Integrating Quantum Cryptosystems in Next Generation Networks

Alejandro Aguado, Vicente Martin

Center for Computational Simulation

POLITÉCNICA

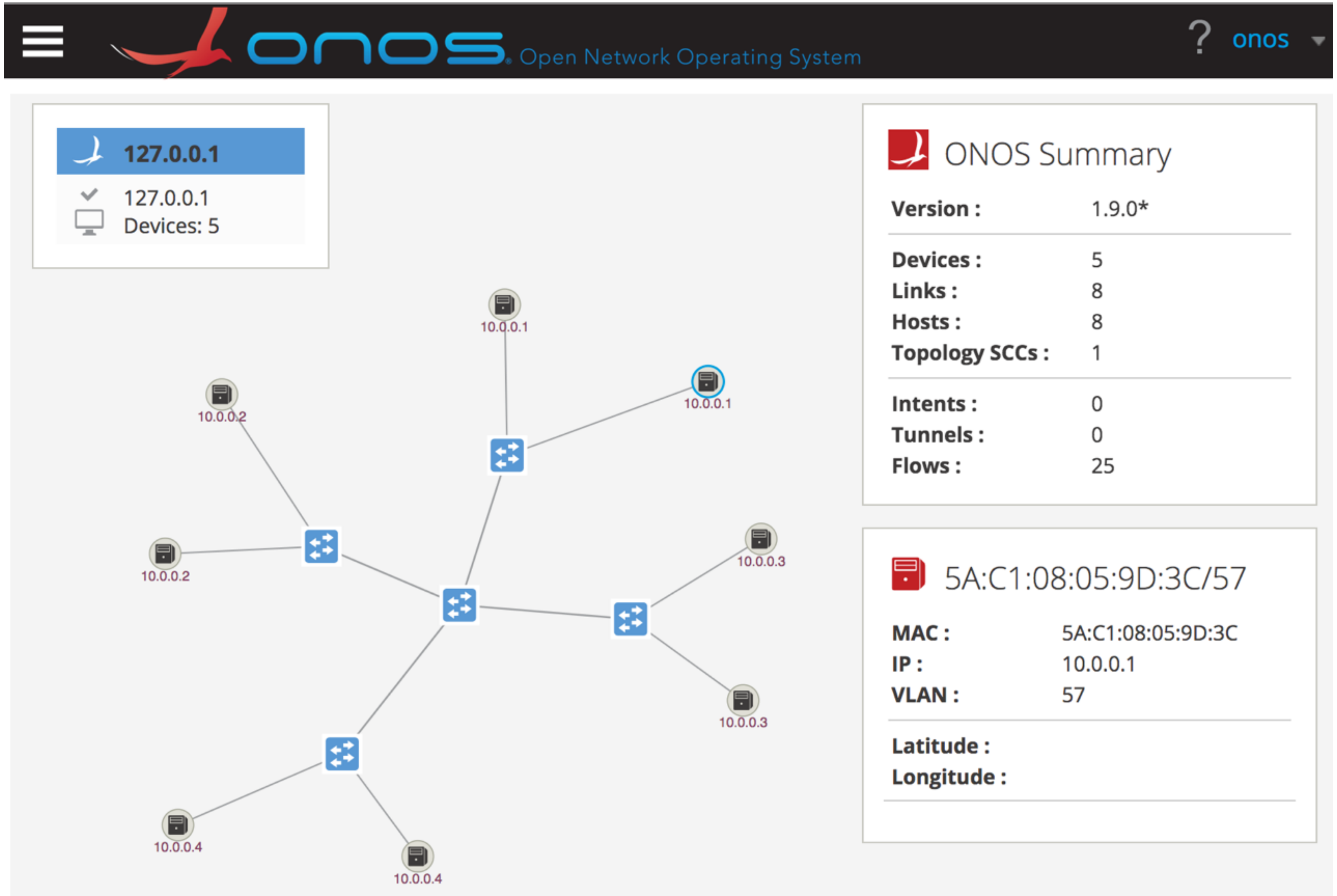
Outline

- Introduction
  - Software-Defined Networking
  - Network Functions Virtualization
  - Quantum Key Distribution
- Enabling End-to-End Services with Quantum Encryption
- Securing Control Plane Communications
- Conclusions

Introduction

- **Software-Defined Networking** is a novel network paradigm that allows to decouple the forwarding (data) and management (control) planes of a network, traditionally encapsulated on each network device.
- This network paradigm is based on the concepts of abstraction and network programmability.
- All these techniques allow to centrally manage an entire network, deploying and optimizing end-to-end services using standard protocols, such as OpenFlow and NETCONF.

Introduction:
Example of SDN controller view of a domain

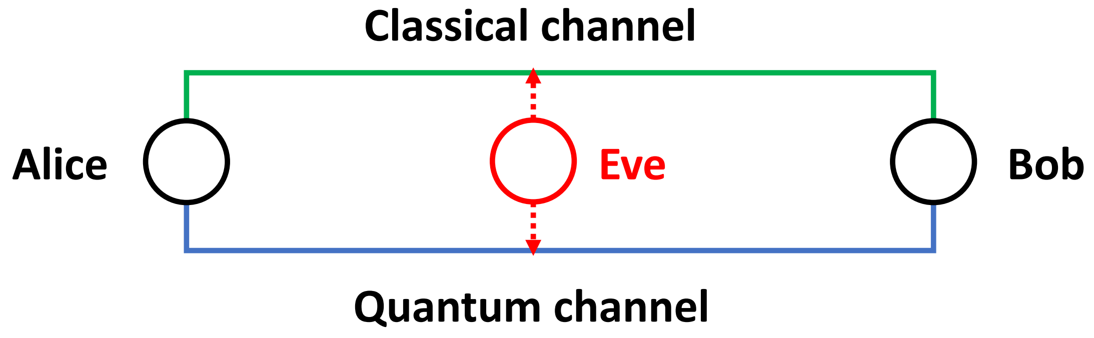


Introduction

- **Network virtualization** allows to simulate network resources that do not physically exist as hardware appliances.
- Virtualization in a network environment can happen in different ways
  - Creating virtual links or tunnels across multiple devices that are given to the user as a single link (VNTM).
  - Abstracting several network resources as a single entity to be controlled by end users (FlowVisor/OpenVirteX/Strauss arch.).
  - Encapsulating network functionalities inside software images / virtual machines (NFV).

Examples of NFV MANO projects

Introduction: Quantum Key Distribution

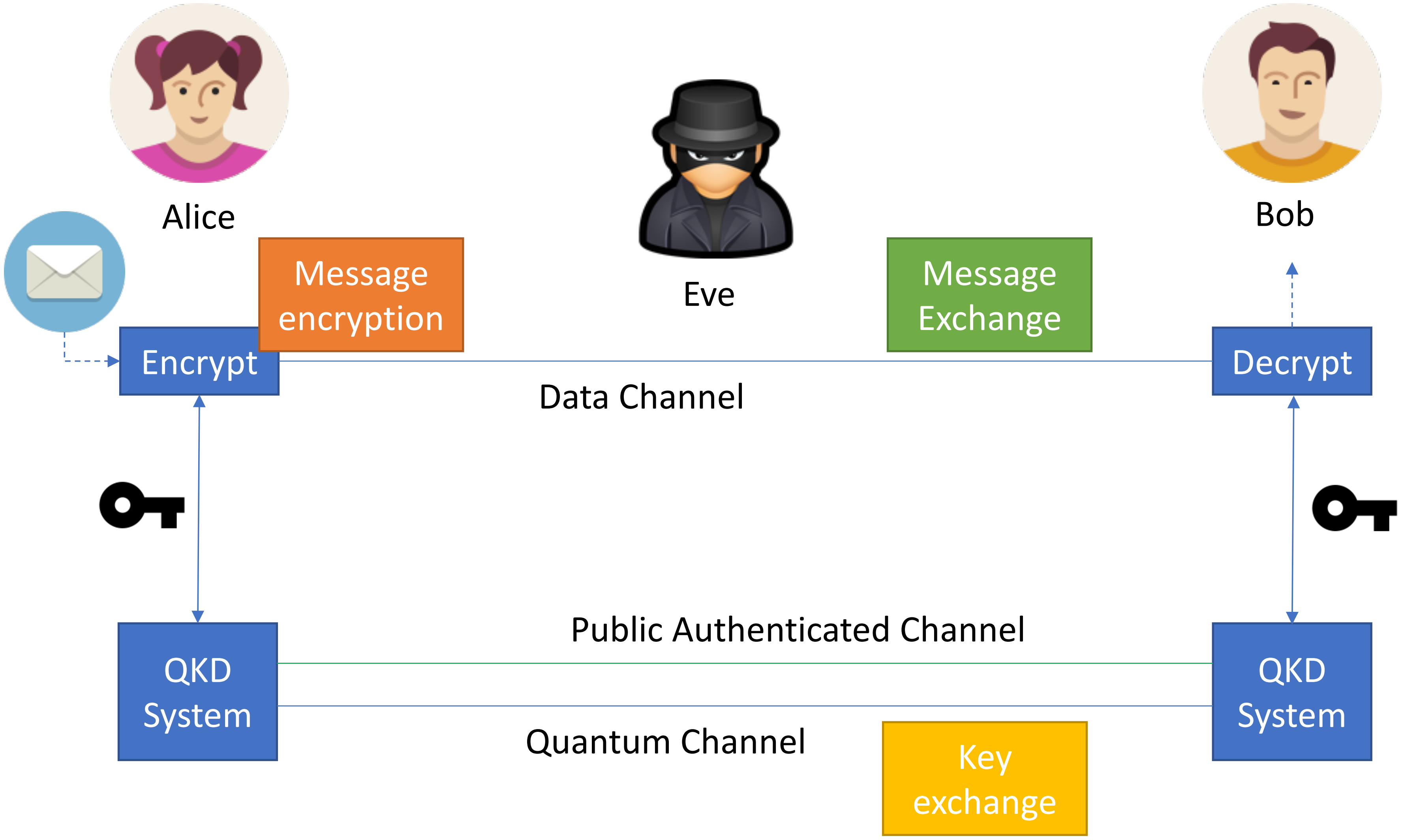


- **QKD** technology can be regarded as **two sources of synchronized random numbers** that are **separated physically.**
- QKD **does not depend on computational assumptions** (i.e. it will be safe however the computational power of the attacker). It **provides backward and forward security**.
- It can be **mathematically proven to be secure** (in principle, an information theoretic secure (ITS) primitive)
- A correct implementation will deliver keys of the **highest security**

LIMITATIONS

- QKD has some limitations that do not affect the **conventional cryptosystems, usually based on computational complexity.**
- Any kind of **amplifiers or active components** that can modify the state of the quantum signals **must be bypassed**.
- This sets a **limit to the maximum distance** (or absorptions) that a QKD protocol can tolerate, well suited to be used within a metropolitan area or with links of **up to 150 km**

Introduction: Quantum Key Distribution



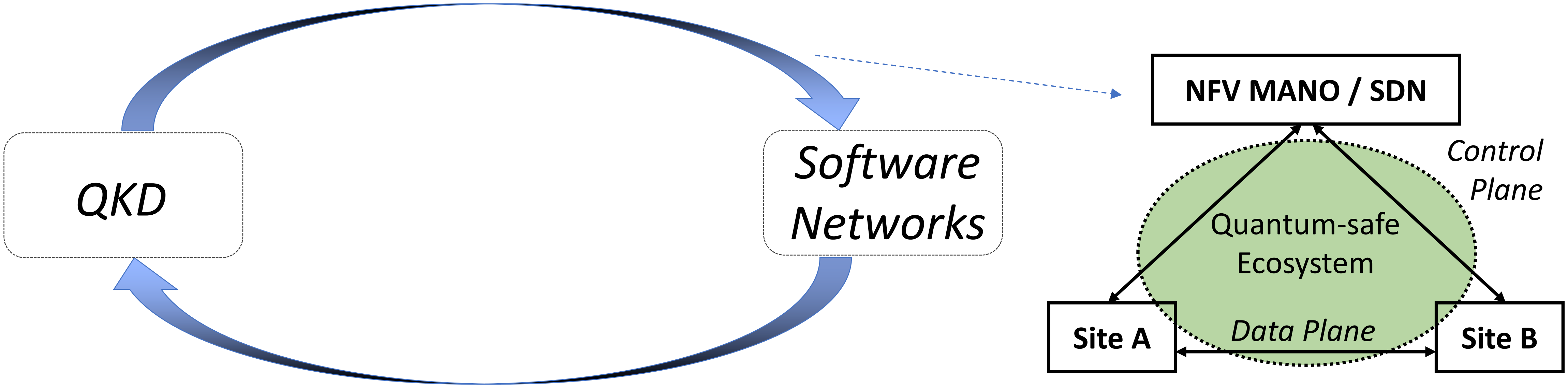
Ingredients:

- Qubit transmitter (typically photons), Alice.
- Single qubit receivers, Bob.
- Quantum channel (capable of transmitting qubits from Alice to Bob, in our case fibre).
- Classical channel (public, but authenticated).

A mutually-beneficial relation

The integration of QKD technologies in novel network paradigms must be seen as a mutually beneficial agreement, as both worlds can easily improve by being combined.

- Alleviates current and new (SDN+NFV) security threats
- Brings a physical security layer composable with traditional schemes

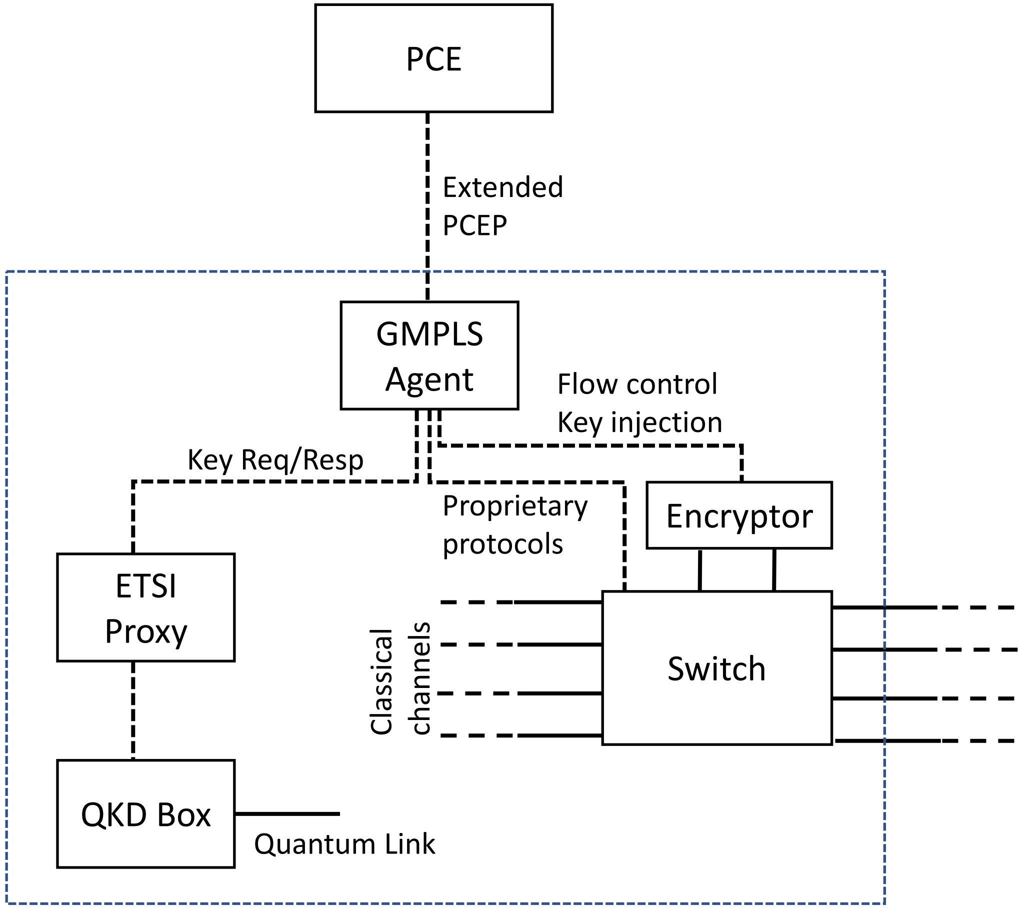


- Allows to easily integrate and manage QKD systems, reducing costs
- Allows to use the trusted node model without additional assumptions.

Introduction

- Network services are **increasingly requesting more flexibility and network resources**.
- One of the biggest demands is to **increase the level of security** for the transmission between remote premises.
- Here we show an example of a **node architecture** and the **protocol requirements** in a GMPLS environment to provide **QKD-enhanced security in end-to-end services**.

Example of QKD-enabled network node architecture



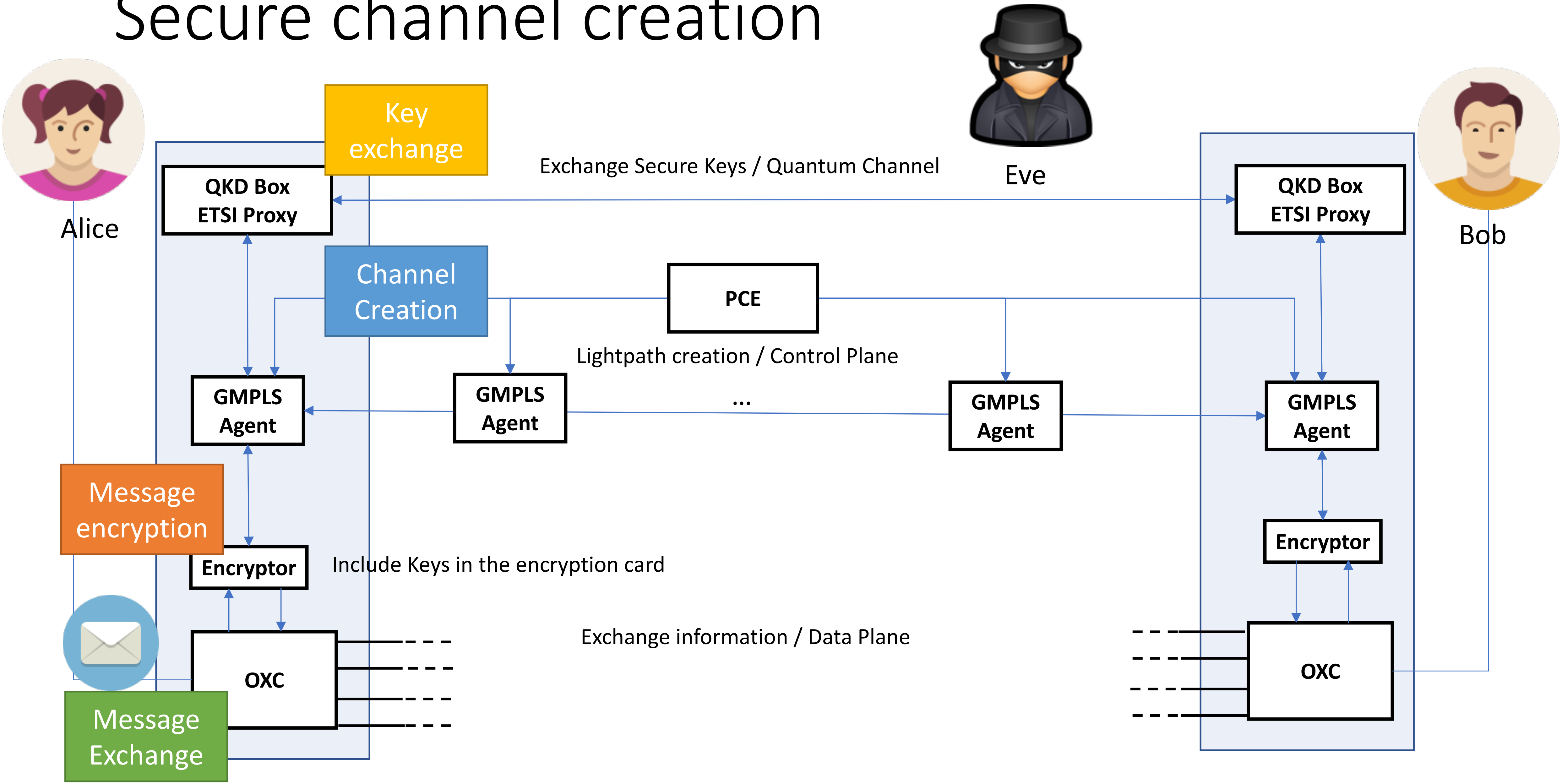
Desired capabilities:

- Access to QKD-generated keys.
- Encryption in upstream services (Data encryptor, security module, etc.).
- Switching/Routing.
- Control plane interface enabling automation

Definition of requirements in terms of parameters

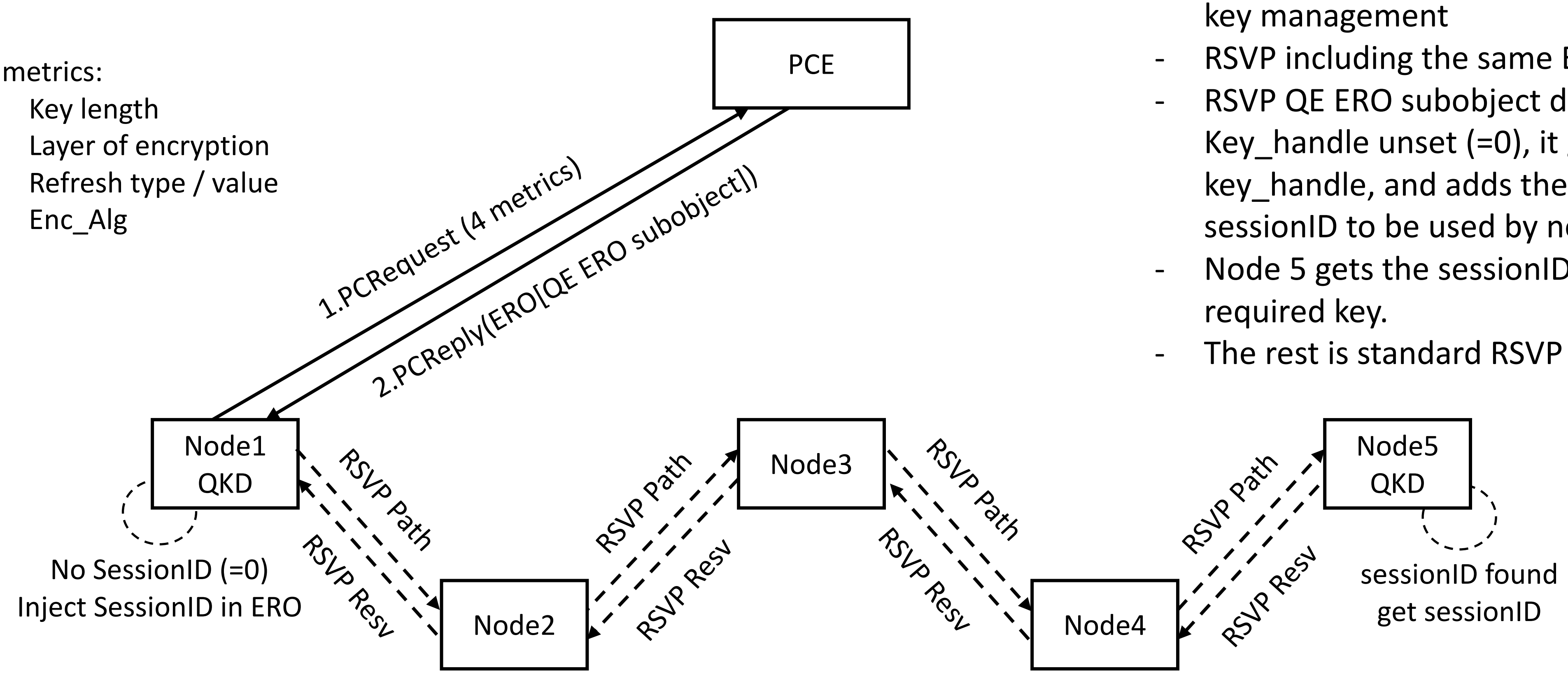
- Parameters required to be exchanged (point-to-point encryption):
  - **Session ID (key_handle)**: Initially set as 0, session ID gets the value of the first Key handle extracted by the source agent in the initial setup. The source agent will be in charge of updates (future work).
  - **Key length**: Length of the key to be used for the encryption.
  - **Destination**: It defines the other peer (encryptor/decryptor) to synchronise with. Currently defined by an IP address.
  - **Encryption Layer**: Layer where encryption is performed.
  - **Refresh type and value**: Type of refresh to be done for a key (time/traffic/etc) and the value to be considered as a threshold.
  - **Algorithm**: Encryption algorithm to be used.

Secure channel creation



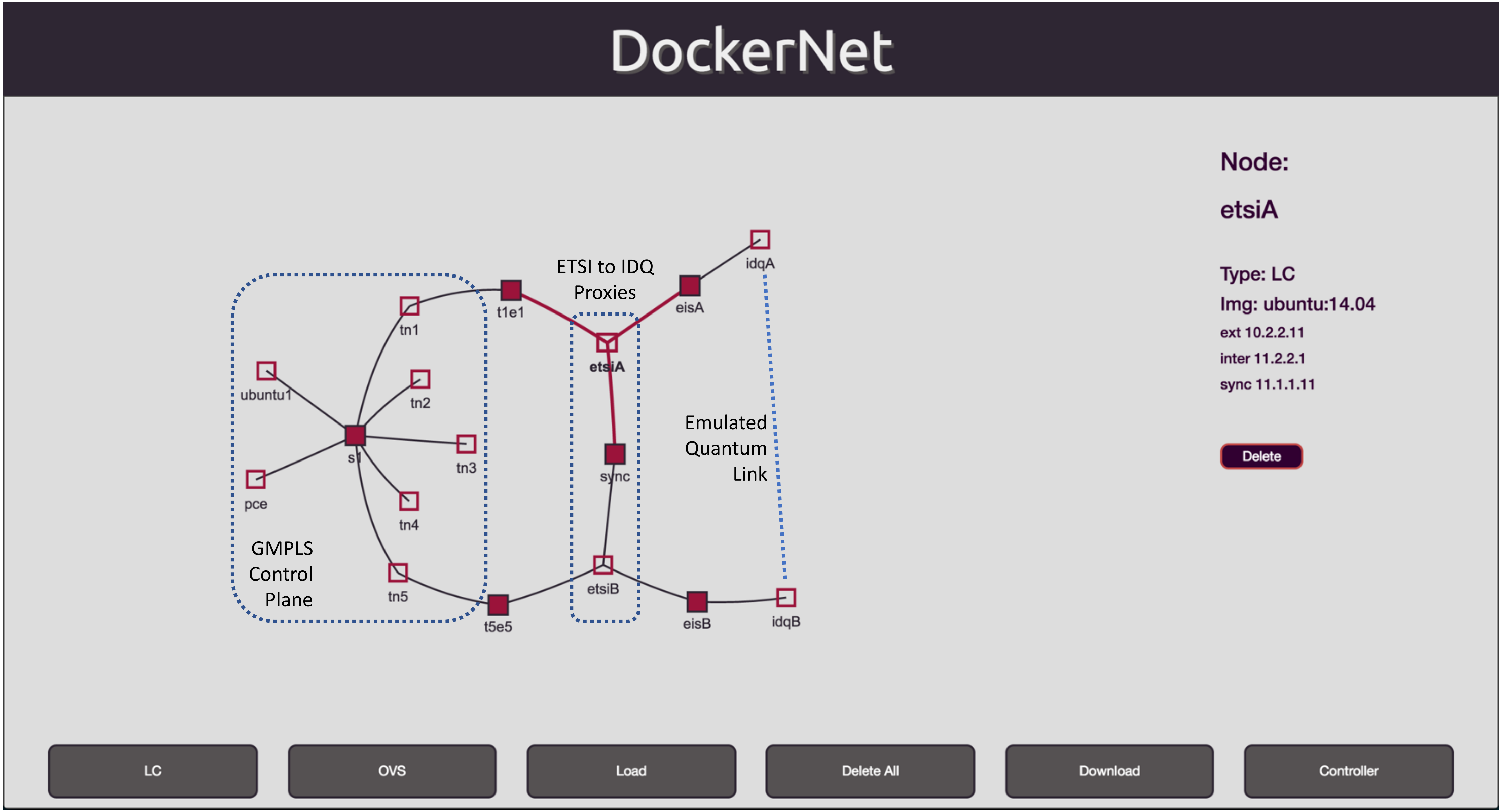
GMPLS+PCE Architecture
Proposed workflow: Case “Node starts”

4 metrics:
- Key length
- Layer of encryption
- Refresh type / value
- Enc_Alg



GMPLS case:
- PCRequest including metric for inline encryption.
- PCReply including new ERO subobjects for key management
- RSVP including the same ERO
- RSVP QE ERO subobject detected by node 1. Key_handle unset (=0), it gets a new key and key_handle, and adds the key_handle as sessionID to be used by node5
- Node 5 gets the sessionID and extracts the required key.
- The rest is standard RSVP

Experimental validation



Experimental validation RSVP (signalling)

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|L|    Type     |     Length    |  IDQ Session ID  (64 bytes)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             ...  (64 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  IDQ Session ID  (64 bytes)   |     KEY LEN     |    Ref Type  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Refresh value  (2 bytes)   |       Enc_Algo (2 bytes)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

10.1.1.1	10.1.1.2	RSVP	PATH Message.
10.1.1.2	10.1.1.3	RSVP	PATH Message.
10.1.1.3	10.1.1.4	RSVP	PATH Message.
10.1.1.4	10.1.1.5	RSVP	PATH Message.
10.1.1.5	10.1.1.4	RSVP	RESV Message.
10.1.1.4	10.1.1.3	RSVP	RESV Message.
10.1.1.3	10.1.1.2	RSVP	RESV Message.
10.1.1.2	10.1.1.1	RSVP	RESV Message.

Node 4 QE ERO subobject.
(before node 2)
Type: 0x67
Value: ”00..00” (64 bytes)
KeyLenght: 32
Enc_layer: 2
RefType: 0xfd
RefValue: 60
Alg: 10 (TBD)

Node 4 QE ERO subobject.
(before node 2)
Type: 0x67
Value: “4a0e…052f” (64 bytes)
KeyLenght: 32
Enc_layer: 2
RefType: 0xfd
RefValue: 60
Alg: 10 (TBD)

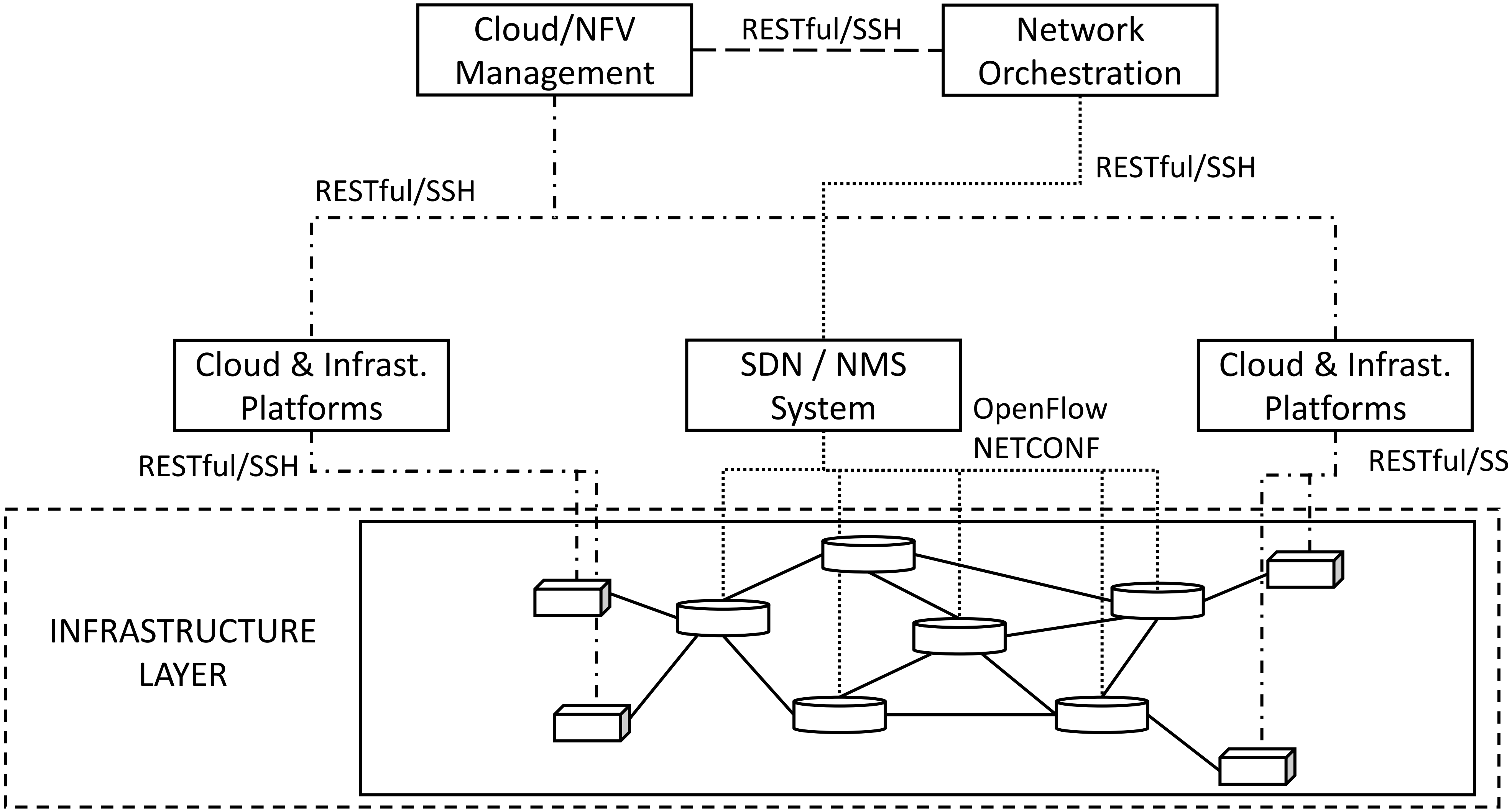
```
0120  20 00 67 4a 00 00 00 00  00 00 00 00 00 00 00 00
0130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160  00 00 00 00 00 20 02 fc  03 e8 00 0a 05 30 00 10
```

```
00f0  00 00 01 08 0a 01 01 05  20 00 67 4a 4a 0e 75 e8
0100  03 d7 f6 9e 9a 29 a1 0d  1c 7b 31 10 ac c3 95 98
0110  b4 78 9f 4f 0d 0e c1 40  fb ca 46 1d 6c a5 d2 a8
0120  a8 cc f0 d4 95 71 76 7d  31 b6 e0 69 4e a0 10 a0
0130  95 89 98 eb df 7d 35 85  e3 e6 05 2f 00 20 02 fc
0140  ff e8 00 0a 00 08 13 01  00 00 00 01 00 0c 0b 07
```

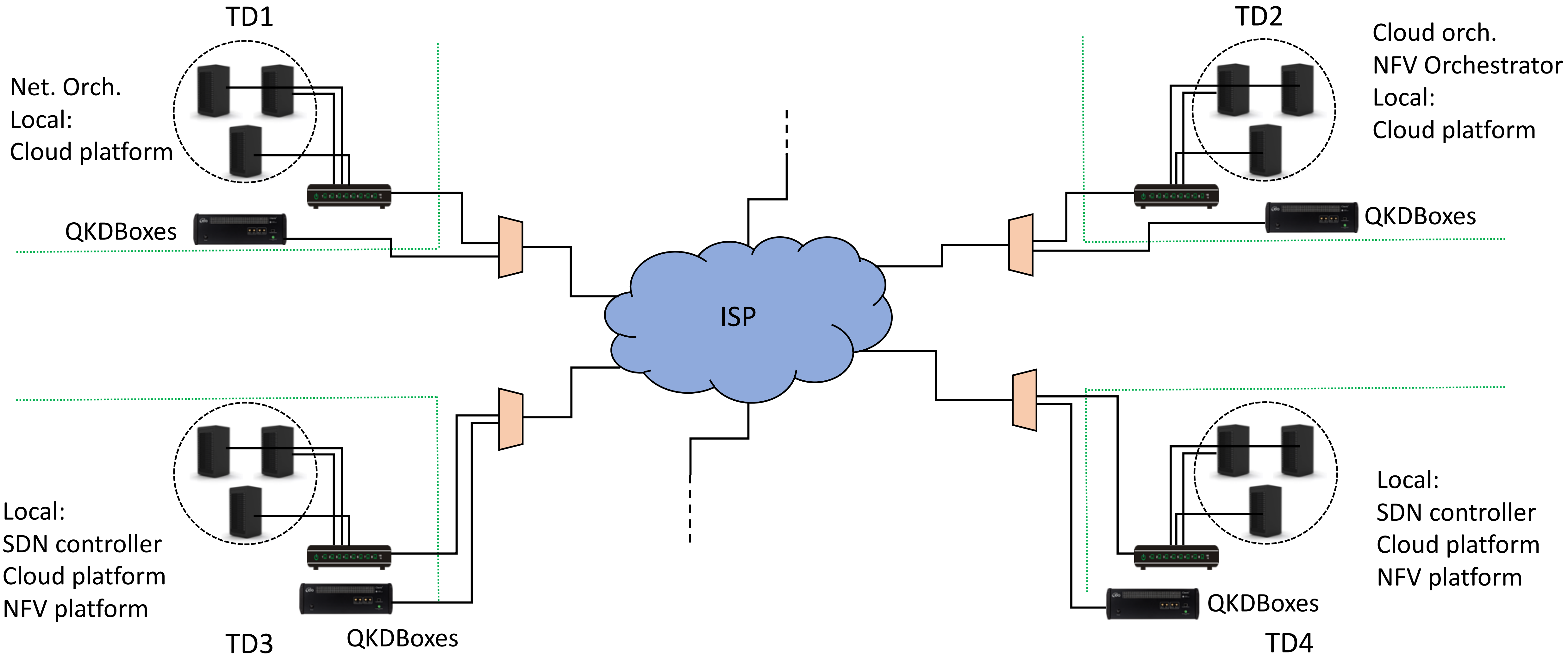
Securing SDN and NFV control plane operations

- Current network architectures and devices communicate with each other utilizing different protocols and standards.
- Some of these protocols are open and therefore vulnerable to attacks while others rely on security protocols, which internally use public key encryption (at least for key exchange).
- Here we propose the integration of SSH-based interfaces for control plane communication, replacing or reinforcing the public-key-based key exchange (Diffie-Hellman) for QKD.

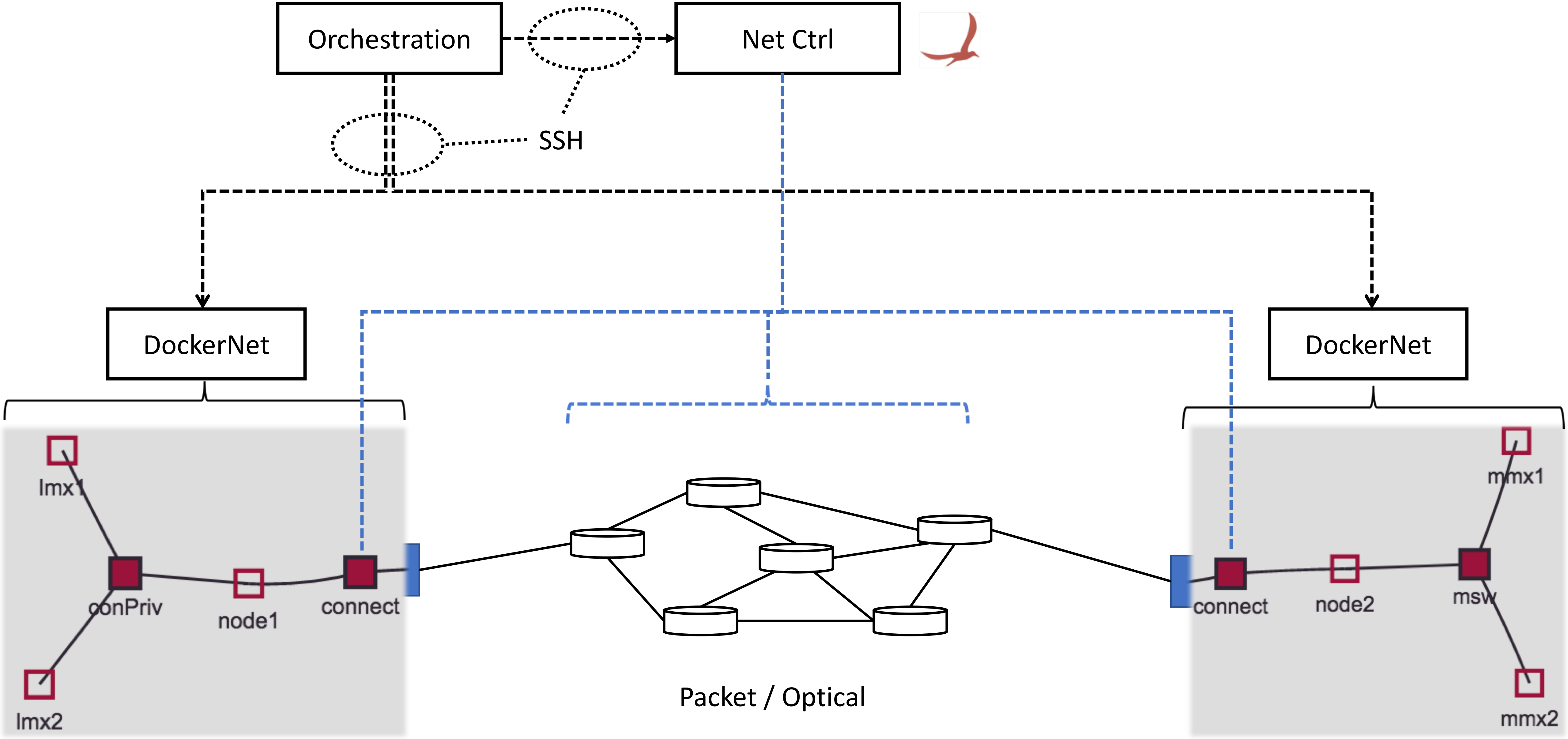
Abstract view



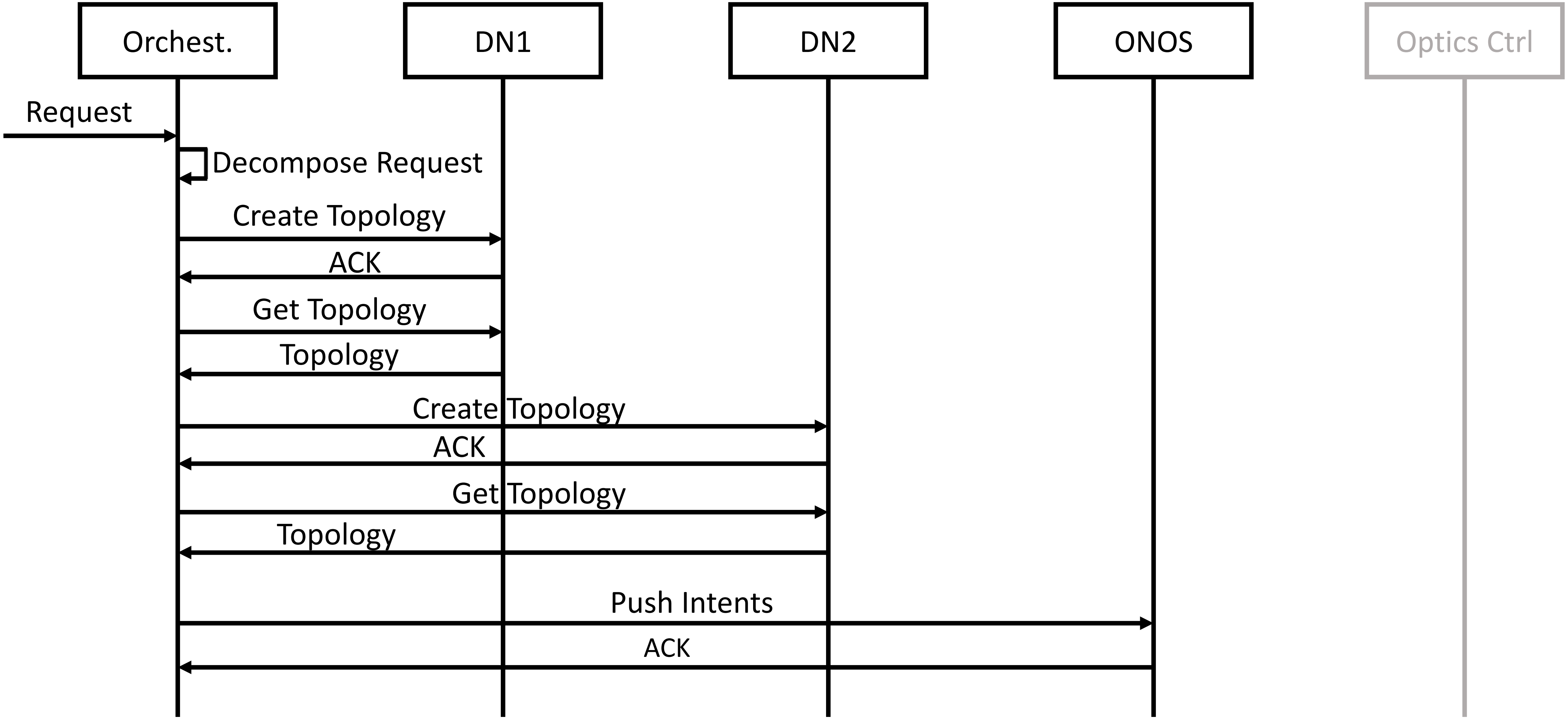
Logical view



Proposed implementation

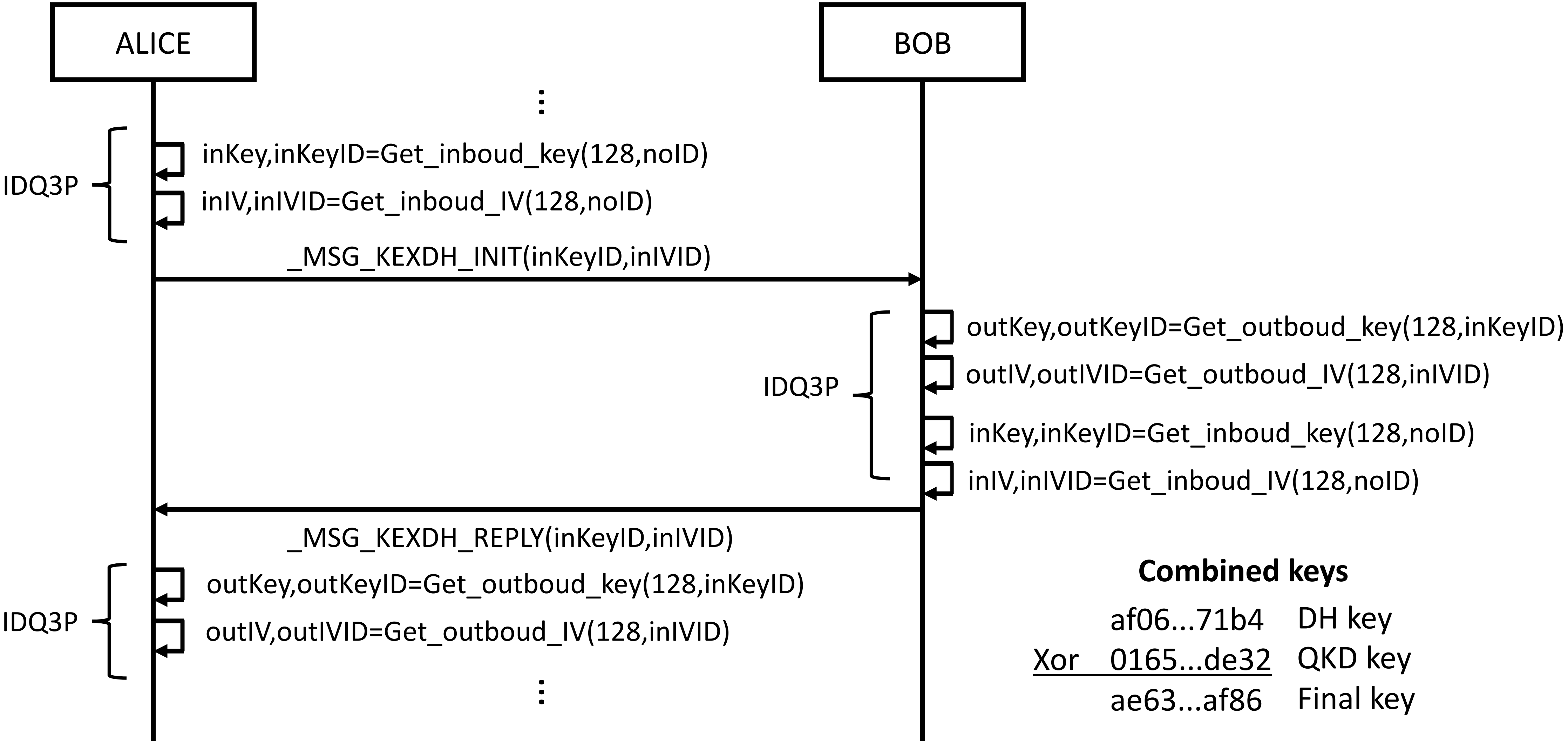


Demo workflow



Key exchange operation (SSH)

Example using extended DH_group1



Combined keys

	af06...71b4	DH key
Xor	0165...de32	QKD key
	ae63...af86	Final key

Captures, Workflow (local example)

SSH Session messages

34	5.023716	127.0.0.1	127.0.0.1	SSH…	716	Client: Key Exchange Init
35	5.023887	127.0.0.1	127.0.0.1	SSH…	756	Server: Key Exchange Init
40	5.037080	127.0.0.1	127.0.0.1	SSH…	252	Client: Diffie-Hellman Key Exchange Init
49	5.052209	127.0.0.1	127.0.0.1	SSH…	556	Server: Diffie-Hellman Key Exchange Reply
54	5.063034	127.0.0.1	127.0.0.1	SSH…	84	Client: New Keys
55	5.063049	127.0.0.1	127.0.0.1	SSH…	84	Server: New Keys
56	5.096277	127.0.0.1	127.0.0.1	SSH…	132	Client: Encrypted packet (len=64)
57	5.096494	127.0.0.1	127.0.0.1	SSH…	132	Server: Encrypted packet (len=64)
58	5.096684	127.0.0.1	127.0.0.1	SSH…	164	Client: Encrypted packet (len=96)

Preferred KEX

```
aa 1c c2 b8 d3 f2 34 e5  93 9e 00 00 00 82 71 6b   ......4. ......qk
64 2d 64 69 66 66 69 65  2d 68 65 6c 6c 6d 61 6e   d-diffie -hellman
2d 67 72 6f 75 70 31 2d  73 68 61 31 2c 64 69 66   -group1- sha1,dif
66 69 65 2d 68 65 6c 6c  6d 61 6e 2d 67 72 6f 75   fie-hell man-grou
70 2d 65 78 63 68 61 6e  67 65 2d 73 68 61 31 2c   p-exchan ge-sha1,
64 69 66 66 69 65 2d 68  65 6c 6c 6d 61 6e 2d 67   diffie-h ellman-g
72 6f 75 70 31 34 2d 73  68 61 31 2c 64 69 66 66   roup14-s ha1,diff
69 65 2d 68 65 6c 6c 6d  61 6e 2d 67 72 6f 75 70   ie-hellm an-group
2d 65 78 63 68 61 6e 67  65 2d 73 68 61 32 35 36   -exchang e-sha256
```

QKey extraction

39	5.036890	127.0.0.1	127.0.0.1	UDP	106	Source port: 5323	Destination port: 57518
41	5.038411	127.0.0.1	127.0.0.1	UDP	73	Source port: 46851	Destination port: 5323
42	5.039415	127.0.0.1	127.0.0.1	UDP	106	Source port: 5323	Destination port: 46851
43	5.040462	127.0.0.1	127.0.0.1	UDP	73	Source port: 53708	Destination port: 5323

Few OF messages

303	7.924126	138.100.…	192.168.1…	Ope…	76	Type: OFPT_FEATURES_REQUEST
307	7.924436	192.168.…	138.100.1…	Ope…	100	Type: OFPT_FEATURES_REPLY
308	7.924485	192.168.…	138.100.1…	Ope…	76	Type: OFPT_HELLO
310	7.932078	138.100.…	192.168.1…	Ope…	84	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
311	7.932248	192.168.…	138.100.1…	Ope…	276	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
313	7.937106	138.100.…	192.168.1…	Ope…	84	Type: OFPT_HELLO
315	7.937130	138.100.…	192.168.1…	Ope…	76	Type: OFPT_FEATURES_REQUEST

Captures, Workflow (local example)

```
127.0.0.1   127.0.0.1   TLSv1.2 Client Hello
127.0.0.1   127.0.0.1   TCP     4443→54448 [ACK] Seq=1 Ack=169 Win=44800 Len=0 TSval=1774576205 TSec
127.0.0.1   127.0.0.1   UDP     Source port: 47584  Destination port: 5323
127.0.0.1   127.0.0.1   UDP     Source port: 5323  Destination port: 47584
127.0.0.1   127.0.0.1   TLSv1.2 Server Hello, Certificate, Server Key Exchange, Server Hello Done
```

```
Secure Sockets Layer
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 111
  ▼ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 107
```

Server KeyID

```
0c 2a ea 31 78 03 2f 45  f9 a1 de 33 66 27 1e 8b
27 a3 c6 52 71 a9 43 51  9c 60 f6 73 3a 51 cc 37
8b 0c 5c 91 e6 a2 df 1a  a2 1a 1b 4d 1d 08 cc 49
db 8d 27 3f 68 b0 14 03  03 00 01 01 16 03 03 00
```

```
127.0.0.1   127.0.0.1   UDP     Source port: 57816  Destination port: 5323
127.0.0.1   127.0.0.1   UDP     Source port: 5323  Destination port: 57816
127.0.0.1   127.0.0.1   UDP     Source port: 48884  Destination port: 5323
127.0.0.1   127.0.0.1   UDP     Source port: 5323  Destination port: 48884
127.0.0.1   127.0.0.1   TLSv1.2 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
```

```
Secure Sockets Layer
▶ TLSv1.2 Record Layer: Handshake Protocol: Server Hell
▶ TLSv1.2 Record Layer: Handshake Protocol: Certificate
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
```

```
cb 39 31 ca 1b b0 5b ff  13 b1 4e 7d 82 eb b8 8e
cc 9b 7d 27 6c 3e d5 b8  73 7d 83 df 4f 7f 2d 4f
a4 84 ab 2b b8 bd 41 5b  41 16 ef 92 2f a8 f2 44
56 38 08 23 98 d6 1a 21  52 52 a7 16 03 03 00 04
```

Client KeyID

Future developments

- We are defining new use cases for the integration of QKD technologies in future network paradigms and services.
- We are currently collaborating with different standardization groups from IEEE and ETSI in order to integrate QKD systems in current control plane frameworks.
- We would like to create a physically distributed testbed to demonstrate our solutions in a realistic scenario. We are currently discussing these possibilities with network operators and vendors.

THAN K YOU!!!

Alejandro Aguado and Vicente Martin

Center for Computational Simulation

POLITÉCNICA