

Traditional Networking vs. Software Defined Networking

Jorge López

<jorge.lopez@telecom-sudparis.eu>
Télécom SudParis / Université Paris-Saclay

November 30th, 2018

Outline

Introduction

Traditional Computer Networks

Software Defined Networking

Conclusion

Our goals for today

- ▶ Make you remember (or give you a brief introduction to) traditional TCP/IP networking
 - ↪ You will easily recall how you get `http://google.ru/` in your computer 😊
- ▶ Make you understand the Software Defined Networking (SDN) advantages, architecture, functioning and possibilities
 - ↪ You will also get to play with an SDN through a simulator 😊 (more on this later)
- ▶ Give you an example of multi-component / complex systems. . .

A side note: an increasing amount of efforts in the industrial / academic world to these type of frameworks

Any contributions to the field are welcome!

Outline

Introduction

Traditional Computer Networks

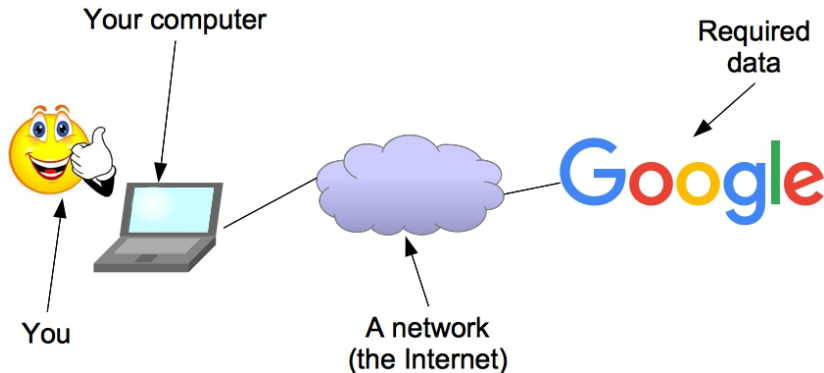
Software Defined Networking

Conclusion

How data are transmitted in computer networks?

... a long story

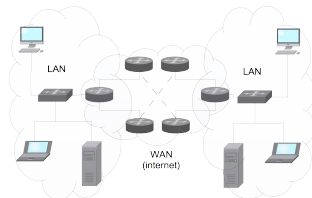
What are the necessary steps to get google's content into your web browser?






Computer networks

Typical notations

Simplified Network Architecture



- ▶ LAN = Local Area Network. WAN = Wide Area Network
- ▶ Router (intra-network communication): 
- ▶ Switch (inter-network communication): 
- ▶ Hosts (computers, data sources): 

Basic communication

Let's understand how data can reach one host to the other

- ▶ It all starts by the local host sending data to the “source” LAN
- ▶ From the LAN the data are sent to the WAN
- ▶ From the WAN the data are forwarded iteratively until it reaches the “destination” LAN
- ▶ From the “destination” LAN the data are transferred to the remote host

There are two layers in the communication process

- ▶ For the communication between hosts and LAN
- ▶ From the LAN to WAN, and WAN to WAN (this is grouped equally)

The physical link (PHY) layer

To communicate we need a **physical** link, a connection

A physical connection between the hosts and the switch (or host to host, etc.)

- ▶ To transmit data ($D \in \{0, 1\}^*$) from one host to another over different media, e.g., IEEE 802.3 wired Ethernet standard, IEEE 802.11 Wireless Ethernet standard, and IEEE 802.15 Bluetooth standard
- ▶ The voltage used to interpret 0s and 1s, which pin/cable has which data, the speed to transmit, the radio-frequency, the multiplexing, the time-slots to transmit, how to handle collisions between wireless-packets and many other parameters are part of the PHY layer
- ▶ Extensive! Out of the scope of our topics :)

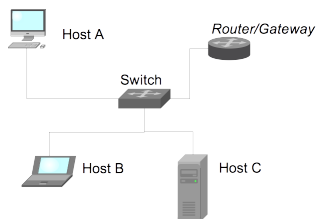
message.zip: Used to transmit data from a device to other *physically* connected device(s)

The Data Link layer

Data-Link (A.K.A. Layer 2) Layer functions?

- ▶ PHY is clearly not enough. . .
- ▶ Allows hosts to send data to several hosts (addressing) with a single physical connection
- ▶ Also important, it detects potential physical transmission errors (CRC)

Typical LAN architecture

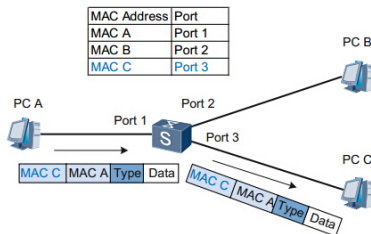


The Data Link layer II

Switching

- ▶ Switched networks have stood the test of time, differently from other, e.g., Token Rings
- ▶ Easy to expand (perhaps that's why 😊)
- ▶ All the magic of switching, a **switching table**
Device \mapsto Port mapping (A non-injective non-surjective function)

Switching tables



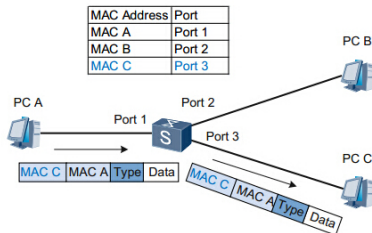
- ▶ Think about two switches connected together, what would happen?

The Data Link layer II

Switching

- ▶ Switched networks have stood the test of time, differently from other, e.g., Token Rings
- ▶ Easy to expand (perhaps that's why 😊)
- ▶ All the magic of switching, a **switching table**
Device \mapsto Port mapping (A non-injective non-surjective function)

Switching tables



- ▶ Think about two switches connected together, what would happen?
- ▶ Answer: In a single port many devices!

The Data Link layer III

The **Ethernet** Protocol Frame Structure

Preamble	Start Delim	Dest MAC	Src MAC	Ether Type	Payload	FCS
(7 bytes)	(1 byte)	(6 bytes)	(6 bytes)	(2 bytes)	(1500-46 bytes)	(4 bytes)

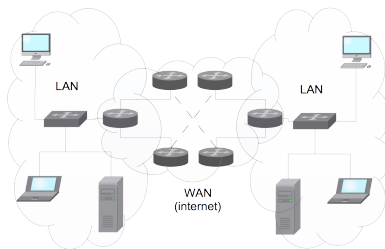
- ▶ Preamble + Start of Frame Delimiter: used by the operating system to determine / synchronize the packet data (sequential data)
- ▶ Destination Media Access Control (MAC) address: destination *unique* (2^{48}) Identifier for network interfaces
- ▶ Source MAC: source device address, typically represented as 6 hex pairs separated by colons (b8:2a:14:36:dc:86)
- ▶ Ether Type: what type of *data* it carries (0x806=ARP, 0x800=IPv4, more on this later...)
- ▶ Payload: the *data*
- ▶ Frame Checking Sequence: error detection and correction code (ask Prof. Yevtushenko)

The Network Layer

Network (A.K.A. Layer 3) Layer functions?

- ▶ Routing / inter-network communication
- ▶ Connectionless (best effort) communication
- ▶ Hierarchical host addressing

Typical Inter-network architecture



The Network Layer II

Routing

- ▶ The core of L3, based on the routing (moving *data*) from one network to the other
- ▶ Based on Internet Protocol (IP) addresses, e.g.: 91.221.61.55 (typically represented with 4 decimal octets separated by a '.')
- ▶ Local networks have IP addresses, which belong to the **same** network
- ▶ Network masks (netmasks) determine the network, a 4 byte *and* bit-wise mask
 - ↪ A common notation, the most significant bits are the network address For example, “/16” means the netmask is equal to: 11111111111111110000000000000000
- ▶ Simply put: 192.168.1.0/24 \implies all devices with IP addresses 192.168.x.x belong to the same network

The Network Layer III

The IPv4 Datagram Structure

Version <i>(1 nibble)</i>	IHL <i>(1 nibble)</i>	DSCP <i>(6 bits)</i>	ECN <i>(2 bits)</i>	Total Length <i>(2 bytes)</i>			
Identification <i>(2 bytes)</i>				Flags			Fragment offset <i>(13 bits)</i>
				0 <i>(1 bit)</i>	DF <i>(1 bit)</i>	MF <i>(1 bit)</i>	
Time to Live <i>(1 byte)</i>	Protocol <i>(1 byte)</i>		Header checksum <i>(2 bytes)</i>				
Source IP address <i>(4 bytes)</i>							
Destination IP address <i>(4 bytes)</i>							
Options <i>(IHL – 20 bytes)</i>							

Wow... those are lots of fields... let's take a look at some of them

- ▶ Protocol: Which protocol (*data*) the IP datagram carries (0x06=TCP, 0x11=UDP, more on this later)
- ▶ Source IP address: Sending host IP address
- ▶ Destination IP address: Receiving host IP address

Routing protocols

Working principles

- ▶ Basic and most important idea: find the gateway and deliver the data to it, gateway IP in the same network, MAC found via the Address Resolution Protocol (ARP), $IP \mapsto MAC$ translation
- ▶ The gateway decides to which peer (router) to forward the traffic based on the destination IP address
- ▶ Classical shortest path-finding algorithm application, e.g., Bellman-Ford algorithm for classical Routing Information Protocol (RIP)
- ▶ Shortest path is not not always the “best” path
 - ↪ Internet service providers might prefer to pass traffic to a longer path which is less expensive (in terms of P, €, \$), e.g., the Border Gateway Protocol — governs the Internet

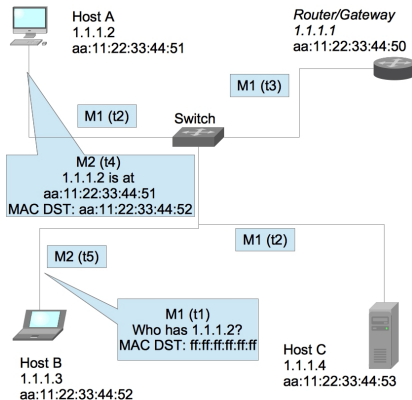
The Address Resolution Protocol (ARP)

Why it is so important?

Data sources are commonly hosts, running a 'network-compliant' Operating System (OS)

- ↪ Even if it is not required for hosts to know the target MAC address, the OSs will not send subsequent packets if they do not obtain one
- ↪ If IP is used in some rules but ARP is blocked, there is no traffic generated!

How ARP works?



The Transport Layer

Differentiating application data

- ▶ 1 city different ports \mapsto 1 host different ports

The User Datagram Protocol (UDP)

Source Port <i>(2 bytes)</i>	Destination Port <i>(2 bytes)</i>
Length <i>(2 bytes)</i>	Checksum <i>(2 bytes)</i>

- ▶ Source Port: The port of the originating host
- ▶ Destination Port: The target port
- ▶ Length: The size of the UDP packet (header + data)
- ▶ Checksum: error detection for the UDP header

UDP is unreliable (no facilities to guarantee data are delivered) but, lightweight (no complications \implies good for real-time protocols), no order, etc.

The Transport Layer II

Transmission Control Protocol (TCP)

Source Port (2 bytes)										Destination Port (2 bytes)									
Sequence Number (4 bytes)																			
Acknowledgement Number (4 bytes)																			
Data Offset (1 nibble)		Reserved (3 bits)		N S	C R	E C	U R	A C	P S	R S	F I	Window Size (2 bytes)							
Checksum (2 bytes)										Urgent Pointer (2 bytes)									
Options (Data offset -5 bytes)																			

Lots of fields, again... Let's consider for now only one:

- Acknowledgment Number: if the field ACK =1, acknowledges all previous data sent with the corresponding sequence number, if this is in response to a SYN, it means just the initial sequence number setup is being acknowledged (no data)

TCP establishes connection with a “3-way handshake” SYN; SYN-ACK; ACK \implies Ordered, reliable, congestion avoidance data delivery!

The Application Layer

Finally! The data actual applications care about! Take HTTP data as an example

Request:

```
GET / HTTP/1.1
Host: google.ru
Connection: Closed
```

Response:

```
HTTP/1.1 200 OK
Date: Thu, 29 Nov 2018 15:30:09 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2018-11-29-15; expires=Sat, 29-Dec-2018 15:30:09 GMT; path=/; domain=
Set-Cookie: NID=148=CZfK5GBk2twglYCHa6fB776FQC4QrKzBCZDifHtD7b34KKnvg7kIWNF0Q9P3jW7tpKw
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
```

5ad9

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="fr"><head>
...
```

Packet Encapsulation

Packet Encapsulation

HTTP
Request

Packet Encapsulation

TCP Headers <i>DST PORT 80</i>	HTTP Request
-----------------------------------	-----------------

Packet Encapsulation

IP Headers <i>DST IP 79.136.239.38</i>	TCP Headers <i>DST PORT 80</i>	HTTP Request
---	-----------------------------------	-----------------

Packet Encapsulation

Data-Link Headers	IP Headers <i>DST IP 79.136.239.38</i>	TCP Headers <i>DST PORT 80</i>	HTTP Request	Data-Link Footer
----------------------	---	-----------------------------------	-----------------	---------------------

Packet Encapsulation

Data-Link Headers	IP Headers <i>DST IP 79.136.239.38</i>	TCP Headers <i>DST PORT 80</i>	HTTP Request	Data-Link Footer
----------------------	---	-----------------------------------	-----------------	---------------------

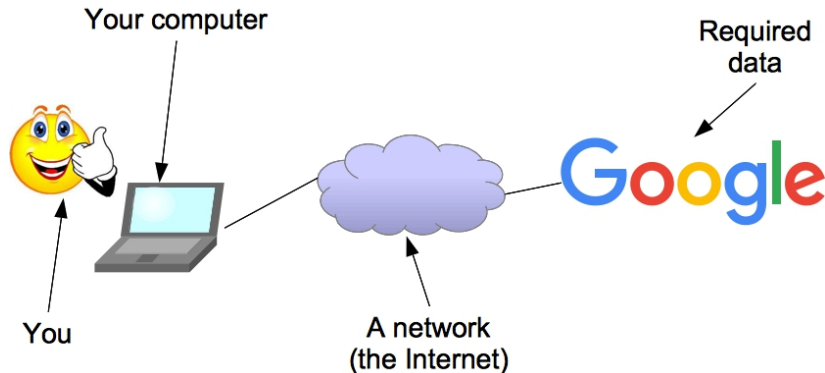
After encapsulation?

- ▶ First the packet is delivered to the default gateway, MAC address is obtained with ARP
- ▶ Default GW forwards the packet to the other routers until it reaches the destination host, the destination host IP address is obtained using the Domain Name System Protocol (maps from domain names, e.g., google.ru to IP addresses 79.136.239.38)
- ▶ Along the way, the packet is opened just in the corresponding layer
- ▶ The packet gets “decapsulated”, and delivered to the receiving application

How data are transmitted in computer networks?

The story...

What are the necessary steps to get google's content into your web browser?



For more information, see [Kozierok, 2005]

Outline

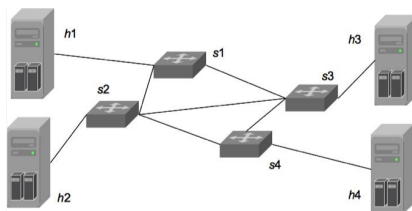
Introduction

Traditional Computer Networks

Software Defined Networking

Conclusion

Traditionally, what it means to configure this small network?



Assuming these are heterogeneous (incl. different brands) multilayer switches, i.e., they can switch data based on IP, or TCP, etc.

1. **For each** switch s
 - 1.1. Log in to s via a console
 - 1.2. **If** you know how to configure s **go to** 1.4.
 - 1.3. Learn how to configure s
 - 1.4. Grab your network design and translate to proper commands
2. Check if you did well. . .

Limitations with traditional networks 🤖

- ▶ Configuring network forwarding devices requires knowledge of ALL configuration interfaces / languages
- ▶ (Re-)configuring network forwarding device can be error prone
- ▶ (Re-)configuring network forwarding device can be time consuming
- ▶ (Re-)configuring network forwarding device can yield undesired results as it is done 1 by 1 (inconsistent state)
- ▶ All previous problems become even worse as the network size increases

How can we avoid these limitations? 🤔

N.B. nowadays fast deployment is easily attained through virtualization, i.e., network function virtualization, however, we won't cover this topic today...

Software Defined Networking (SDN)

The proposed architecture...

SDN is a networking paradigm, seminal paper:
[Open-Networking-Foundation, 2014]

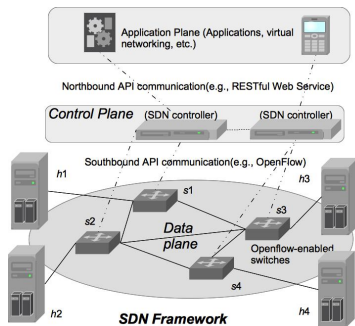


Figure: Example of an SDN Architecture

- ▶ The forwarding of network traffic is configured centrally, at the controller(-s), a.k.a. the *control plane*
- ▶ Data devices (hosts, switches, etc.) form the *data plane*
- ▶ Controllers translate and execute the particular configuration as instructed by the *applications*

Southbound communication

- ▶ These are (mostly) protocols to manage how packets are forwarded
- ▶ They manage the connection to the controller, the management of flow rules, etc., a popular protocol, Open Flow (OF) [Open-Networking-Foundation, 2013]

An OF rule

Location		Matching					Action		
Table	Priority	In port	MAC src	IP src	TCP src port	...	Nat	Output port	...

An example of implemented rules in an SDN-enabled switch:

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x9b00004050d3d0, duration=1091.387s, table=0, n_packets=1018, n_bytes=99764, idle_age=65, priority=500, ip,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:2
  cookie=0x9b0000e74c0915, duration=1091.764s, table=0, n_packets=58, n_bytes=2436, idle_age=82, priority=501,arp,in_port=1 actions=ALL
  cookie=0x9b0000e4b51319, duration=1091.777s, table=0, n_packets=58, n_bytes=2436, idle_age=82, priority=500,arp actions=output:1
```

The behavior is quite simple, starting from the first table (T0) find the rule with the highest priority that matches the input packet and apply the actions of that rule (incl. asking the controller)

Northbound communication

- ▶ These communication is done usually through Applications Program Interfaces (APIs)
- ▶ Common web REST APIs (through typical JSON) or direct Java library/API
- ▶ Interaction with the controller though these interface allows to retrieve (and push) different data: devices, flow rules, dataplane topology (incl. link information), etc.

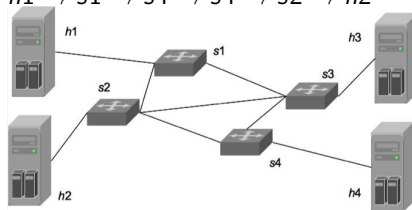
SDN applications

- ▶ Can be executed at the controller (using for example a Java lib.)
- ▶ Can be externally executed (using for example the REST API)

Example SDN application

Real-time traffic balancing application

When the traffic load is high in the link $(s1, s2)$, configure UDP traffic in the path $h1 \rightarrow s1 \rightarrow s2 \rightarrow h2$ but, the rest on the path $h1 \rightarrow s1 \rightarrow s4 \rightarrow s4 \rightarrow s2 \rightarrow h2$



This is possible with SDN! 😊

But not limited to this... e.g., traffic steering based on QoE
[Calvigioni et al., 2018]

With traditional networking

- ▶ Configure s2 to receive “the rest” of the traffic via s4 (ongoing issues...)
- ▶ Configure s4, to forward “the rest” of the traffic from s3 to s2 (ongoing issues...)
- ▶ ...
- ▶ Traffic spike is over... 🙄
- ▶ ...
- ▶ Finally, configure s1...

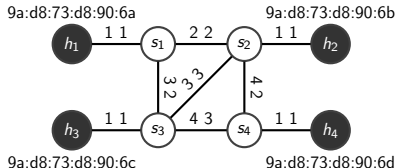
Example northbound requests

Installing a Layer 2 data path

To configure the path $h1 \rightarrow s1 \rightarrow s2 \rightarrow h2$:

Switch	IN_PORT	ETH_SRC	ETH_DST	OUTPUT
s1	1	9a:d8:73:d8:90:6a	ff:ff:ff:ff:ff:ff	ALL
s1	1	9a:d8:73:d8:90:6a	9a:d8:73:d8:90:6b	2
s2	2	9a:d8:73:d8:90:6a	ff:ff:ff:ff:ff:ff	1
s2	2	9a:d8:73:d8:90:6a	9a:d8:73:d8:90:6b	1

Example Dataplane



- Note that in reality the device s1 is assigned an ID like "of:00000000000000001"
- We are taking care of ARP!

REST request

```
{
  "flows": [{
    "priority": 40000, "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:00000000000000001",
    "treatment": {
      "instructions": [
        { "type": "OUTPUT", "port": "ALL" }
      ],
      "selector": {
        "criteria": [
          { "type": "IN_PORT", "port": "1" },
          { "type": "ETH_SRC",
            "mac": "9a:d8:73:d8:90:6a" },
          { "type": "ETH_SRC",
            "mac": "ff:ff:ff:ff:ff:ff" }
        ]
      }
    }
  ]
}
```

Demo

Outline

Introduction

Traditional Computer Networks

Software Defined Networking

Conclusion

Conclusion

- ▶ SDN helps removing some limitations of “traditional” networks, e.g., avoids error-prone human configurations, time consuming (re-)configuration, etc.
- ▶ The SDN paradigm separates (and centralizes) the data plane from the control plane
- ▶ A single interface is used to configure the whole data plane
- ▶ SDN is an active field of research, please do think of contributing
- ▶ Please, do not confuse or mix SDN with having network functions (switches, firewalls, etc.) in software / virtualized, this is a common misconception
 - ↪ However, they can be combined, and it is commonly done

Installing your components

- ▶ The ONOS [Berde et al., 2014] (SDN) controller

↪ Pro tip: A really quick way to have it running, use docker:

`docker pull onosproject/onos`, and execute it:

```
docker run -d -p 8181:8181 -p 6633:6633 -p 6653:6655 onosproject/onos
```

- ▶ The Containernet [Peuster et al., 2016] (SDN) “emulator”

↪ Pro tip: A really quick way to have it running, use docker:

`docker pull containernet/containernet`, and execute it:

```
docker run --rm -it -v /var/run/docker.sock:/var/run/docker.sock \
--privileged --pid='host' containernet/containernet /bin/bash
```

Run a custom topology of your own (in Containernet)

- ▶ Use an external controller, i.e., the Onos you installed
- ▶ Take the custom file as an **example**:

https://raw.githubusercontent.com/jorgelopezcoronado/SDNLab/master/custom_ctnnnet_topology.py

Lab (cont.)

Connect one host to the rest in your topology

- ▶ Though MAC addresses might be the easiest (and recommended) way, of course you are free to experiment as you like
- ▶ Tip: Do not forget about ARP
- ▶ Tip: You can use the default generated documentation, i.e., `http://localhost:8181/onos/v1/docs/#!/flows/post_flows`, although you are not forced, you can write your own REST requests

What to send?

- ▶ The python file with your topology
- ▶ The contents of the rules posted to Onos

References I



Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al. (2014).
Onos: towards an open, distributed sdn os.

In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM.



Calvigioni, G., Aparicio-Pardo, R., Sassatelli, L., Leguay, J., Medagliani, P., and Paris, S. (2018).

Quality of experience-based routing of video traffic for overlay and ISP networks.

In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*, pages 935–943.



Kozierok, C. M. (2005).

The TCP/IP guide: a comprehensive, illustrated Internet protocols reference.
No Starch Press.



Open-Networking-Foundation (2013).

Openflow switch specification v1. 4.0.



Open-Networking-Foundation (2014).

Sdn testing & validation onf sdn solutions showcase theme demonstrations.

<https://www.opennetworking.org/wp-content/uploads/2014/07/IXIA-demo.pdf>.



Peuster, M., Karl, H., and van Rossem, S. (2016).

Medicine: Rapid prototyping of production-ready network services in multi-pop environments.

In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 148–153.