Journal of Network and Systems Management manuscript No.
(will be inserted by the editor)

A framework for dynamic configuration of TLS connections based on standards

Javier Pastor-Galindo* · Gabriel López-Millán · Rafael Marín-López · Fernando Pereñíguez-García · Óscar Cánovas

November 2021

Abstract The *Transport Layer Security* (TLS) protocol is widely used for protecting end-to-end communications between network peers (applications or nodes). However, the administrators usually have to configure parameters (e.g., cryptography algorithms or authentication credentials) to establish TLS connections manually. However, this way of managing security connections becomes infeasible when the number of network peers is high.

This paper proposes a TLS management framework that configures and manages TLS connections in a dynamic and autonomous manner. The solution is based on well-known standardized protocols and models that allow providing the necessary configuration parameters to establish a TLS connection between two network nodes. Nowadays, this is required in several application scenarios such as virtual private networks (VPNs), virtualized network functions (VNFs), or service function chains (SFCs). Our framework is based on standard elements of the Software Defined Networking (SDN) paradigm, widely adopted to provide flexibility to network management, such as for the scenarios aforementioned.

The proposed framework has been implemented in a proof of concept to validate the suitability of the proposed solution to manage the dynamic configuration of TLS connections. The experimental results confirm that the implementation of this framework enables an operable and flexible procedure to manage TLS connections between network nodes in different scenarios.

* Corresponding author

Javier Pastor-Galindo, Gabriel López-Millán, Rafael Marín-López
Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
E-mail: {javierpg,gabilm,rafa}@um.es

Fernando Pereñíguez-García
Department of Engineering and Applied Technologies, University Defense Center - Spanish Air Force Academy, 30720, Spain
E-mail: fernando.pereniguez@cud.upct.es

Óscar Cánovas
Department of Computer Engineering, University of Murcia, 30100 Murcia, Spain
E-mail: ocanovas@um.es

Keywords TLS · Management · SDN · YANG

1 Introduction

Nowadays, the deployment of cloud-based datacenters with thousand of virtual network devices [15], the raising of Service Function Chaining (SFC) to provide quick and light network services deployments [14], or the current concept of Software Defined Wide Area Networks (SD-WANs) [25] have posed new challenges for the network administrators due to the complexity associated to these type of networks. Despite this complexity, security is still essential, and the protection of communications must be ensured between the different network peers involved in these types of networks.

To this end, there is a handful set of standard network security protocols currently being used to protect communications in these scenarios, such as Internet Protocol Security (IPsec) or Transport Layer Security (TLS). However, the management of the configuration parameters (e.g., cryptographic material, peers IP addresses, etc.) by these protocols to operate is an error-prone, non-scalable and time-consuming task when performed manually, especially in network scenarios like the ones mentioned above. Given this situation, the research community is called to find ways to dynamically protect the communication channel between a potentially high number of network peers (either nodes or application services) and in a constantly changing environment.

The management of the forthcoming network scenarios is expected to take advantage of the Software Defined Network (SDN) [21] paradigm. SDN proposes to break the traditional concept of networking based on static decisions and fixed architectures by allowing the dynamic configuration of network nodes [35]. This paradigm brings a new opportunity to implement sophisticated mechanisms of protection and defense. This is possible because SDN gives the responsibility of taking decisions to a centralized entity named *SDN controller* (control plane) while the traffic forwarding activity is performed by network nodes (data plane). This architecture provides a flexible and agile management of the network from the administrator's viewpoint, especially when the number of nodes is high and the network is frequently reconfigured [5].

SDN is strategic to address the challenge of protecting communications channels in complex network scenarios [8]. In the context of standardization, some steps in this direction have already been undertaken within the Internet Engineering Task Force (IETF), through the I2NSF and IPSECME working groups [17]. They discuss the framework and security policy data models necessary for the establishment of IPsec security associations in network nodes by a security controller (an SDN controller for the management of security protocols). Regarding the TLS protocol, the IETF has also been working on the definition of generic data models for TLS configurations [41]. Unfortunately, there is no proposal for a standard framework, at the time of writing, to implement SDN-inspired management of TLS security associations.

This paper proposes an standard-based SDN application to manage TLS connections. More specifically, the main contributions of our work are: 1) the definition of a SDN framework to automate the configuration of TLS parameters that allow the establishment of TLS connections between nodes in complex networks integrated by a high number of them; 2) definition of the architecture by selecting the standard technologies that better fit to achieve this goal. As a consequence NETCONF is the protocol selected for network management while YANG language is used for security data modeling; 3) design of novel TLS client and server configuration models, based on the standard YANG by extending the existing IETF model for TLS management; 4) validation of the proposal in realistic use cases: experiments in star topology or full-mesh topology have not been tested at the level of detail we show in our contribution. These two types of scenarios are relevant because full-mesh represents a model for datacenter cases while the star topology can be found, for example, in SD-WANs.

The rest of this paper is organized as follows. Section 2 analyzes related works found in the literature. Section 3 contains the description of some background technologies included in our proposal. Section 4 presents the SDN-based TLS framework designed for the autonomous management of secure connections. Section 5 describes the data model necessary to specify the required TLS configuration parameters. Section 6 describes the implementation of the proof of concept and the testbed, and discusses the results of the experiments performed. Section 7 provides important security considerations associated with the proposed SDN framework. Finally, Section 8 concludes with some key remarks, as well as future research directions.

2 Related work

The need to simplify the management of secure communications channels is a reality in current networks. This has led the industry to develop commercial solutions that centralize the management of secure channels in an entity known as controller or orchestrator.

For example, regarding the provisioning of VPN services, we can find solutions like *Cisco Network Services VPN Orchestrator* [7] that accelerates the establishment and reconfiguration of VPNs to attend to customers demands rapidly.

Similarly, for clusters with a large number of virtualized network devices, several commercial products have arisen to automate the configuration of secure communications channels. Solutions like *Red Hat OpenShift* [33], *IBM Cloud Private* [16] or *AWS Cloud* [2] give to a central entity the responsibility of controlling the establishment of IPsec protected communications among network devices in the data plane.

This line of work has also been applied to the protection of communications in service mesh networks. Despite there are proprietary commercial solutions like *AWS App Mesh* [3], two open source projects are gaining momentum for the deployment of microservice architectures: *Istio* [18] and *Linkerd* [22]. On the one hand, we can find Istio integrated into products like *Red Hat OpenShift Service Mesh* [28] and *Google Anthos Service Mesh* [12], since both companies are contributors of the Istio project. On the other hand, Linkerd is being boosted as incubating project in the Cloud Native Computing Foundation. In any case, it is important to note that both solutions adopt the same approach to protect communications between microservices: a controller is responsible for securing communications using the TLS protocol. However, those solutions are not based on IETF-defined standards for autonomous configuration of security associations.

In parallel to the development of commercial solutions, academic research has paid attention to the application of SDN technology to achieve a flexible and dynamic management of network security. Ranjbar et al. [32] propose a solution that gives an SDN controller the ability to inspect the TLS handshake negotiation between peers operating under its supervision. Thus, the controller can apply security policies to avoid weak cryptographic algorithms, self-signed certificates, etc. Only if the parameters exchanged during the handshake are considered valid, the controller allows the establishment of the TLS connection. The authors of this work use Openflow as southbound protocol to forward TLS handshake packets to the controller where they are analyzed. However, the SDN controller cannot enforce TLS configuration parameters into the peers in that proposal. It can only inspect the plaintext handshake messages and, if necessary, block the establishment of a TLS connection.

Vajaranta et al. [38] develop a secure overlay network following the SDN paradigm. Authors use VXLAN as link-layer virtualization technique and OpenVPN (based on TLS) for encryption. This work develops the components allowing the SDN controller to forward link-layer flows through OpenFlow switches. Unfortunately, the management of secure communications is still expected to be manually configured by the user. This problem is solved in [36], where authors propose a Network Function Virtualization (NFV) orchestrator able to manage an SFC-enabled processing architecture for SSL/TLS encrypted traffic. This work defines a set of *primitives* to apply tunnels, routes, and filters to manage services, but there are no details about configuration policies or data models. A trust model for the management of SSL certificates is also missing.

Authors in [39] propose a solution for centralized management of dynamic IPsec security associations. IPsec perfectly integrates into the SDN architecture since the IPsec protocol operation (data plane) can be decoupled from the protocol used for key management (control plane), such as Internet Key Exchange (IKEv2). In fact, this aspect is also used in [23] to propose a complete framework to manage IPsec security associations. The SDN controller is able to configure, using the standard NETCONF protocol, the network devices to allow them to establish IPsec security associations. Two different operation modes are defined (IKE case and IKE-less case) that can be applied through YANG data models. This work has been adopted by the IETF I2NSF working group and has been recently published as Proposed Standard RFC (RFC 9061) by the IETF [24].

The standardization efforts to increase the flexibility and level of automation when managing secure communications have also reached the TLS protocol. More precisely, authors in [41] define a set of YANG modules for configuring different parameters in the TLS client and server. The model provides YANG *grouping* elements that are expected to be reused by applications using the TLS protocol. However, there is no standard framework describing how use of these models in complex network scenarios.

3 Background

This section provides a basic description of the standard technologies employed to develop the autonomous management framework for TLS security associations proposed in this paper.

3.1 Software-Defined Networks (SDNs)

Software Defined Networking (SDN) [21] is a paradigm in which network control functions are separated from network nodes (e.g., routers or switches) and centralized in an entity called *SDN controller*. Moreover, a new higher level is defined to abstract administrators from knowing the details of the underlying network. For this reason, SDN infrastructures are said to be composed of three planes: *application*, *control* and *data*.

The data plane is the lowest layer and where network nodes reside. They are devices that do not take decisions on their own but communicate with the SDN controller (located in the control plane) through the *southbound interface* to receive instructions indicating how to manage network traffic. Examples of southbound protocols are OpenFlow [27], NETCONF [10] or SNMP [11].

The SDN controller, in turn, receives from administrators (located in the application plane) security policies to govern the network. This communication happens through the *northbound interface*. This layer is implemented through business software and is out of the scope of this contribution.

3.2 NETCONF and YANG

Network Configuration Protocol (NETCONF) [10] is a southbound protocol developed by the IETF (*Internet Engineering Task Force)* for network management. It follows a typical client-server model where the NETCONF client can install, manipulate and remove configurations on the NETCONF server that is expected to reside in data plane network nodes.

To protect the communication between client and server, NETCONF relies on using a secure transport protocol such as SSH [9] or TLS [4]. NETCONF uses a communication model based on Remote Procedure Call (RPC). According to this model, requests and responses are represented with *rpc* and *rpc-reply* messages, respectively. These RPC messages can be used to invoke an operation (encoded in the Extensible Markup Language - XML [40]) to manage a device. For example, the *edit-config* operation applies a concrete configuration on a device.

NETCONF uses a language called YANG [6] to model configuration data exchanged between client and server. A YANG data model works hierarchically and follows a tree structure of data nodes. YANG defines different types of data nodes. For example, the type *container* is used for nodes residing in intermediate levels of the tree that contain other nodes, thus grouping data logically. Conversely, the type *leaf* is used for end nodes containing simple data (e.g. a numeric value). Using different types of nodes, the designer is able to customize the desired structure and create a YANG data model. YANG modules can be translated into an equivalent

XML syntax called YANG Independent Notation (YIN). The catalog of current standardized YANG models can be found in [42].

3.3 Transport Layer Security (TLS)

Several secure protocols have been developed to ensure integrity, confidentiality, and authentication to communication channels. Without a doubt, *Transport Layer Security* (TLS) [34] is one of the most widely used protocols in current networks for this purpose. Although its usage has become popular to protect web traffic [13], TLS is also employed to implement other security services like, for example, VPNs [30].

TLS establishes a secure connection between two peers (i.e. network nodes) to protect application data exchanged among them. TLS assigns the role of *TLS client* to the peer initiating the TLS connection, while the other is designated as *TLS server*. At the time of writing, the last version of TLS is 1.3, which is the one considered in the paper.

TLS 1.3 is composed of two basic components: the one for establishing a secure connection between two nodes (*Handshake Protocol*), and the other for protecting data (*Record Protocol*). The former is started by the TLS client and allows both TLS client and server to mutually authenticate each other, as well as to negotiate the keys needed by the cryptographic algorithms (used by the Record Protocol) to implement confidentiality and integrity services for the data protection. Among the handshake modes supported by TLS 1.3, in the context of this paper we consider the basic one (called *Full TLS Handshake*) since it is the default mode.

4 The standard framework for TLS autonomous management

This section describes the framework to automate the establishment of TLS connections between (network) nodes using standard protocols and interfaces. Below we describe the architectural components, the operation, and the procedure designed to assist the configuration process of TLS connections.

4.1 Architecture

The solution is based on the general SDN architecture. It is composed by two types of entities: the *Node*, representing an entity in the network needing to establish a TLS connection (data plane); and the *Security Controller (SC)*, acting as the central entity in charge of provisioning nodes with the necessary configuration parameters (control plane).

Figure 1 illustrates the entities defined in this framework. For completeness, this figure depicts the application plane, although it is outside the scope of this work. As expected, we find the Security Controller in the control plane, which maintains a complete view of the network located in the data plane and holds the *TLS Security Policies* that must be applied to the nodes residing in the data plane. Communication between the entities residing in the control (Security Controller) and data (nodes) planes may happen through a dedicated high-speed management

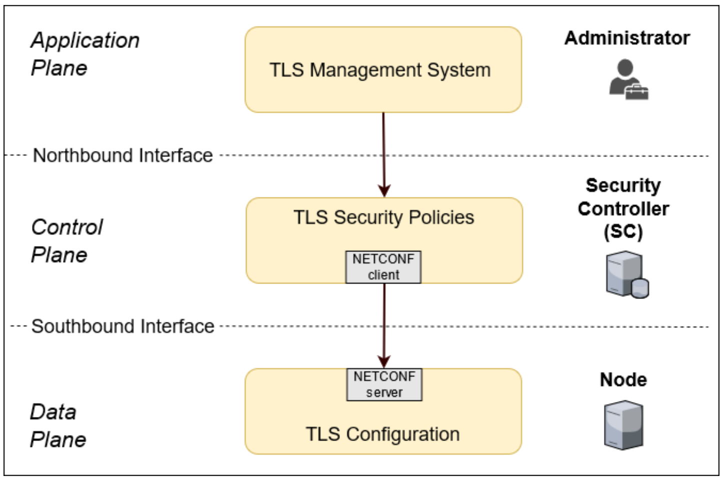


Fig. 1 Proposed architecture

network. However, other schemes are possible, like those where the controller communicates with data plane entities through the data network.

We assume that the nodes in the data plane need to establish TLS connections to exchange data securely. For example, they are requested to establish a SD-WAN, which implies the establishment of TLS connections with a customer's branch network; to protect communications between virtualized nodes belonging to a SFC in a cloud; or to protect information exchanged between microservices in a service mesh network.

It is also important to note that the nodes deploy a typical TLS implementation, which is divided into two main layers: the TLS handshake layer, which is the protocol that allows establishing the TLS connections, and the TLS record layer, which is in charge of protecting data traffic based on the cryptographic material and algorithms exchanged during the TLS handshake. As such, the main task of the Security Controller in this architecture is to send the required configuration information that allows the node to run the TLS handshake to establish the TLS connections.

The interaction between the Security Controller and the nodes will be carried out through the *southbound interface*, so that the former will provide the latter with proper parameters. In this particular case, we have chosen NETCONF [10] to implement this interface. NETCONF, together with the standard modeling language YANG, is considered more flexible, modular and extensible than other alternatives, such as SNMP, to conveniently represent the required security information.

We integrate the NETCONF client functionality in the Security Controller, while the NETCONF server role is assigned to the nodes. Therefore, the nodes require a NETCONF server agent that is able to translate the XML information received from the controller (based on the YANG model) into specific TLS implementations (for example OpenSSL [29], OpenVPN [30], etc.).

As a consequence, a standard YANG model for the definition of TLS configuration parameters is needed. This model must include cryptographic material and TLS roles (client or servers), endpoints contact information, and information about how to manage the TLS connection. In summary, all the information that TLS handshake requires to operate properly to establish a TLS connection. A description about the proposed model can be found in Section 5.

In order to create and maintain the TLS-based secure connections between the nodes, the Security Controller assumes a trust relationship with each node. In the particular case of the proposed solution, this trust relationship is implicitly inherited from the *Secure Transport Layer* used by NETCONF, which typically results in the establishment of a secure channel between the controller and the nodes through SSH (by default) or TLS.

4.2 General operation

In the following, we detail the general framework operation to automatize the establishment of TLS connections. Despite our proposal is designed to support any number of nodes or topologies, we provide a simple scenario for the sake of clarity, where a Security Controller has to configure a TLS connection between two nodes. We assume that:

- The Security Controller has already received from the administrator (through the Northbound Interface) the Security Policies describing the network nodes requiring TLS connection. As mentioned earlier, this process is out of the scope of this work.
- The Security Controller knows the contact information of the nodes (i.e., IPv4 or IPv6 addresses).
- The nodes are shipped with the credentials necessary to establish a secure transport for the NETCONF session with the Security Controller. Typically, this is achieved with an RSA key pair to set up an SSH connection.
- The Security Controller has an implementation of a NETCONF client, and nodes has the software to run a NETCONF server. They should be able to manage configurations and verify YANG models.
- The nodes have the proper support to create TLS connections (when acting as TLS client) and receive them (when deployed as TLS server).
- The key material used by TLS in the data plane can be either directly provided by the controller or locally stored in the node (for example in a keystore or hardware security module). The YANG model used in NETCONF should allow both options.

The general workflow is divided into two parts: a) the Security Controller configures the first node as TLS server; b) the Security Controller configures the second node as TLS client to start the TLS handshake. Note that when the latter starts the TLS handshake, the TLS server must be prepared to negotiate the TLS connection. The details about the workflow are depicted in Figure 2, and they are as follows:

1. According to the TLS Security Policies, the Security Controller (SC) decides the role played by each node: TLS server (N1) or TLS client (N2).
2. The SC sends the required TLS server configuration for N1 based on the standard YANG model. It includes information about the IP and port for listening

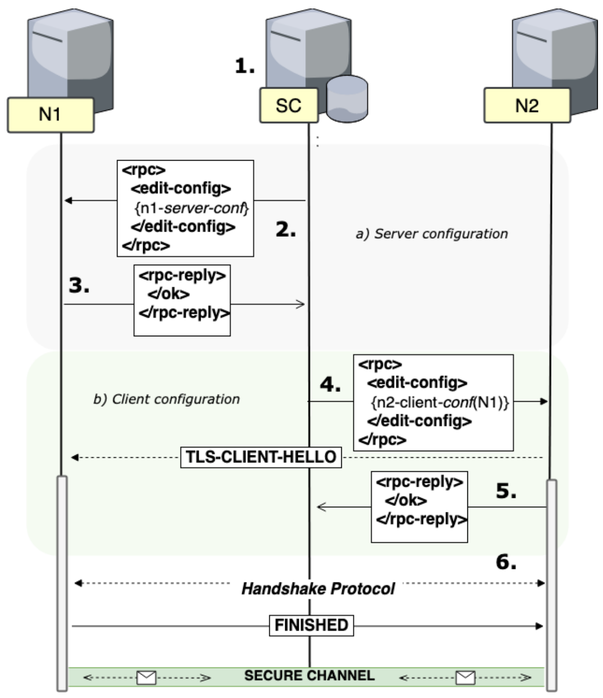

N1
SC
N2
1.
<rpc> <edit-config> {n1-server-conf} </edit-config> </rpc>
2.
a) Server configuration
3.
<rpc-reply> </ok> </rpc-reply>
<rpc> <edit-config> {n2-client-conf(N1)} </edit-config> </rpc>
4.
b) Client configuration
TLS-CLIENT-HELLO
<rpc-reply> </ok> </rpc-reply>
5.
6.
Handshake Protocol
FINISHED
SECURE CHANNEL

Fig. 2 Configuration of a TLS connection between two nodes.

to incoming TLS handshake and information about the required cryptographic material (e.g., shared secret or X.509 certificate and private key, as well as Certification Authority information) depending on the Security Policies. The Security Controller can explicitly include this key material in the configuration sent to the node, or information about where to locate that in the own node.

3. The NETCONF server agent at N1 translates the XML received from the Security Controller into system-specific TLS commands to run a TLS server with the provided configuration. If the configuration is successfully applied, it responds with a *RPC OK* response.
4. The SC sends now the required TLS client configuration for N2 based on the standard YANG model. It includes information about the TLS server's IP and port for outgoing TLS connections and the required cryptographic material (e.g., shared Secret or X.509 certificate, private key, and Certification Authority information) depending on the Security Policies.
5. Once the configuration is applied and validated in N2, it may start the TLS handshake with N1 and responds with a *RPC OK* to the SC. Note that N2 is informing the SC with the *RPC OK* response that the configuration has been applied properly. However, the responsibility for eventually establishing the TLS connection falls on N2, who may decide to start the TLS handshake just before sending the *RPC OK* (between steps 4 and 5 as shown in Figure 2) or further on (when required). In any case, it is an implementation decision of the node.

6. The TLS security channel will be established once the TLS handshake finishes, being used to protect the traffic between N1 and N2.

It is worth noting that when a node acting as TLS client must establish several TLS connections with different nodes acting as TLS servers, the controller can send multiple TLS client configurations in the same *edit-config* message to the same node. This may happen, for example, in a mesh scenario that involves n nodes, where N1, N2,...,Nn must establish a full TLS mesh. In this case, the Security Controller would send the required $i - 1$ TLS client configurations to Ni in a single *edit-config* message. If the configuration was successfully applied, Ni would be able to start TLS handshakes respectively with the rest of the nodes (which were previously configured and acting as TLS servers). Particularly, the client configurations of the nodes for a full mesh of n nodes would be as follows:

– N2: {*n2-client-conf(N1)*}
– N3: {*n3-client-conf(N1,N2)*}
– Nn: {*nn-client-conf(N1,N2,..., Nn* $- 1$*)*}

Where *ni-client-conf(N1,...,Ni* $- 1$*)* represents the TLS client configuration for Ni to connect to servers in N1, N2 to N$i - 1$.

Finally, it is important to note that though Figure 2 is used for describing the process to configure two nodes, the same exchange is performed at NETCONF level to update or remove a configuration. Indeed, to set up new configurations in two nodes the Security Controller sends *edit-config* NETCONF messages with the operation *merge*. If replacement of an existing inter-node connection is required, the Security Controller has to send a new *edit-config* NETCONF message with the *replace* operation and including the updated configuration to the nodes involved. First, the controller has to update the node acting as TLS server. Then, this node will shut down the established connection, apply the new configuration and wait for the client node to establish the new connection. Then, the controller has to update the configuration of the TLS client node with another *edit-config* with the *replace* operation. The client applies the new configuration (it was shut down by the server) and starts a new TLS connection with this configuration.

In case of needing to remove nodes, something similar happens. If the Security Controller wants to evict one of the nodes from the mesh or the star topology, it must send an edit-config message with the *delete* operation, and information pointing out the configuration that wants to be removed in the node acting as TLS server. The node then deletes the configuration and shuts down the TLS connection with the node acting as TLS client. Then, the Security Controller may now send the corresponding delete operation to the node acting as TLS client.

Therefore, adding, updating and removing configurations in two nodes involve the same number of exchanges. That is, they all involve the same number of *edit-config* NETCONF messages. The only substantial change is the information carried in these messages. The *create* operation includes a complete configuration to establish a TLS channel. The *replace* operation only carries those parameters that want to be updated and the *delete* operation carries the identifier of the particular configuration to be removed.

5 Modelling of TLS client and server configuration

This section describes the YANG data model designed to configure a TLS connection between two nodes, one acting as TLS client and the other as TLS server.

5.1 Relevant parameters

After analyzing the TLS standard specification [34] and some of the current widely used open source implementations, we have identified four different groups of configuration parameters essential for the creation of secure TLS connections. These are the following:

- *Connection data.* Group of parameters containing data for the transport layer to initiate the TLS connection. It includes:
  - *Server-port*: listening TCP port for the reception of the TLS connection. It is necessary for both TLS client and server configurations. In the server configuration, this piece of information is used for binding the TLS service, whereas it indicates to the TLS client the server port where to establish the TLS connection.
  - *Server-address*: IP address of the TLS server. Only necessary for TLS client configurations.
- *Client/Server Identity.* It contains the information required by a peer to authenticate against the remote peer during the establishment of the TLS session. For example, when configuring the TLS client, this information refers to the identity of the client.
  - *Auth-algorithm*: Authentication method, RSA or PSK.
  - *PSK value*: Shared key in case PSK authentication method is used.
  - *Private-key*: its interpretation depends on the *Auth-algorithm.* If RSA is used, it would be *RSAPrivateKey.*
  - *Public-key*: its meaning is also determined by the *Auth-algorithm.* If RSA, it would be *RSAPublicKey.* It is optional if certificate is used.
  - *Certificates*: X.509 certificate that contains the identity and the *Public-key* associated with the *Private-key.*
- *Client/Server Authentication.* This group of parameters contains information required to authenticate the remote peer in the TLS connection:
  - *ca-certs*: set of trusted CA certificates.
  - *client/server-certs*: when configuring a TLS server, it contains the set of trusted client certificates. Conversely, the TLS client configuration contains the set of trusted server certificates.
- *Handshake Parameters.* Group of parameters for customizing different features of TLS communication:
  - *TLS version*: version that the server prefers, for example *tls1.2* or *tls1.3.* In the context of this paper, we use tls1.3.
  - *Cipher-Suites*: cryptographic suites acceptable and supported by the server. They include key exchange, encryption and integrity algorithms.
  - *KeepAlives*: time configuration for detecting peer connection alive. For example, time to wait for a connection before considering it is down and the number of attempts.

Table 1 shows a configuration example for a TLS client and server, considering the security parameters previously described.

	Server configuration	**Client configuration**
Connection data	*Server-port = 666*	*Server-address = 10.0.0.2* *Server-port = 666*
Server/Client Identity	*Auth-algorithm = rsa* *Private-key = N1 private key* *Certificates = N1 X.509 certificate*	*Auth-algorithm = rsa* *Private-key = N2 private key* *Certificates = N2 X.509 certificate*
Client/Server Authentication	*ca-certs = X.509 CA certificate* *client-certs = null*	*ca-certs = X.509 CA Certificate* *server-certs = null*
Handshake Parameters	*TLS version = tls1.3* *Cipher-Suites = ecdhe-rsa-with-aes-256-gcm-sha384* *KeepAlive-Max/Attempts: 30/3*	*TLS version = tls1.3* *Cipher-Suites = ecdhe-rsa-with-aes-256-gcm-sha384* *KeepAlive-Max/Attempts: 30/3*

Table 1 Example TLS configuration parameters

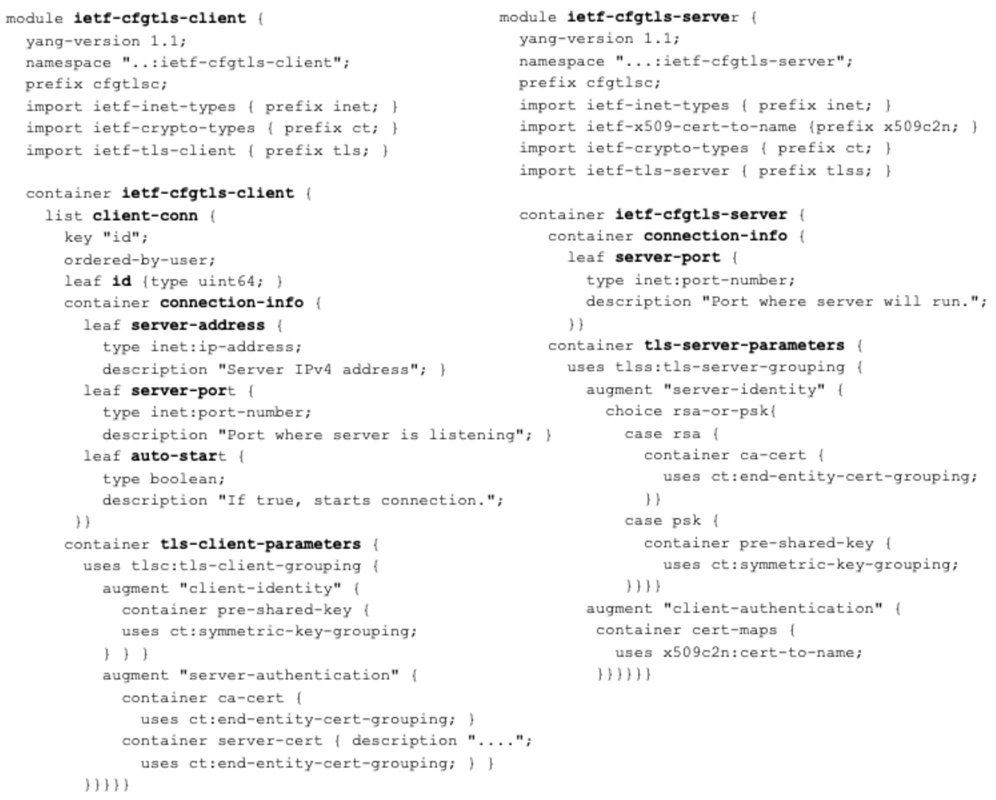
5.2 YANG data model

Since NETCONF is used to implement the communication interface (i.e., southbound) between the security controller and the nodes, and NETCONF transports data in XML format, we need to define some template representing the possible data format.

In order to represent the TLS configuration parameters described in Section 5.1, we have defined two different YANG modules: one for the TLS server configuration and the other for the TLS client configuration. These models play the role of TLS applications that make use of the YANG groupings for TLS defined in [41], which, at the time of writing, is being standardized by the IETF. In fact, this draft aims to provide the YANG elements (containers, types, etc.) to services willing to automatize the establishment of TLS connections.

Taking as reference this model, we have developed the new ones compliant with the requirements of our framework (Figure 3). The additions can be summarized as follows:

– Regarding the client module, we define a list of TLS client connections (not a single one). For each connection, the required information to connect with the TLS server is specified (*server-address* and *server-port*). We have also added the element (*auto-start*) to indicate whether the TLS connection must start immediately after applying the configuration. Furthermore, we augment the *tls-client-grouping* defined in [41] to support pre-shared key (PSK) or RSA authentication and X.509 certificates for trusted entities.
– Regarding the server module, we have included the information about the TLS server's listening port in the node. We also need to augment the YANG model to provide PSK and RSA authentication support. Besides, it allows the client authentication by client identity by means of the *cert-to-name* element.



```
module ietf-cfgtls-client {
  yang-version 1.1;
  namespace "..:ietf-cfgtls-client";
  prefix cfgtlsc;
  import ietf-inet-types { prefix inet; }
  import ietf-crypto-types { prefix ct; }
  import ietf-tls-client { prefix tls; }

  container ietf-cfgtls-client {
    list client-conn {
      key "id";
      ordered-by-user;
      leaf id {type uint64; }
      container connection-info {
        leaf server-address {
          type inet:ip-address;
          description "Server IPv4 address"; }
        leaf server-port {
          type inet:port-number;
          description "Port where server is listening"; }
        leaf auto-start {
          type boolean;
          description "If true, starts connection.";
       }}
      container tls-client-parameters {
        uses tlsc:tls-client-grouping {
          augment "client-identity" {
            container pre-shared-key {
            uses ct:symmetric-key-grouping;
          } } }
          augment "server-authentication" {
            container ca-cert {
              uses ct:end-entity-cert-grouping; }
            container server-cert { description "....";
              uses ct:end-entity-cert-grouping; } }
        }}}}}
```

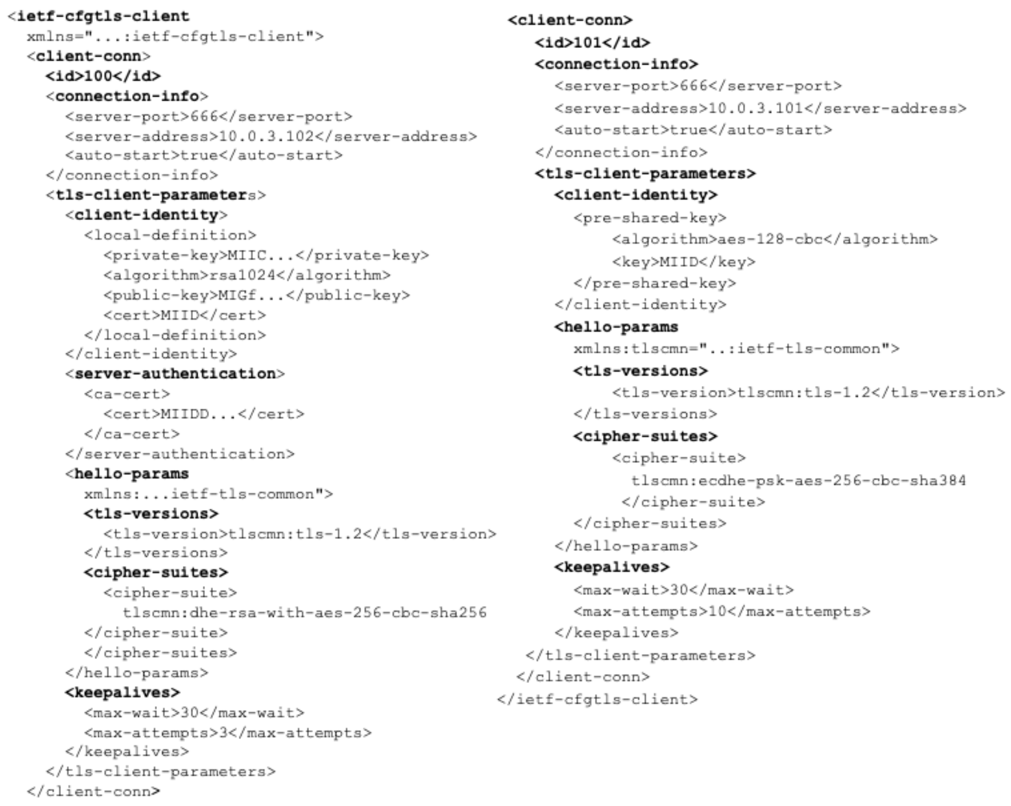
```
module ietf-cfgtls-server {
  yang-version 1.1;
  namespace "...:ietf-cfgtls-server";
  prefix cfgtlsc;
  import ietf-inet-types { prefix inet; }
  import ietf-x509-cert-to-name {prefix x509c2n; }
  import ietf-crypto-types { prefix ct; }
  import ietf-tls-server { prefix tlss; }

  container ietf-cfgtls-server {
     container connection-info {
       leaf server-port {
         type inet:port-number;
         description "Port where server will run.";
       }}
     container tls-server-parameters {
       uses tlss:tls-server-grouping {
         augment "server-identity" {
           choice rsa-or-psk{
             case rsa {
               container ca-cert {
                 uses ct:end-entity-cert-grouping;
               }}
             case psk {
               container pre-shared-key {
                 uses ct:symmetric-key-grouping;
             }}}}
         augment "client-authentication" {
          container cert-maps {
            uses x509c2n:cert-to-name;
          }}}}}}
```

Fig. 3 TLS client and server application models

It is worth noting that the definition of required elements such as TLS version, cipher-suites, keep-alive information or locally stored key material is already provided by the groupings provided by the models *ietf-tls-client* and *ietf-tls-server* in [41].

In order to illustrate the use of the model defined in this work, Figure 4 shows an example of a TLS client configuration including information about two connections. Connection *100* is based on RSA and includes information about client public and private key, TLS versions, ciphersuite and keep-alive. Alternatively, connection *101* is based on PSK.

Finally, it is important to note that these modules have to be loaded in both the controller and nodes to enable a common language for interpreting the configurations. YANG models play a key role, not only because they set the format of message content, but they also allow the nodes to validate the settings instantiated by the Security Controller.



```
<ietf-cfgtls-client
  xmlns="...:ietf-cfgtls-client">
  <client-conn>
    <id>100</id>
    <connection-info>
      <server-port>666</server-port>
      <server-address>10.0.3.102</server-address>
      <auto-start>true</auto-start>
    </connection-info>
    <tls-client-parameters>
      <client-identity>
        <local-definition>
          <private-key>MIIC...</private-key>
          <algorithm>rsa1024</algorithm>
          <public-key>MIGf...</public-key>
          <cert>MIID</cert>
        </local-definition>
      </client-identity>
      <server-authentication>
        <ca-cert>
          <cert>MIIDD...</cert>
        </ca-cert>
      </server-authentication>
      <hello-params
        xmlns:...ietf-tls-common">
        <tls-versions>
          <tls-version>tlscmn:tls-1.2</tls-version>
        </tls-versions>
        <cipher-suites>
          <cipher-suite>
            tlscmn:dhe-rsa-with-aes-256-cbc-sha256
        </cipher-suite>
        </cipher-suites>
      </hello-params>
      <keepalives>
        <max-wait>30</max-wait>
        <max-attempts>3</max-attempts>
      </keepalives>
    </tls-client-parameters>
  </client-conn>
```

```
  <client-conn>
    <id>101</id>
    <connection-info>
      <server-port>666</server-port>
      <server-address>10.0.3.101</server-address>
      <auto-start>true</auto-start>
    </connection-info>
    <tls-client-parameters>
      <client-identity>
        <pre-shared-key>
            <algorithm>aes-128-cbc</algorithm>
            <key>MIID</key>
        </pre-shared-key>
      </client-identity>
      <hello-params
        xmlns:tlscmn="..:ietf-tls-common">
        <tls-versions>
            <tls-version>tlscmn:tls-1.2</tls-version>
        </tls-versions>
        <cipher-suites>
            <cipher-suite>
              tlscmn:ecdhe-psk-aes-256-cbc-sha384
             </cipher-suite>
        </cipher-suites>
      </hello-params>
      <keepalives>
        <max-wait>30</max-wait>
        <max-attempts>10</max-attempts>
      </keepalives>
    </tls-client-parameters>
  </client-conn>
</ietf-cfgtls-client>
```

Fig. 4 TLS client example configuration

6 Implementation and experimental results

In order to test the validity of this proposal, we have implemented a proof-of-concept scenario to extract experimental results of the performance of the framework. The source code of the scenario, implementation of the tests, network traces, and experimental results are publicly available[1].

6.1 Deployed scenario

As shown in Figure 5, the SDN scenario deployed consists of N nodes and a single Security Controller (SC). Only a physical equipment has been necessary for the proof-of-concept because the topology is virtually deployed with *Docker*[2]. The server features 2 Intel Xeon E5-2630 v4 CPUs (a total of 20 cores at 2.2 GHz) and

[1] https://github.com/javier-pg/sysrepo-cfgssl
[2] https://www.docker.com/

80 GB of DDR4 memory at 2400 MHz. In relation with the link capacity, which is based on RAM speed, it supports 11.5 Gbps. The elements considered in the solution are presented below:

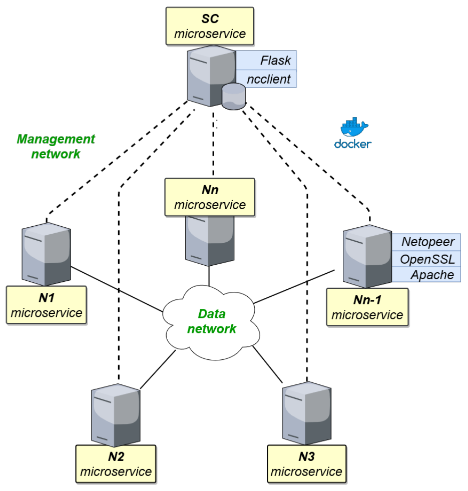


Fig. 5 Deployed virtual SDN scenario

- **SC**: It is the container deploying the security controller, whose functionality is implemented in Python. It manages NETCONF sessions with the nodes by sending TLS configurations according to the YANG model. Particularly, the NETCONF client is implemented with *ncclient*[3]. We have also implemented a simple registration process for the nodes to merely notify they are ready to receive the configuration at a specific network address and start the experiments. This is REST service implemented with *Flask*[4].
- **Ni**: It is a container implementing a node. With *Docker*, we can automatically scale this container to n replicas. The node will register in the SC with a REST message to trigger the NETCONF session for the southbound communication. Once it is contacted by the SC, and depending on the configuration received, it will launch a TLS server or establish TLS connections as a client.
  The NETCONF server functionality is implemented with *Netopeer*[5], which allows loading our proposed YANG models and scheduling callbacks on the arrival of new configurations. When the node receives a server configuration, it runs a TLS server implemented with *Apache 2.4*[6]. Alternatively, when a client configuration is received, TLS connections are established making use of *OpenSSL 1.1.1*[7].

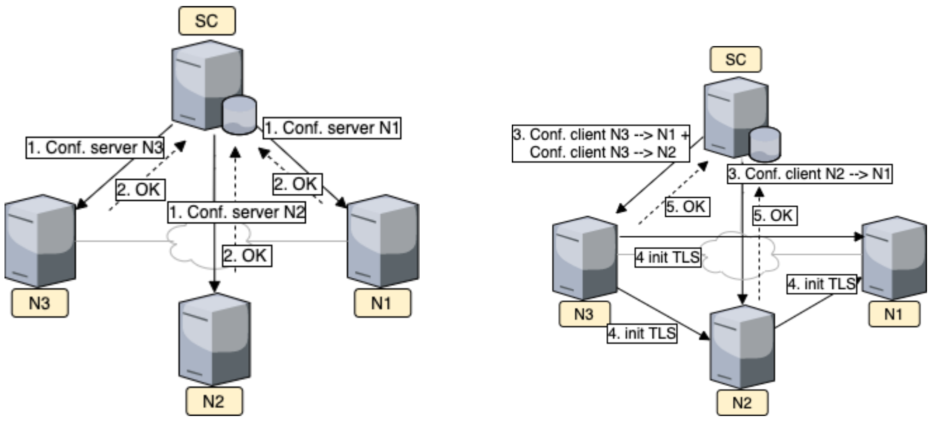
[3] https://pypi.org/project/ncclient/
[4] https://flask.palletsprojects.com
[5] https://github.com/CESNET/netopeer
[6] https://httpd.apache.org
[7] https://openssl.org

- **Management network**: It is the virtual management network that interconnects the SC container with the different Ni containers to exchange configuration information by means of NETCONF messages.
- **Data network**: It is the virtual network which interconnects the Ni containers. For this reason, each node container incorporates a second interface. It is the network where the TLS associations are established.

6.2 Implemented use cases

From the scenario explained above, we have defined three use cases in which we make use of the SC to configure TLS connections between the nodes. They represent typical uses cases similar to those discussed in Section 2 and permit us to evaluate different scenario workloads. Particularly, we consider these use cases to be *tasks* to be performed by the SC.

- **Task 1: To form a mesh of n nodes**. In this task, the SC is committed to configure a mesh of n nodes from scratch, an application scenario that we can find in mesh networks composed by microservices and protected using TLS, like Istio. This task has been implemented in the SC in two parts: firstly, the SC sends the specific TLS server configuration to the n nodes simultaneously so they are ready to process the TLS handshake that will start afterward; secondly, once the SC has configured the nodes as TLS servers, it sends the TLS client configurations. More specifically, the configurations sent to node Ni includes the N$i-1$ TLS client configurations to connect to the TLS servers from node N1 to N$i-1$.

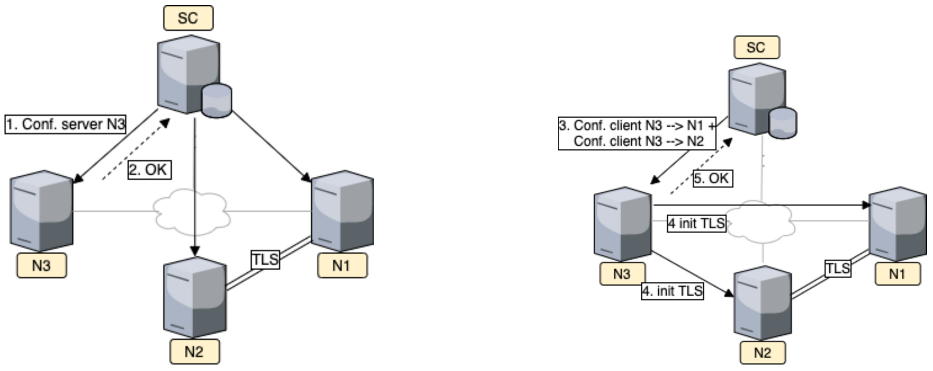


(a) Server configuration (b) Client configuration

Fig. 6 Task 1: To form a mesh of N nodes

Figure 6 exemplifies this task for n=3. Firstly, the SC configures simultaneously N1, N2 and N3 as TLS servers (steps 1 and 2 in Fig. 6a), so they become ready to process the TLS handshake with the rest of nodes. Once the SC gets the confirmation that nodes are configured as TLS servers, the SC proceeds to configure, in parallel, N1, N2 and N3 with the corresponding TLS client configuration (steps 3 to 5 in Fig. 6b). In particular, N2 receives the TLS configuration to start a TLS handshake with N1, and N3 is configured to start the TLS connection with N1 and N2.

– **Task 2: To add a new node to an existing mesh**. In this task, a new node Nn is incorporated to an existing mesh of size n-1. This case covers, for example, the case where a new replica needs to be included in an existing mesh of microservices for scalability purposes. Firstly, the node Nn must be configured as TLS server so that it is prepared in case another node joins the mesh later. Secondly, the SC sends the configuration as TLS client so that Nn can start the TLS handshake with the n-1 nodes in the existing mesh.

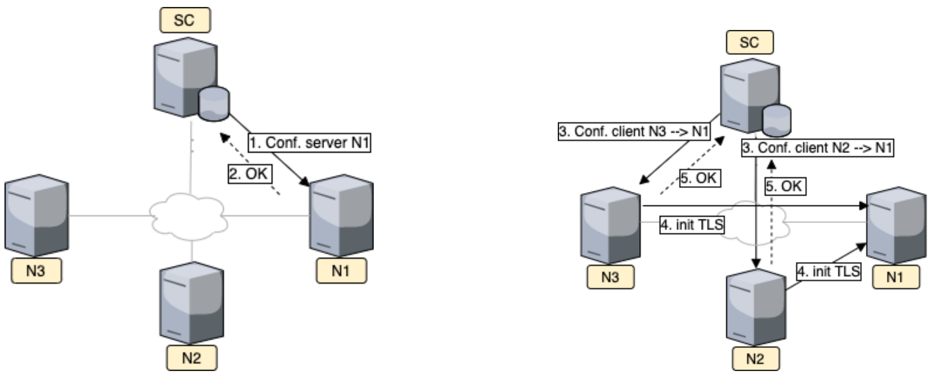


(a) Server configuration (b) Client configuration

Fig. 7 Task 2: To add a new node to an existing mesh

Figure 7 shows an example with n=3. N1 and N2 already form a simple mesh and N3 should establish a TLS connection with both nodes to complete a mesh of size n=3. Firstly, the SC configures the N3 as TLS server (steps 1 and 2 in Fig. 7a). Once the N3 is configured as a server, the SC configures N3 with the TLS client configurations (steps 3 to 5 in Fig. 7b) to establish TLS connections with N1 and N2.

– **Task 3: To form a star topology of n nodes**. In this task, one of the Ni nodes (referred as central node) acts as TLS server and the rest of n-1 nodes as TLS clients, like in some SD-WAN scenarios. This task has been implemented by the SC in two parts: firstly, it configures the central node as TLS server and secondly, it configures the rest of nodes as TLS clients so they can connect with the central node.



(a) Server configuration (b) Client configuration

Fig. 8 Task 3: To form a star topology of N nodes

Figure 8 shows an example with n=3. N1 is selected as TLS server (central node), and N2 and N3 act as TLS clients so that they can both establish a TLS connection with the central node. Firstly, the SC configures N1 as TLS server (steps 1 and 2 in Fig. 8a). Then, the SC configures N2 and N3 in parallel with the TLS client configuration (steps 3 to 5 in Fig. 8b) to establish TLS connections with N1.

6.3 Testbeds and experimental results

We have conducted three types of experiments to evaluate the system for the previously discussed tasks. For each one, we incrementally vary the number of nodes involved, simulating from light to more stressful situations for the SC. When configuring a TLS server, the SC built the payload (XML) containing the *server-port*, the *auto-start* activated, a *RSA-2048 server certificate* as identity, a trusted *CA certificate* for client authentication, two *tls-versions* supported, and two *cipher-suites* suggested. In the case of the TLS client configuration, the SC included a list of needed TLS connections that specified, in each entry, the *server-port*, the *server-address*, the *auto-start* flag activated, a *RSA-2048 client certificate* as identity, a trusted *CA certificate* for server authentication, two *tls-versions* supported, and two *cipher-suites* suggested.

The ultimate goal of these experiments is to measure the *service time* of the controller, that is, how long it takes for the SC to configure the TLS-based security of the network, depending on the total number of nodes (n). In particular, the *service time* to configure a single node comprises from the first outgoing message sent by the SC to the port 830 of the node (NETCONF session establishment), to the last message received in the controller from that port (end of NETCONF session). Therefore, the interval includes not only configuration operations, but also the establishment and termination of the SSH session being used by NETCONF. When configuring a network of nodes, the *service time* covers from the first NETCONF message sent to the first node to the last message received from the last node. Note that the timestamps are always obtained within the SC.

- **Experimental results for Task 1**. In the first type of experiments, we have measured the *service time* of the SC to configure TLS in mesh networks of different sizes (that is, the time elapsed between step 1 and step 5 of Figure 6). Particularly, for each size we run 20 executions, from $n = 5$ to $n = 80$ with steps of 5 nodes, to observe the behavior and scalability of the system as the size increases. Figure 9 shows the average time required by the SC to complete the configuration (Y-axis) in seconds of a mesh of n nodes (X-axis). For example, the SC can agilely configure TLS-based mesh networks of 15, 45, and 80 nodes in approximately 1, 4, and 11 seconds, respectively. In this sense, although the southbound communication with nodes is mainly executed in parallel (we actually employ a pool of threads), the *service time* is neither uniform nor linear. Therefore, we infer from the quadratic tendency that the SC experiences an increasing and no-linear load as the size of the mesh rises (e.g., nodes queuing on the thread pool, processing of parallel connections, and congestion of the control network).

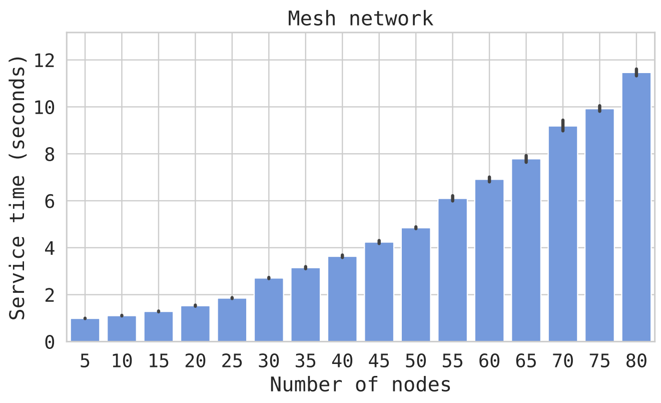


Fig. 9 Service time required by the SC to configure a mesh network of n nodes

- **Experimental results for Task 2**. In this case, we have calculated the *service time* required by the SC to resize a mesh network by including a new node. Concretely, we have run 20 tests in which the SC scales a mesh network from $n = 1$ to $n = 80$, configuring a new node as TLS server and client (following the phases depicted in Figure 7). Figure 10 presents the average time (Y-axis) the SC takes to configure the node Ni (X-axis) when a mesh of size $i - 1$ is already established. As observed, N1 is particularly fast for being only configured as TLS server (and not as client). The SC configures the rest of the nodes employing approximately between 0.8 seconds and 1.4 seconds.
  For example, it can dynamically enforce the security parameters for 44 TLS connections (n=45) in just 1.2 seconds. This logarithmic tendency implies that sending only one NETCONF message with multiple client configurations is a good choice to amortize the cost of sending a NETCONF message through a SSH connection and the impact of increasing the number of configurations affects less than sending NETCONF messages over the network.

In any case, as shown in Figure 7b, step 3, for a new node i, it is configured by the SC with $i-1$ client configurations, increasing this way the payload size of the NETCONF message when i increases so that the time to complete the configuration process.
It is worth mentioning that the fixed offset of 0.7 seconds in the Y-axis corresponds to the latency of initiating the NETCONF SSH session, the NETCONF operations to send the server configuration, and closing the NETCONF session. In this sense, the variable factor in the service time of the nodes only resides on the incremental size of the client configuration (and the fragmentation in more TCP messages), which seems to have a proportionally low influence compared to the rest of the operation.

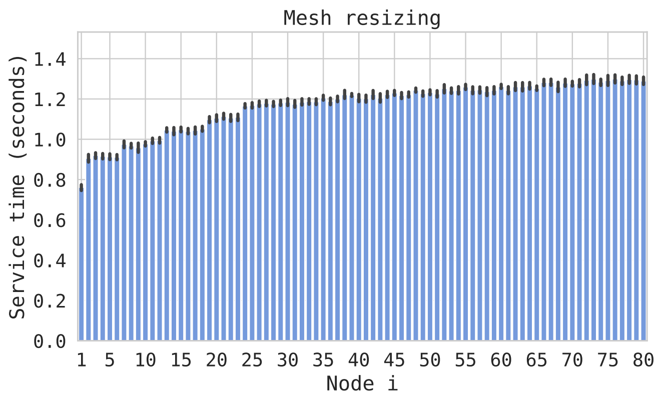


Fig. 10 Service time required by the SC to configure Node i within a mesh of $i-1$ nodes

– **Experimental results for Task 3**. Finally, we have evaluated the efficiency of the proposed system to guarantee the security configuration of a star topology. We have measured the time required by the SC to complete the configuration of a central node as TLS server and the rest of nodes as TLS clients (as shown in Figure 8), forming star topologies from size $n = 5$ to $n = 80$, with steps of 5 nodes. The average times of 20 tests per star topology are presented in Figure 11, revealing that the SC can configure TLS in star topologies of 25, 45, 60, and 80 nodes in approximately 2, 4, 6, and 9 seconds, respectively. The quadratic function is similar to the one obtained for Task 1, sharing close *service times* until $n = 45$ approximately. The reason of this tendency is that, from this size onward, the load of the system (mainly the SC) increases and impacts the general performance due to the number of nodes establishing simultaneously TLS connections.

The aforementioned service times allow us to evaluate the centralized TLS configuration of nodes through the control plane, which is the main contribution of the framework. However, for the sake of completeness, we have launched some experiments to demonstrate that nodes in the data plane, once configured, are able to establish TLS connections. We have deployed and configured star networks of

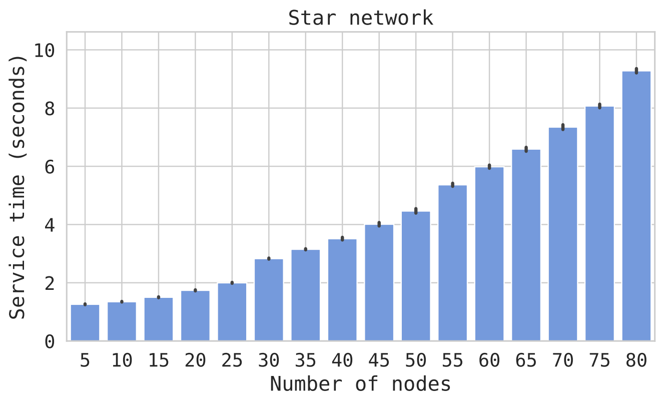


Fig. 11 Service time required by the SC to configure TLS in a star topology of n nodes

5 nodes and up to 80 nodes (with steps of 5 nodes) to measure the time required to complete all the secure connections. All nodes are configured by the controller with the auto-start flag activated, so they automatically start the TLS handshake with the central node when the configuration is received.

Figure 12 shows the results of the experiments, exposing that the TLS protection of a star network takes approximately one second for 30 nodes, two seconds for 50 nodes, and five seconds for 80 nodes. The TLS times cover the whole list of handshakes in the data plane and are decoupled from the configuration process of the control plane. The temporal data and traffic captures are publicly available on the project repository.

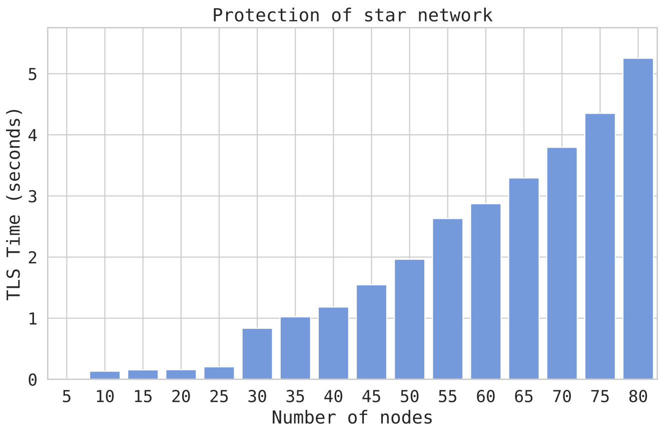


Fig. 12 TLS time required for n nodes to protect the data plane of star network

6.4 Overview and discussion on experimental results

Once we have shown the behaviour of this proof-of-concept for different tasks, we discuss some implications from the point of view of the application scenarios where they might be applied.

In relation to mesh networks, if the goal is to be as fast as possible configuring a mesh network of TLS connections, the best option would be to configure TLS in a mesh network of "n" nodes simultaneously (Task 1), rather than gradually configuring from the first node to the last node (Task 2). In that way, we take advantage of the parallel configuration of nodes. However, the no-linear performance of the former approach may not be suitable for environments where resources are limited. In that case, the resizing strategy would be more appropriate as only one configuration procedure is running at a time.

That non-linear behaviour for Task 1 is also present when configuring a star network in Task 3, provoking a similar tendency in both use cases, but with a lower curve for Task 3. The core of the client configuration process is similar in both cases, differing only in the payload. The client configuration for Task 1 includes entries for $i-1$ servers, whereas for Task 3 the SC only specifies the information to secure the channel with one server.

Finally, it is worth noting that performance results would not vary substantially if the link capacity were lower than 11.5Gbps (for example, 1Gbps). In the most demanding situation, which is concurrently configuring a mesh network of 80 nodes, we have measured that the average overhead transmission of configuration messages is 4.861 Mb/s, with a peak of 20Mb/s. Therefore, the maximum overhead is two orders of magnitude lower than the link capacity.

7 Security considerations

In general, different security aspects in SDN environments have been widely analyzed in the literature, such as [19], [20], or [1]. Not in vain, in the SDN framework, the controller manages node configurations, which can include cryptographic material (as happens in our framework) and other important configuration parameters that can affect the operation of the nodes. Therefore it is a key entity in the infrastructure. In this section, we discuss different security aspects related to our proposal, beginning with the threat model considered to study the security in the proposed framework.

7.1 Threat model

In this context, we assume that an attacker is potentially capable of carrying out different types of attacks over the communication path between the Security Controller and the nodes, if no security measurements are put in place. This includes eavesdropping, traffic analysis, spoofing, insertion, modification, deletion, delay, replay or impersonate any of the entities. This means that the attacker can read messages on the network and remove, change, or inject forged messages. However, it is not able to break cryptographic primitives and has not been able to compromise the entities involved (controller and nodes). Nevertheless, we do explore the consequences and the impact in case the controller or nodes are compromised.

7.2 Protecting communication between controller and the nodes

To avoid an attacker of this nature to succeed, it is required to enforce strong access control mechanisms (i.e. authentication of the entities), ensure availability and to establish authenticated, and well-protected communication channels with integrity and confidentiality between the controller and nodes (southbound interface), and application services (northbound interface) [43].

In particular, a security association must be established between the Security Controller and each node in order to protect the configuration exchanged between these entities. The proposed framework makes use of NETCONF as southbound protocol. NETCONF defines, as mandatory, the usage of a secure transport protocol like SSH (by default) or TLS, which are considered secure to provide confidentiality and integrity, avoiding replay and man-in-the middle attacks with mutual authentication.

7.3 Key material distribution

In the proposed framework, the Security Controller may send TLS cryptographic material (public/private keys, certificates, PSK, etc.) for the establishment of the TLS channels between the nodes. We provide some requirements in order to avoid or reduce the possibility of an attacker to access this key material, namely:

- The Security Controller must ensure the node configuration is compliant with TLS 1.3 security considerations [34].
- The node must not allow the reading of private cryptographic material once it has been applied by the Security Controller (i.e., write-only operations).
- If PSK authentication is used in TLS between the nodes, the Security Controller must generate randomly the PSK and remove it immediately just after being distributed in order to reduce the impact if an attacker has been able to compromise the Security Controller.
- If RSA keys are used, and the Security Controller generates both certificate and private key, it must remove the associated private keys immediately after distributing them to the nodes for the same reasons as above.
- The ciphersuite used for either SSH or TLS secure channel must generate key material that allows protecting the information exchanged between the Security Controller and the nodes, at least with the same security strength that the distributed key material (e.g. if the Security Controller distributes a 128-bit PSK, SSH or TLS must encrypt that information with, at least, a 128-bit symmetric key).

7.4 Security Controller as a single point of failure

As already mentioned, the considered framework has a core entity which is the Security Controller. If the Security Controller is not operative it may affect the node operation. In general, any SDN-based network has to deal with this type of problem [37]. In this sense, a solution that allows replicating the SDN controller case of a failover is typical and valid for our case [31].

Nevertheless, it is worth describing the impact in case nodes cannot access the Security Controller. In fact, this might happen if the attacker is able to provoke a denial-of-service attack over the Security Controller. Firstly, if a node cannot access the Security Controller, it cannot be configured. Therefore, the node cannot be considered available to establish any TLS channel with other nodes. If the node was however configured, the framework is more resilience because a node ships a TLS implementation that can be used to maintain the existing TLS channels.

7.5 Other considerations

When a node needs to be deployed, it must be pre-provisioned with Security Controller's information (e.g. Security Controller's certificate and IP address). Moreover, the Security Controller's must be provisioned with the node's information (e.g. node certificate and/or CA certificate and IP address). After this pre-provision process, they can both establish the SSH or TLS secure channel demanded by NETCONF.

Nevertheless, the specific mechanism that allows deploying a node in a secure fashion under the control of the Security Controller is out of scope of this paper. The reason is that this initial process must happen for any SDN-based application before the Security Controller can operate over a node. This is our case, since the SDN-based TLS management we describe in this paper is the application. As an example, something similar happens with OpenFlow networks [27] , where the OFCONFIG protocol [26] is used for configuring switch and controller certificates to establish TLS between them. This is a requirement for OpenFlow operation, which is the protocol used between the controller and the switches (southbound protocol).

In any case, it is at least necessary to establish some minimum but strong security requirements for this pre-provision of information. In particular, the pre-provision process must securely deliver the required information by preventing an attacker from modifying or gaining access to the pre-provisioned information. For example, if the attacker is able to change Security Controller's certificate and/or IP address, then the node would connect to a rogue Security Controller).

8 Conclusions and future work

The power of Software Defined Networking brings new opportunities to research and develop innovative solutions. That is the reason why it will be present in future cloud environments, virtualized services and 5G networks. Apart from improving the management of the network, it has also a direct application in network security, and particularly in the creation and management of secure communication channels.

The paper has described a framework for the management of TLS connections in SDN based on existing standards. In particular, the solution is a flexible, dynamic and agile scheme for configuring network devices from the security controller using the NETCONF protocol. Using as reference ongoing standardization work, YANG models have been adapted to our work in order to allow the negotiation of security parameters and the establishment of TLS connections in the data plane.

Associated with the theoretical proposal, we have developed a proof-of-concept implementation to deploy a testbed able to run the different tasks associated with current application scenarios, such as mesh network in datacenter, virtualization environments or SD-WAN scenarios. It is a virtualized prototype with a security controller and an incremental number of network nodes. In addition, some experiments to measure the system performance are also described for each one of the tasks proposed: mesh, node addition and star topologies. Those experiments provide promising results and could help administrator in order to design how TLS channels could be established depending on the required topology, network resources, number of nodes, etc.

Nevertheless, the presented solution is a first approach to address the management of TLS connections. As a statement of direction, we could highlight the inclusion of more mature controllers (with technologies such as OpenDayLight or ONOS), and the implementation of notifications and state date in the data plane. Another aspect deserving more research is the communication between controllers (*inter-SC communication*) in order to implement distributed solutions and support the establishment of TLS connections between devices belonging to different administrative domains.

Declarations

Funding

This study was partially funded by grant from the Spanish Government with code FPU18/00304.

Conflicts of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Availability of code, material, and data

The full project is available at `https://github.com/javier-pg/sysrepo-cfgssl`.

References

1. Ahmad, S., Mir, A.H.: Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers. Journal of Network and Systems Management **29**(1), 9 (2020). DOI 10.1007/s10922-020-09575-4. URL `https://doi.org/10.1007/s10922-020-09575-4`

2. Amazon Web Services Cloud. Deploying an opportunistic IPsec mesh on the AWS Cloud. https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls (Accessed 2020-12-02)
3. AWS App Mesh. Transport layer Security (TLS). https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls (Accessed 2020-12-02)
4. Badra, M.: NETCONF over Transport Layer Security (TLS). RFC 5539 (2009). DOI 10.17487/RFC5539. URL https://rfc-editor.org/rfc/rfc5539.txt
5. Bellavista, P., Dolci, A., Giannelli, C., Padalino Montenero, D.D.: SDN-Based Traffic Management Middleware for Spontaneous WMNs. Journal of Network and Systems Management **28**(4), 1575–1609 (2020). DOI 10.1007/s10922-020-09551-y. URL https://doi.org/10.1007/s10922-020-09551-y
6. Björklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (2010). DOI 10.17487/RFC6020. URL https://rfc-editor.org/rfc/rfc6020.txt
7. Network Services Orchestrator VPN Solution Overview. https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/solution-overview-c22-734917.html (Accessed 2020-12-02)
8. da Costa Cordeiro, W.L., Marques, J.A., Gaspary, L.P.: Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management. Journal of Network and Systems Management **25**(4), 784–818 (2017). DOI 10.1007/s10922-017-9423-2. URL https://doi.org/10.1007/s10922-017-9423-2
9. Cullen, M.: Using the NETCONF Protocol over Secure Shell (SSH). RFC 6242 (2011). DOI 10.17487/RFC6242. URL https://rfc-editor.org/rfc/rfc6242.txt
10. Enns, R., Björklund, M., Bierman, A., Schönwälder, J.: Network Configuration Protocol (NETCONF). RFC 6241 (2011). DOI 10.17487/RFC6241. URL https://rfc-editor.org/rfc/rfc6241.txt
11. Fedor, M., Schoffstall, M.L., Davin, J.R., Case, D.J.D.: Simple Network Management Protocol (SNMP). RFC 1157 (1990). DOI 10.17487/RFC1157. URL https://rfc-editor.org/rfc/rfc1157.txt
12. Google Anthos Service Mesh. https://cloud.google.com/anthos/service-mesh5 (Accessed 2020-12-02)
13. Google Transparency Report. https://transparencyreport.google.com/https/overview (Accessed 2020-12-05)
14. Hantouti, H., Benamar, N., Taleb, T.: Service Function Chaining in 5G Beyond Networks: Challenges and Open Research Issues. IEEE Network **34**(4), 320–327 (2020). DOI 10.1109/MNET.001.1900554
15. Helali, L., Omri, M.N.: A survey of data center consolidation in cloud computing systems. Computer Science Review **39**, 100366 (2021). DOI https://doi.org/10.1016/j.cosrev.2021.100366. URL https://www.sciencedirect.com/science/article/pii/S157401372100006X
16. IBM Cloud Private. Encrypting cluster data network traffic with IPsec. https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.0/installing/ipsec_mesh.html (Accessed 2020-12-02)
17. Interface to Network Security Functions (I2NSF) Working Group. https://datatracker.ietf.org/wg/i2nsf/about/ (Accessed 2020-12-09)
18. Istio 1.8. Security architecture. https://istio.io/latest/docs/concepts/security/ (Accessed 2020-12-02)
19. (ITU-T), I.T.U.: Framework of software-defined networking (itu-t y.3300) (2014)
20. Kreutz, D., Ramos, F.M., Verissimo, P.: Towards Secure and Dependable Software-Defined Networks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, p. 55–60. Association for Computing Machinery, New York, NY, USA (2013). DOI 10.1145/2491185.2491199. URL https://doi.org/10.1145/2491185.2491199
21. Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE **103**(1), 14–76 (2015). DOI 10.1109/JPROC.2014.2371999
22. Linkerd 2.x. Securing your service. https://linkerd.io/2/tasks/securing-your-service/ (Accessed 2020-12-02)
23. Lopez-Millan, G., Marin-Lopez, R., Pereniguez-Garcia, F.: Towards a standard SDN-based IPsec management framework. Computer Standards & Interfaces **66**, 103357 (2019).

DOI https://doi.org/10.1016/j.csi.2019.103357. URL http://www.sciencedirect.com/science/article/pii/S0920548918303052
24. Marin-Lopez, R., Lopez-Millan, G., Pereniguez-Garcia, F.: A YANG Data Model for IPsec Flow Protection Based on Software-Defined Networking (SDN). RFC 9061 (2021). DOI 10.17487/RFC9061. URL https://rfc-editor.org/rfc/rfc9061.txt
25. Michel, O., Keller, E.: Sdn in wide-area networks: A survey. In: 2017 Fourth International Conference on Software Defined Systems (SDS), pp. 37–42 (2017). DOI 10.1109/SDS.2017.7939138
26. Open Networking Foundation: OF-CONFIG Version 1.2 (2014)
27. Open Networking Foundation: OpenFlow Switch Specification Version 1.5.1 (2015)
28. RedHat OpenShift Service Mesh. https://docs.openshift.com/container-platform/4.6/service_mesh/v2x/ossm-security.html (Accessed 2020-12-02)
29. OpenSSL. Cryptography and SSL/TLS Toolkit. https://www.openssl.org/ (Accessed 2020-12-03)
30. OpenVPN. https://openvpn.net/ (Accessed 2020-12-03)
31. Pashkov, V., Shalimov, A., Smeliansky, R.: Controller failover for SDN enterprise networks. In: 2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), pp. 1–6 (2014). DOI 10.1109/MoNeTeC.2014.6995594
32. Ranjbar, A., Komu, M., Salmela, P., Aura, T.: An SDN-based approach to enhance the end-to-end security: SSL/TLS case study. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, pp. 281–288 (2016). DOI 10.1109/NOMS.2016.7502823
33. RedHat OpenShift. Encrypting traffic between nodes with IPsec. https://docs.openshift.com/container-platform/3.11/admin_guide/ipsec.html (Accessed 2020-12-02)
34. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (2018). DOI 10.17487/RFC8446. URL https://rfc-editor.org/rfc/rfc8446.txt
35. Singh, S., Jha, R.: A Survey on Software Defined Networking: Architecture for Next Generation Network. Journal of Network and Systems Management **25**(2), 321–374 (2017). DOI 10.1007/s10922-016-9393-9. URL http://doi.org/10.1007/s10922-016-9393-9
36. Sousa, E., Cunha, V.A., de Carvalho, M.B., Corujo, D., Barraca, J.P., Gomes, D., Schaeffer-Filho, A.E., dos Santos, C.R.P., Granville, L.Z., Aguiar, R.L.: Orchestrating an SFC-enabled SSL/TLS traffic processing architecture using MANO. In: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1–7 (2018). DOI 10.1109/NFV-SDN.2018.8725675
37. Suartana, I.M., Anggraini, M.A.N., Pramudita, A.Z.: High Availability in Software-Defined Networking using Cluster Controller: A Simulation Approach. In: 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE), pp. 1–5 (2020). DOI 10.1109/ICVEE50212.2020.9243173
38. Vajaranta, M., Kannisto, J., Harju, J.: Implementation Experiences and Design Challenges for Resilient SDN Based Secure WAN Overlays. In: 2016 11th Asia Joint Conference on Information Security (AsiaJCIS), pp. 17–23 (2016). DOI 10.1109/AsiaJCIS.2016.25
39. Vajaranta, M., Kannisto, J., Harju, J.: IPsec and IKE as Functions in SDN Controlled Network. In: Z. Yan, R. Molva, W. Mazurczyk, R. Kantola (eds.) Network and System Security, pp. 521–530. Springer International Publishing, Cham (2017)
40. (W3C), W.W.W.C.: Extensible Markup Language (XML). www.w3.org/TR/xml/ (2013)
41. Watsen, K.: YANG Groupings for TLS Clients and TLS Servers. Internet-Draft draft-ietf-netconf-tls-client-server-22, Internet Engineering Task Force (2020). URL https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-22. Work in Progress
42. YANG Catalog. https://yangcatalog.org/ (Accessed 2021-04-05)
43. Yin, H., Xie, H., Tsou, T., Lopez, D.R., Aranda, P.A., Sidi, R.: Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases. RFC 8192 (2017). DOI 10.17487/RFC8192. URL https://rfc-editor.org/rfc/rfc8192.txt