This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.

tracebox: Topology Measurement and Impairement Discovery

RCM tutorial

Korian Edeline
University of Liège

mami
measurement and architecture for a middleboxed internet

LIÈGE université

Launch the VM

$ git clone https://github.com/mami-project/vpp-mb **-b rcm**

Contains a **Vagrantfile**.

$ cd vagrant && vagrant up && vagrant ssh

Middlebox discovery ?

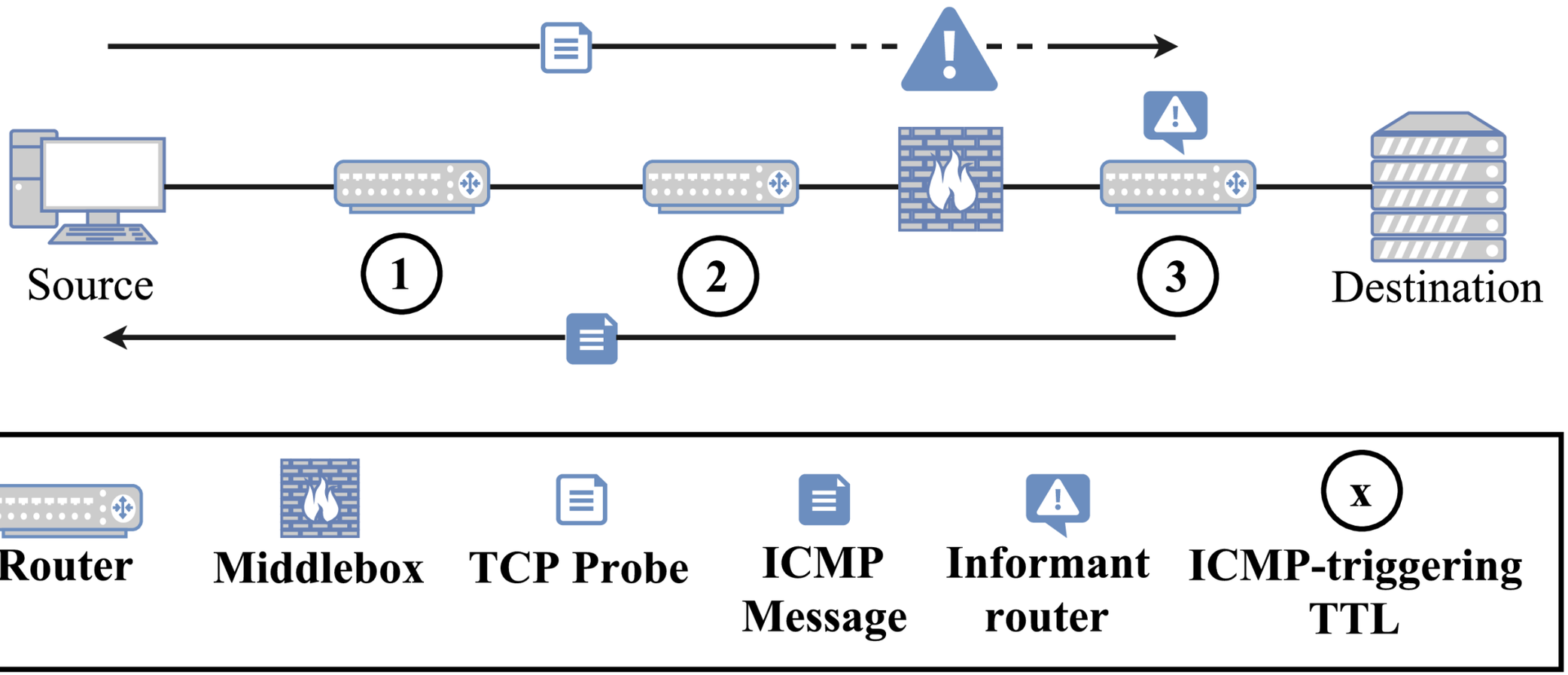
$ ./setup_topology **1**

$ ./setup_topology **2**

Destination is **10.0.0.10**

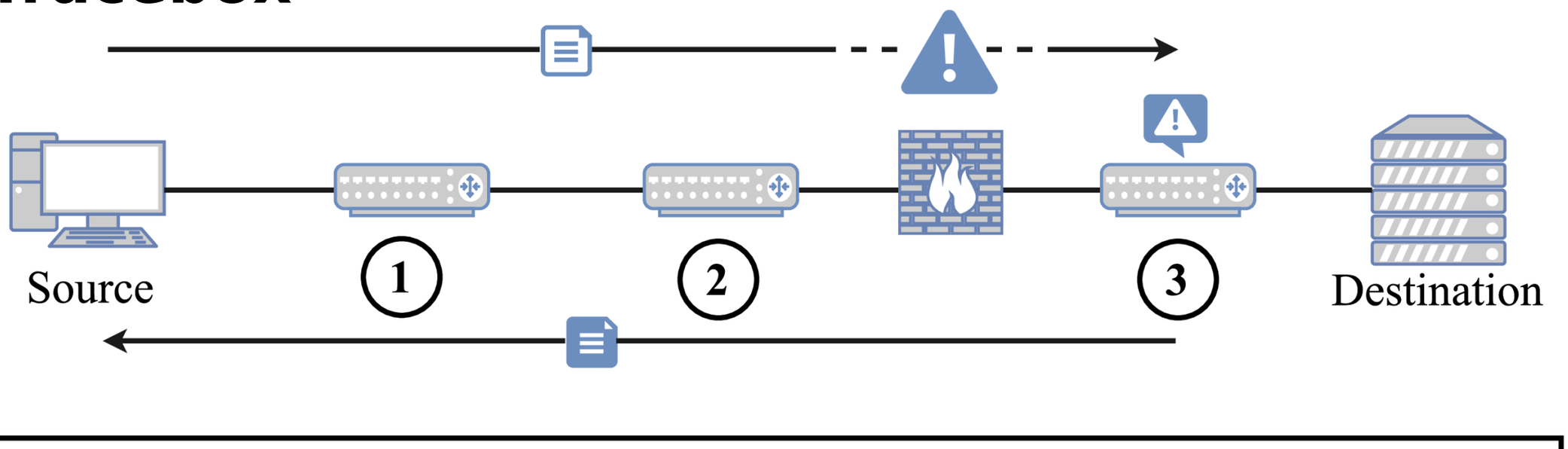
Middlebox discovery ?

- **Tracebox**



Middlebox discovery ?

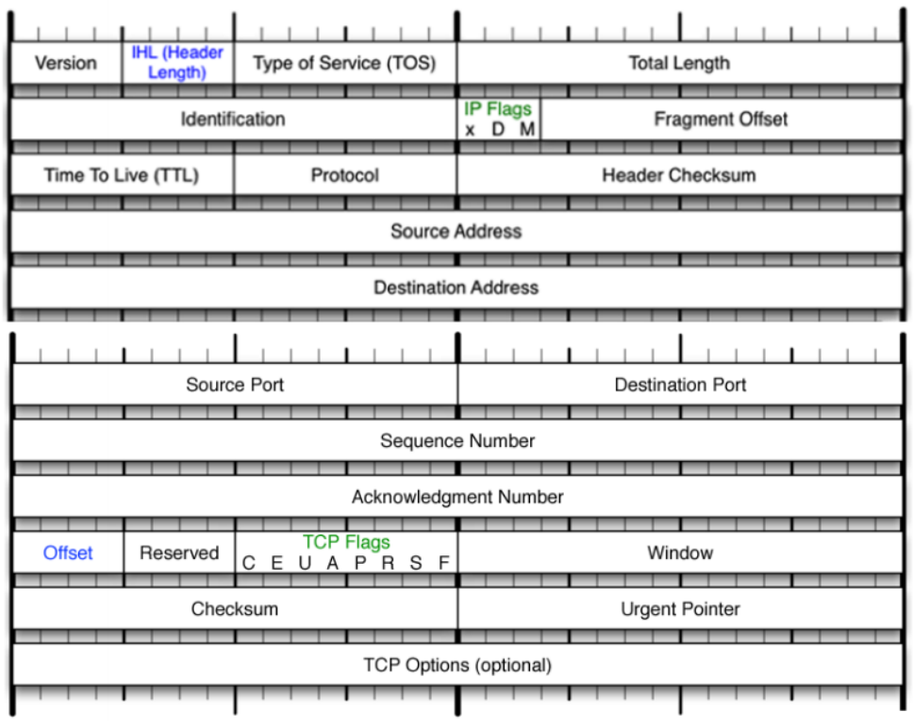
- **Tracebox**



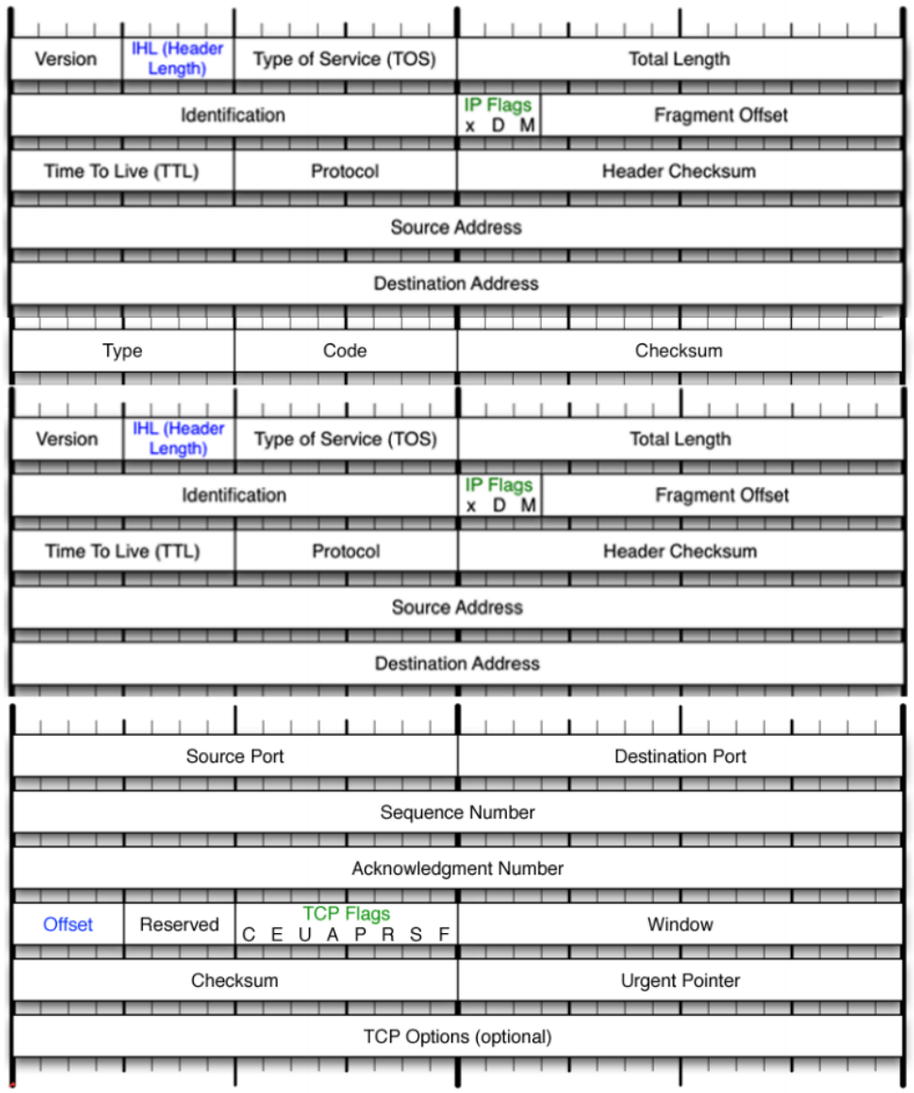
- Middlebox **Detection**

Middlebox discovery ?

Probe Sent

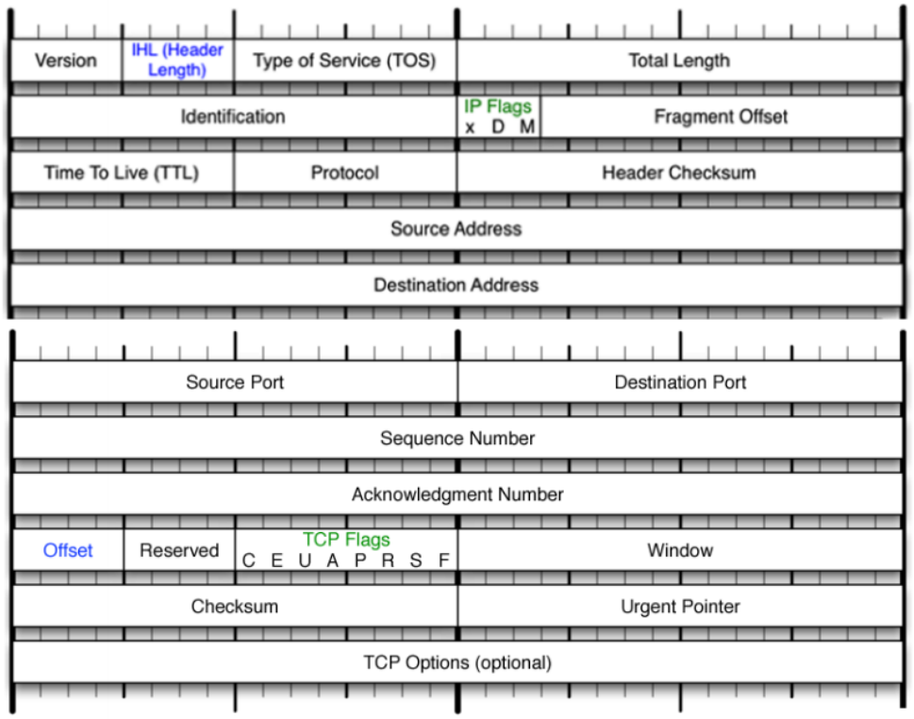


ICMP Received

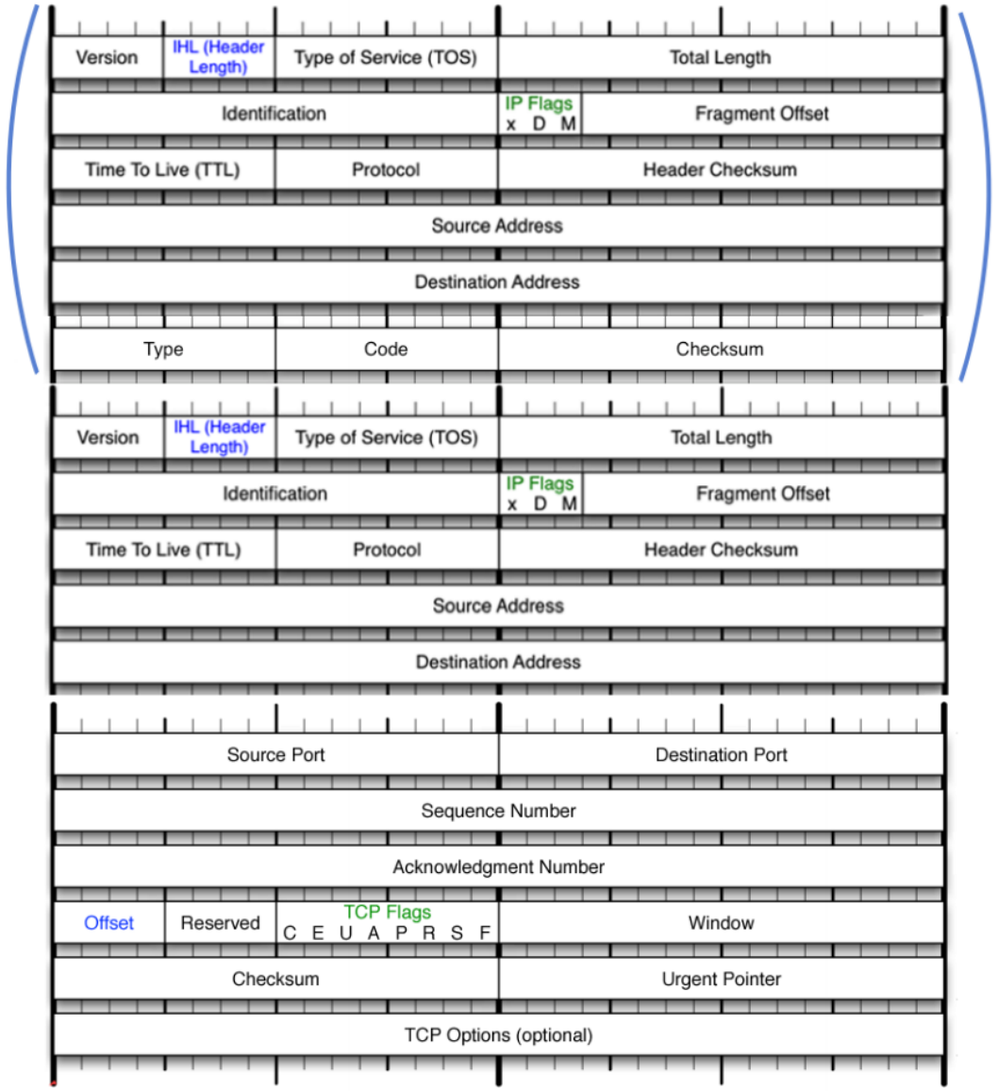


Middlebox discovery ?

Probe Sent

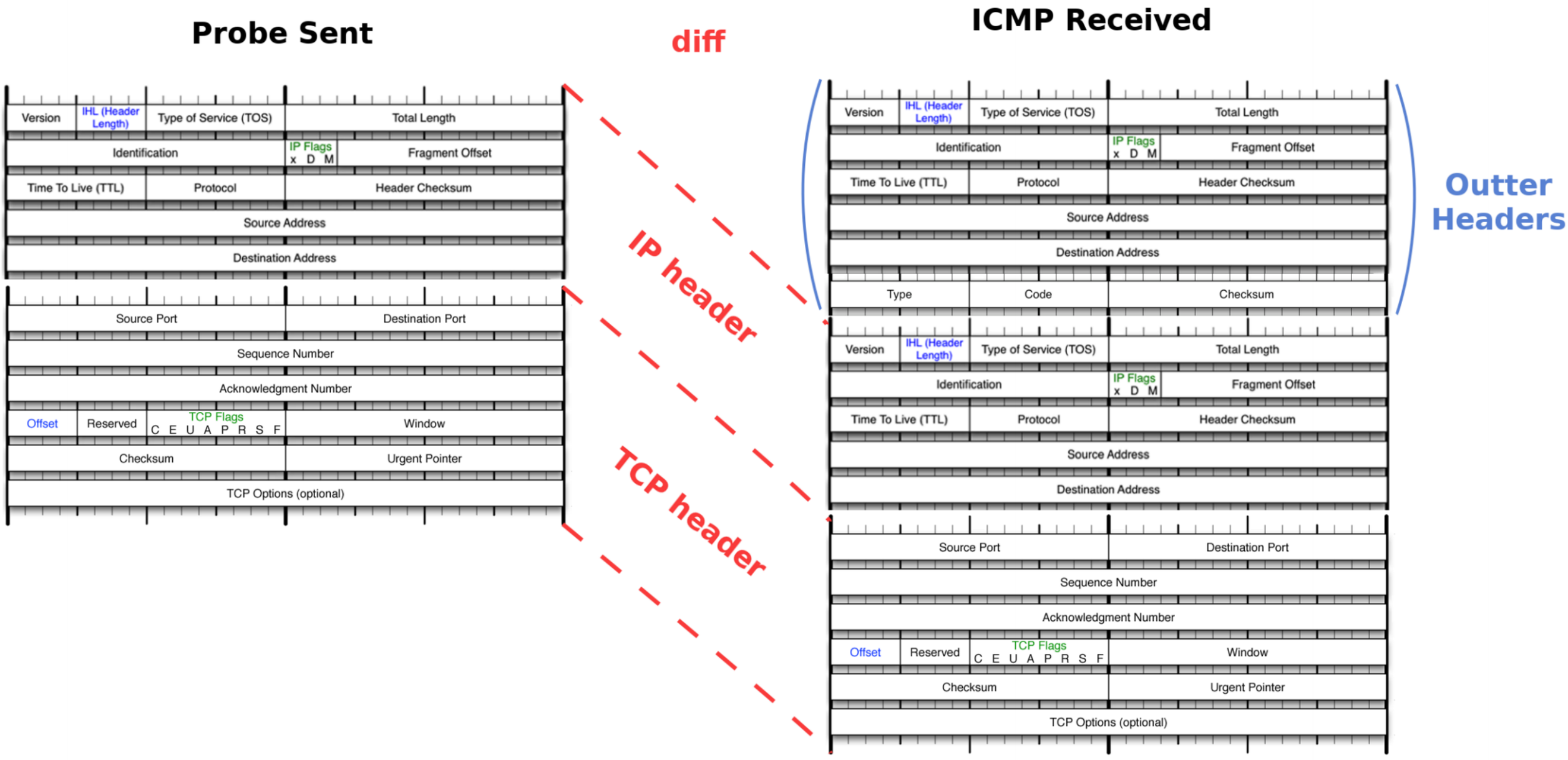


ICMP Received



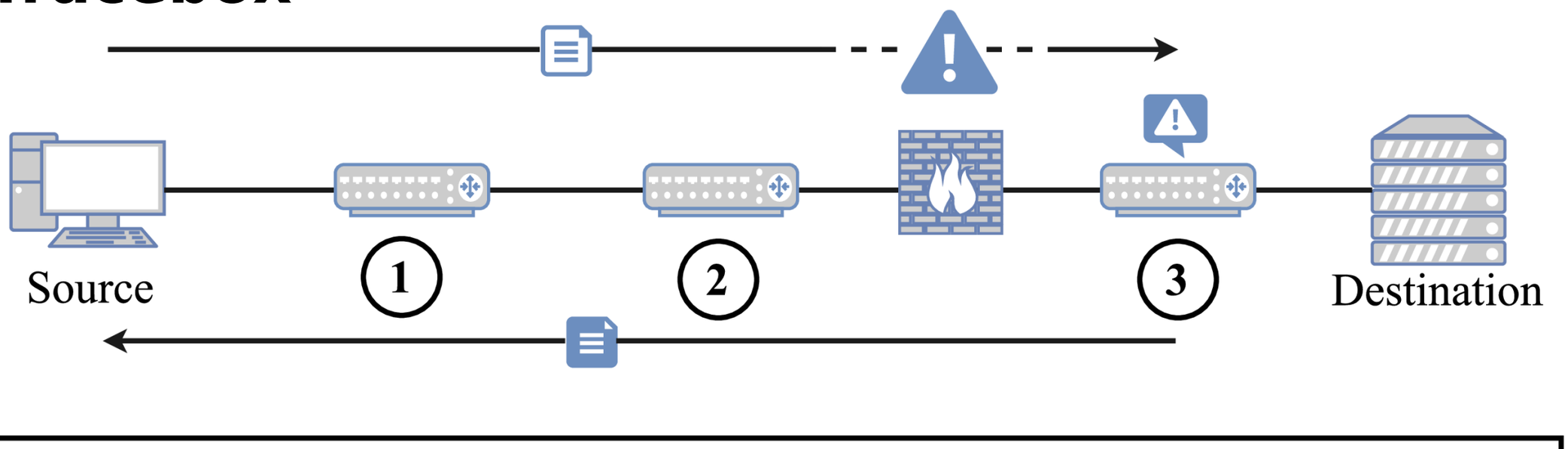
Outter Headers

Middlebox discovery ?



Middlebox discovery ?

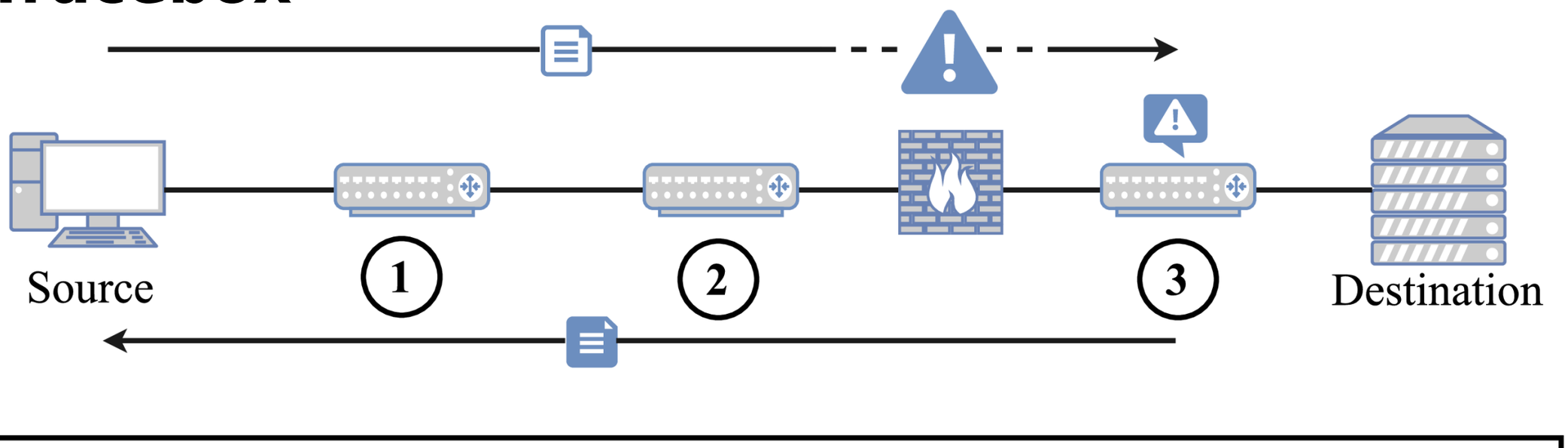
- **Tracebox**



- Middlebox **Detection**

Middlebox discovery ?

- **Tracebox**



- Middlebox **Detection**
- Middlebox **Location**

Location precision

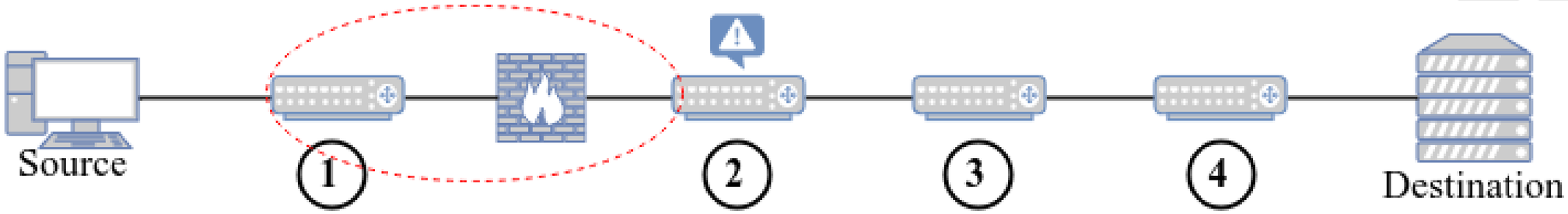
- **Quotation size:**
  - **RFC 792** : “The internet header plus the first 64 bits”
  - **RFC 1812** : “as much [...] as possible” (< 576 B)
    - Default on Linux, Cisco OSX, HP, Alcatel, ...

Location precision

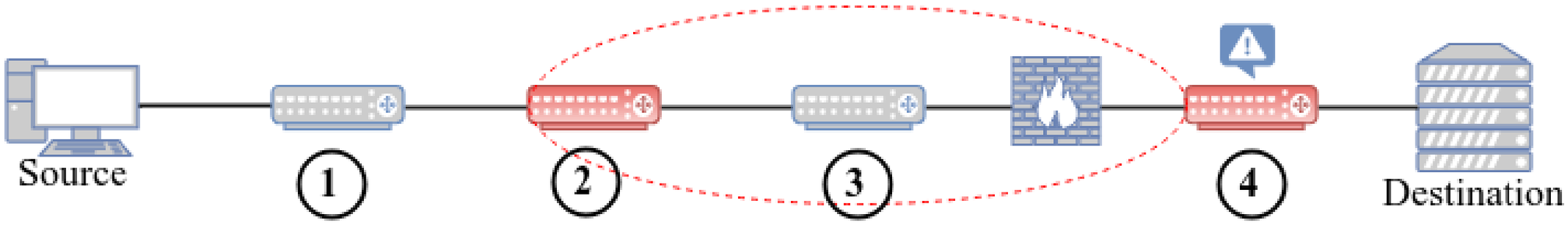
- **Quotation size:**
  - **RFC 792** : “The internet header plus the first 64 bits”
  - **RFC 1812** : “as much [...] as possible” (< 576 B)
    - Default on Linux, Cisco OSX, HP, Alcatel, ...

- 90% of all path at least contain one RFC1812 router
  - Edeline, K., & Donnet, B, “A First Look at the Prevalence and Persistence of Middleboxes in the Wild” In 29th International Teletraffic Congress (ITC 29), 2017.

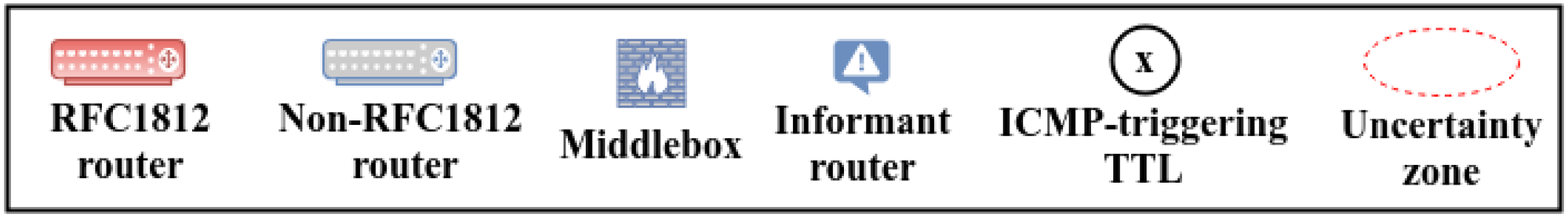
Middlebox discovery ?



1. Modified field is within the first 28 bytes



2. Modified field is outside the first 28 bytes

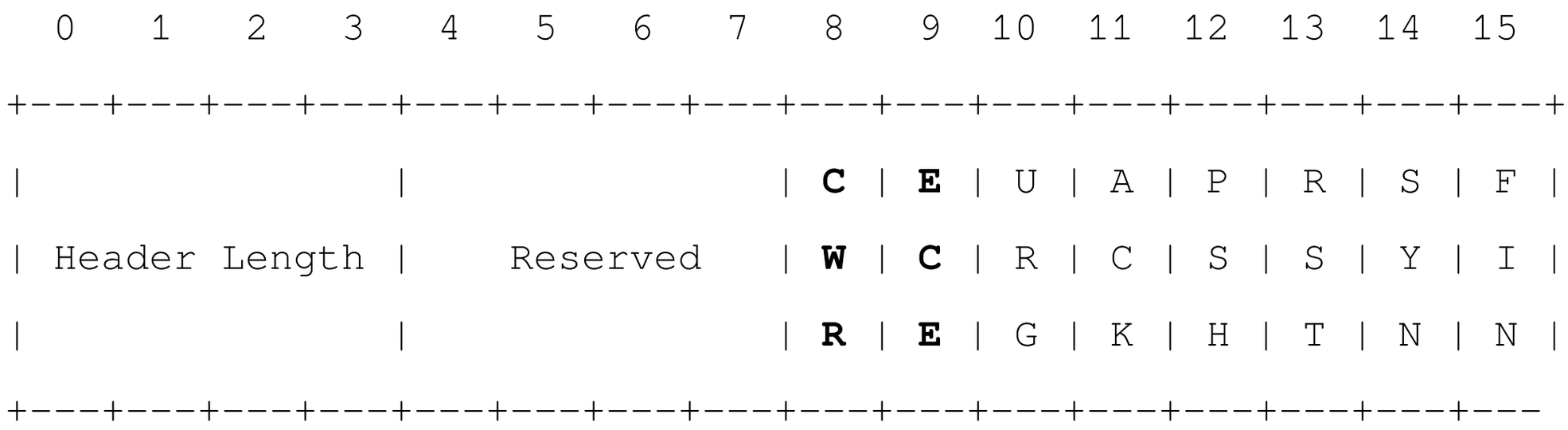


Explicit Congestion Notification (ECN)

- What is ECN ?
  - End-to-end notification of network congestion without dropping packets
  - ECN-aware router may **set a mark** in the IP header **instead of dropping** a packet in order to signal impending congestion.

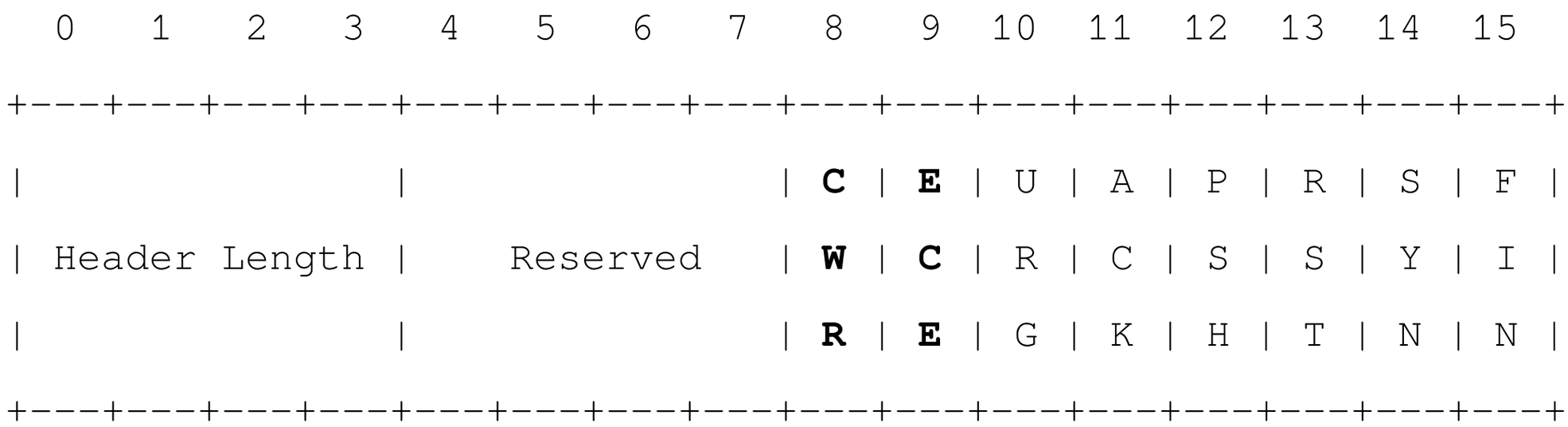
ECN Negotiation

- ECN Negotiation with TCP



ECN Negotiation

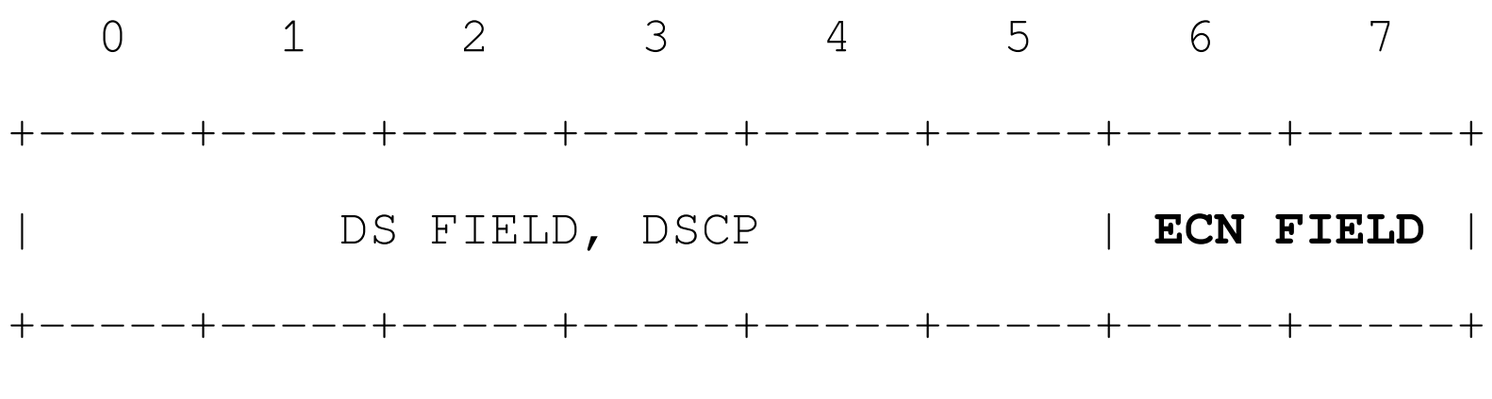
- ECN Negotiation with TCP



  - **ECE** & **CWR** set in SYN packet
  - **ECE** & **!CWR** set in SYN ACK packet

ECN Marking

- ECN in IP:



ECN Marking

- ECN in IP:

```
    0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|          DS FIELD, DSCP                 | ECN FIELD |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

	ECT	**CE**	
	0	0	Not-ECT
ECN-Capable Transport	0	1	**ECT**(1)
ECN-Capable Transport	1	0	**ECT**(0)
Congestion Experienced	1	1	**CE**

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).
- RFC2597 recommends the following Code Points:

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).
- RFC2597 recommends the following Code Points:
  - Best-Effort/Default: 000 000

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).
- RFC2597 recommends the following Code Points:
  - Best-Effort/Default: 000 000
  - Expedited Forwarding (EF): 101 110

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).
- RFC2597 recommends the following Code Points:
  - Best-Effort/Default: 000 000
  - Expedited Forwarding (EF): 101 110
  - Assured Forwarding (AF):

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec	001 010	010 010	011 010	100 010
Medium Drop Prec	001 100	010 100	011 100	100 100
High Drop Prec	001 110	010 110	011 110	100 110

IP Features: DSCP

- **Differentiated Services Code Points (DSCP)** is used to map traffic to **QoS** policies inside a DiffServ domain by associating value (Code Points) to Per Hop Behaviors (PHB).
- RFC2597 recommends the following Code Points:
  - Best-Effort/Default: 000 000
  - Expedited Forwarding (EF): 101 110
  - Assured Forwarding (AF):

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec	001 010	010 010	011 010	100 010
Medium Drop Prec	001 100	010 100	011 100	100 100
High Drop Prec	001 110	010 110	011 110	100 110

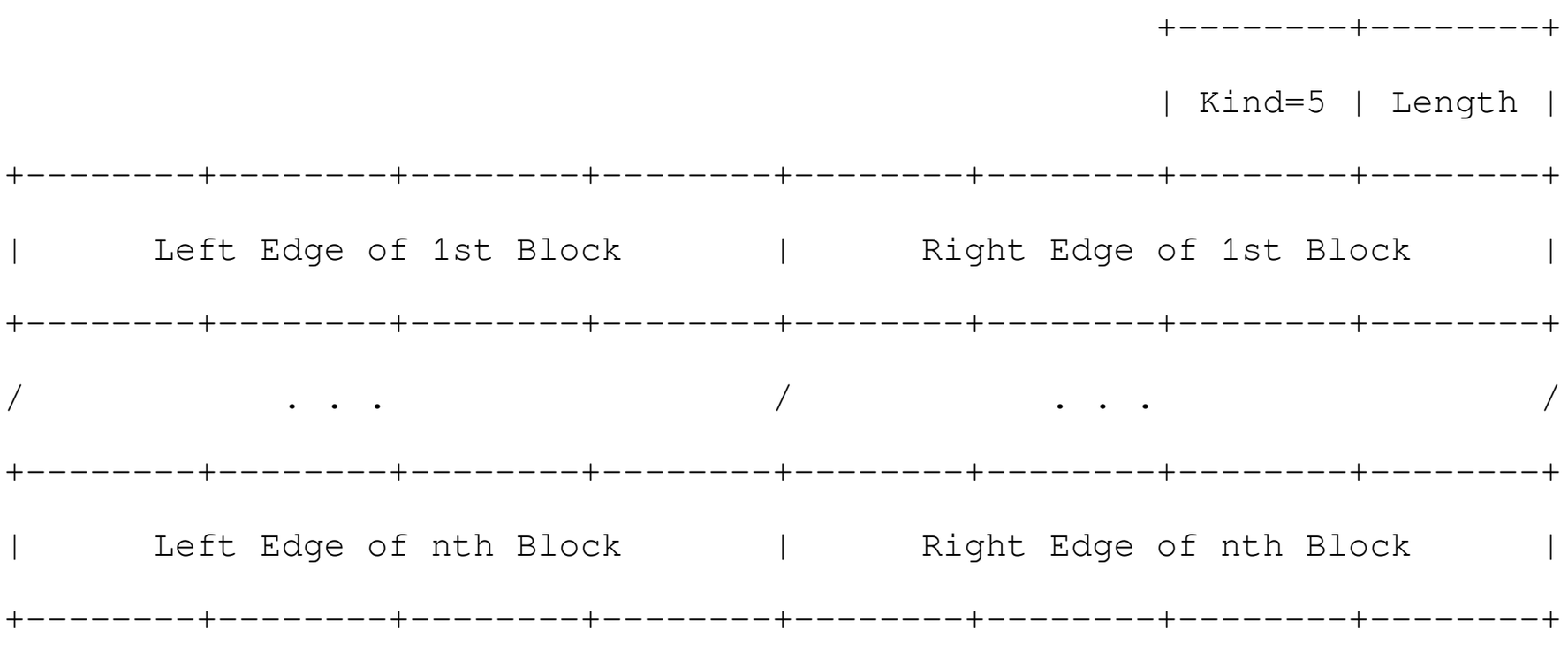
  - 8 Class selectors for backward compatibility with ToS precedence

TCP Features: Selective ACKnowledgements (SACK)

- TCP option to address throughput reduction and unnecessary retransmissions when multiple packets are lost from a single window.

TCP Features: Selective ACKnowledgements (SACK)

- TCP option to address throughput reduction and unnecessary retransmissions when multiple packets are lost from a single window.



All tools you shall need

All tools you shall need

- **ping**
  - **$** ping 10.0.0.10

All tools you shall need

- **ping**
  - **$** ping 10.0.0.10
  - ***run ping first***

All tools you shall need

- **ping**
  - **$** ping 10.0.0.10
  - ***run ping first***
- **traceroute**
  - **$** traceroute 10.0.0.10

All tools you shall need

- **ping**
  - **$** ping 10.0.0.10
  - ***run ping first***
- **traceroute**
  - **$** traceroute 10.0.0.10
- **tcptraceroute**
  - **$** sudo tcptraceroute 10.0.0.10

All tools you shall need

- **ping**
  - **$** ping 10.0.0.10
  - ***run ping first***
- **traceroute**
  - **$** traceroute 10.0.0.10
- **tcptraceroute**
  - **$** sudo tcptraceroute 10.0.0.10
- **nmap**
  - **$** nmap 10.0.0.10

tracebox

tracebox

- **$** sudo scamper -c “**tracebox**” -i 10.0.0.10

tracebox

- **$** sudo scamper -c “**tracebox**” -i 10.0.0.10
- **$** sudo scamper -c “tracebox **-v**” -i 10.0.0.10

tracebox

- **$** sudo scamper -c “**tracebox**” -i 10.0.0.10
- **$** sudo scamper -c “tracebox **-v**” -i 10.0.0.10
- **$** sudo scamper -c “tracebox -v **-p ip/tcp**” …

tracebox

- **$** sudo scamper -c “**tracebox**” -i 10.0.0.10
- **$** sudo scamper -c “tracebox **-v**” -i 10.0.0.10
- **$** sudo scamper -c “tracebox -v **-p ip/tcp**” …
- **$** … “tracebox -v -p ip/tcp/**CWR/ECE**” …
- **$** … “tracebox -v -p ip/tcp/CWR/ECE/**MSS(1460)**” …
- **$** … “tracebox -v -p ip/tcp/**ECT**” …

All tools you shall need (cont.)

All tools you shall need (cont.)

- **wget**
  - download file from server
  - **$** `wget 10.0.0.10`

All tools you shall need (cont.)

- **wget**
  - download file from server
  - **$** `wget 10.0.0.10`
- **sysctl**
  - configure probes

All tools you shall need (cont.)

- **wget**
  - download file from server
  - **$** `wget 10.0.0.10`
- **sysctl**
  - configure probes
    - **$** `sudo sysctl net.ipv4.tcp_<feature>`
    - **$** `sudo sysctl -w net.ipv4.tcp_<feature>=<val>`

All tools you shall need (cont.)

- **wget**
  - download file from server
  - **$** `wget 10.0.0.10`
- **sysctl**
  - configure probes
    - **$** `sudo sysctl net.ipv4.tcp_<feature>`
    - **$** `sudo sysctl -w net.ipv4.tcp_<feature>=<val>`
  - reseted when instanciating topologies
    - **$** `sudo sysctl -p`

Topology #1

Topology #1

```
$ ./setup_topology 1
```

Topology #1

```
$ ./setup_topology 1
```

Scenario: “Slow HTTP traffic from server”

Topology #1

```
$ ./setup_topology 1
```

Scenario: “Slow HTTP traffic from server”

```
$ wget 10.0.0.10/big_buck_bunny_720p_10mb.mp4
```

Topology #1

- Observation #1: Middlebox sets **1452 Bytes** Maximum Segment Size (**MSS**)

Topology #1

- Observation #1: Middlebox sets **1452 Bytes** Maximum Segment Size (**MSS**)
  - **$** sudo scamper –c “tracebox –v –p ip/tcp/**mss**” \
    –i 10.0.0.10

Topology #1

- Observation #2: Middlebox sets **CE** flag on all **ECN-Capable** packets
  - **$** `sudo scamper -c "tracebox -v -p ip/tcp/`**ect**`" -i 10.0.0.10`

Topology #1

- Observation #2: Middlebox sets **CE** flag on all **ECN-Capable** packets
  - **$** `sudo scamper -c "tracebox -v -p ip/tcp/`**ect**`" -i 10.0.0.10`
  - **CE** packets are *treated as* **lost** packets by sender
  - Sender keeps reducing send rate

Topology #1

- Observation #2: Middlebox sets **CE** flag on all **ECN-Capable** packets
  - **$** sudo scamper –c “tracebox –v –p ip/tcp/**ect**” –i 10.0.0.10
  - **CE** packets are *treated as* **lost** packets by sender
  - Sender keeps reducing send rate
  - ECN use can be set by changing the kernel parameters at runtime
    - **$** sudo sysctl –w net.ipv4.**tcp_ecn**=**0**

Topology #1

$ ./setup_topology **1**

Scenario: “Slow HTTP traffic from server”

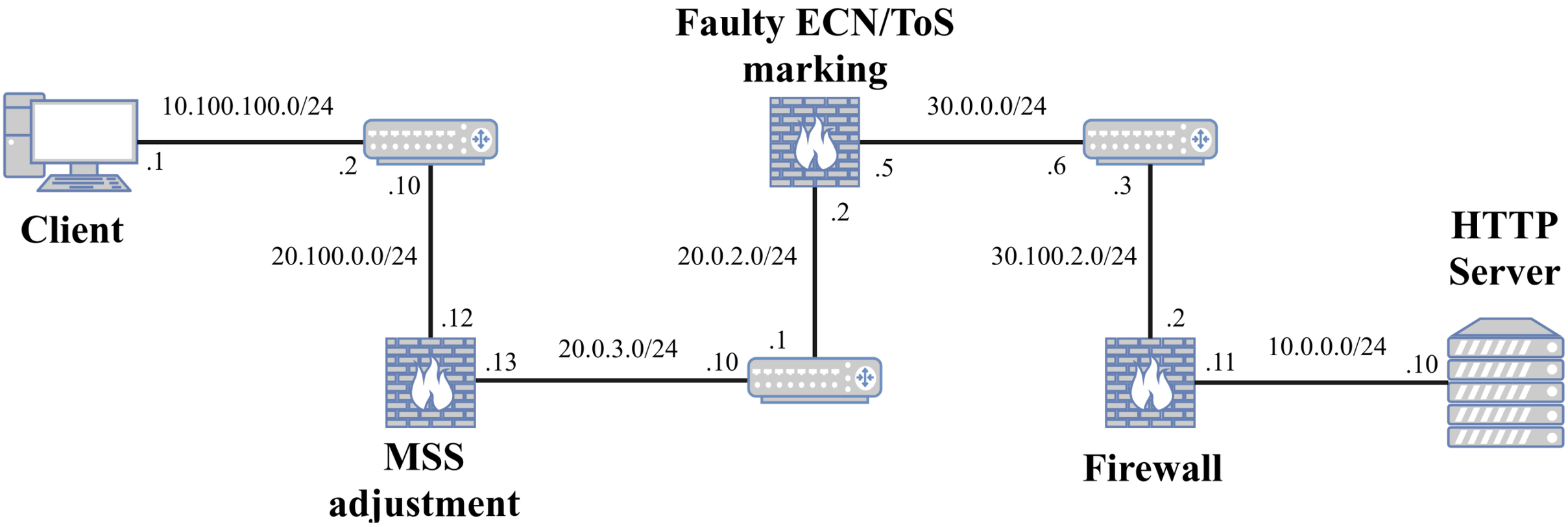
Problem:

- Middlebox reporting inexistant congestion

Solution:

- Disable ECN

Topology #1


Faulty ECN/ToS marking
10.100.100.0/24
.1
.2
.10
Client
20.100.0.0/24
20.0.2.0/24
30.0.0.0/24
.5
.6
.3
.2
30.100.2.0/24
HTTP Server
.12
.13
20.0.3.0/24
.10
.1
.2
.11
10.0.0.0/24
.10
MSS adjustment
Firewall

Topology #2

$ ./setup_topology **2**

Scenario: “No connectivity to HTTP server”

Topology #2

- Observation #1: **ECN-setup SYN** packets are **dropped** by router

Topology #2

- Observation #1: **ECN-setup SYN** packets are **dropped** by router
- ECN **Fallback** mechanism:

  (1) Host A: Sends an ECN-setup SYN.

  (2) Host B: Sends an ECN-setup SYN/ACK, packet is dropped or delayed.

  (3) Host A: Sends a non-ECN-setup SYN.

  (4) Host B: Sends a non-ECN-setup SYN/ACK.

Topology #2

- Observation #1: **ECN-setup SYN** packets are **dropped** by router
- ECN **Fallback** mechanism:

  (1) Host A: Sends an ECN-setup SYN.

  (2) Host B: Sends an ECN-setup SYN/ACK, packet is dropped or delayed.

  (3) Host A: Sends a non-ECN-setup SYN.

  (4) Host B: Sends a non-ECN-setup SYN/ACK.

- **$** `sudo sysctl -w net.ipv4.`**`tcp_ecn_fallback`**`=1`

Topology #2

- Observation #2: Router sets **DSCP** (Differentiated Service Code Point)

Topology #2

- Observation #2: Router sets **DSCP** (Differentiated Service Code Point)
  - Best-Effort/Default: 000 000
  - Expedited Forwarding (EF): 101 110
  - Assured Forwarding (AF):

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec	001 010	010 010	011 010	100 010
Medium Drop Prec	001 100	010 100	011 100	100 100
High Drop Prec	001 110	010 110	011 110	100 110

  - 8 Class selectors for backward compatibility with ToS precedence

Topology #2

- Observation #2: Router sets **DSCP** (Differentiated Service Code Point)
  - Best-Effort/Default: 000 000
  - Expedited Forwarding (EF): 101 110
  - Assured Forwarding (AF):

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec	001 010	010 010	011 010	100 010
Medium Drop Prec	001 100	010 100	011 100	100 100
High Drop Prec	**001 110**	010 110	011 110	100 110

  - 8 Class selectors for backward compatibility with ToS precedence

Topology #2

- Observation #3: Middlebox sets/adds TCP MSS (Maximum Segment Size) option:

Topology #2

- Observation #3: Middlebox sets/adds TCP MSS (Maximum Segment Size) option:
  - 1460 Byte MSS to fit **standard ethernet** 1500-byte data segment

Topology #2

- Observation #3: Middlebox sets/adds TCP MSS (Maximum Segment Size) option:
  - 1460 Byte MSS to fit **standard ethernet** 1500-byte data segment
- Observation #4: Middlebox sets/adds WScale option:

Topology #2

- Observation #3: Middlebox sets/adds TCP MSS (Maximum Segment Size) option:
  - 1460 Byte MSS to fit **standard ethernet** 1500-byte data segment
- Observation #4: Middlebox sets/adds WScale option:
  - TCP receive window is limited to **65,535** bytes (16-bit field)
  - Window Scale Option increases it to **1 GB** by introducing a constant left-shift value (max. 14).

Topology #2

- Observation #3: Middlebox sets/adds TCP MSS (Maximum Segment Size) option:
  - 1460 Byte MSS to fit **standard ethernet** 1500-byte data segment
- Observation #4: Middlebox sets/adds WScale option:
  - TCP receive window is limited to **65,535** bytes (16-bit field)
  - Window Scale Option increases it to **1 GB** by introducing a constant left-shift value (max. 14).
  - TCP Window scale of 8 means that a constant factor of 2^8 must be applied to the receive window.
  - Receive window: $[0;2^{16}-1] * 2^8$

Topology #2

- Observation #5: Middlebox re-shuffle TCP sequence number

Topology #2

- Observation #5: Middlebox re-shuffle TCP sequence number
  - Old TCP stacks generates Initial Sequence Number (ISN) that **lacks of randomness**
    - TCP sequence prediction attack

Topology #2

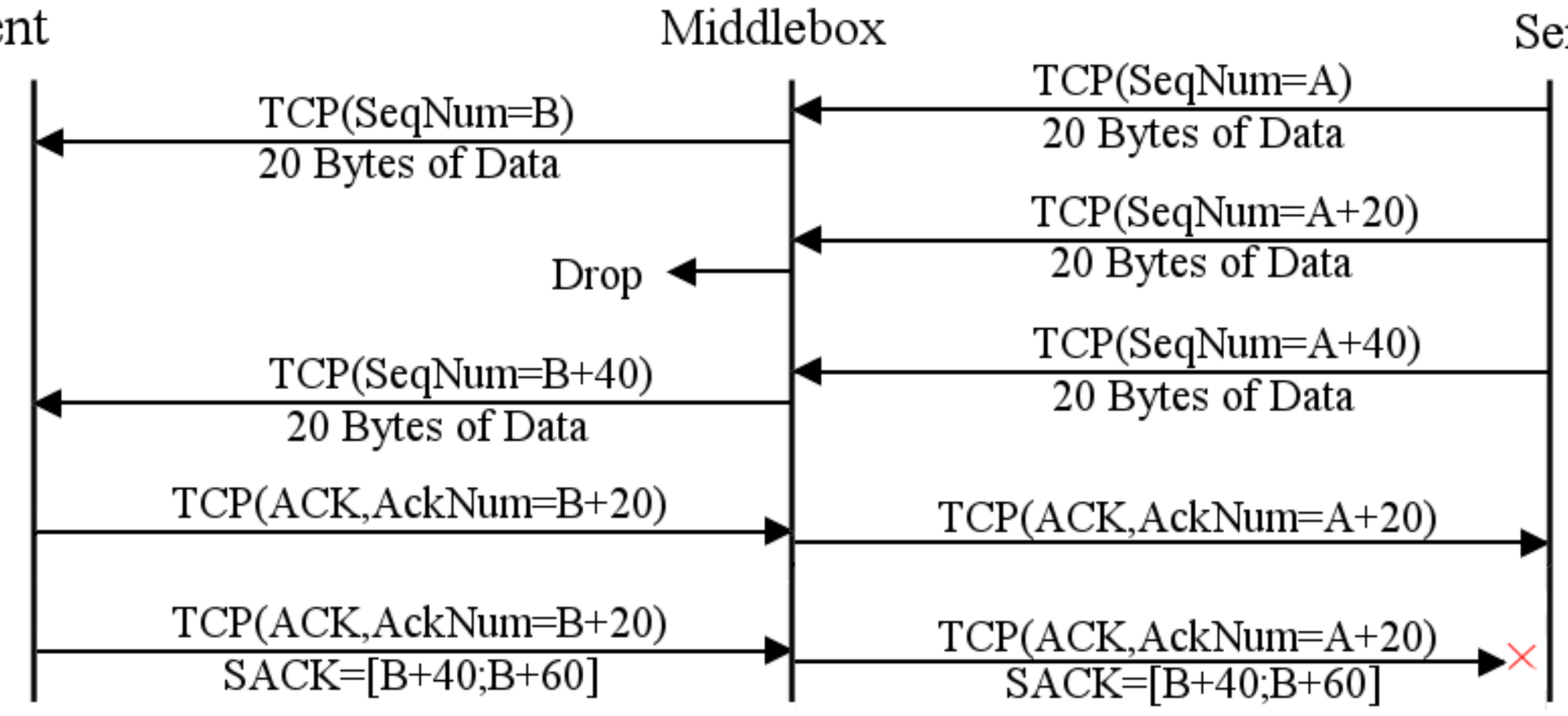
- Observation #5: Middlebox re-shuffle TCP sequence number
  - Old TCP stacks generates Initial Sequence Number (ISN) that **lacks of randomness**
    - TCP sequence prediction attack
  - The re-shuffle is a supposed to be a fix, but some stacks forgot **SACK**

TCP Features: Selective ACKnowledgements (SACK)

- TCP option to address throughput reduction and unnecessary retransmissions when multiple packets are lost from a single window.

TCP Features: Selective ACKnowledgements (SACK)

- TCP option to address throughput reduction and unnecessary retransmissions when multiple packets are lost from a single window.



Topology #2

- Observation #5: Middlebox re-shuffle TCP sequence number
  - Old TCP stacks generates Initial Sequence Number (ISN) that **lacks of randomness**
    - TCP sequence prediction attack
  - The re-shuffle is a supposed to be a fix, but some stacks forgot **SACK**

Topology #2

- Observation #5: Middlebox re-shuffle TCP sequence number
  - Old TCP stacks generates Initial Sequence Number (ISN) that **lacks of randomness**
    - TCP sequence prediction attack
  - The re-shuffle is a supposed to be a fix, but some stacks forgot **SACK**
  - Endpoints may discard a **duplicate ACK** with an invalid **SACK** block
  - Other middleboxes may discards them as invalid packets

Topology #2

- Observation #5: Middlebox re-shuffle TCP sequence number
  - Old TCP stacks generates Initial Sequence Number (ISN) that **lacks of randomness**
    - TCP sequence prediction attack
  - The re-shuffle is a supposed to be a fix, but some stacks forgot **SACK**
  - Endpoints may discard a **duplicate ACK** with an invalid **SACK** block
  - Other middleboxes may discards them as invalid packets
  - `$ sudo sysctl -w net.ipv4.tcp_sack=0`

Topology #2

$ ./setup_topology **2**

Scenario: “No connectivity to HTTP server”

Problem 1:

- ECN-setup blackhole

Solution 1:

- Enable ECN fallback

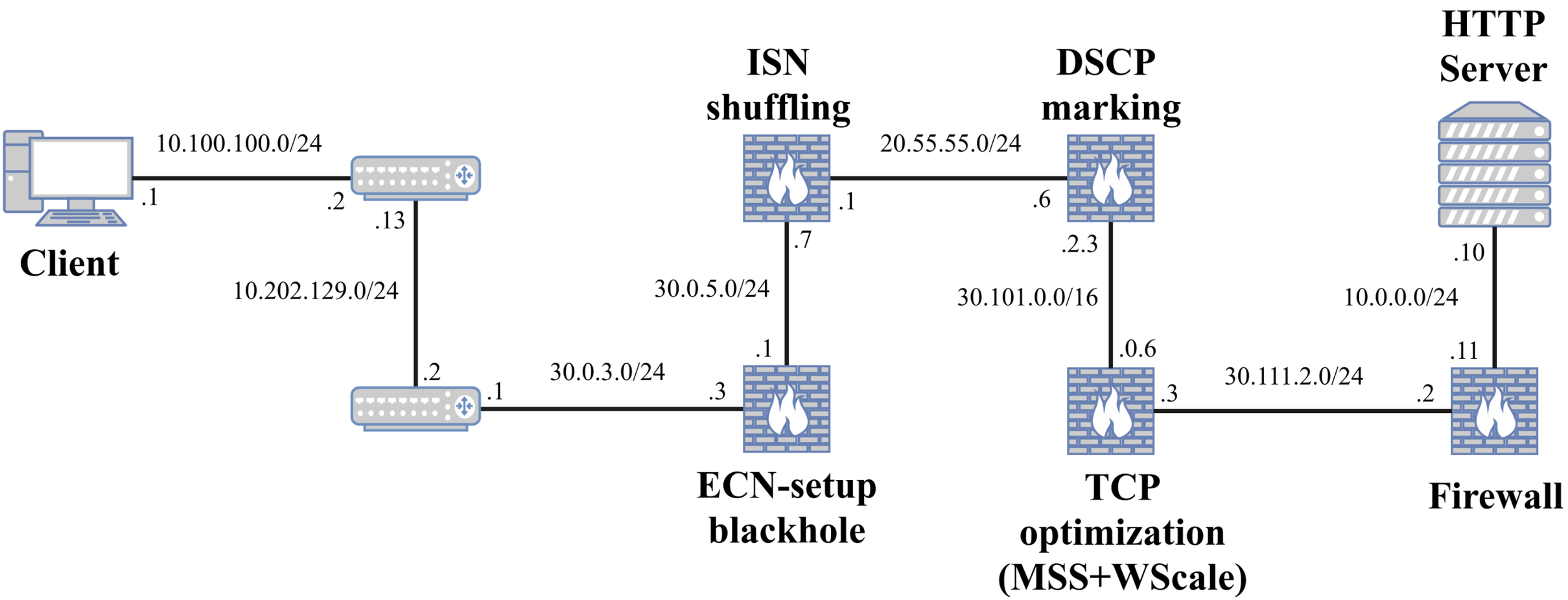
Problem 2:

- Invalid SACK blocks stall the connection

Solution 2:

- Disable SACK

Topology #2


10.100.100.0/24
.1
.2
.13
Client
10.202.129.0/24
.2
.1
30.0.3.0/24
.3
.1
30.0.5.0/24
.7
ECN-setup
blackhole
ISN
shuffling
20.55.55.0/24
.1
.6
.2.3
30.101.0.0/16
.0.6
.3
TCP
optimization
(MSS+WScale)
DSCP
marking
HTTP
Server
.10
10.0.0.0/24
.11
30.111.2.0/24
.2
Firewall

Thanks!

Tools used

- Testbed
  - fd.io
  - Configurable Middlebox plugin
    - https://github.com/mami-project/vpp-mb
- Tracebox
  - https://github.com/mami-project/tracebox