How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP

Costin Raiciu

University Politehnica of Bucharest

Joint work with: **Christoph Paasch, Sebastien Barre, Alan Ford, Fabien Duchene, Michio Honda, Olivier Bonaventure, Mark Handley**

Thanks to

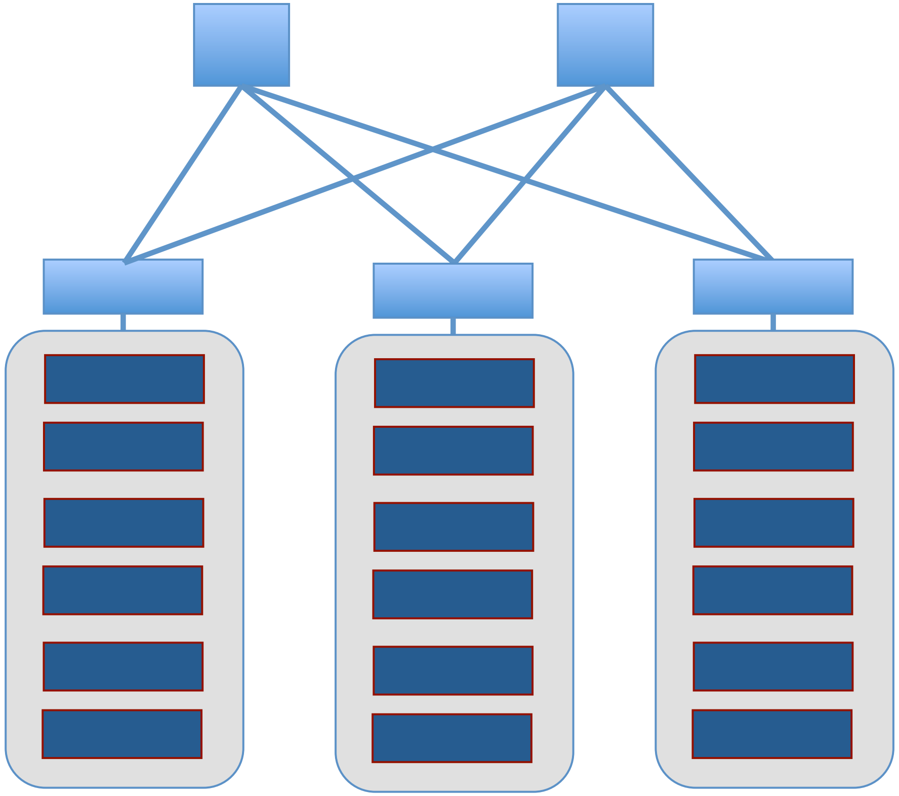
Networks are becoming multipath

Mobile devices have multiple wireless connections

Networks are becoming multipath

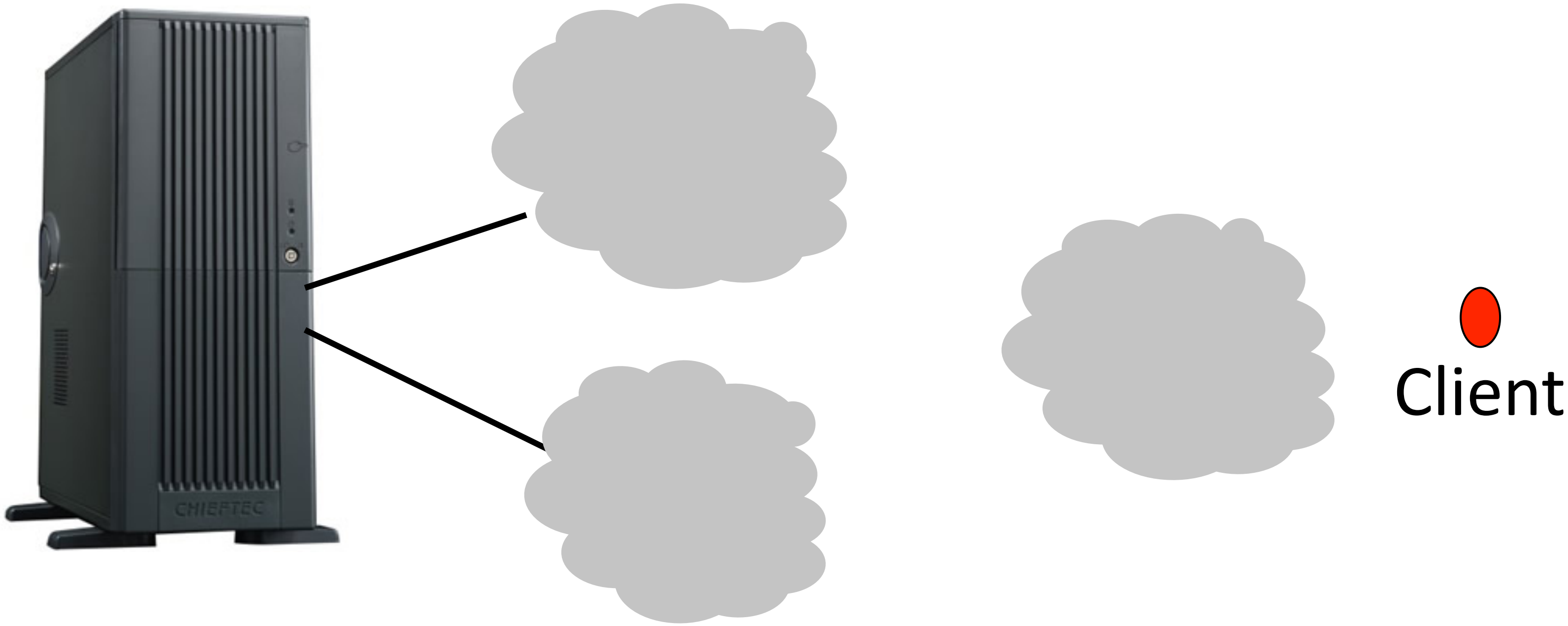
Datacenters have redundant topologies

Networks are becoming multipath



Datacenters have redundant topologies

Networks are becoming multipath



Servers are multi-homed

How do we use these networks?

TCP.

Used by most applications,
offers byte-oriented reliable delivery,
adjusts load to network conditions

TCP is single path

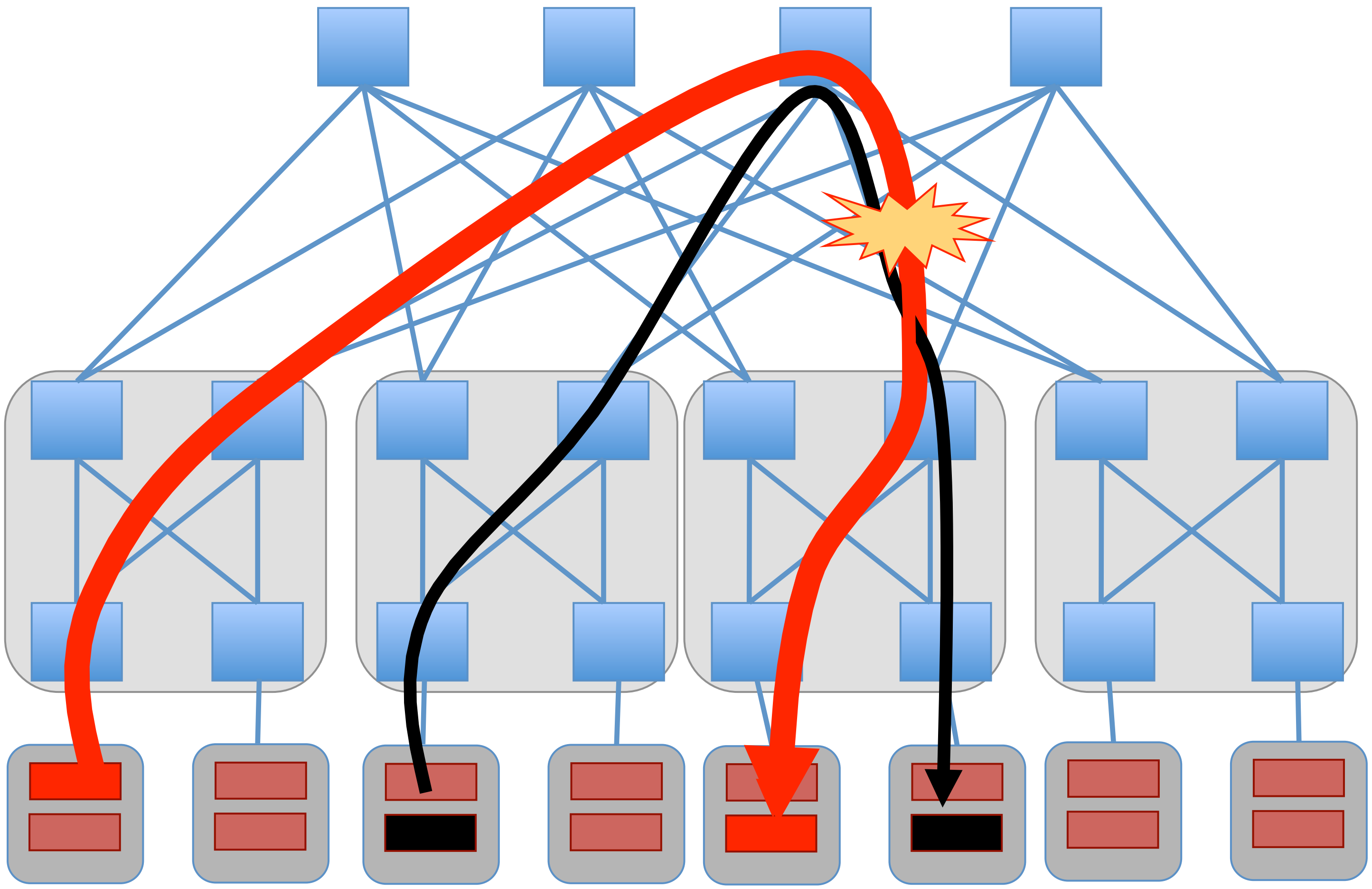
A TCP connection

Uses a **single-path** in the network regardless of network topology

Is **tied** to the **source** and **destination** addresses of the endpoints

Mismatch between network and transport creates **problems**

Collisions in datacenters



[Fares et al - A Scalable, Commodity Data Center Network Architecture - Sigcomm 2008]

How hard can it be?
Designing and
Implementing a
Deployable Multipath TCP

Deployable Multipath TCP

How hard can it be?

Designing

Implementing

Deployable Multipath TCP

How hard can it be?

Designing

Implementing

Goal: A Deployable Multipath TCP

We want to evolve TCP to be able to use multiple paths in the network.

Multipath TCP must meet the following goals:

GOAL 1: Support *unmodified applications*

GOAL 2: Work over *today's networks*

GOAL 3: *Work whenever TCP would work*

Our Linux kernel Multipath TCP implementation

supports legacy apps

and works well over:

deployed 3G and Wifi networks,

existing datacenters and

the Internet at large.

Deployable Multipath TCP

How hard can it be?

Designing

Implementing

It can be pretty hard.

It can be pretty hard.

Mark Handley *suggested* we start working on designing MPTCP in spring 2007

It can be pretty hard.

Mark Handley *suggested* we start working on designing MPTCP in spring 2007

Five years later, here we are –

we finally nailed this!

It can be pretty hard.

Mark Handley *suggested* we start working on designing MPTCP in spring 2007

Five years later, here we are –

we finally nailed this!

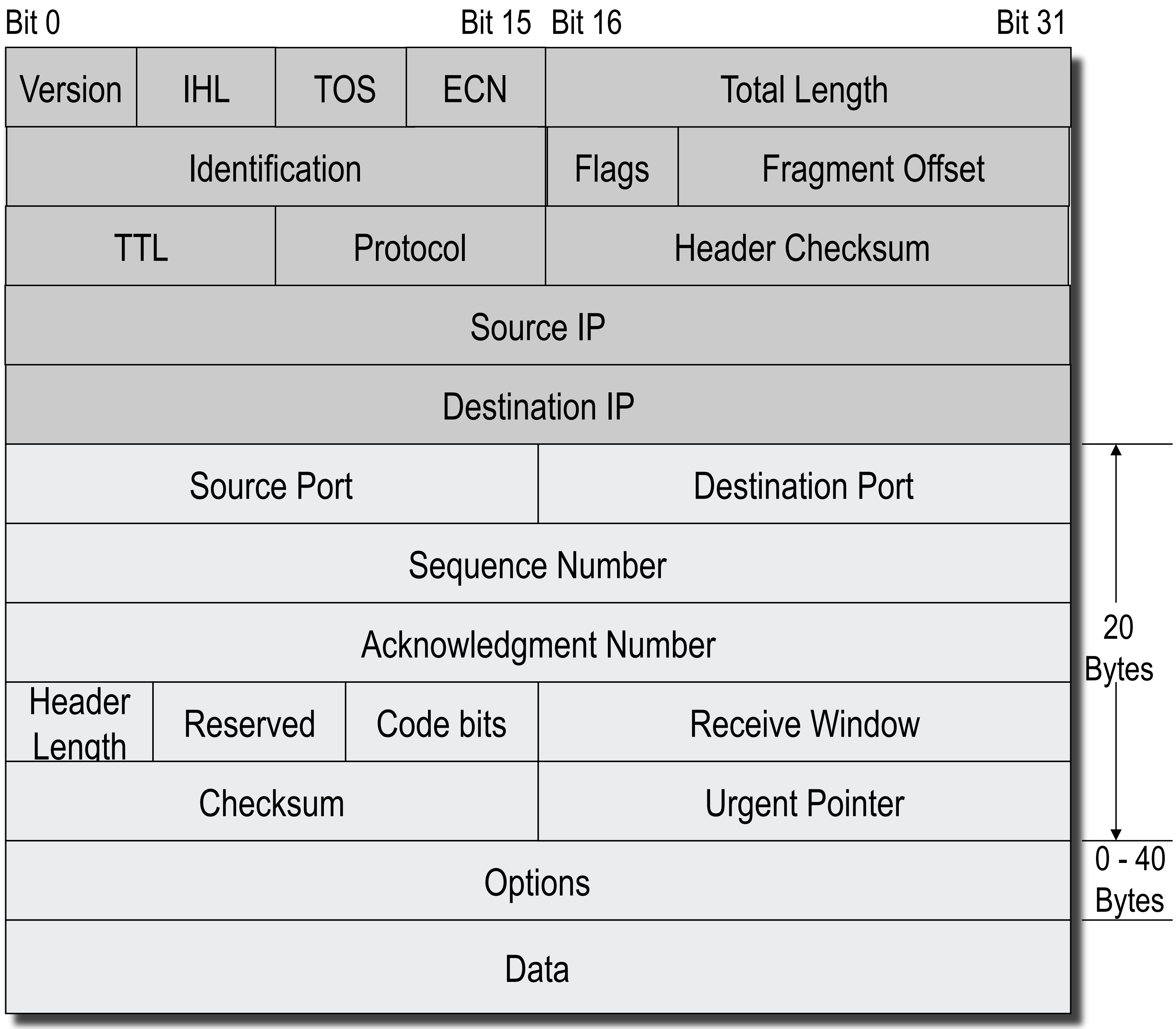
Why was it this difficult?

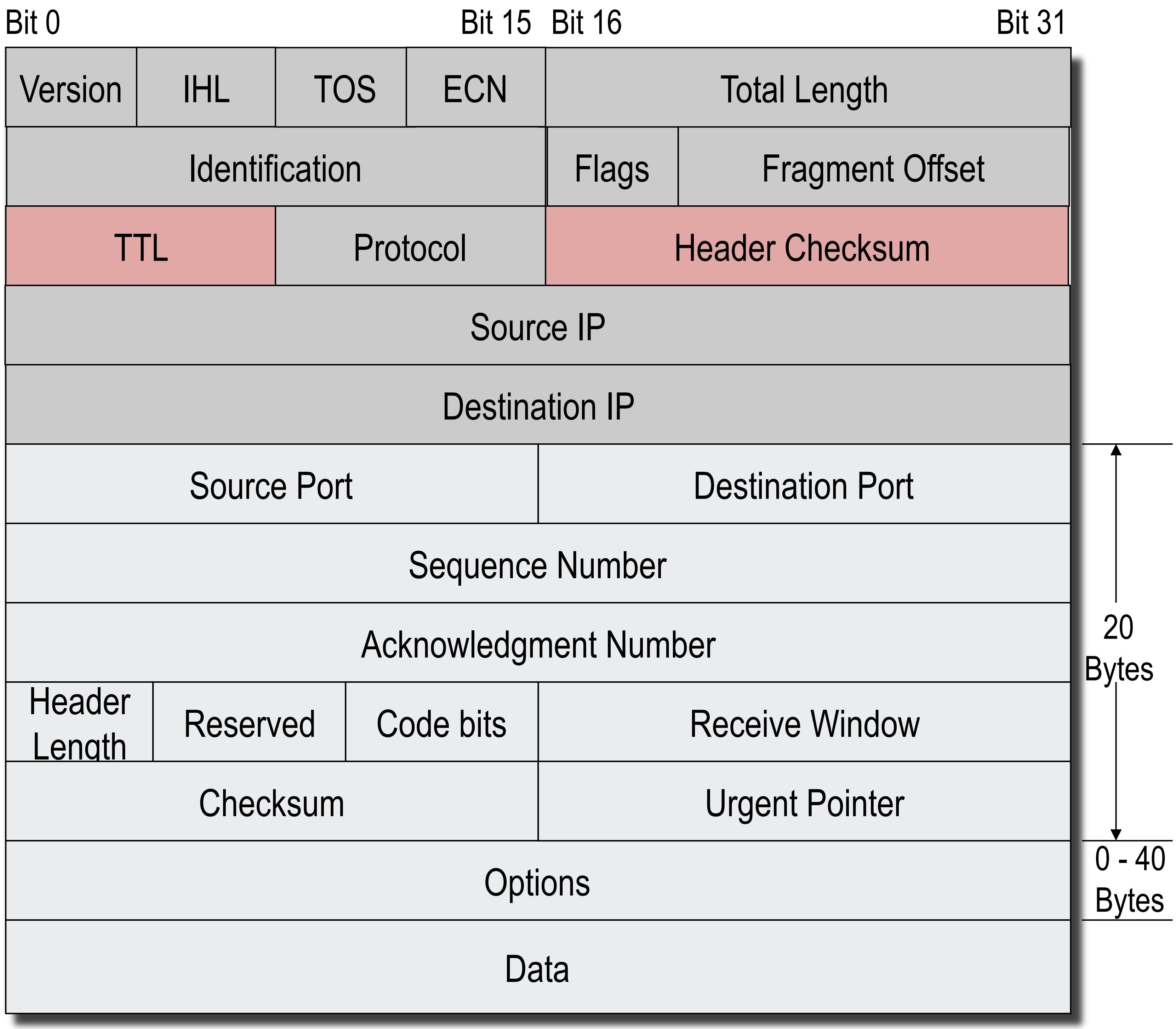
Internet Architecture is a living thing.

Protocol Layering

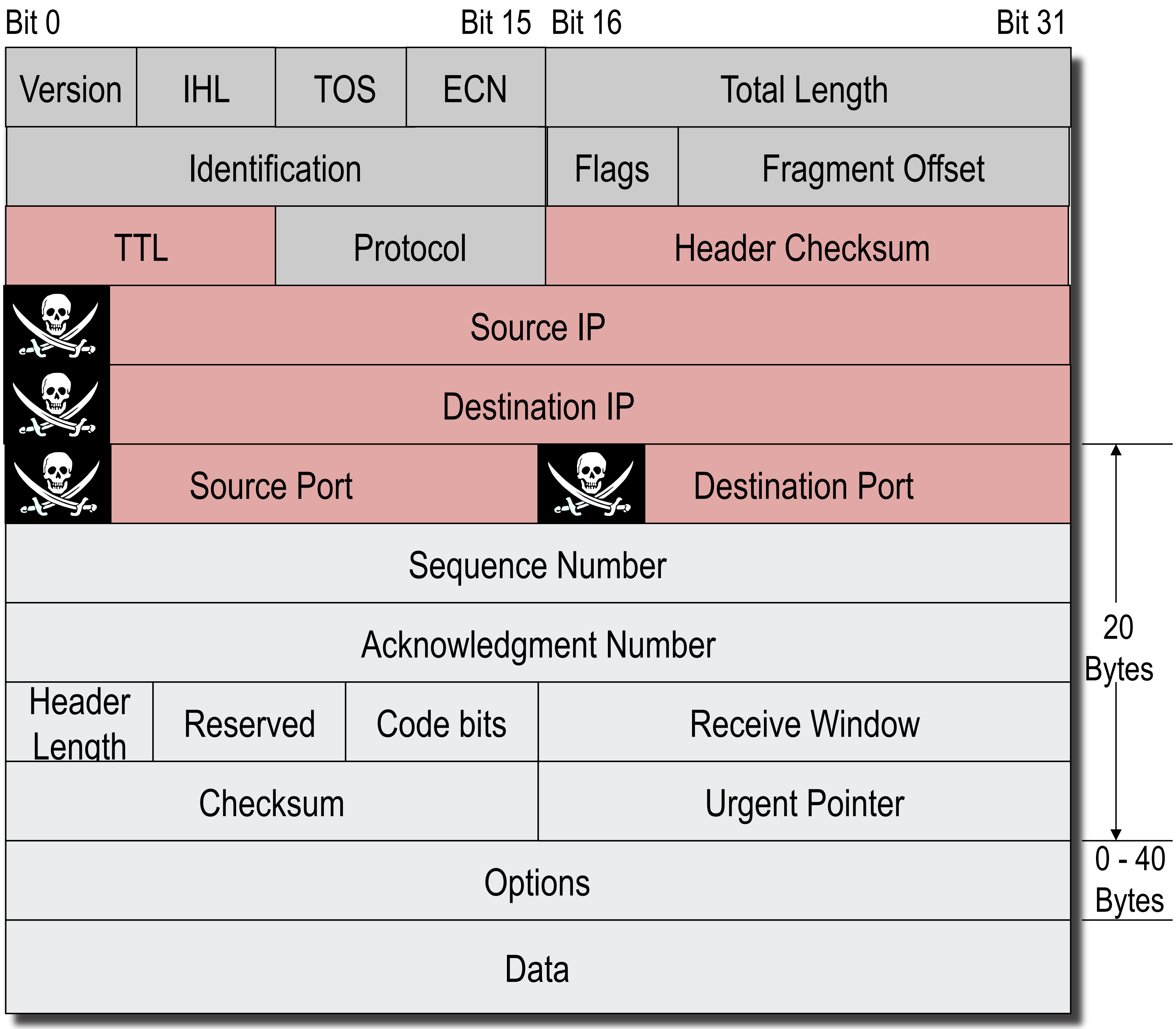
- Link layers (eg Ethernet) are local to a particular link
- Routers look at IP headers to decide how to route a packet.
- TCP provides reliability via retransmission, flow contr etc.
- Application using OS's TCP API to do its job.

Middleboxes

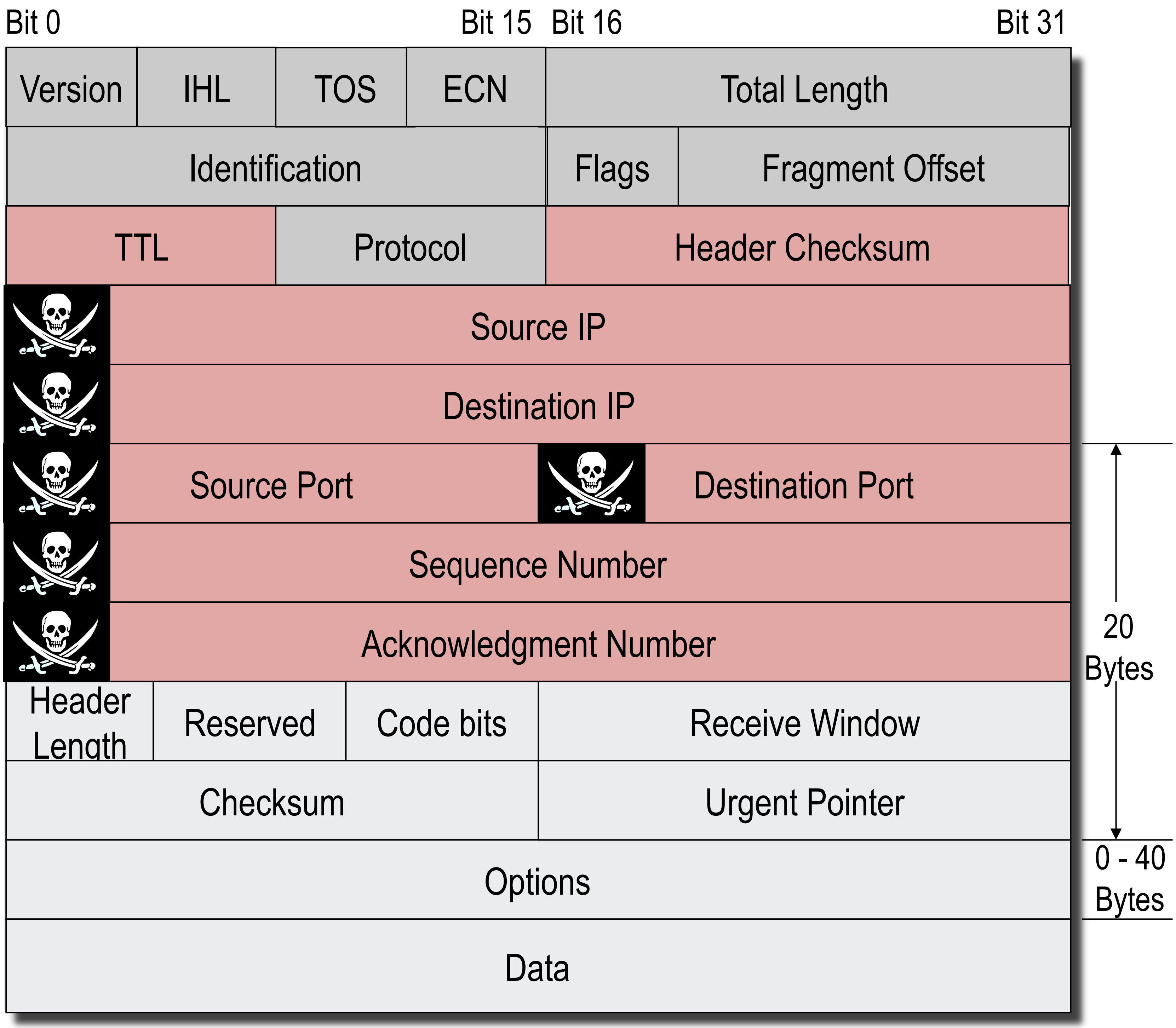

Bit 0
Bit 15
Bit 16
Bit 31
Version
IHL
TOS
ECN
Total Length
Identification
Flags
Fragment Offset
TTL
Protocol
Header Checksum
Source IP
Destination IP
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Header Length
Reserved
Code bits
Receive Window
Checksum
Urgent Pointer
Options
Data
20 Bytes
0 - 40 Bytes

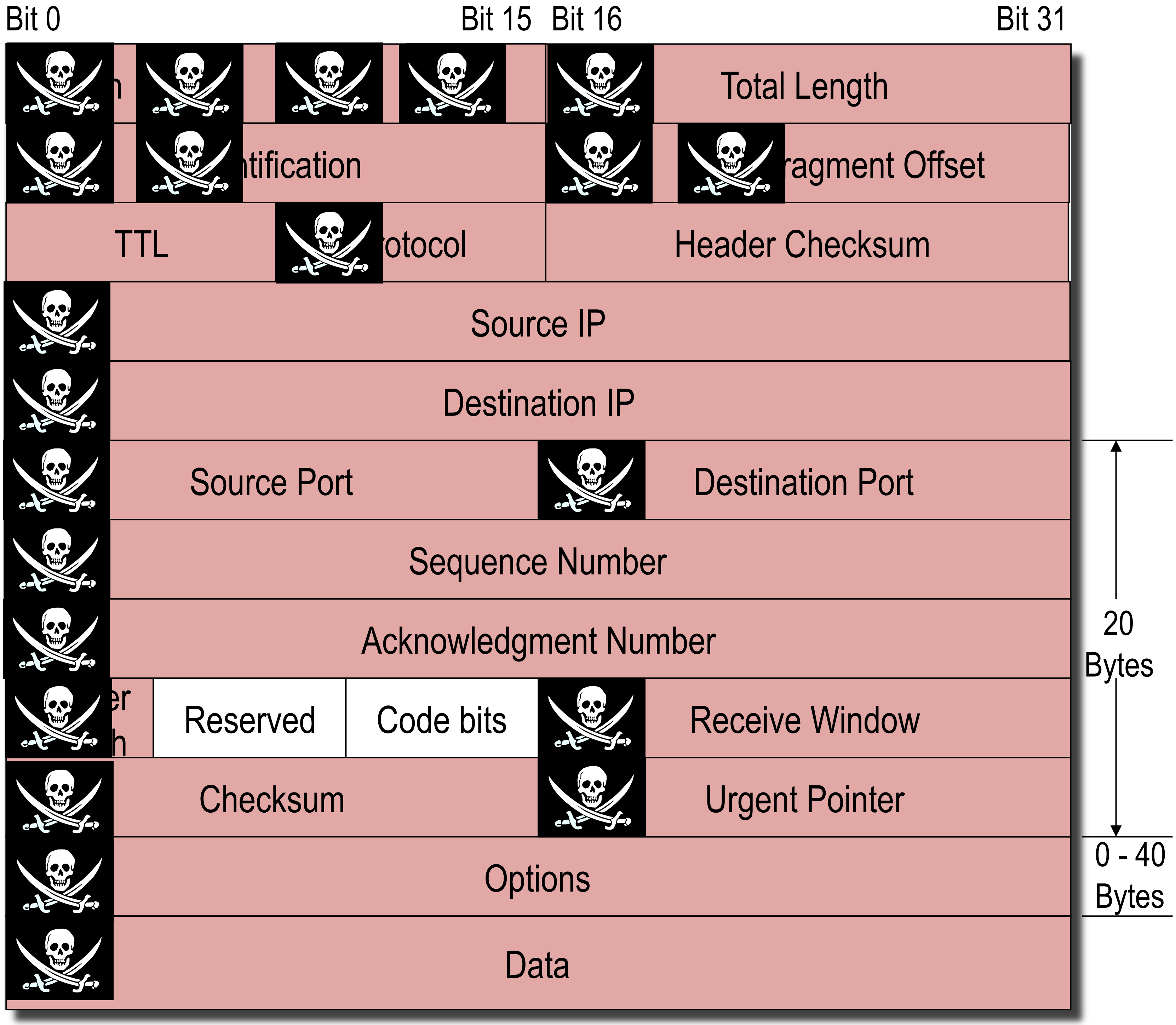


Bit 0
Bit 15
Bit 16
Bit 31
Version
IHL
TOS
ECN
Total Length
Identification
Flags
Fragment Offset
TTL
Protocol
Header Checksum
Source IP
Destination IP
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Header Length
Reserved
Code bits
Receive Window
Checksum
Urgent Pointer
Options
Data
20 Bytes
0 - 40 Bytes



Bit 0
Bit 15
Bit 16
Bit 31
Version
IHL
TOS
ECN
Total Length
Identification
Flags
Fragment Offset
TTL
Protocol
Header Checksum
Source IP
Destination IP
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Header Length
Reserved
Code bits
Receive Window
Checksum
Urgent Pointer
Options
Data
20 Bytes
0 - 40 Bytes


Bit 0
Bit 15
Bit 16
Bit 31
Version
IHL
TOS
ECN
Total Length
Identification
Flags
Fragment Offset
TTL
Protocol
Header Checksum
Source IP
Destination IP
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Header Length
Reserved
Code bits
Receive Window
Checksum
Urgent Pointer
Options
Data
20 Bytes
0 - 40 Bytes



Bit 0
Bit 15
Bit 16
Bit 31
Total Length
TTL
Header Checksum
Source IP
Destination IP
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Reserved
Code bits
Receive Window
Checksum
Urgent Pointer
Options
Data
20 Bytes
0 - 40 Bytes

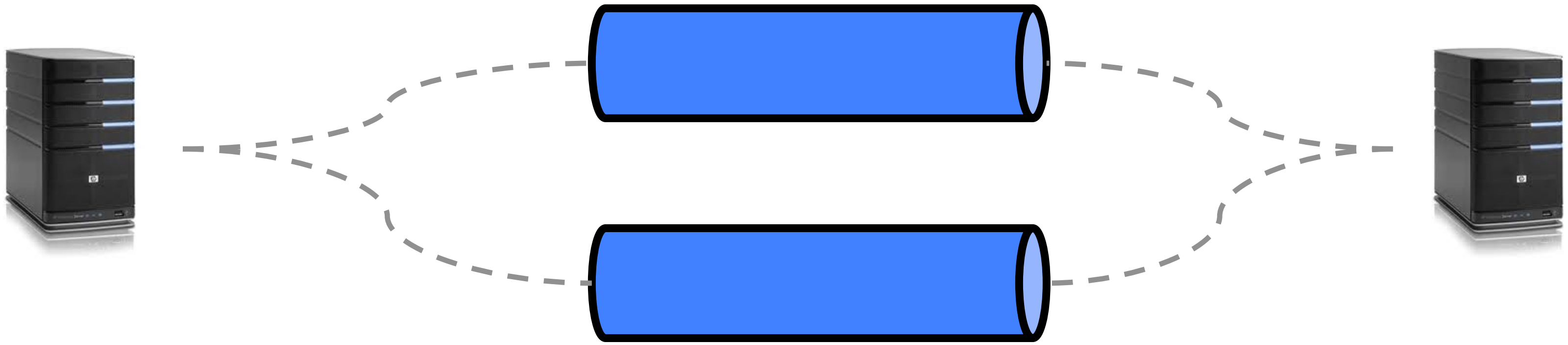
Deployable Multipath TCP

How hard can it be?

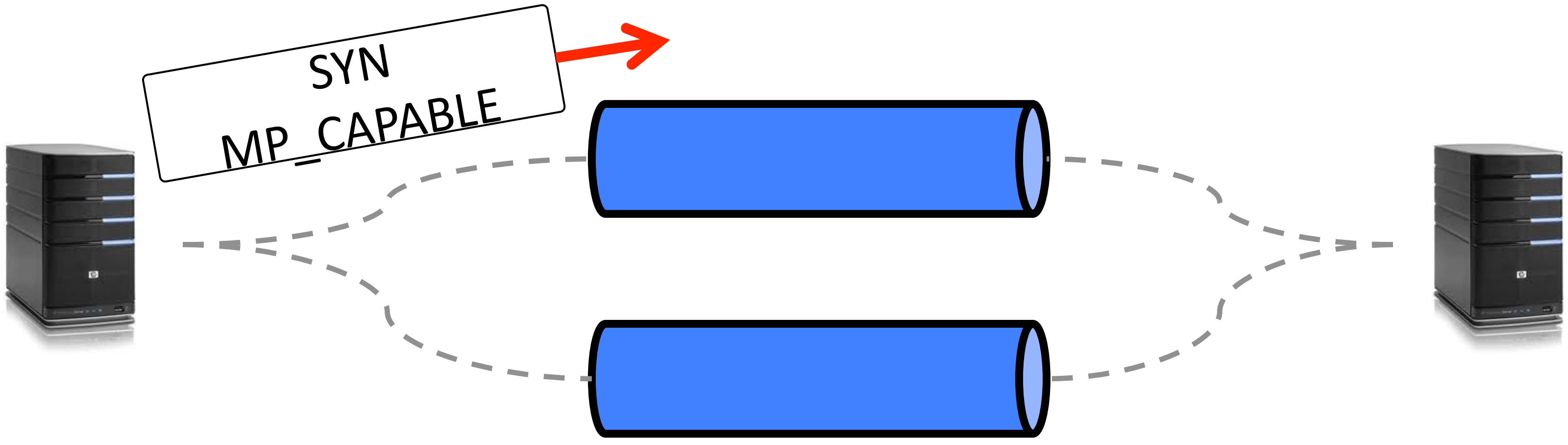
Designing

Implementing

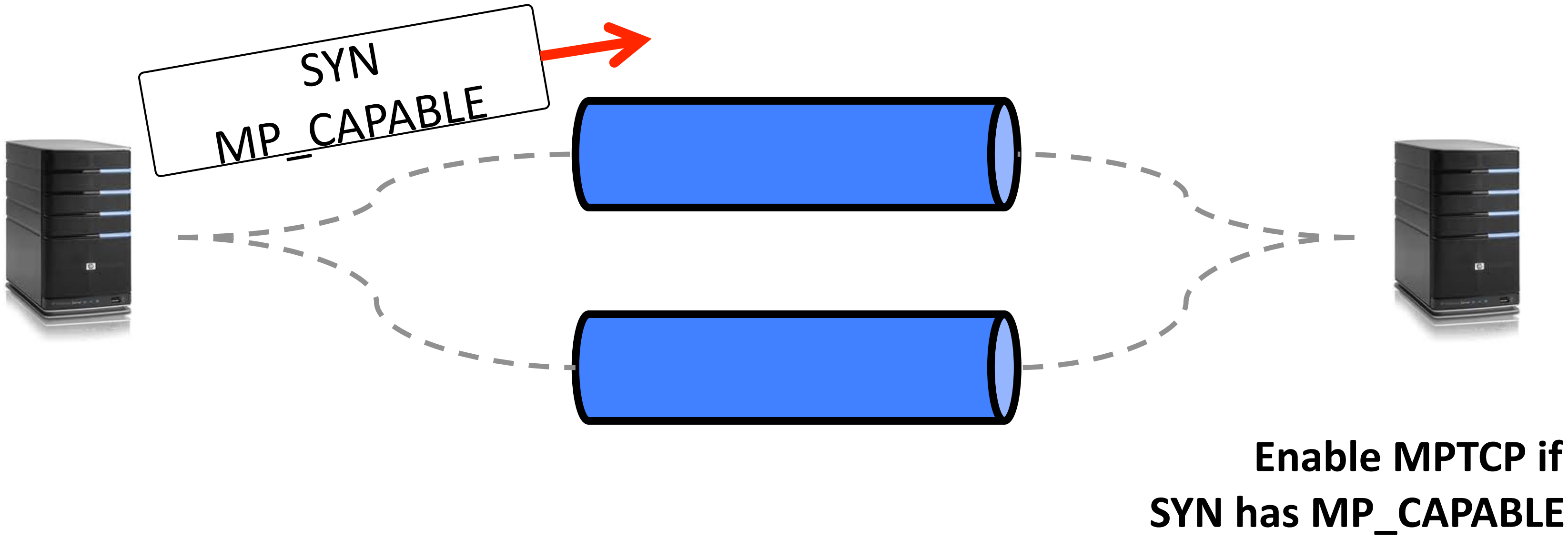
MPTCP Connection Management



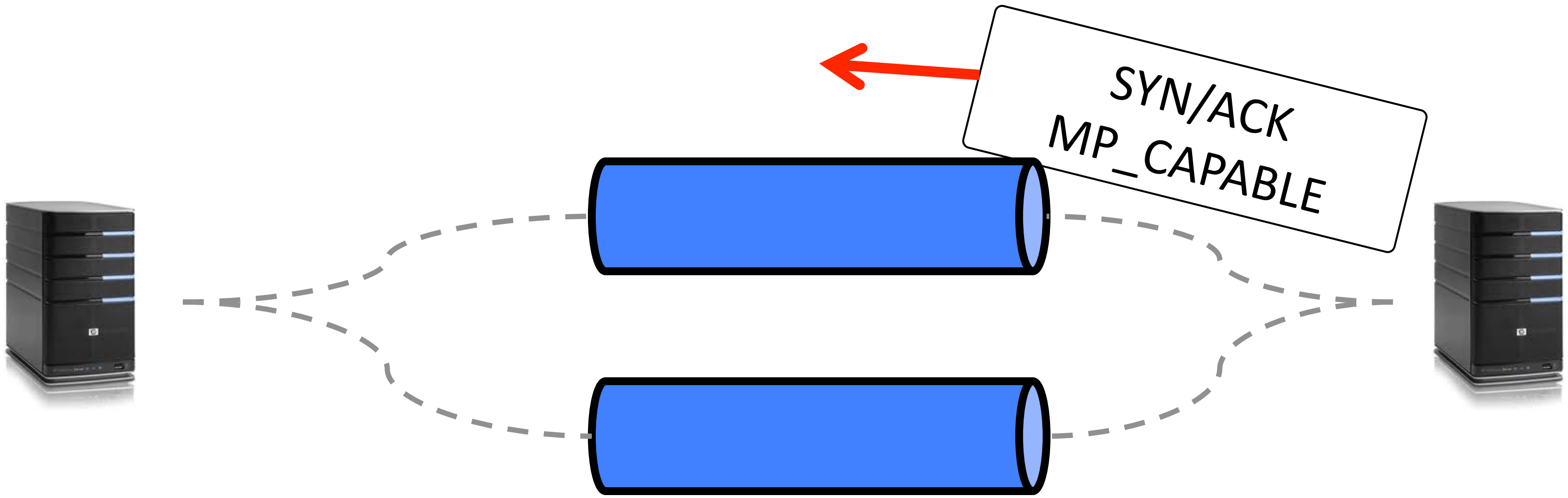
MPTCP Connection Management



MPTCP Connection Management



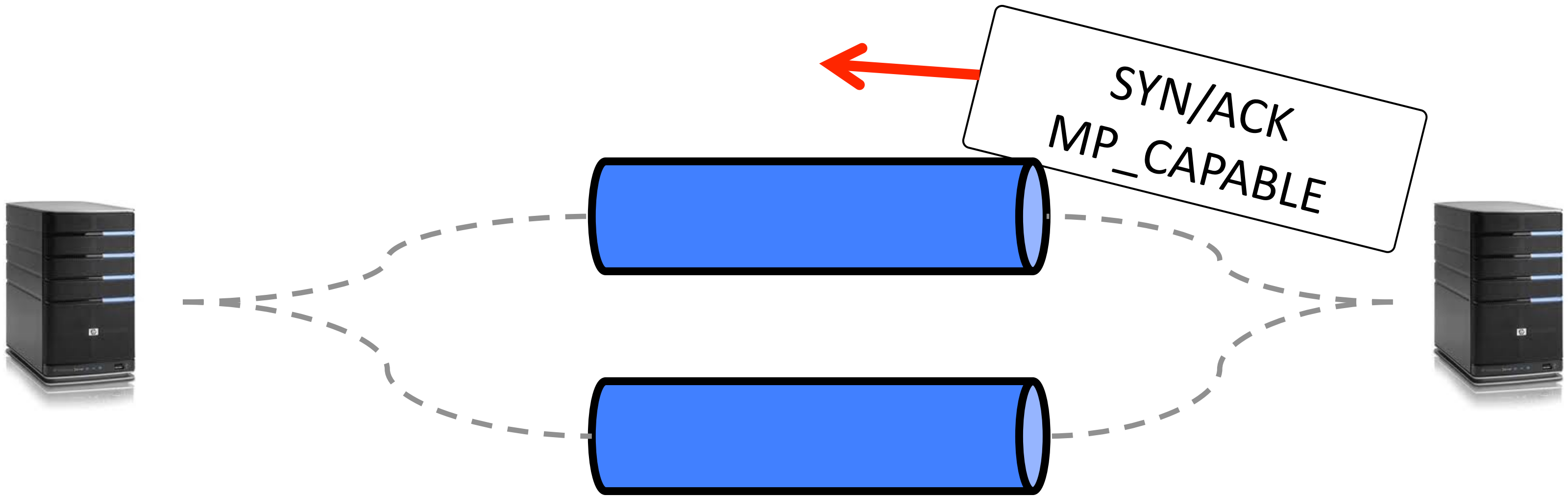
MPTCP Connection Management



Enable MPTCP if SYN has MP_CAPABLE

ENABLED

MPTCP Connection Management

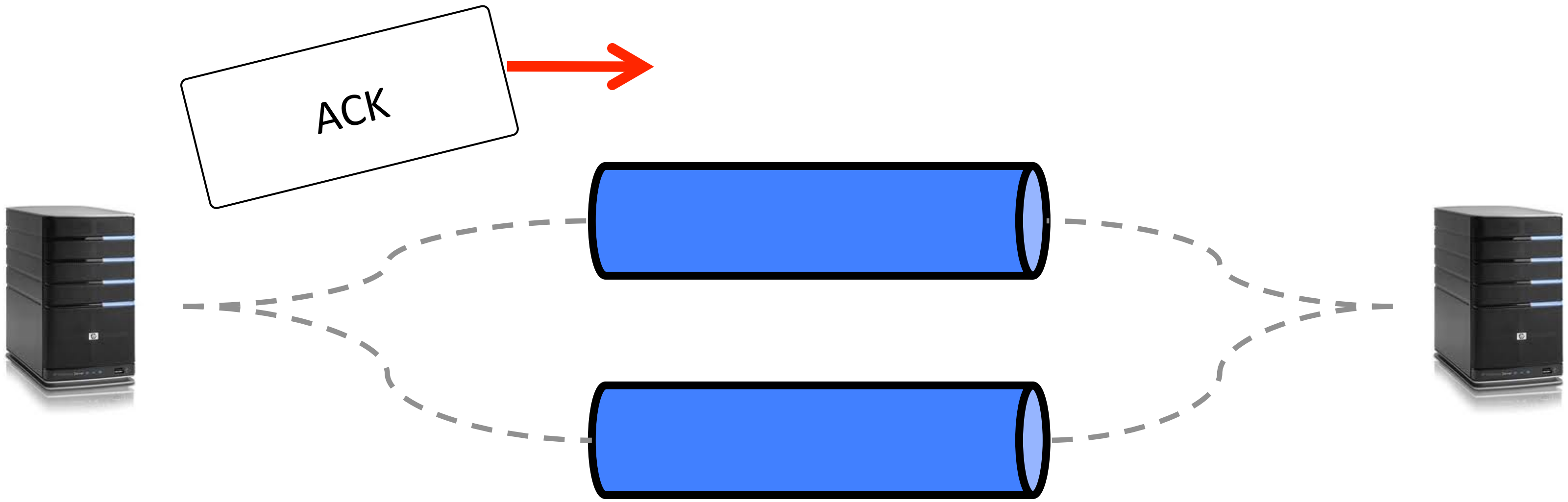


Enable MPTCP if
SYN/ACK has MP_CAPABLE

Enable MPTCP if
SYN has MP_CAPABLE

ENABLED

MPTCP Connection Management



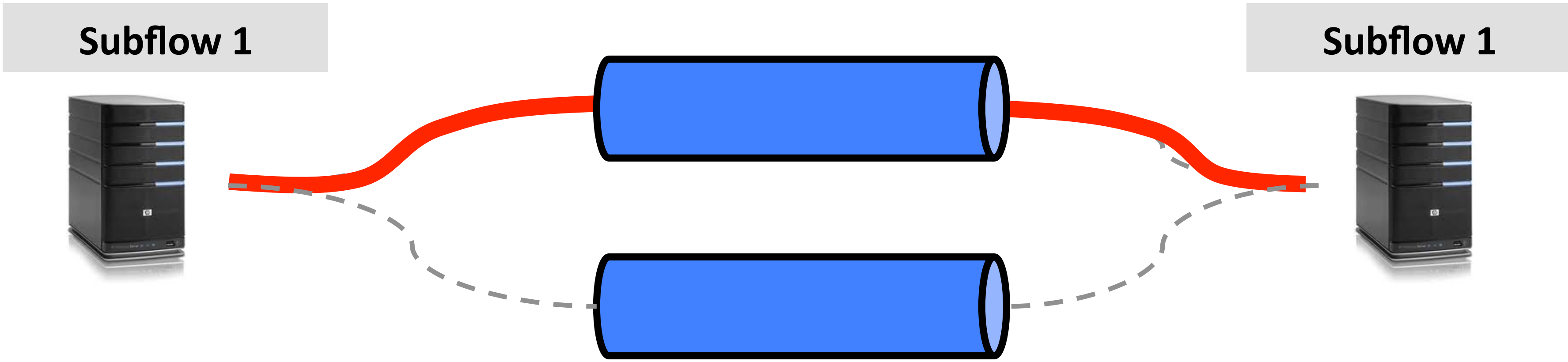
Enable MPTCP if
SYN/ACK has MP_CAPABLE

ENABLED

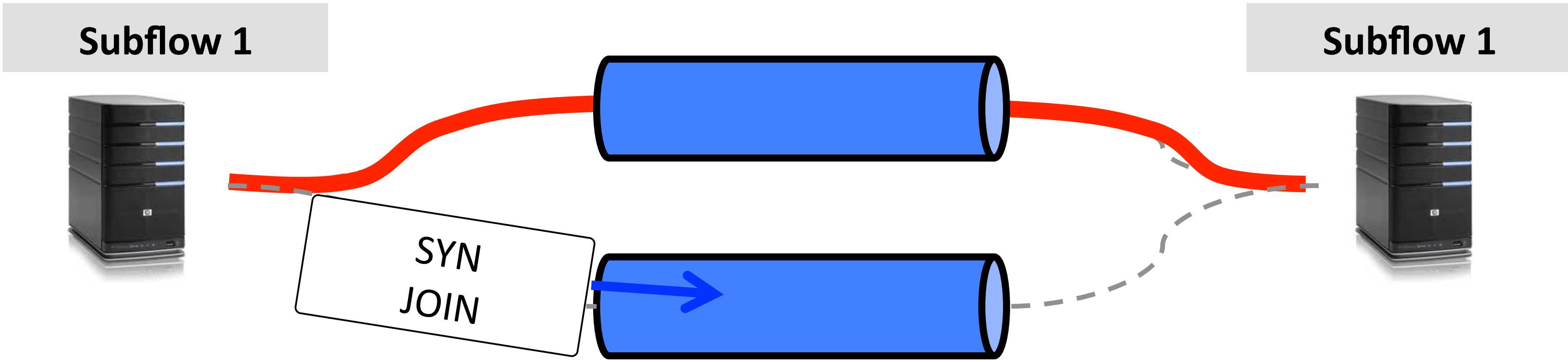
Enable MPTCP if
SYN has MP_CAPABLE

ENABLED

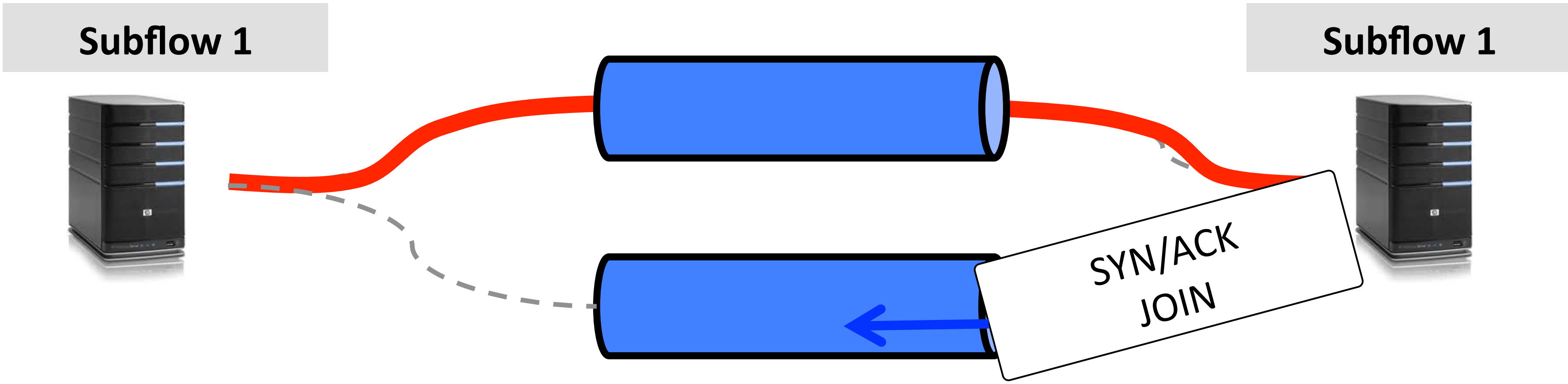
MPTCP Connection Management



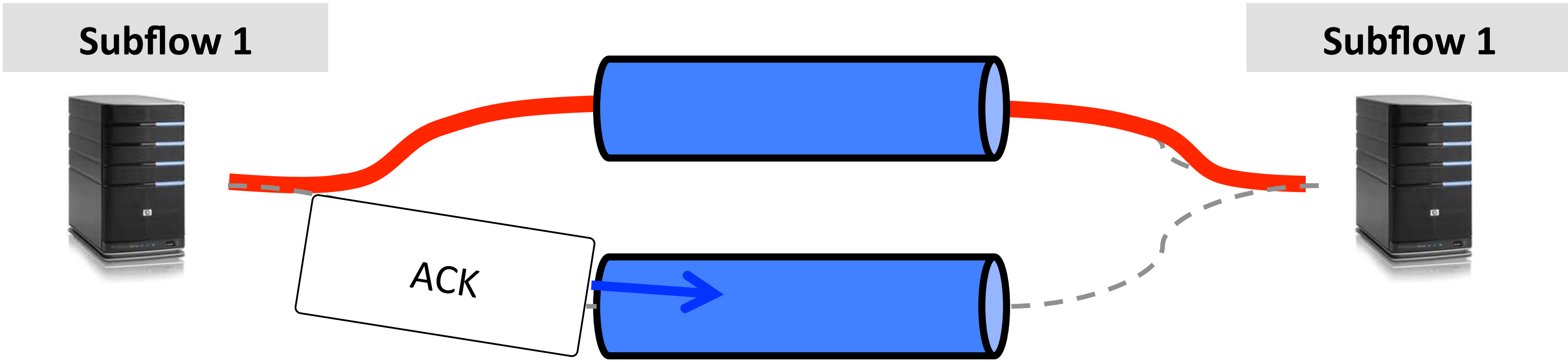
MPTCP Connection Management



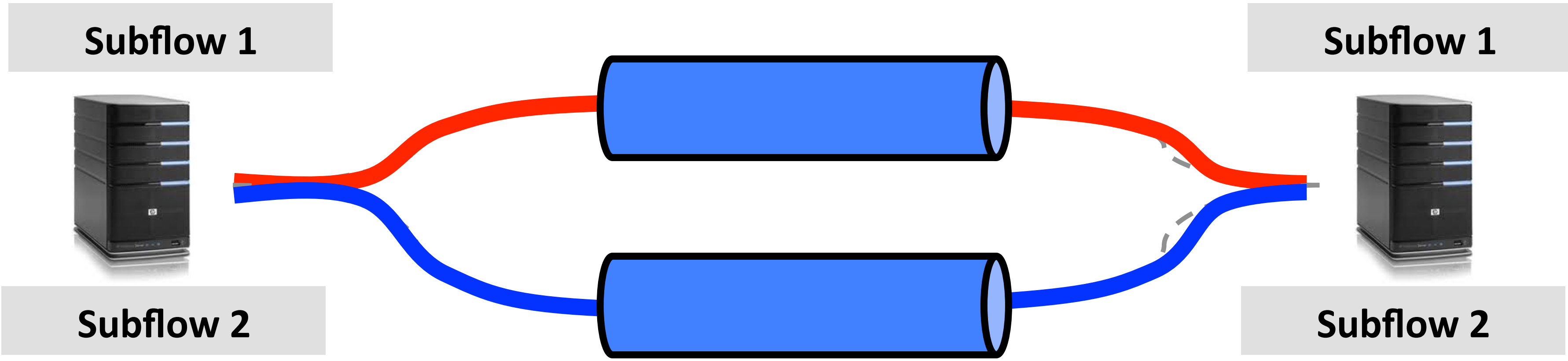
MPTCP Connection Management



MPTCP Connection Management



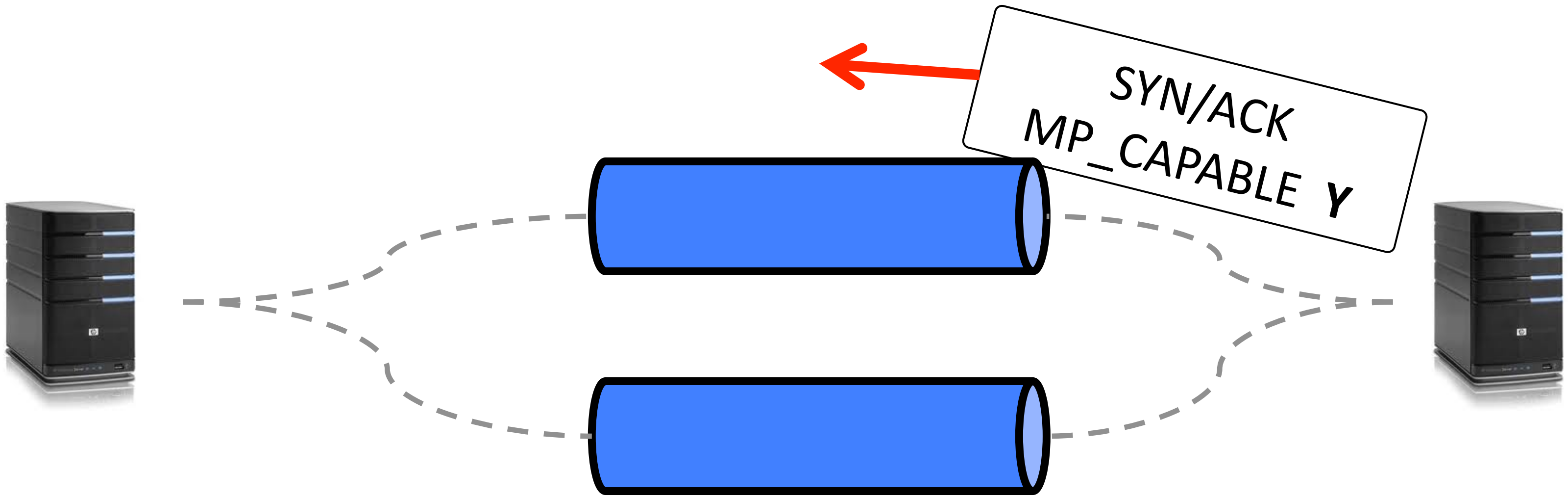
MPTCP Connection Management



That was easy!

Almost too easy…

MPTCP Connection Management

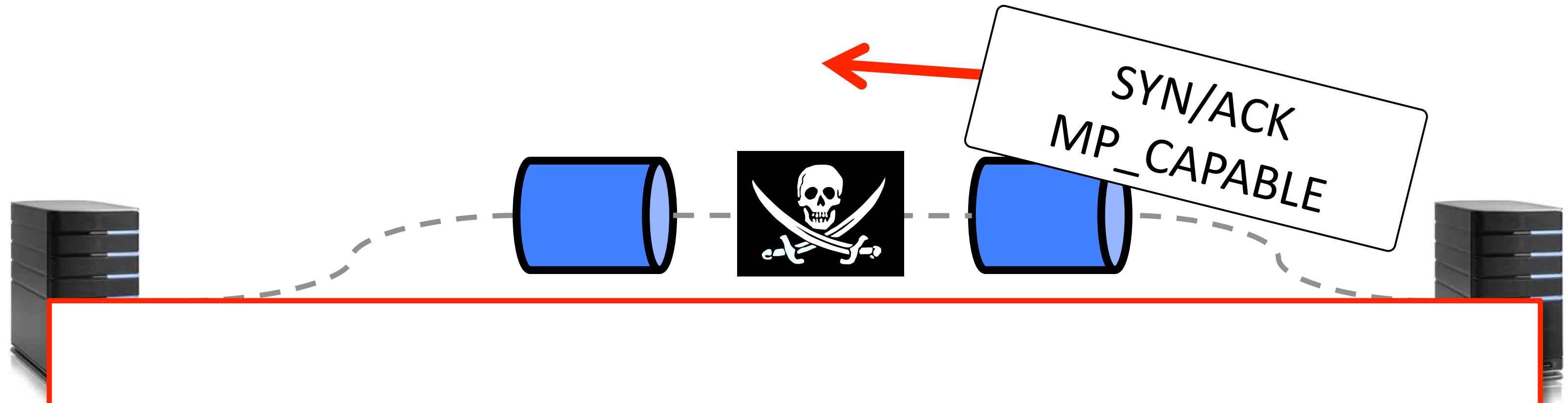


Enable MPTCP if SYN/ACK has MP_CAPABLE

Enable MPTCP if SYN has MP_CAPABLE

ENABLED

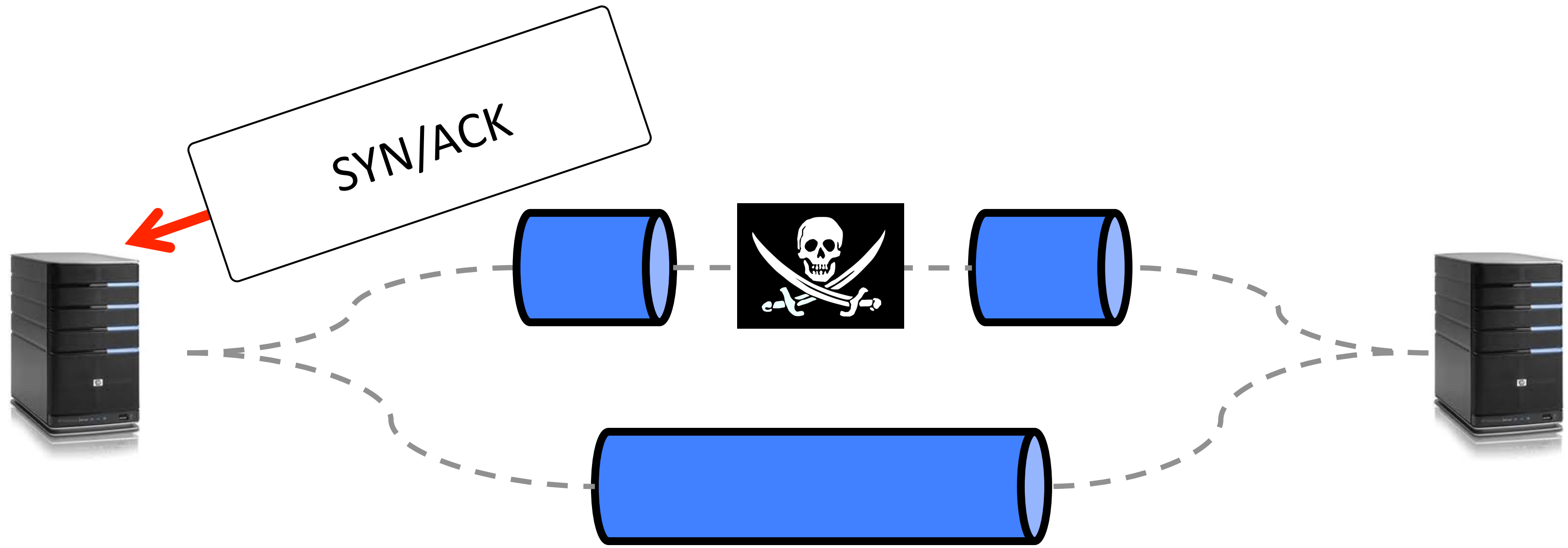
MPTCP Connection Management



6% of access networks remove unknown options
(14% on port 80)

[Honda et al. – Is It Still Possible to Extend TCP? – IMC 2011]

MPTCP Connection Management



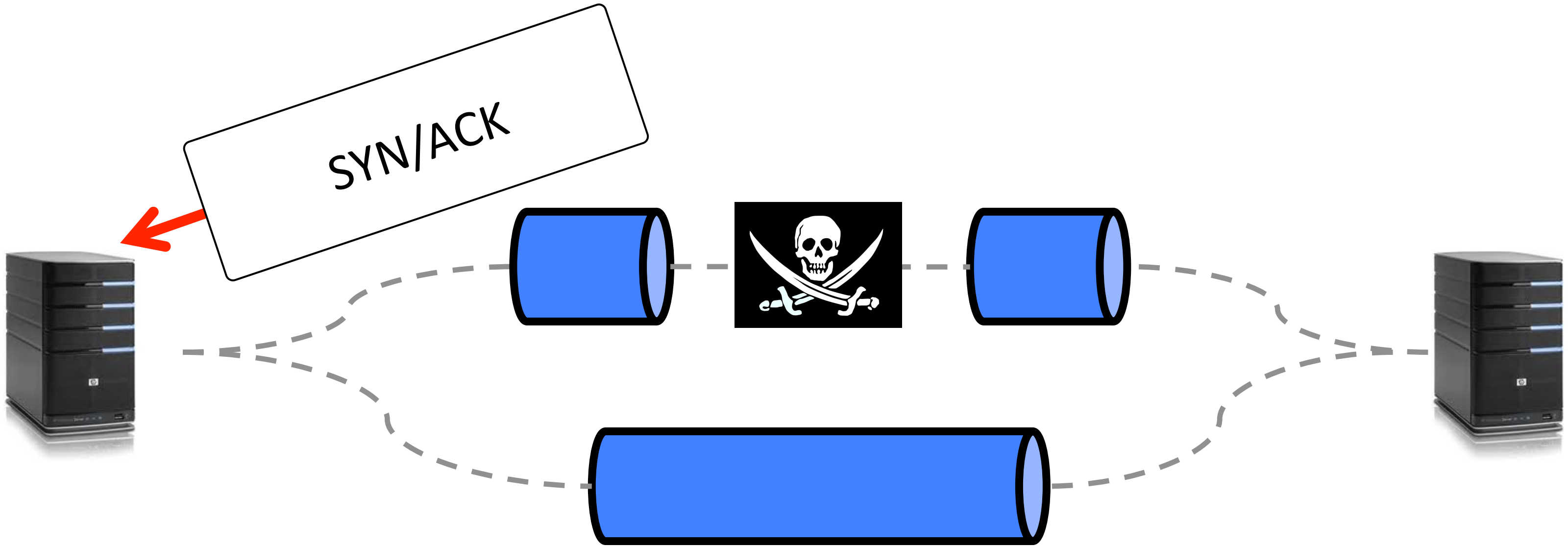
Enable MPTCP if
SYN/ACK has MP_CAPABLE

DISABLED

Enable MPTCP if
SYN has MP_CAPABLE

ENABLED

MPTCP Connection Management

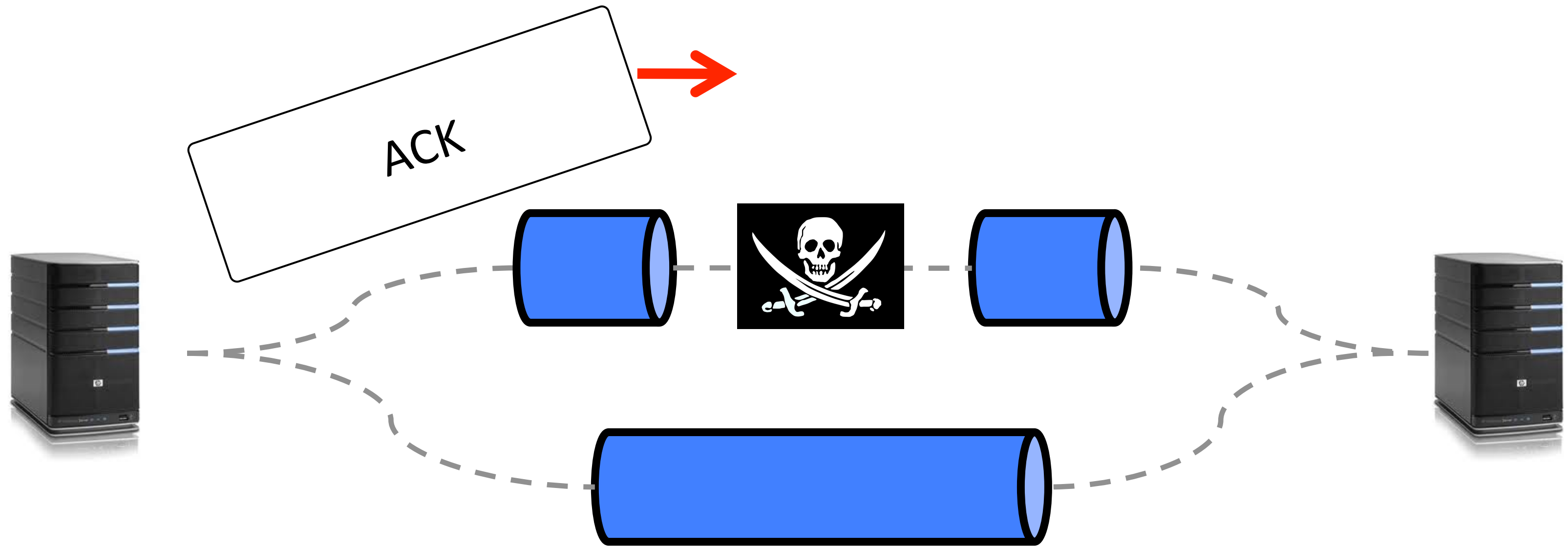


Enable MPTCP if
SYN/ACK has MP_CAPABLE

DISABLED

Enable MPTCP if
SYN has MP_CAPABLE
and ACK has DATA_ACK

MPTCP Connection Management

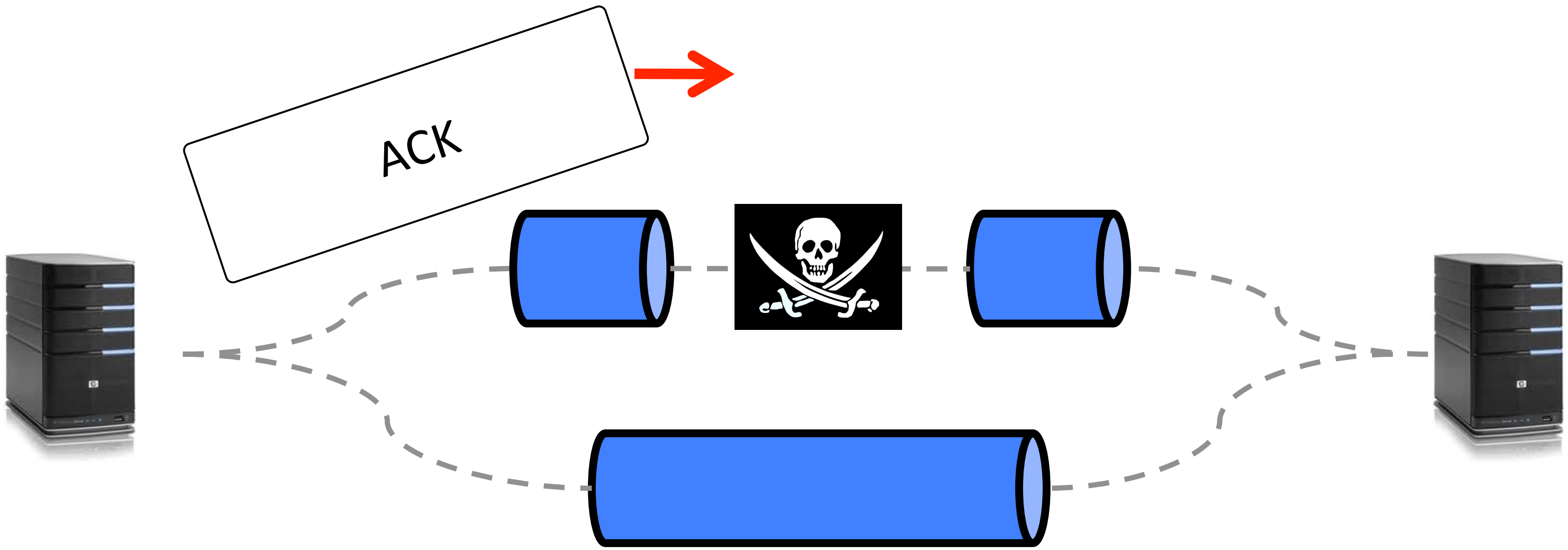


Enable MPTCP if
SYN/ACK has MP_CAPABLE

DISABLED

Enable MPTCP if
SYN has MP_CAPABLE
and ACK has DATA_ACK

MPTCP Connection Management



Enable MPTCP if
SYN/ACK has MP_CAPABLE

DISABLED

Enable MPTCP if
SYN has MP_CAPABLE
and ACK has DATA_ACK

DISABLED

To achieve GOAL 3:
When MPTCP operation is not possible, fallback to TCP.

Lesson

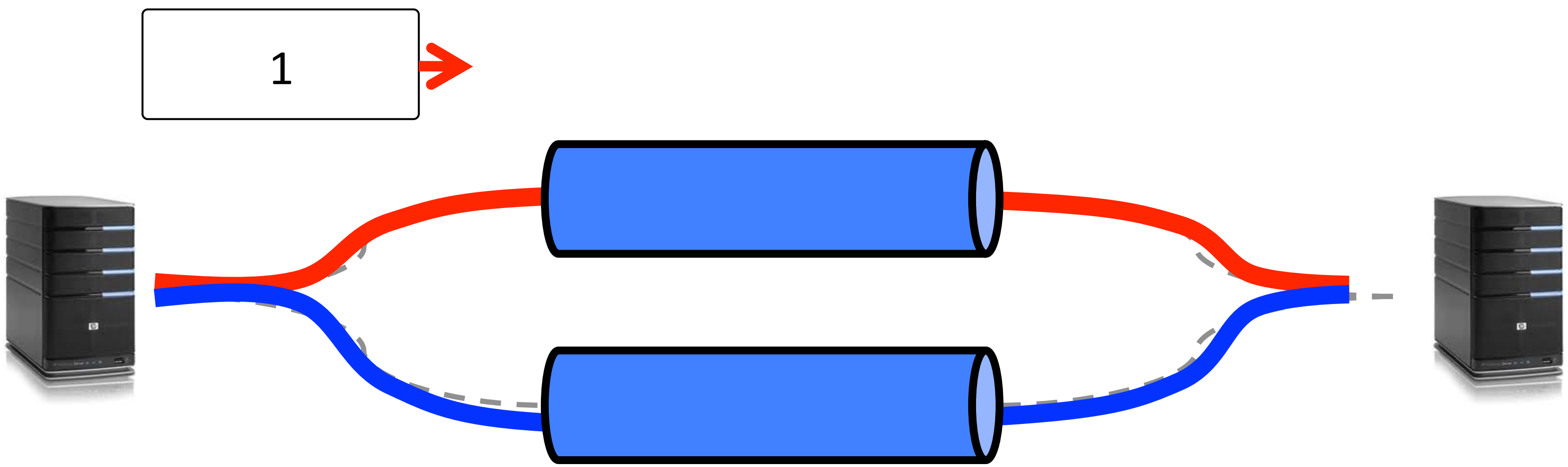
Negotiation used to be between two endpoints

In today's Internet, negotiation is:
between two endpoints
and an unknown number of intermediaries

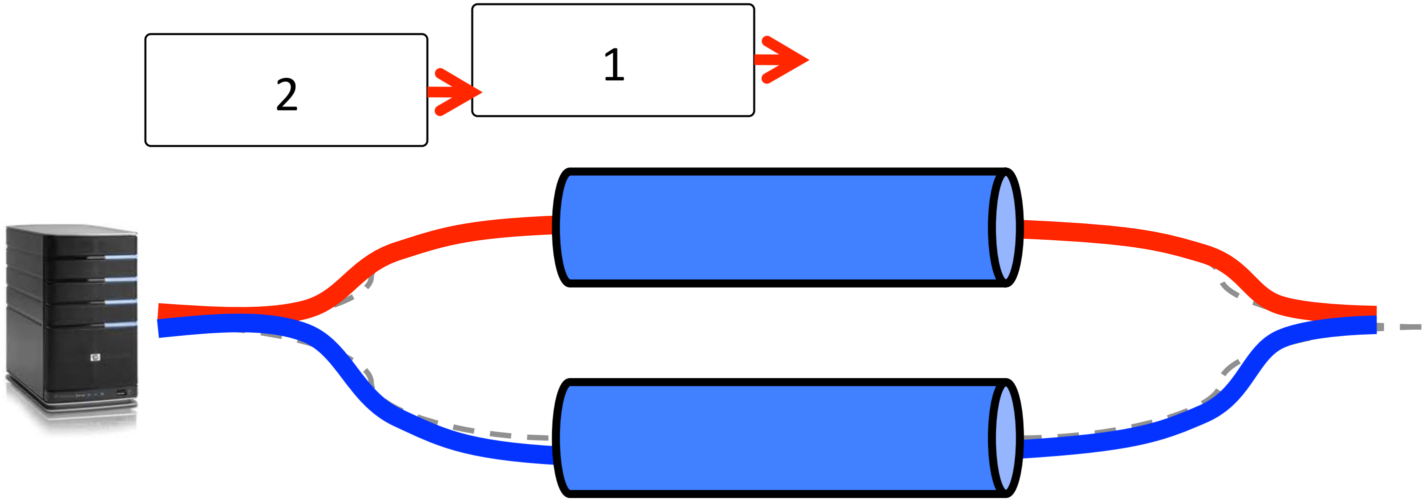
New protocol negotiation has to take this into account or it will fail

Sending Data

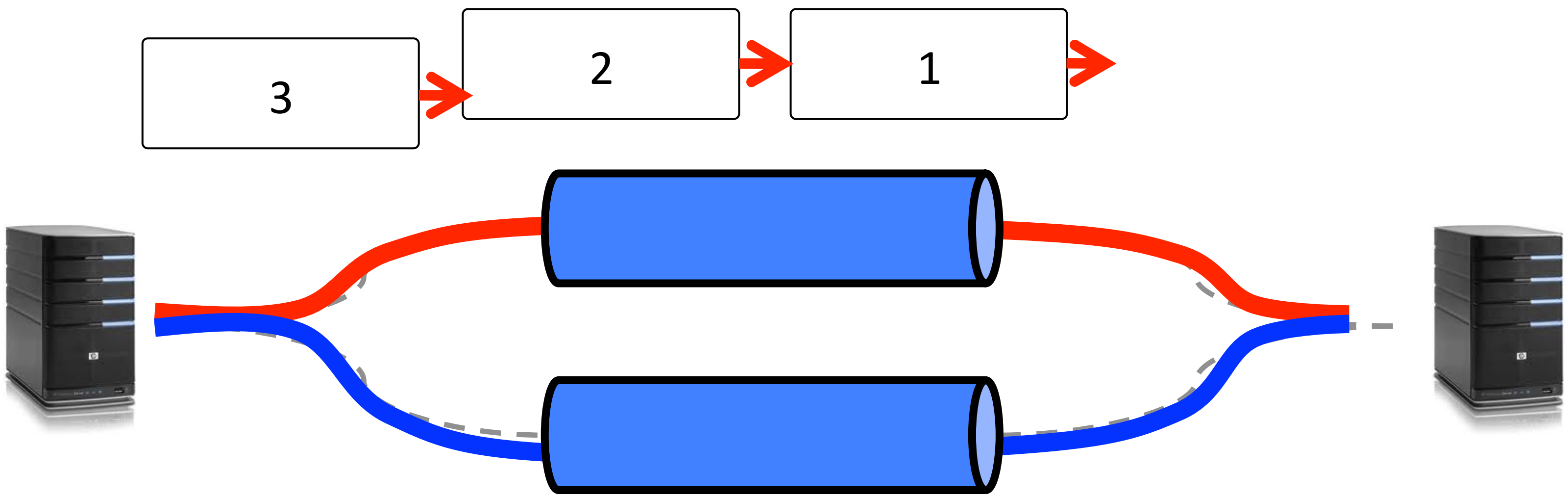
TCP Operation



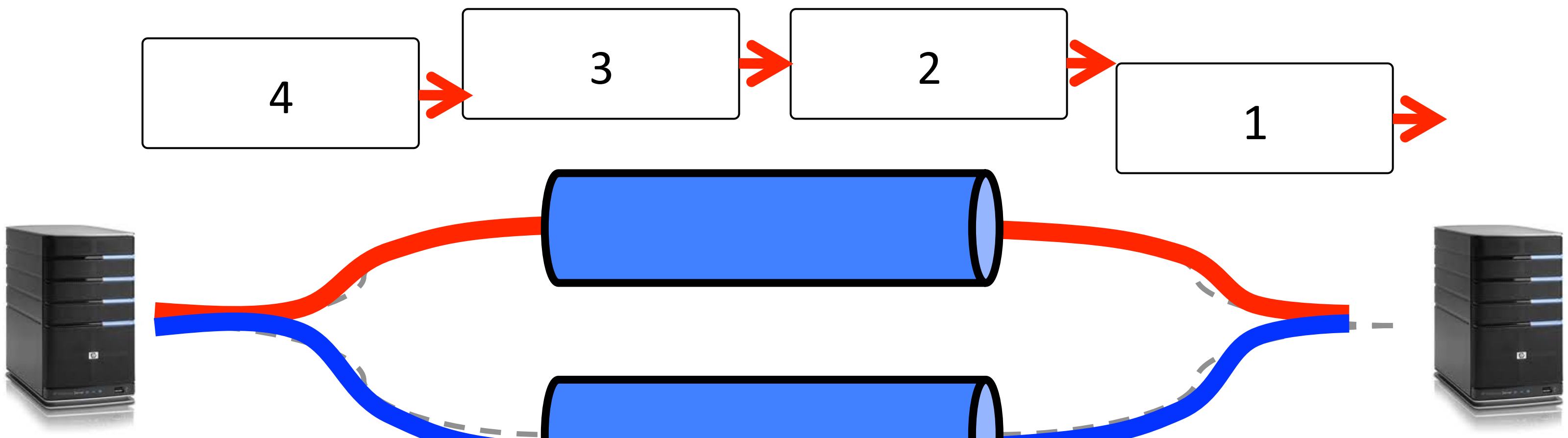
TCP Operation



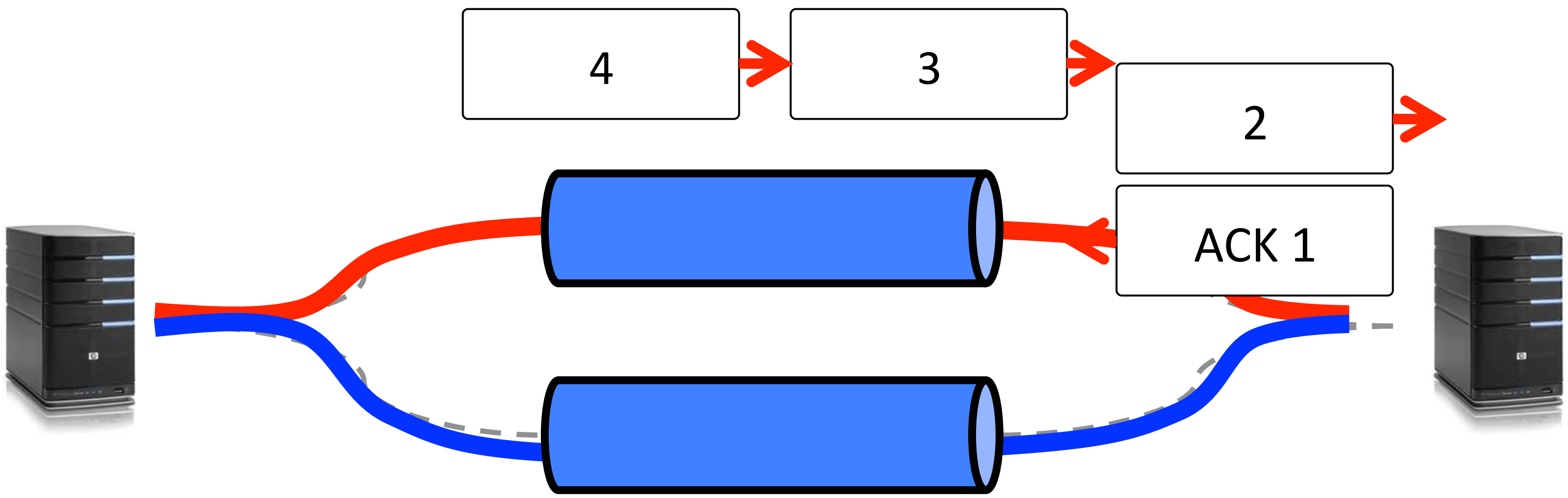
TCP Operation



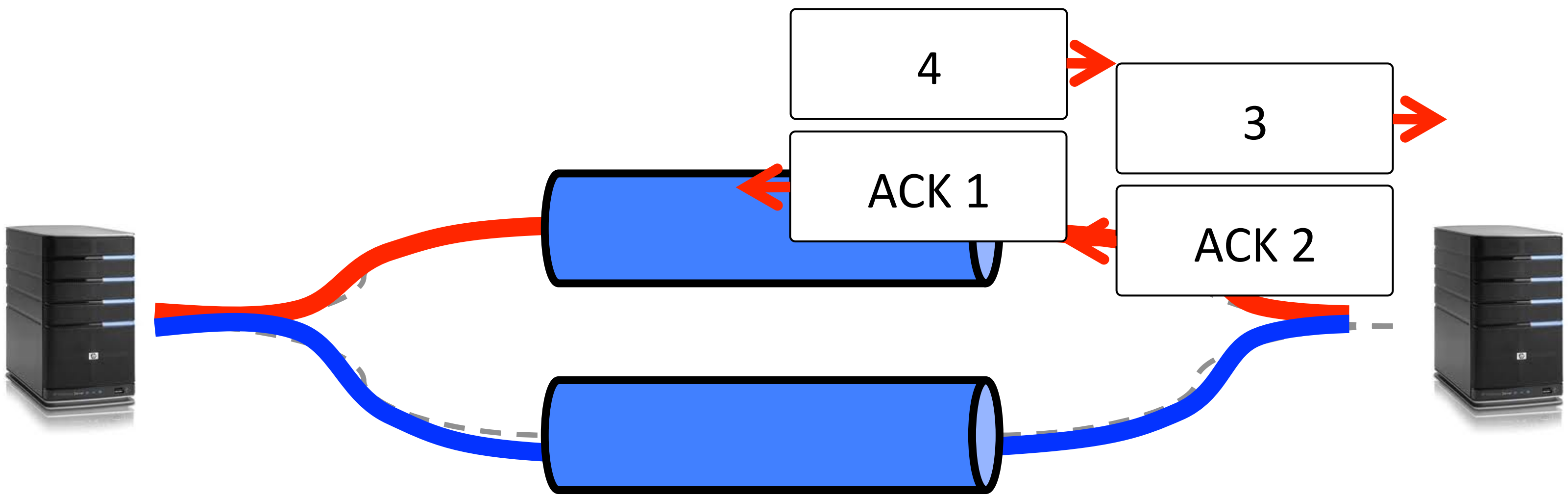
TCP Operation



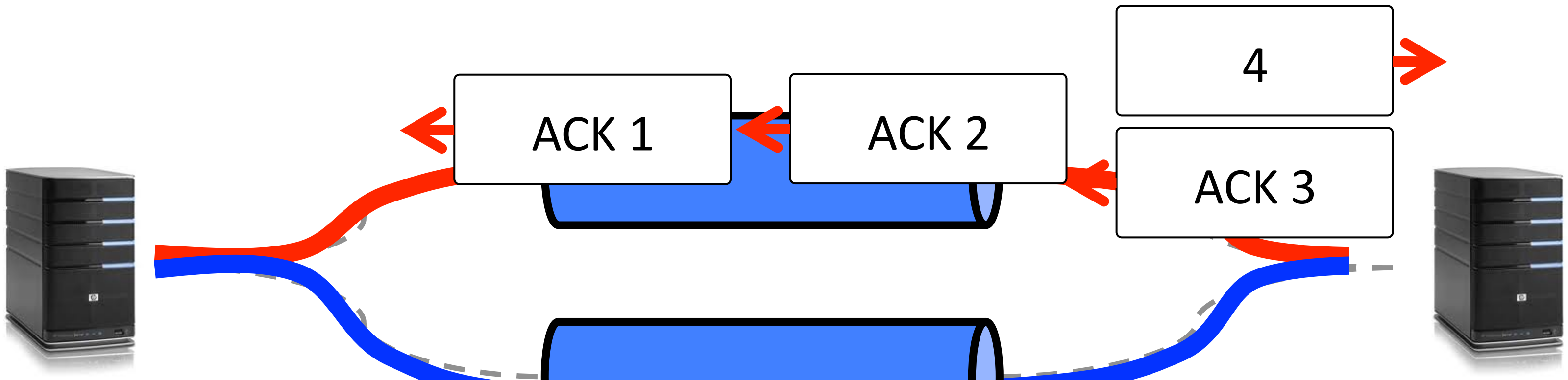
TCP Operation



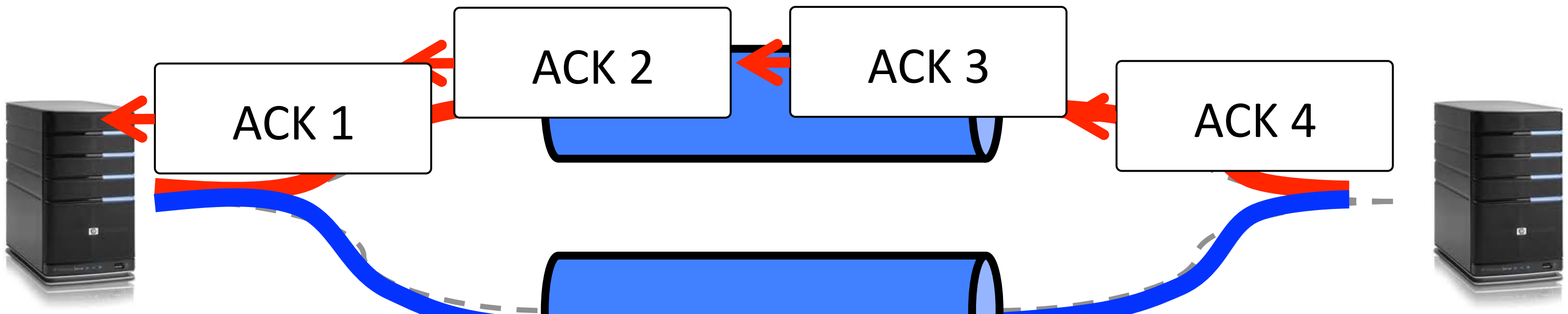
TCP Operation



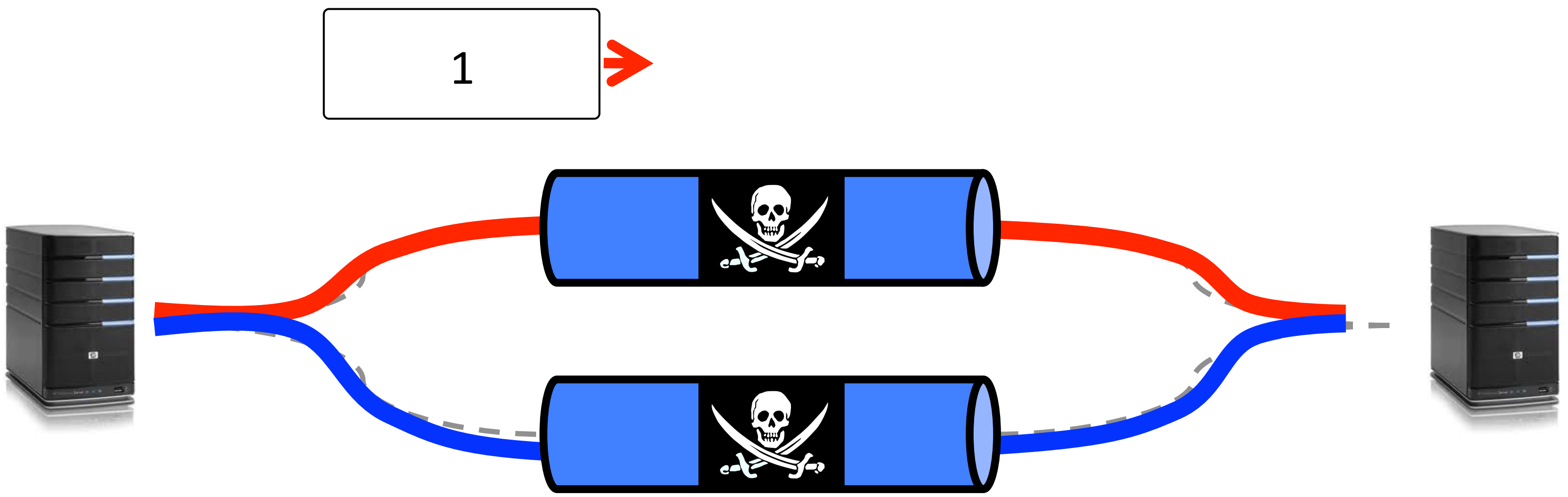
TCP Operation



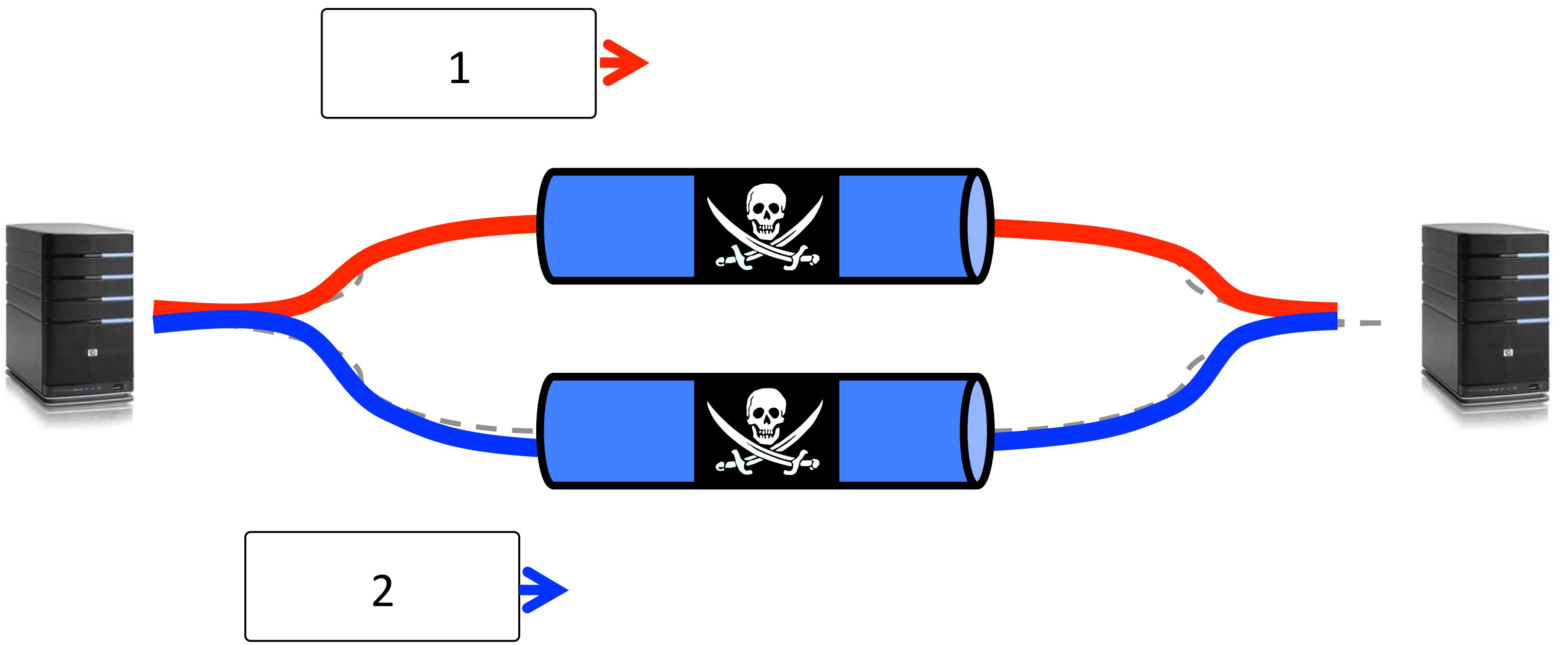
TCP Operation



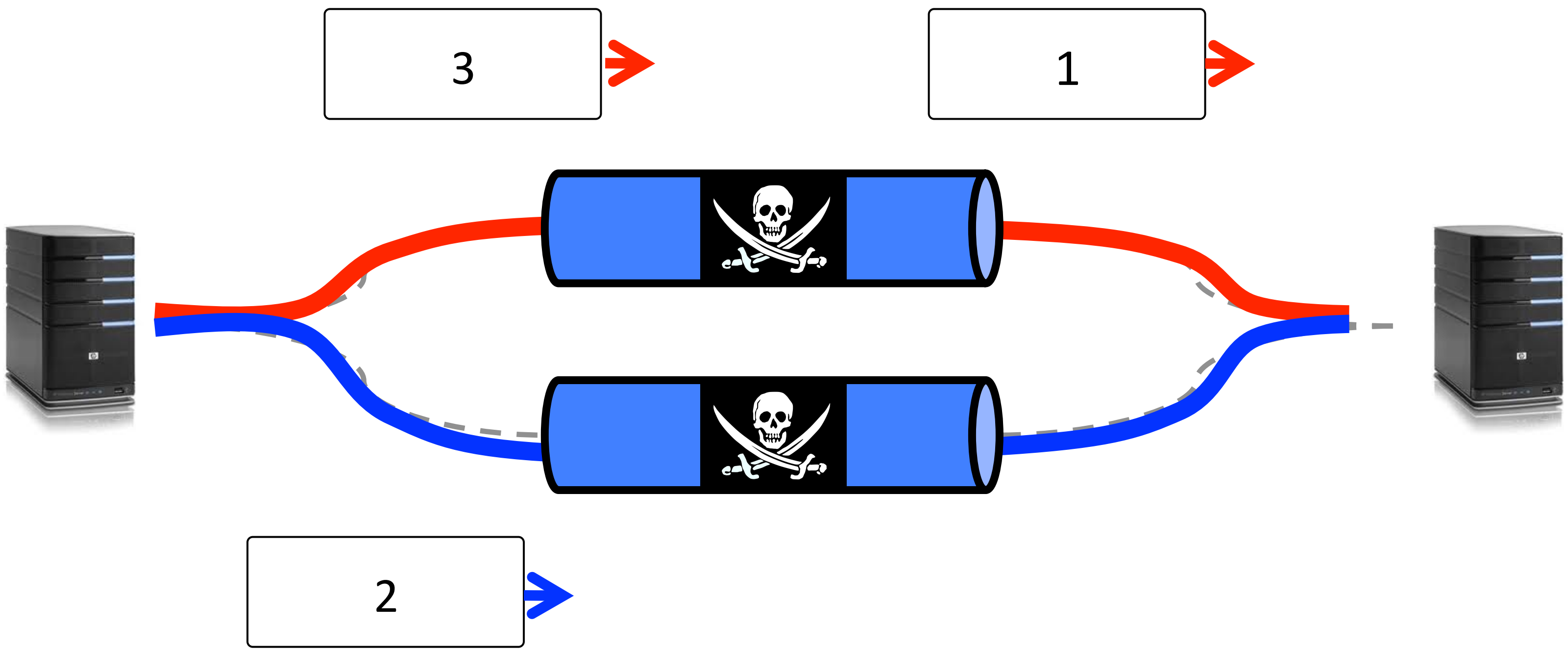
Strawman Design



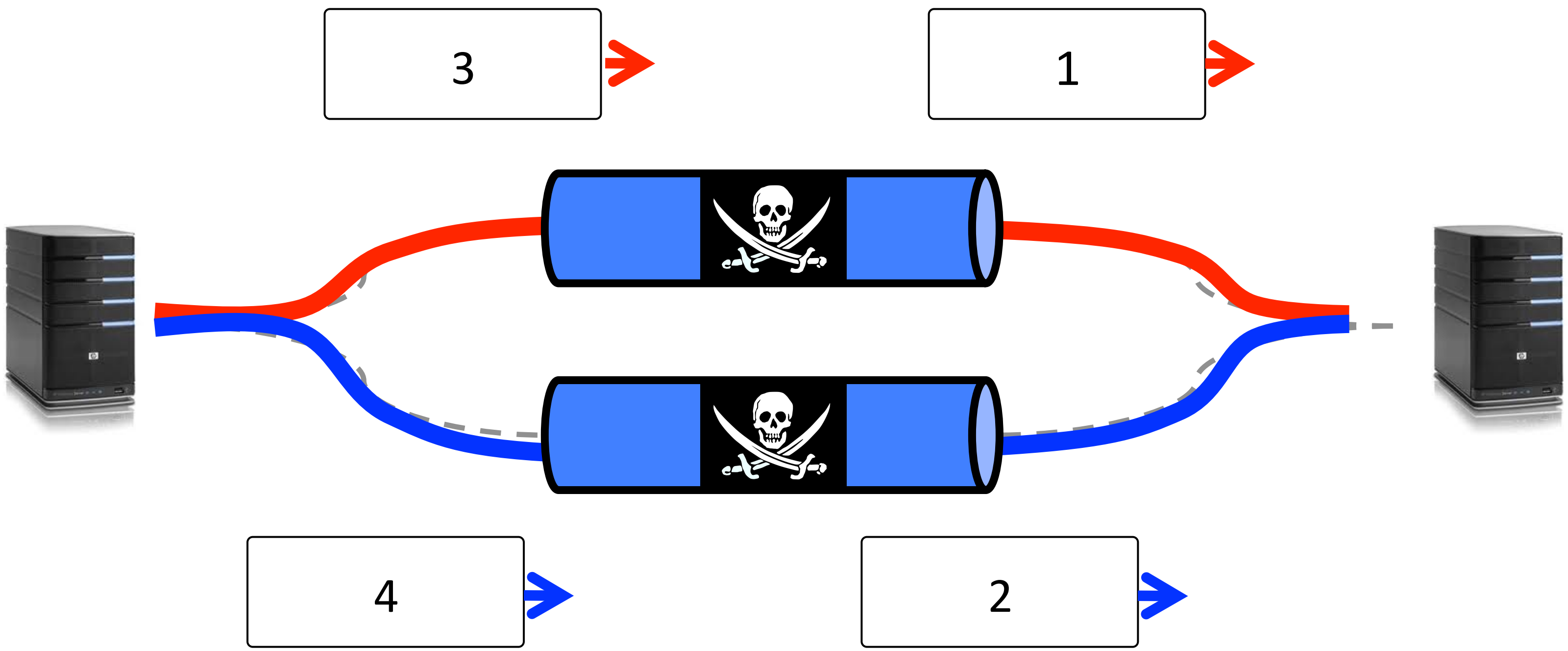
Strawman Design



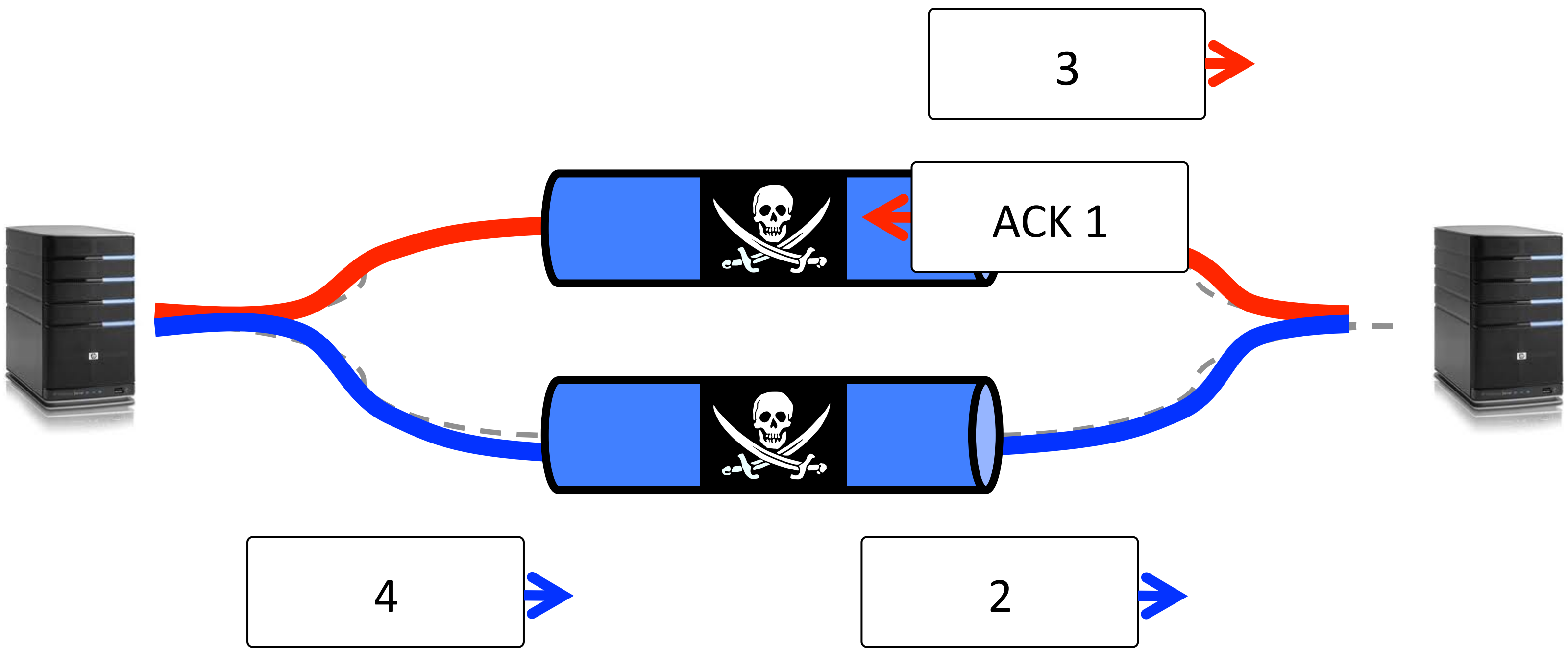
Strawman Design



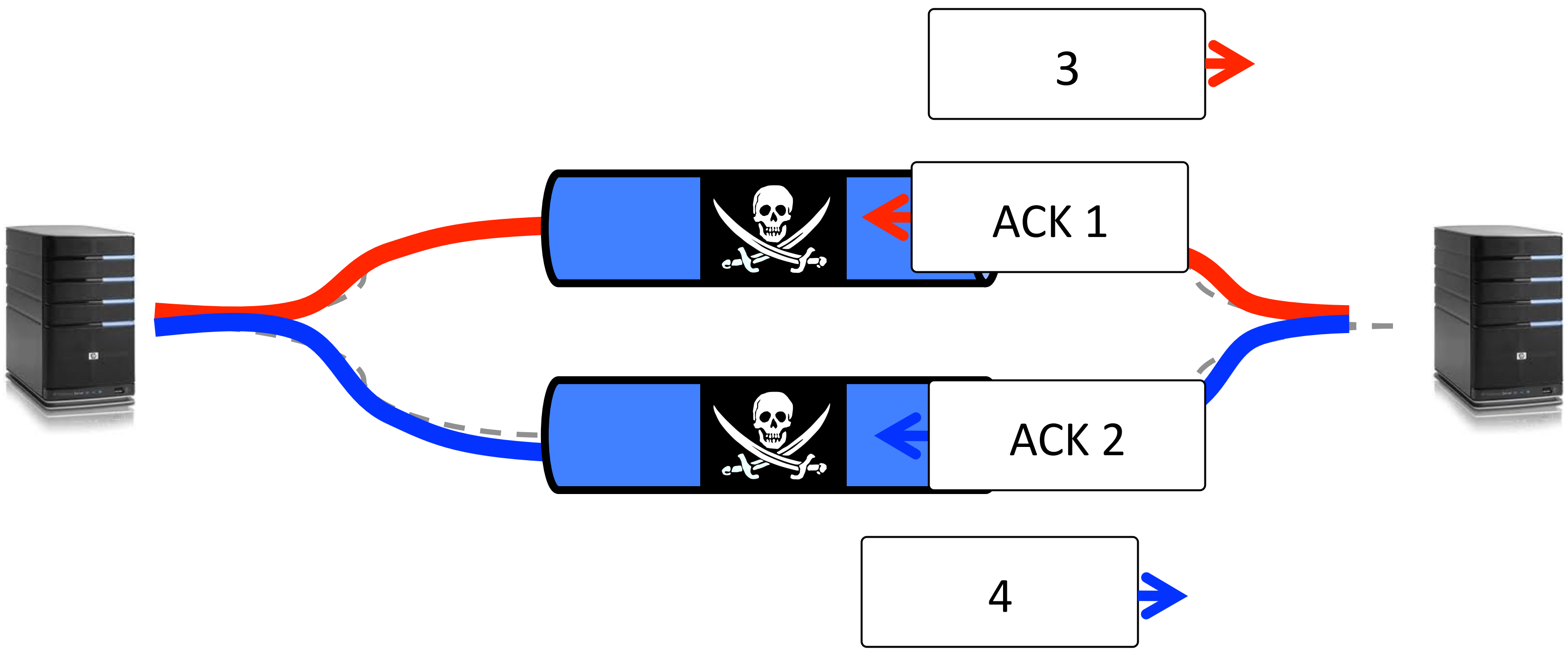
Strawman Design



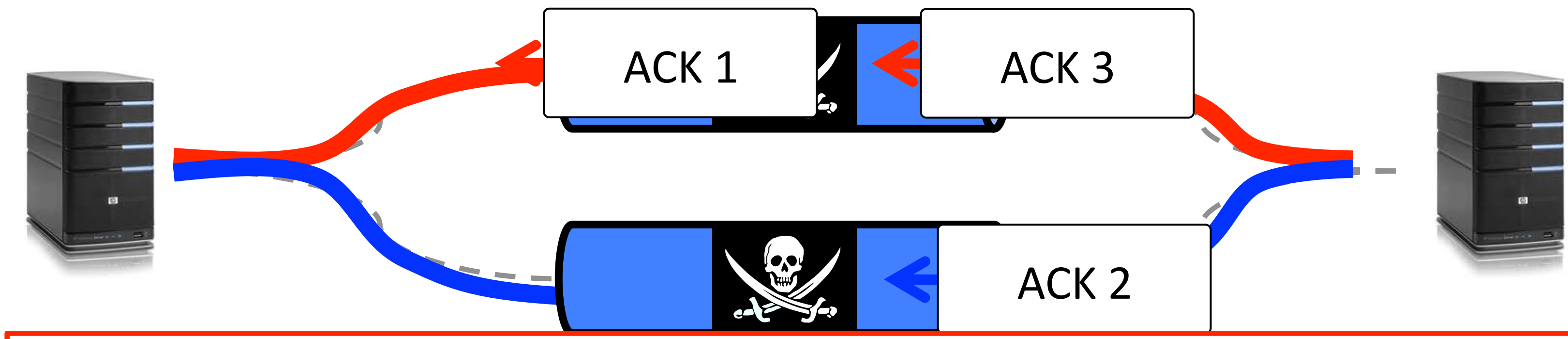
Strawman Design



Strawman Design

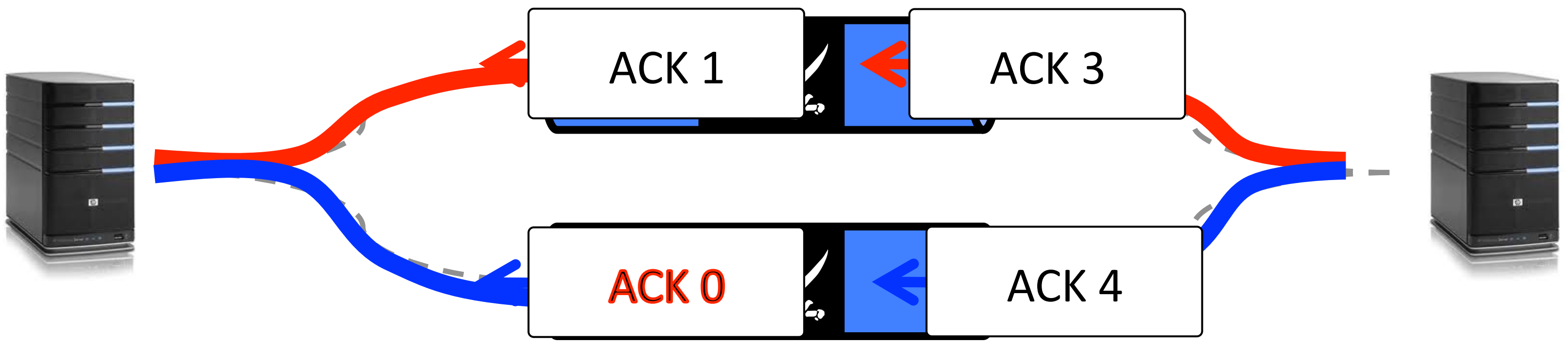


Strawman Design



A third of access networks will "correct" or drop ACKs of unseen data

Strawman Design

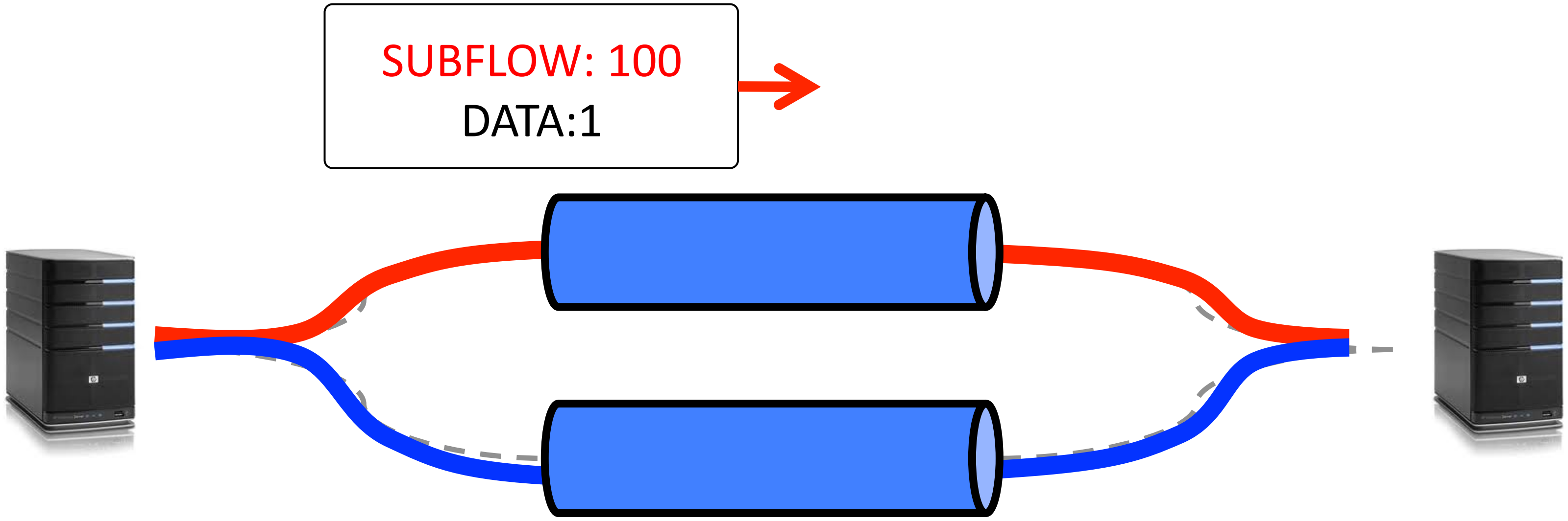


Strawman Design

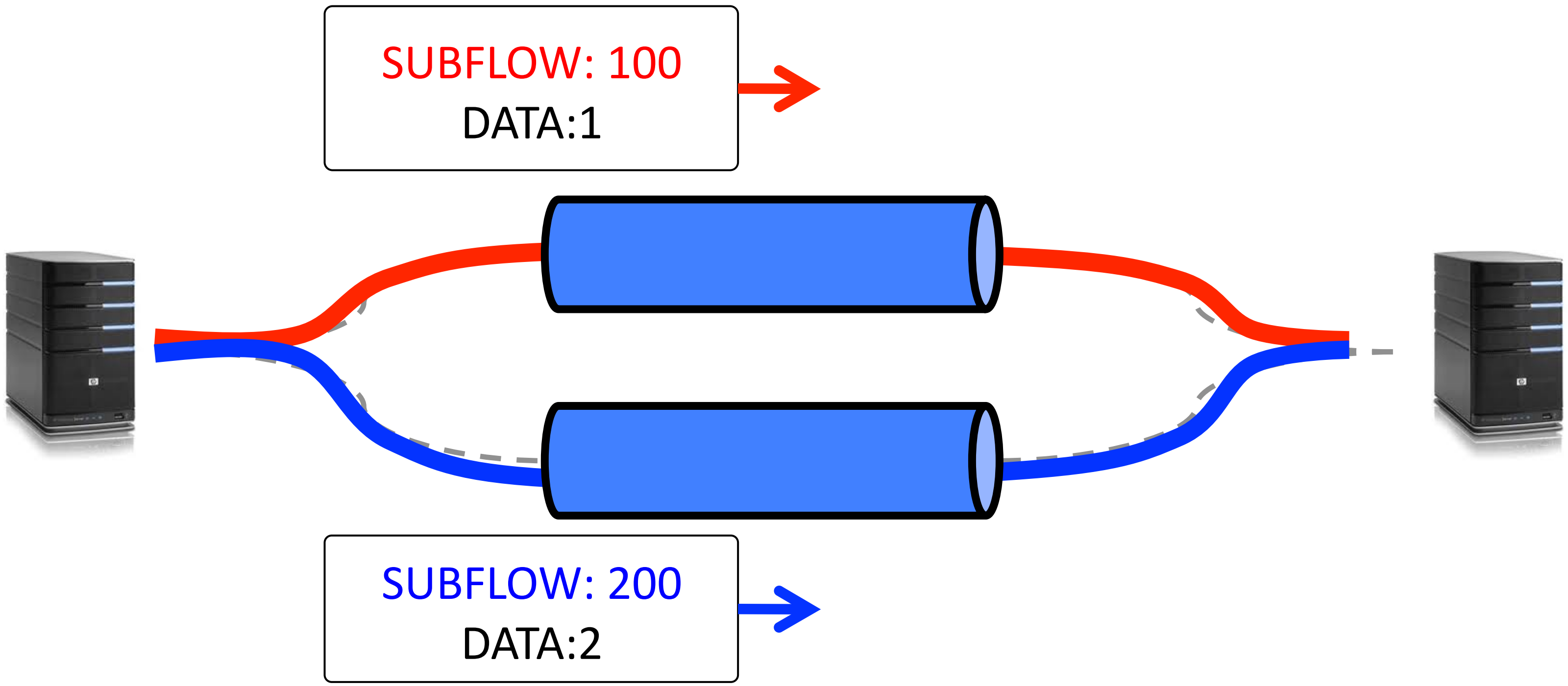
Ok, so what does work?

- We need a sequence space for each subflow
  – This will drive loss detection and retransmissions
- We need a data sequence number
  – This will put segments in order at the receiver
- We need a data ACK for flow control
  – Receive window is relative to Data ACK

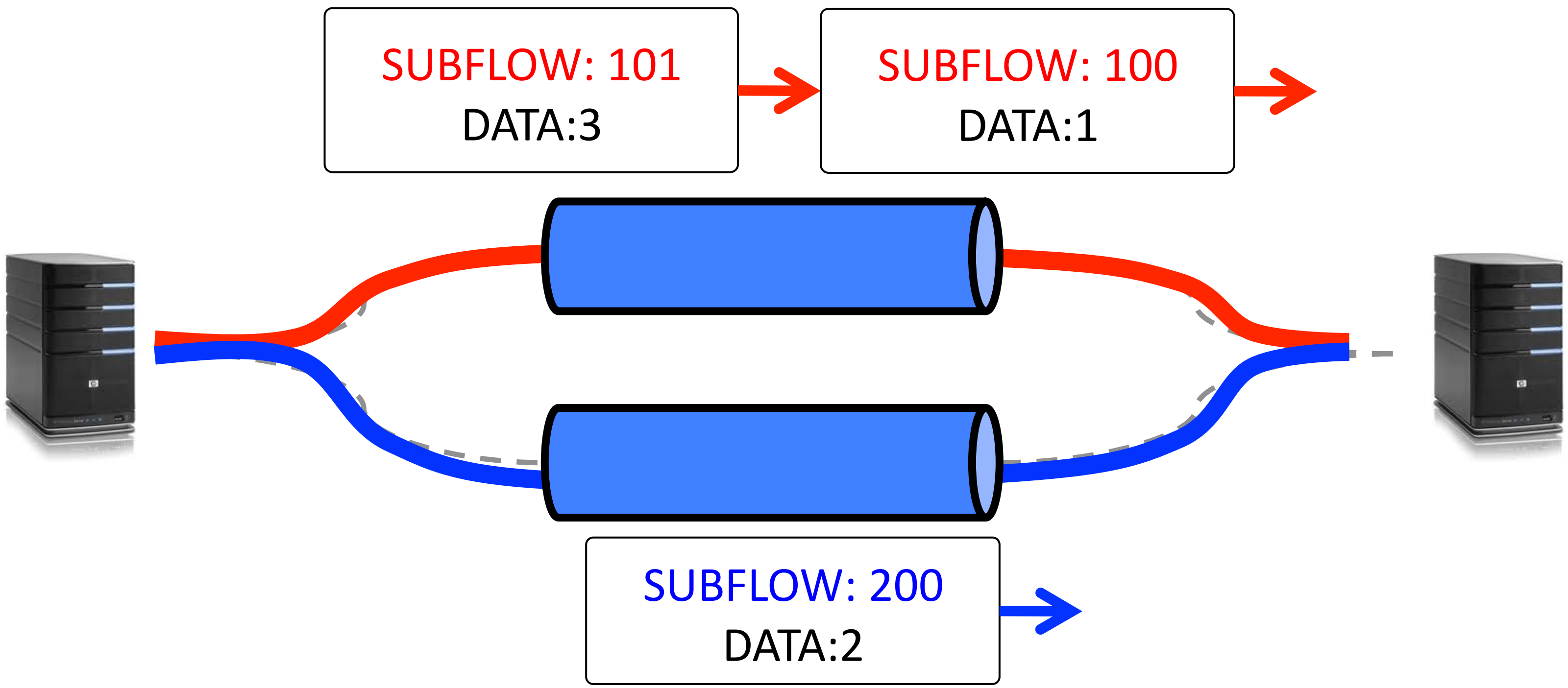
MPTCP Data Transmission



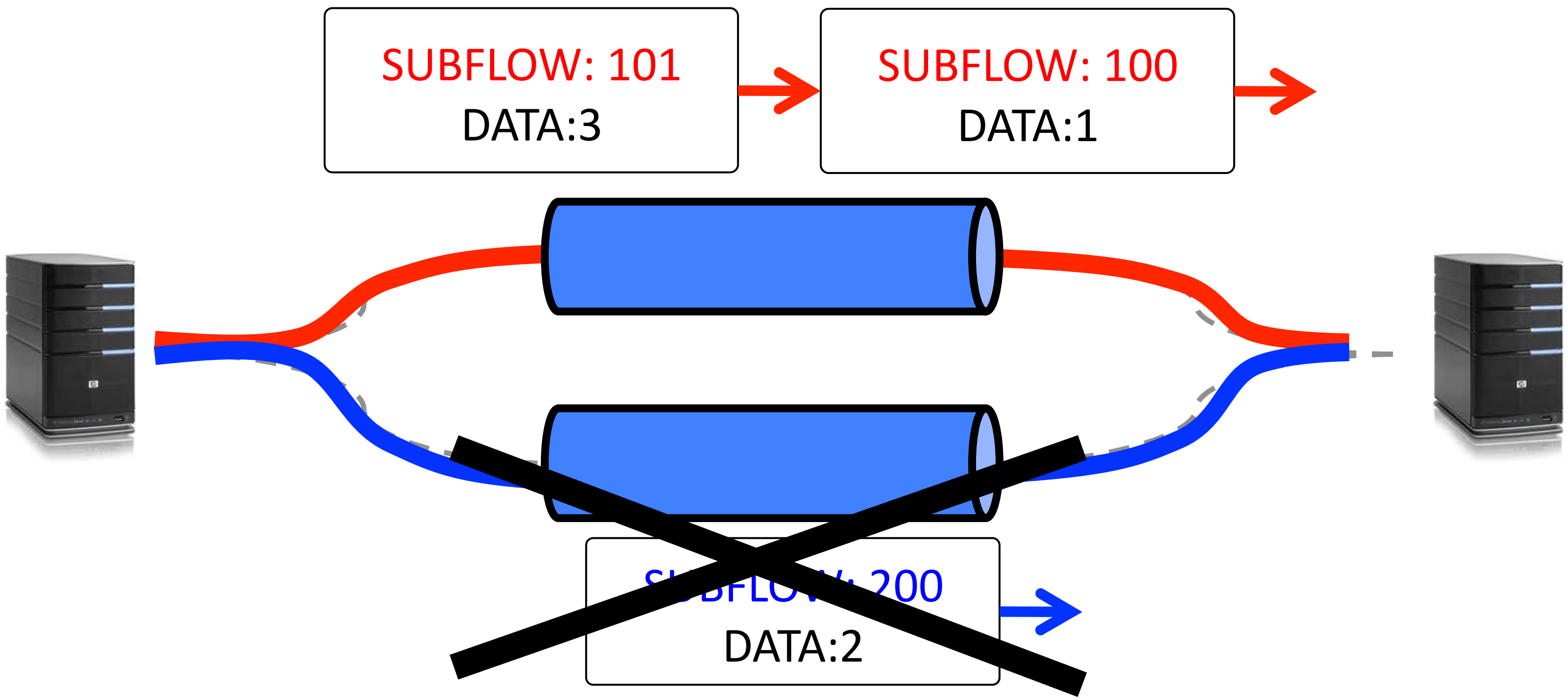
MPTCP Data Transmission



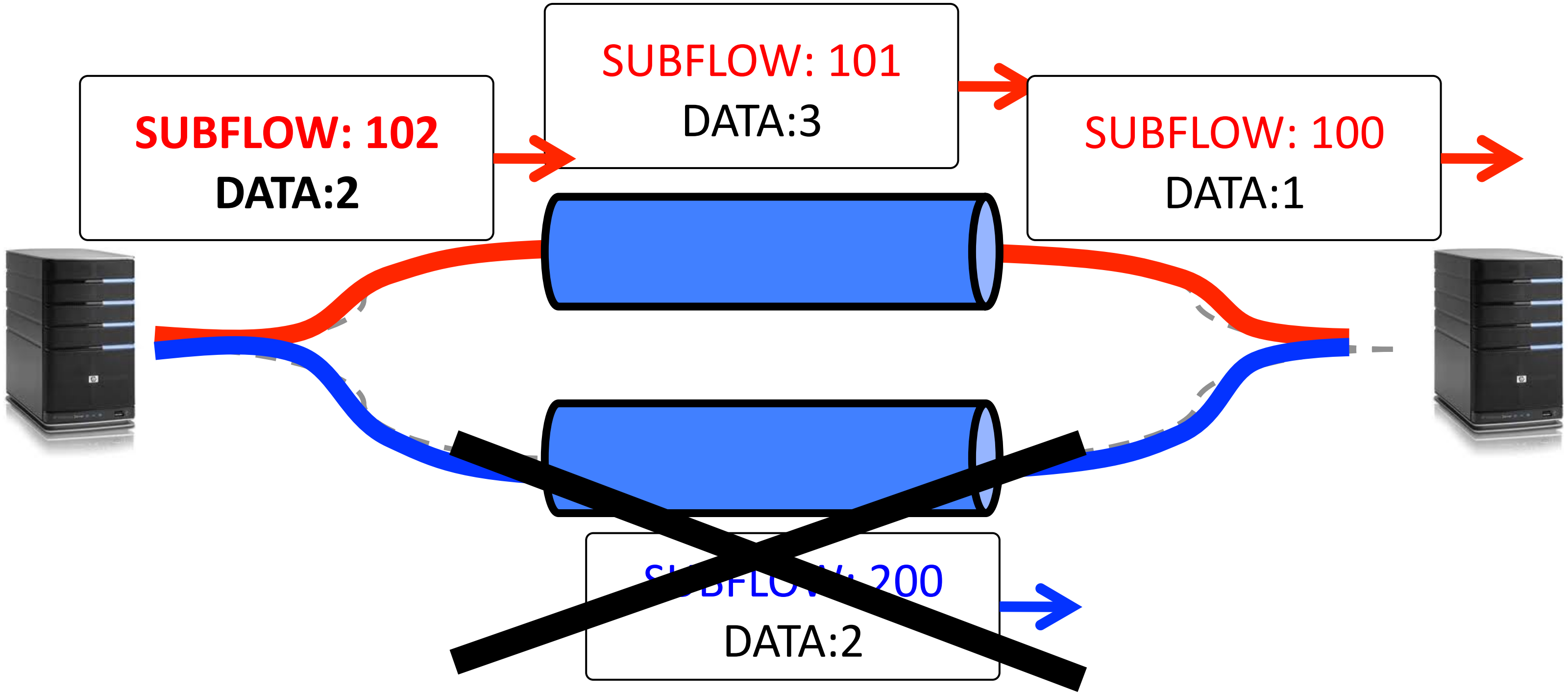
MPTCP Data Transmission



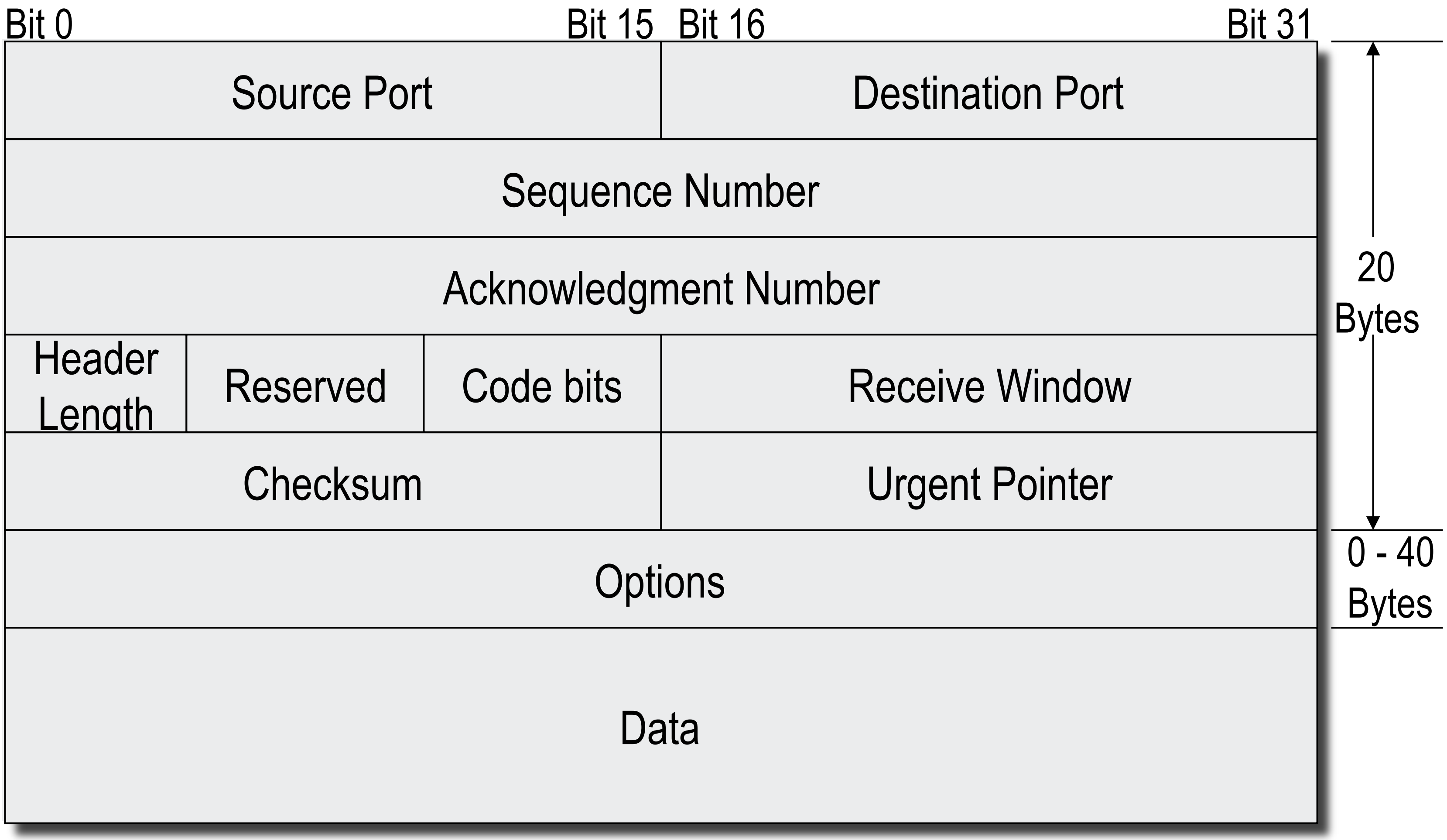
MPTCP Data Transmission



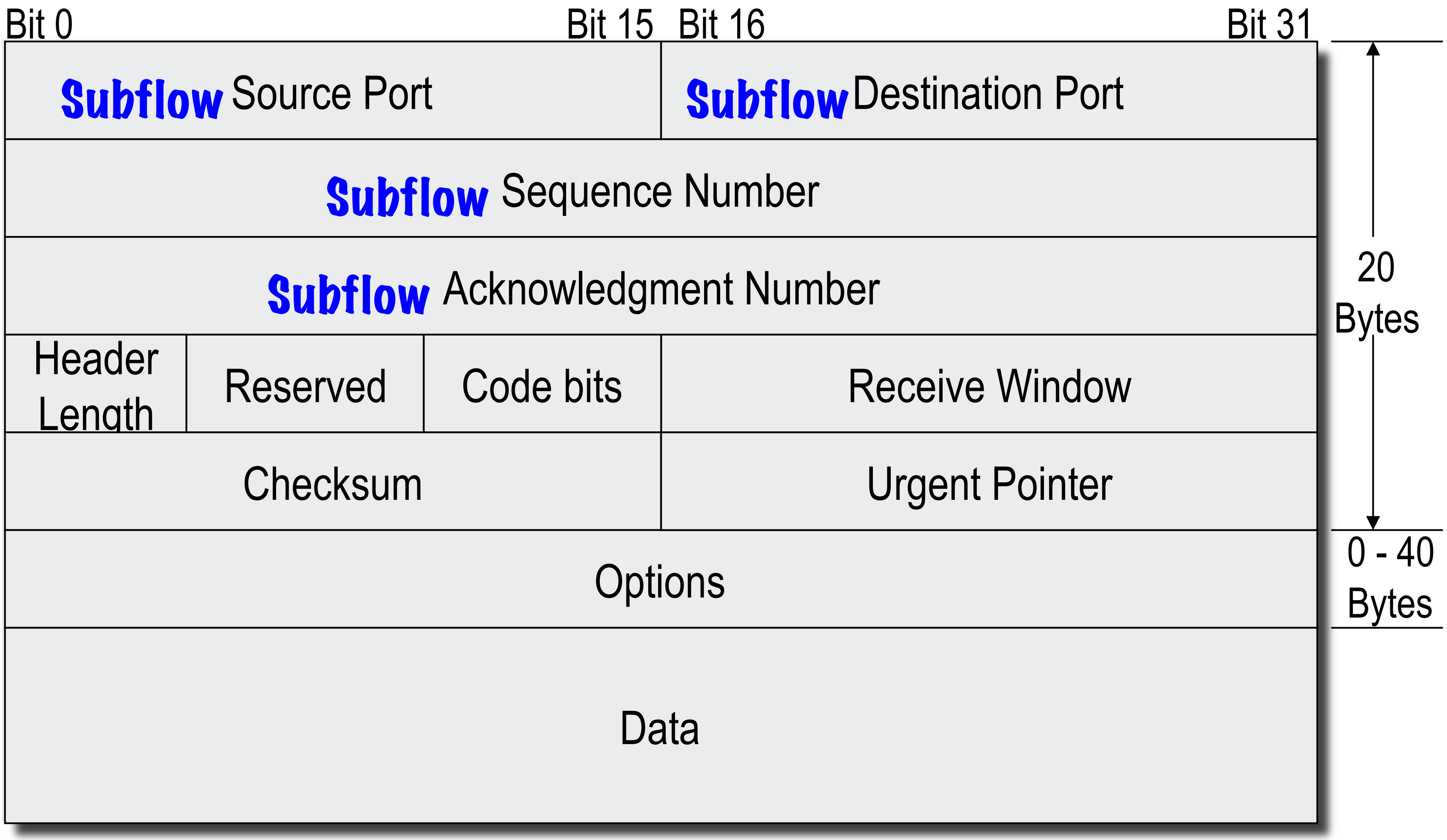
MPTCP Data Transmission



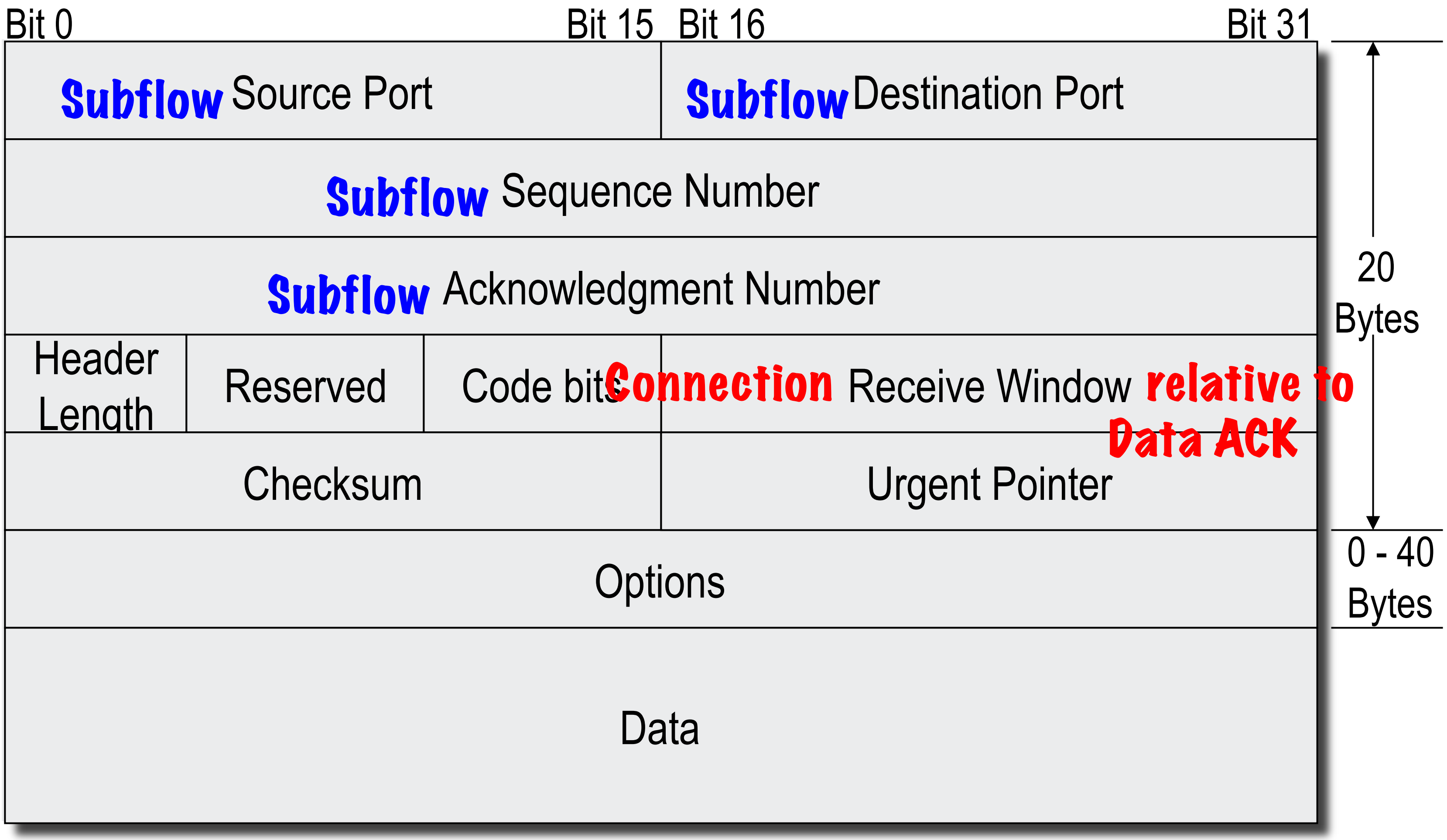
TCP Packet Header



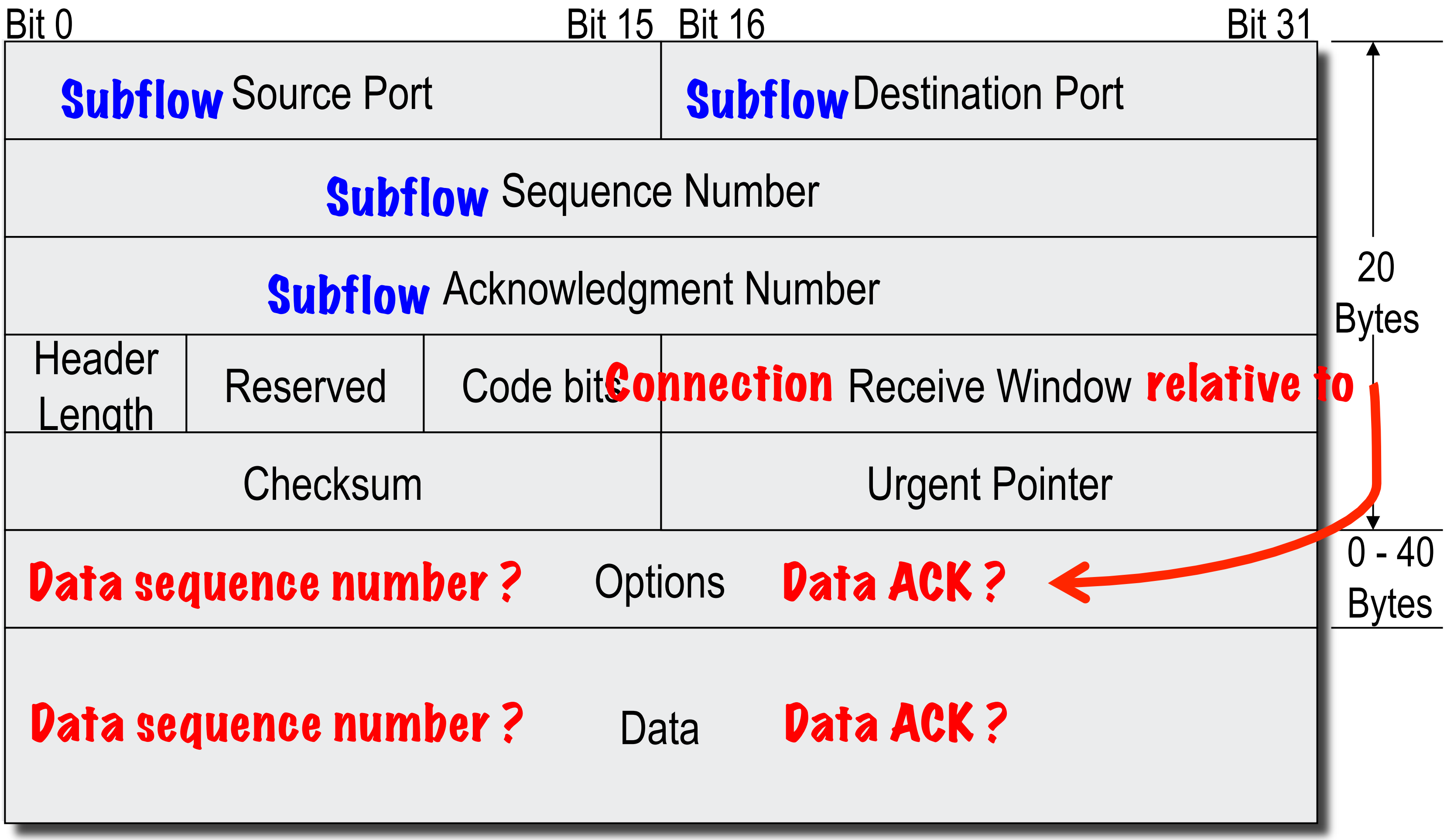
MPTCP Packet Header



MPTCP Packet Header

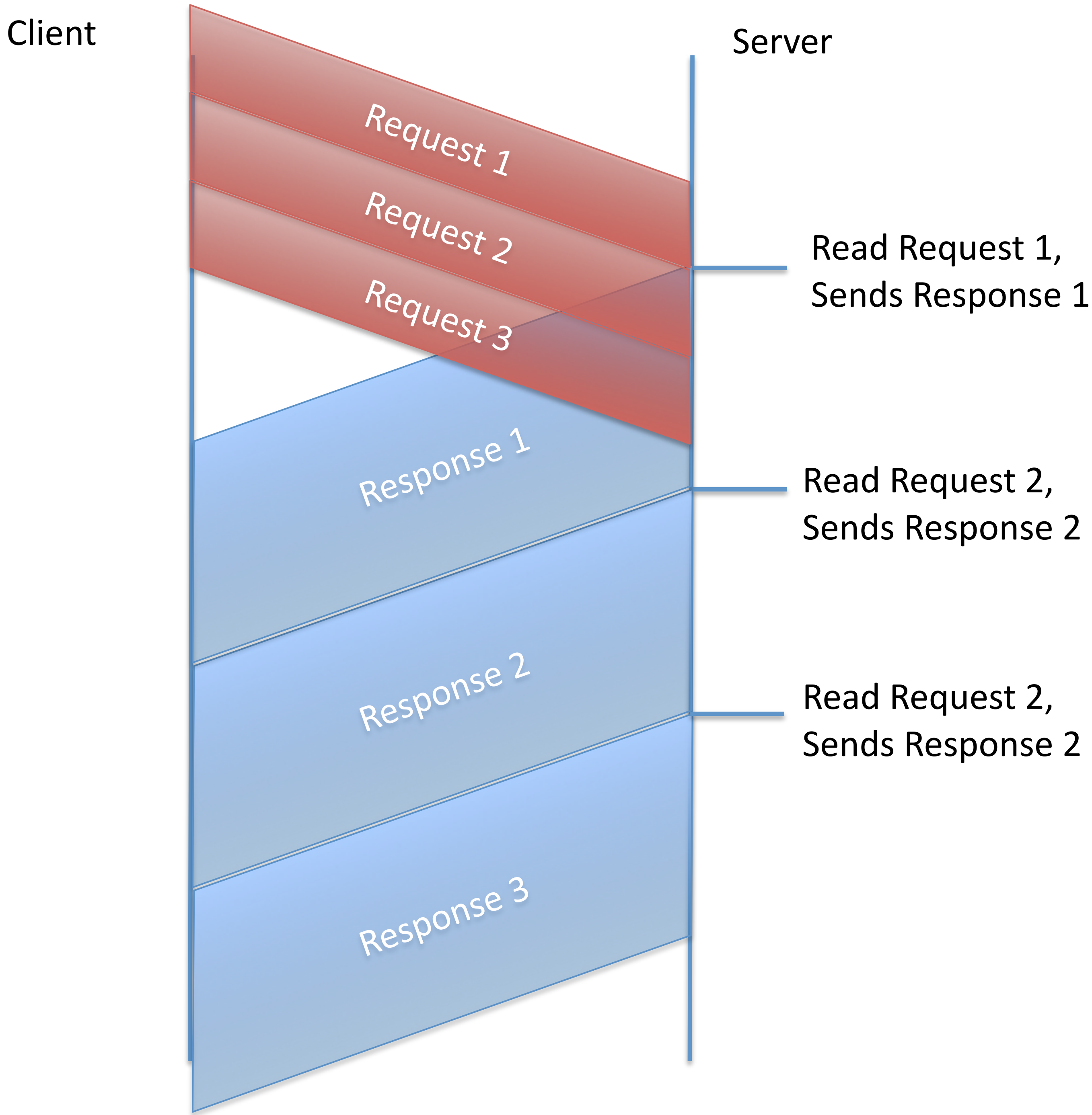


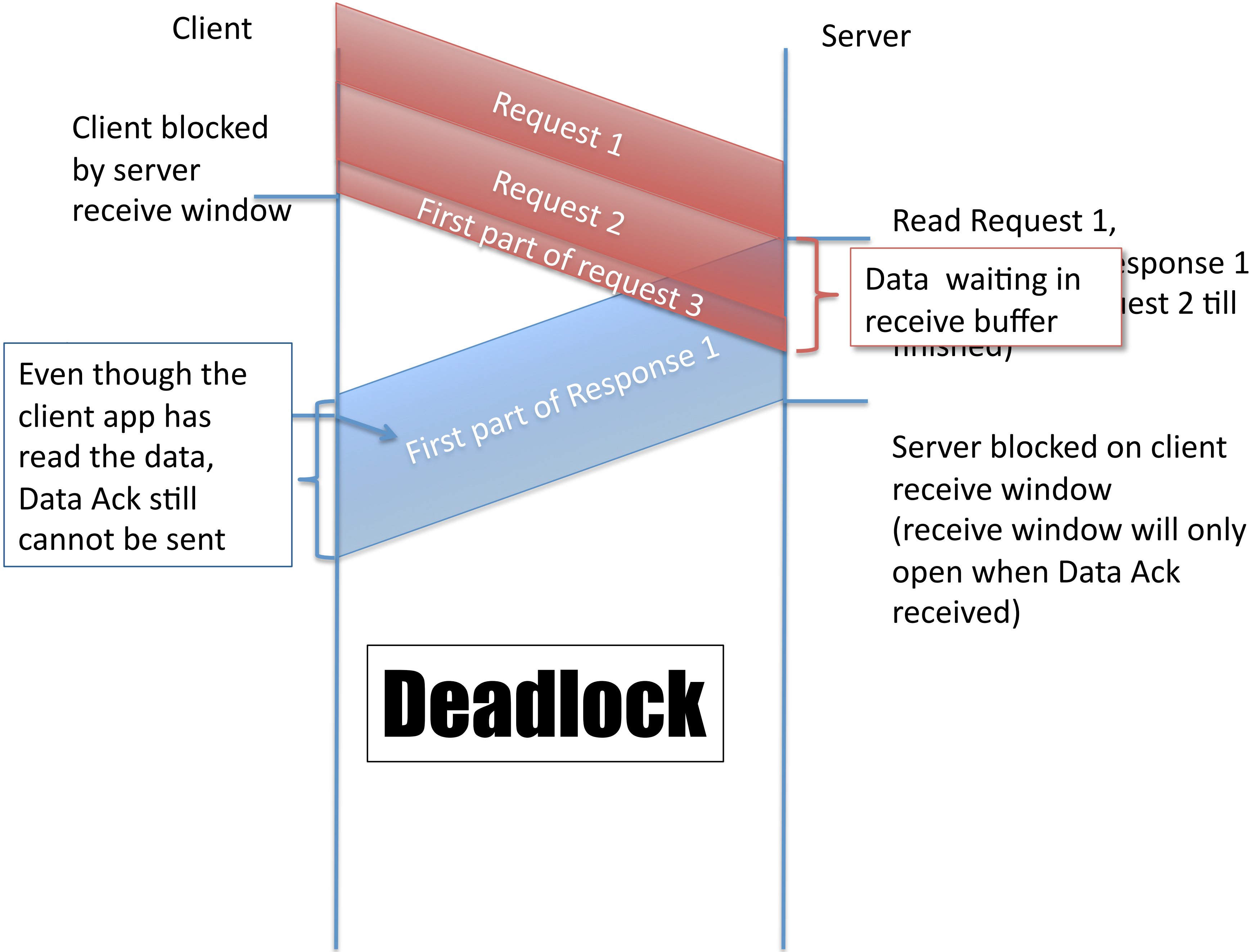
MPTCP Packet Header



Sending Data ACKs in the payload **sucks**

Sending Data ACKs in the payload ~~**sucks**~~ leads to **deadlocks**


Client
Server
Request 1
Request 2
Request 3
Response 1
Response 2
Response 3
Read Request 1,
Sends Response 1
Read Request 2,
Sends Response 2
Read Request 2,
Sends Response 2


Client
Server
Request 1
Request 2
First part of request 3
Client blocked by server receive window
Read Request 1,
Data waiting in receive buffer
First part of Response 1
Even though the client app has read the data, Data Ack still cannot be sent
Server blocked on client receive window (receive window will only open when Data Ack received)
Deadlock

Design space for feasible solutions is quite narrow

There are not too many things that could have been done differently

Read paper for:

- Flow control
- Dealing with content-changing middleboxes
- Dealing with TSO/LRO
- Connection teardown
- Fast receive code
- Middlebox tests
- Evaluation

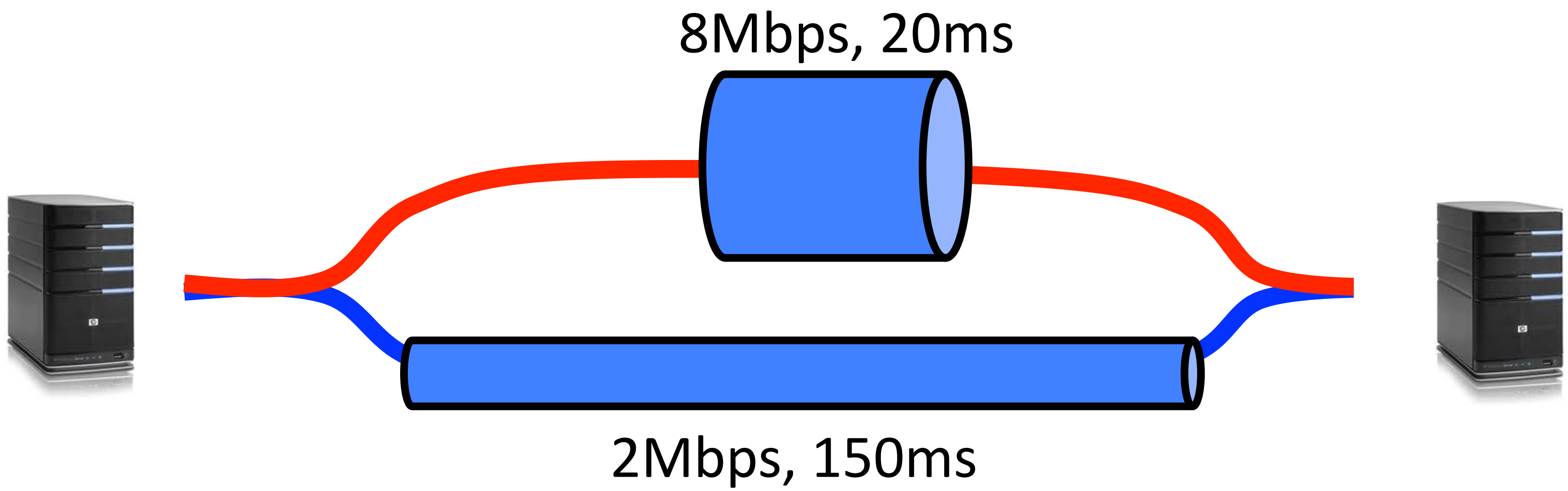
Deployable Multipath TCP

How hard can it be?

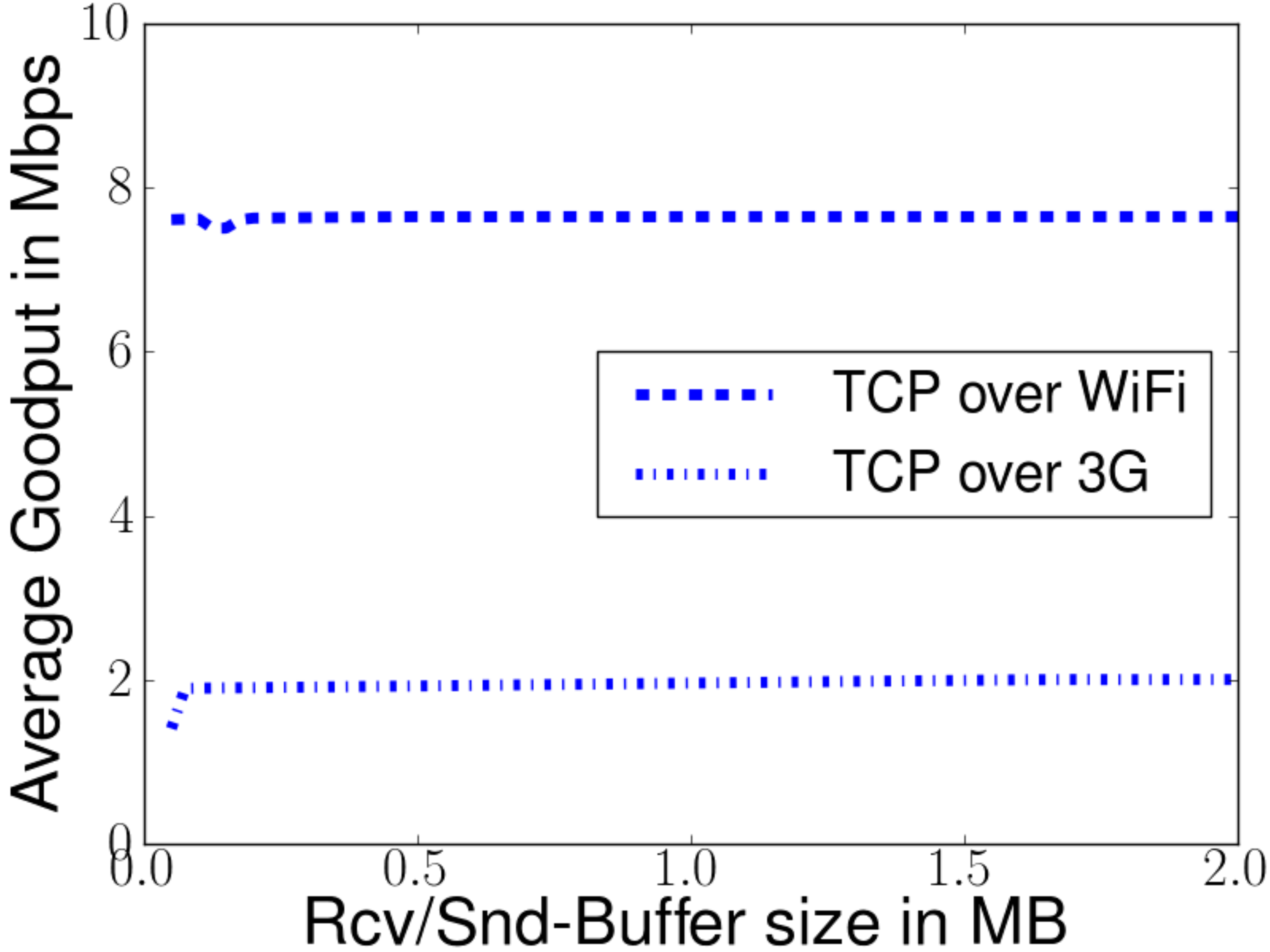
Designing

Implementing

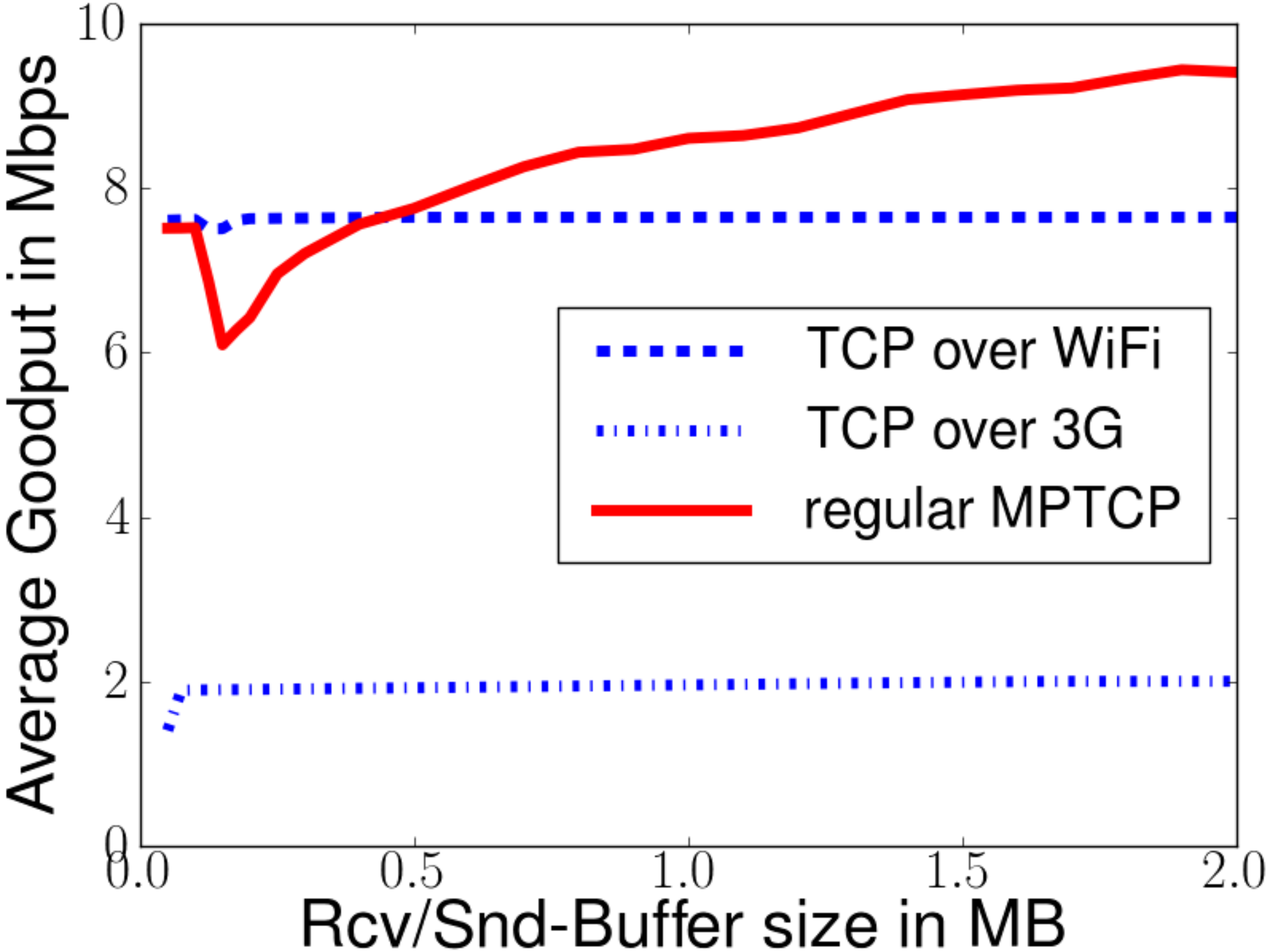
MPTCP over WiFi/3G



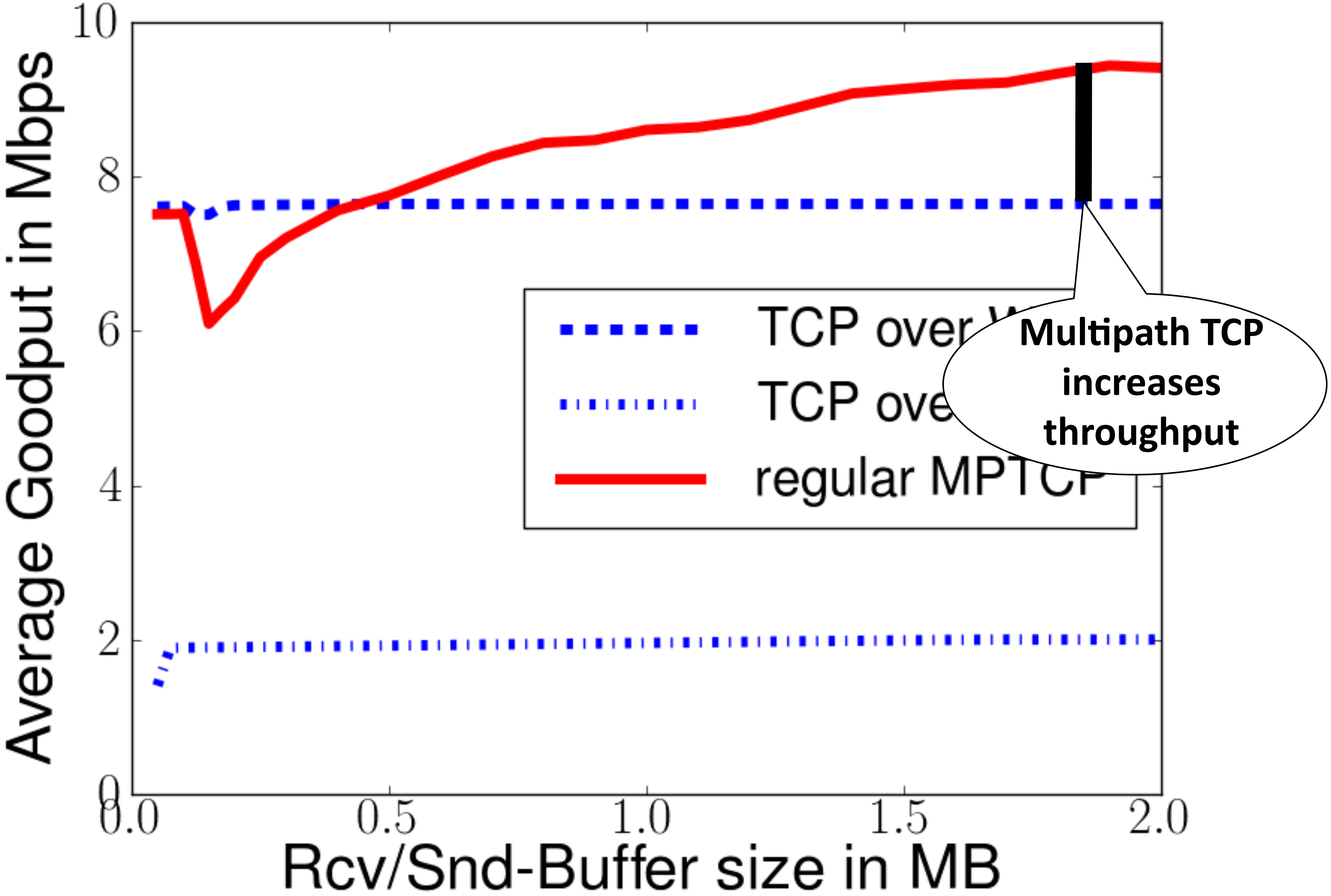
TCP over WiFi/3G



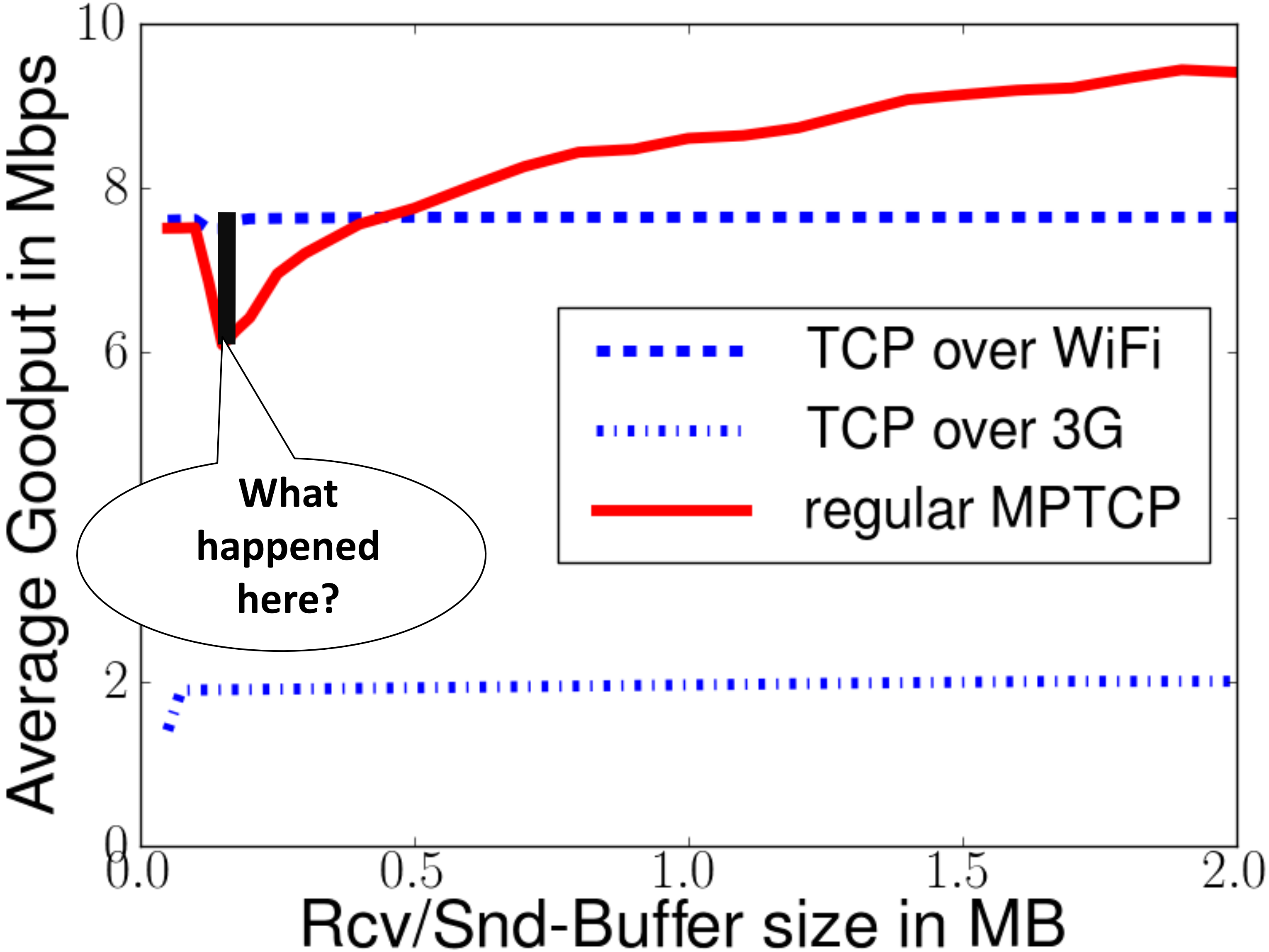
MPTCP over WiFi/3G



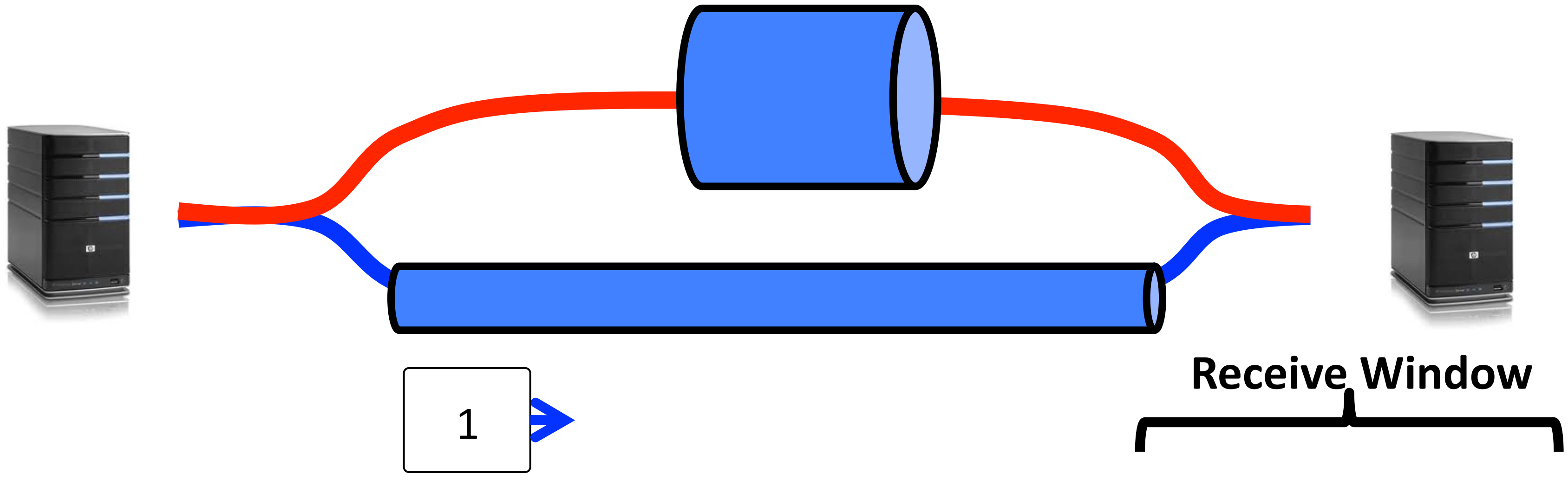
MPTCP over WiFi/3G



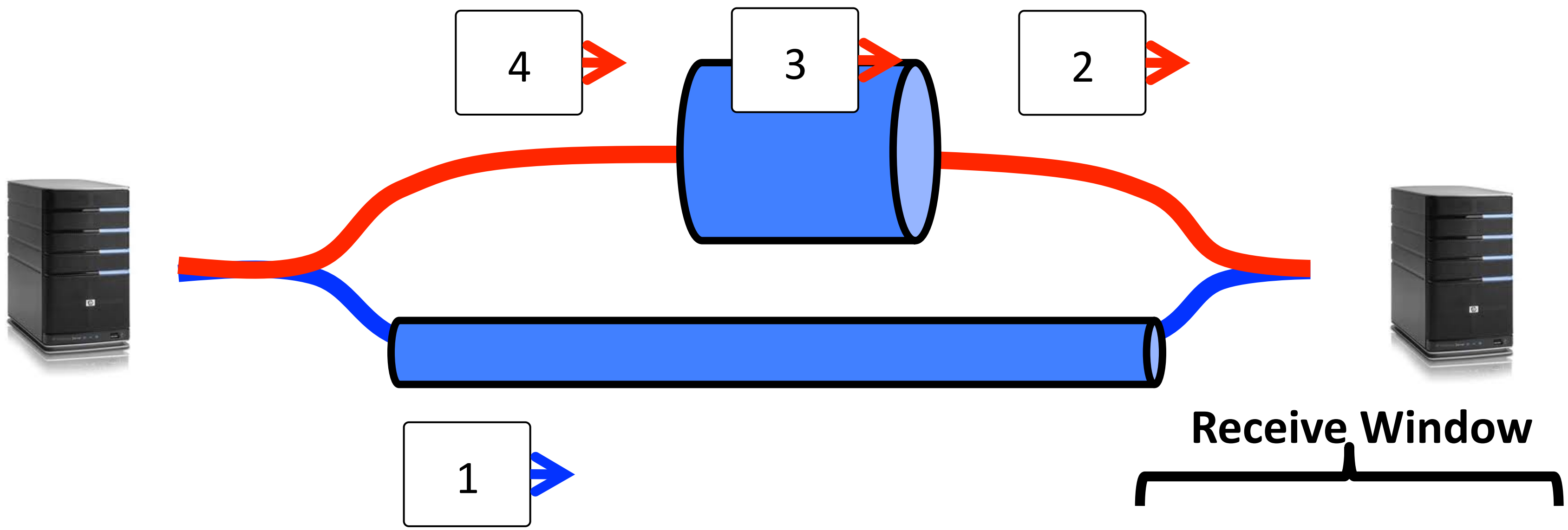
MPTCP over WiFi/3G



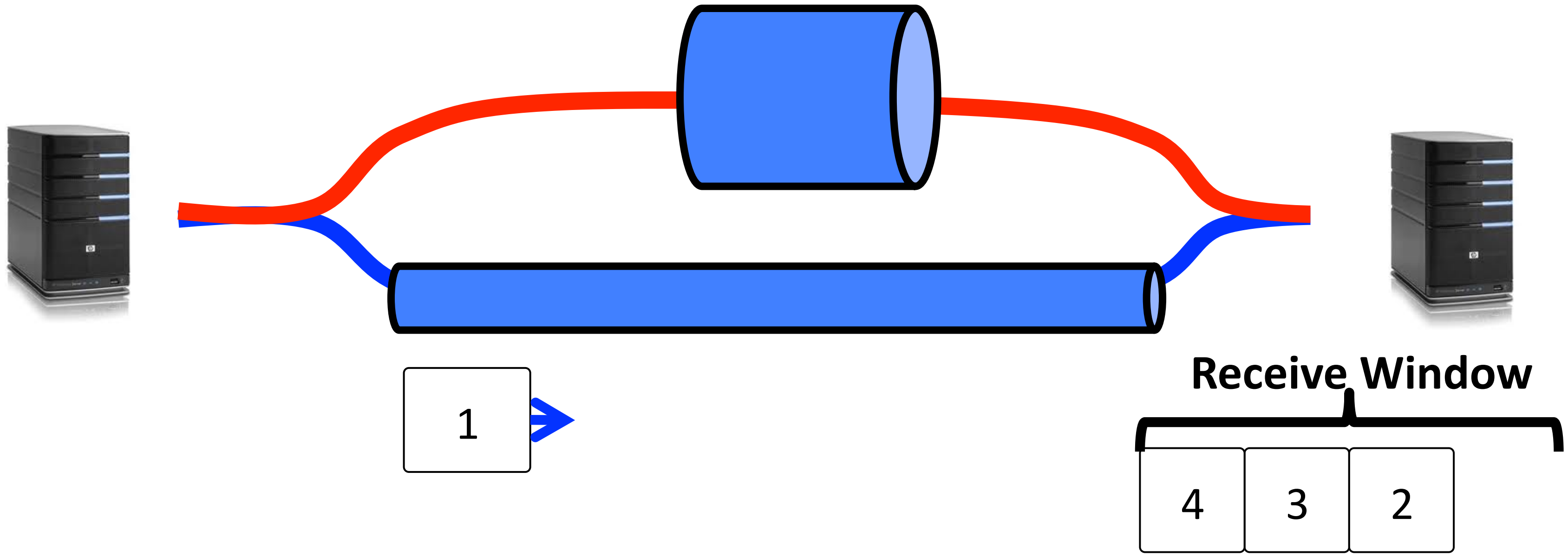
MPTCP over WiFi/3G



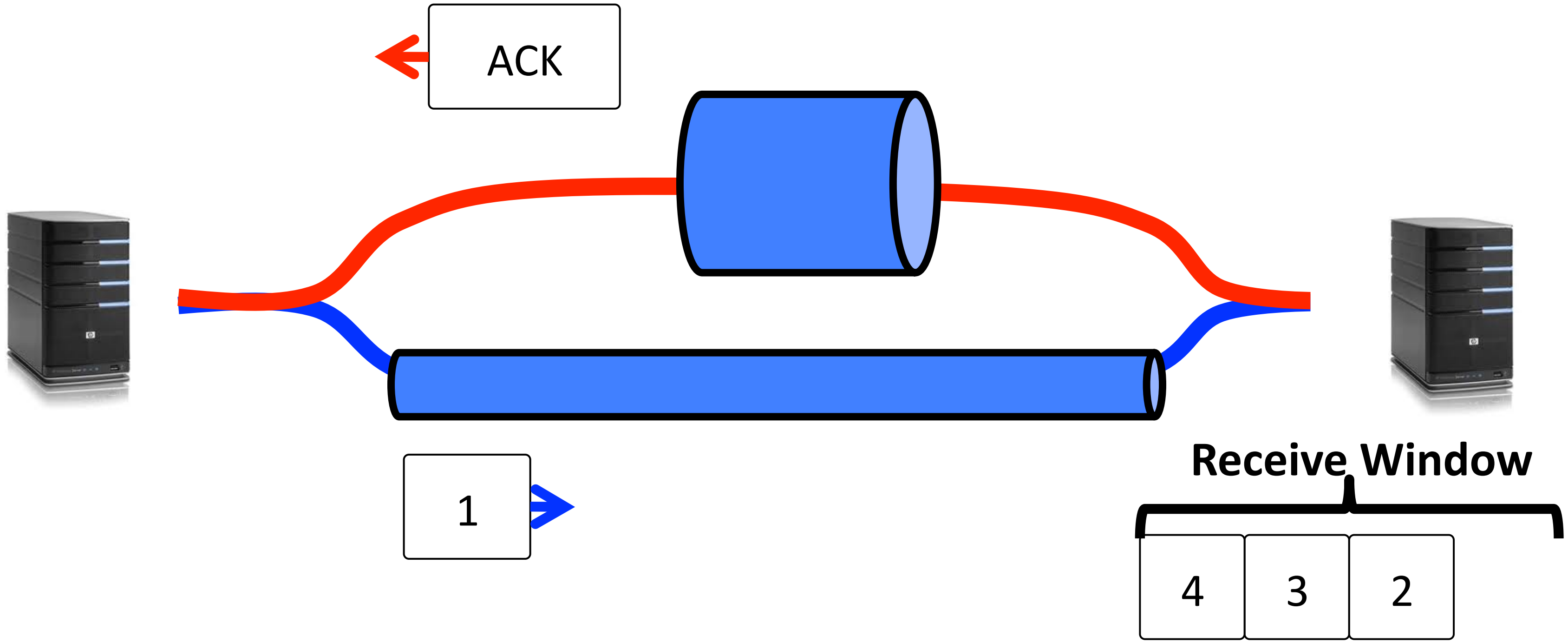
MPTCP over WiFi/3G



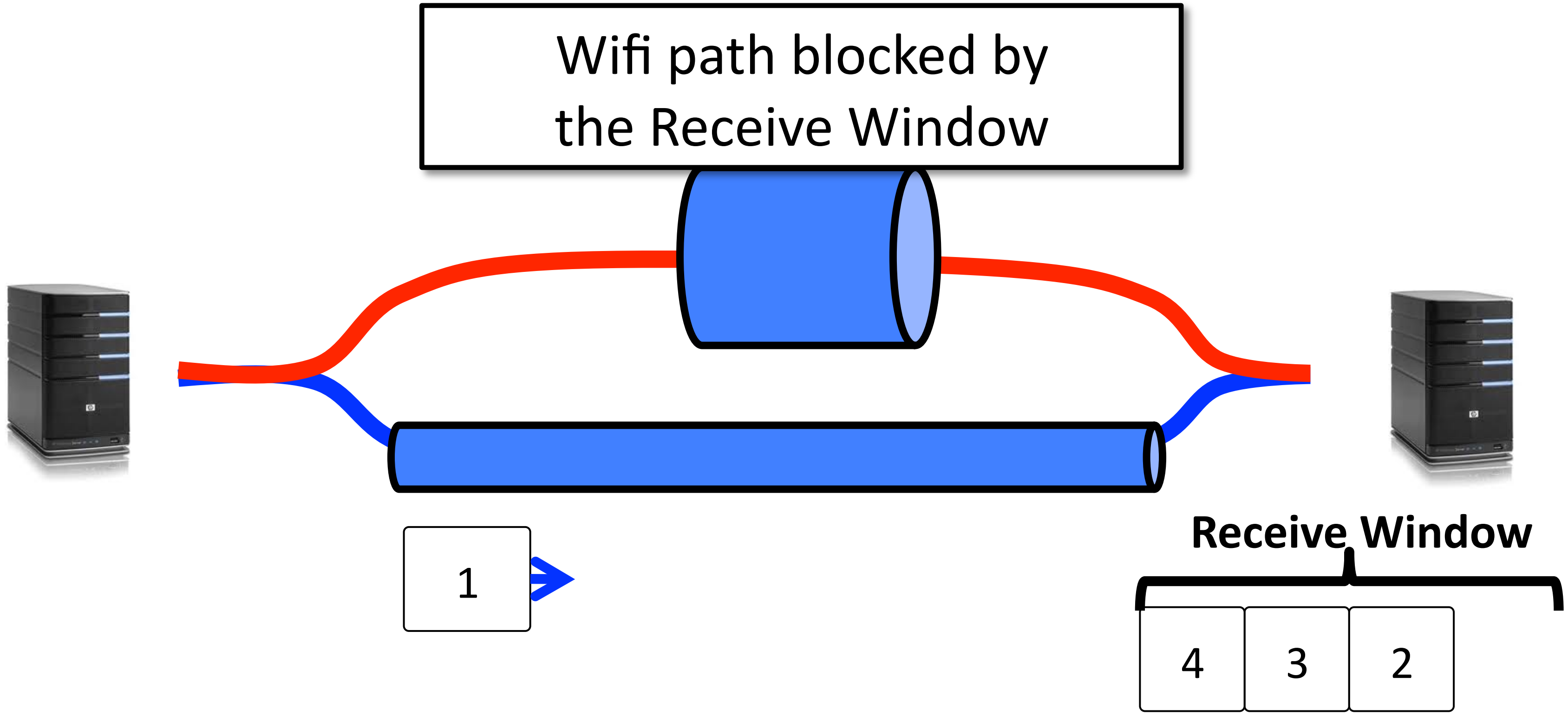
MPTCP over WiFi/3G



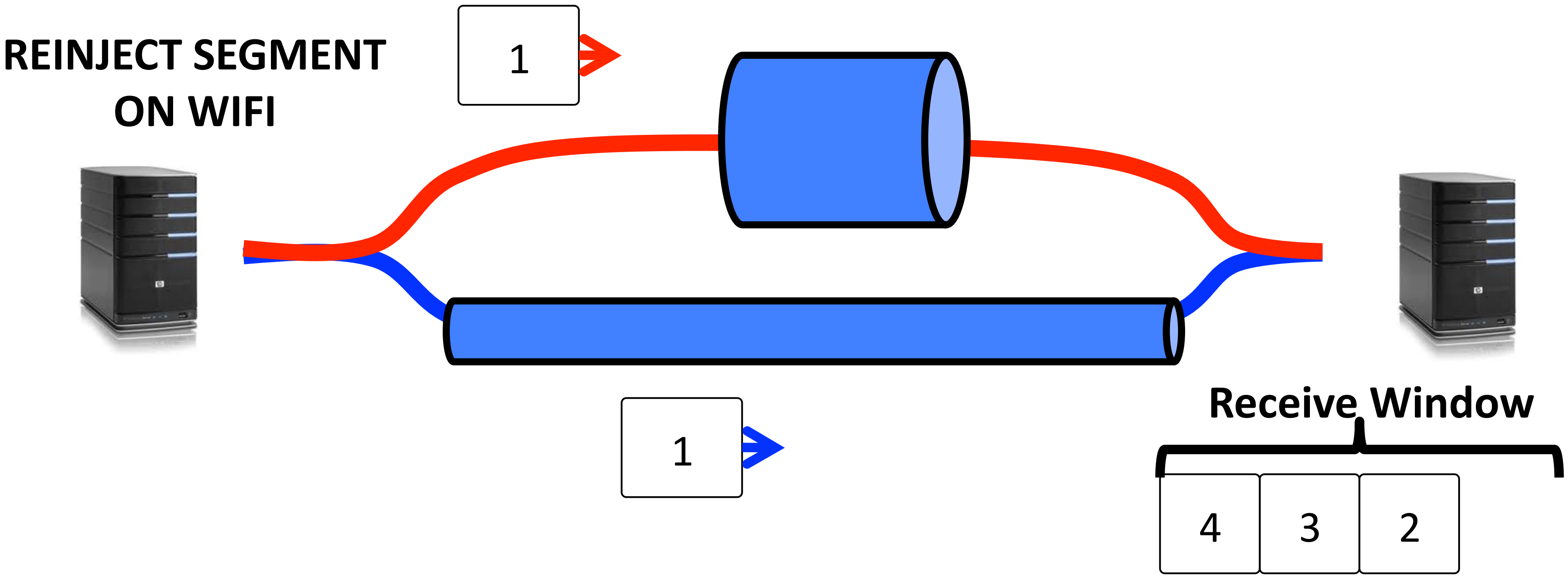
MPTCP over WiFi/3G



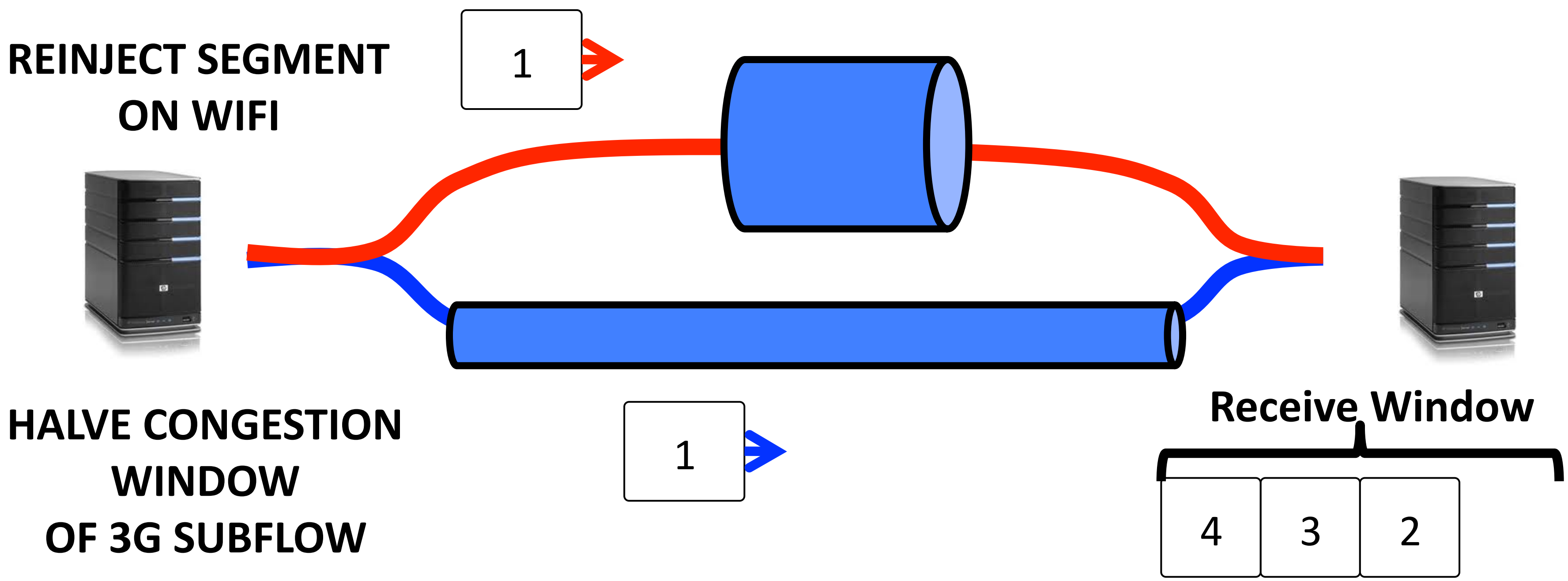
MPTCP over WiFi/3G



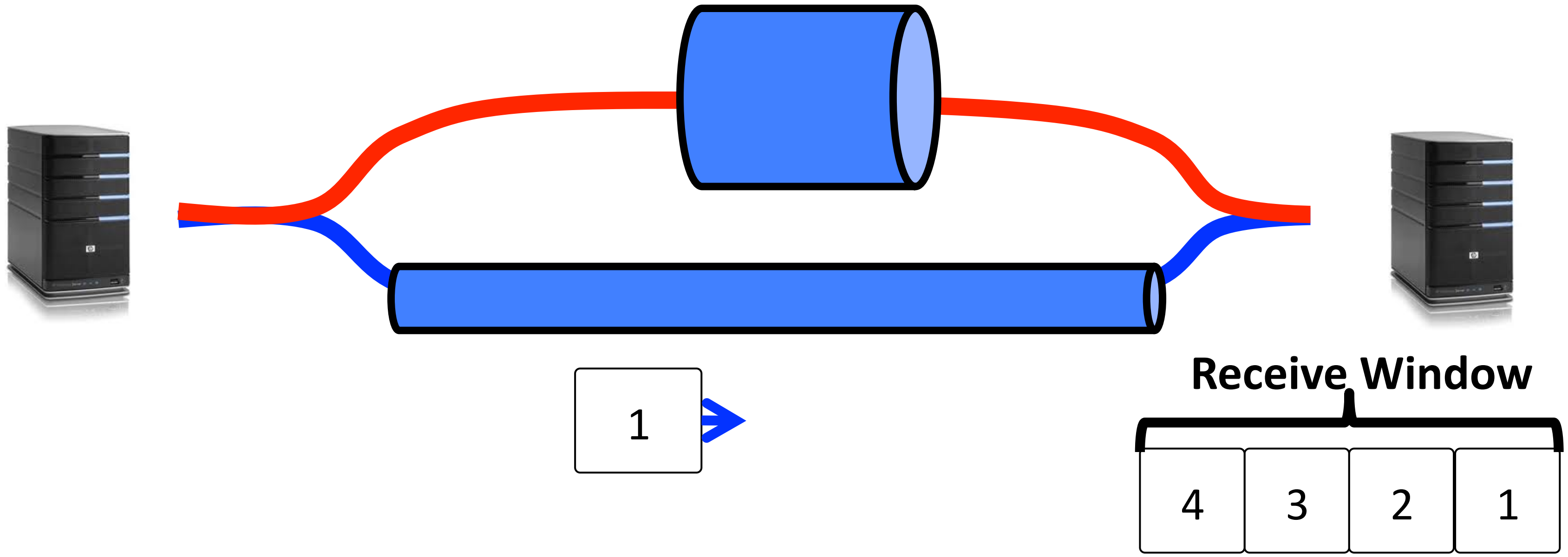
MPTCP over WiFi/3G



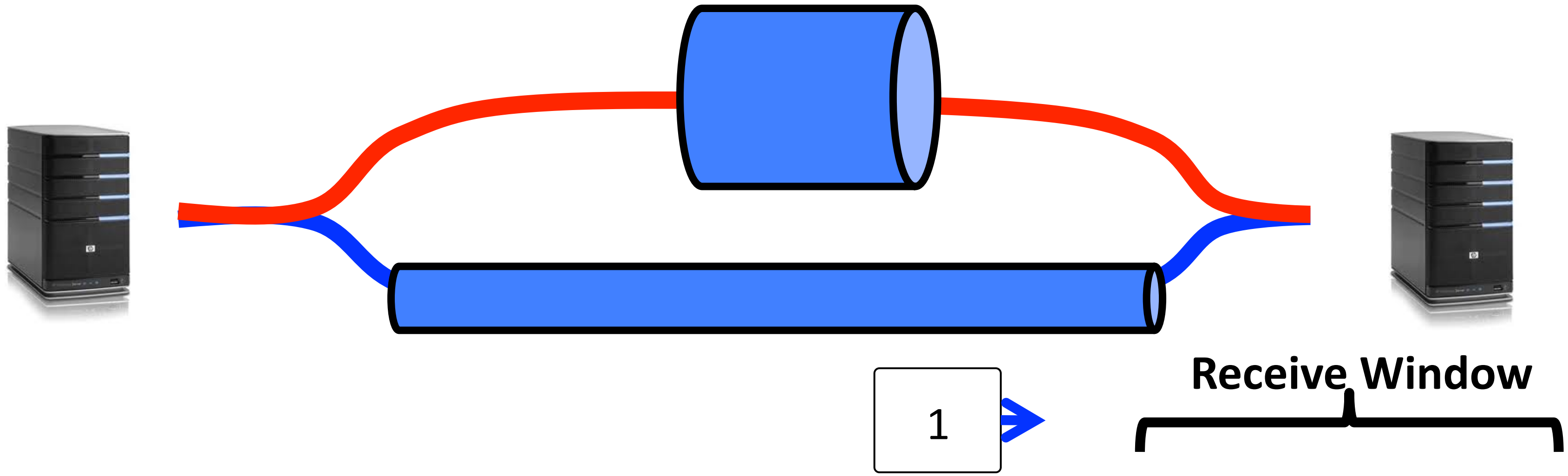
MPTCP over WiFi/3G



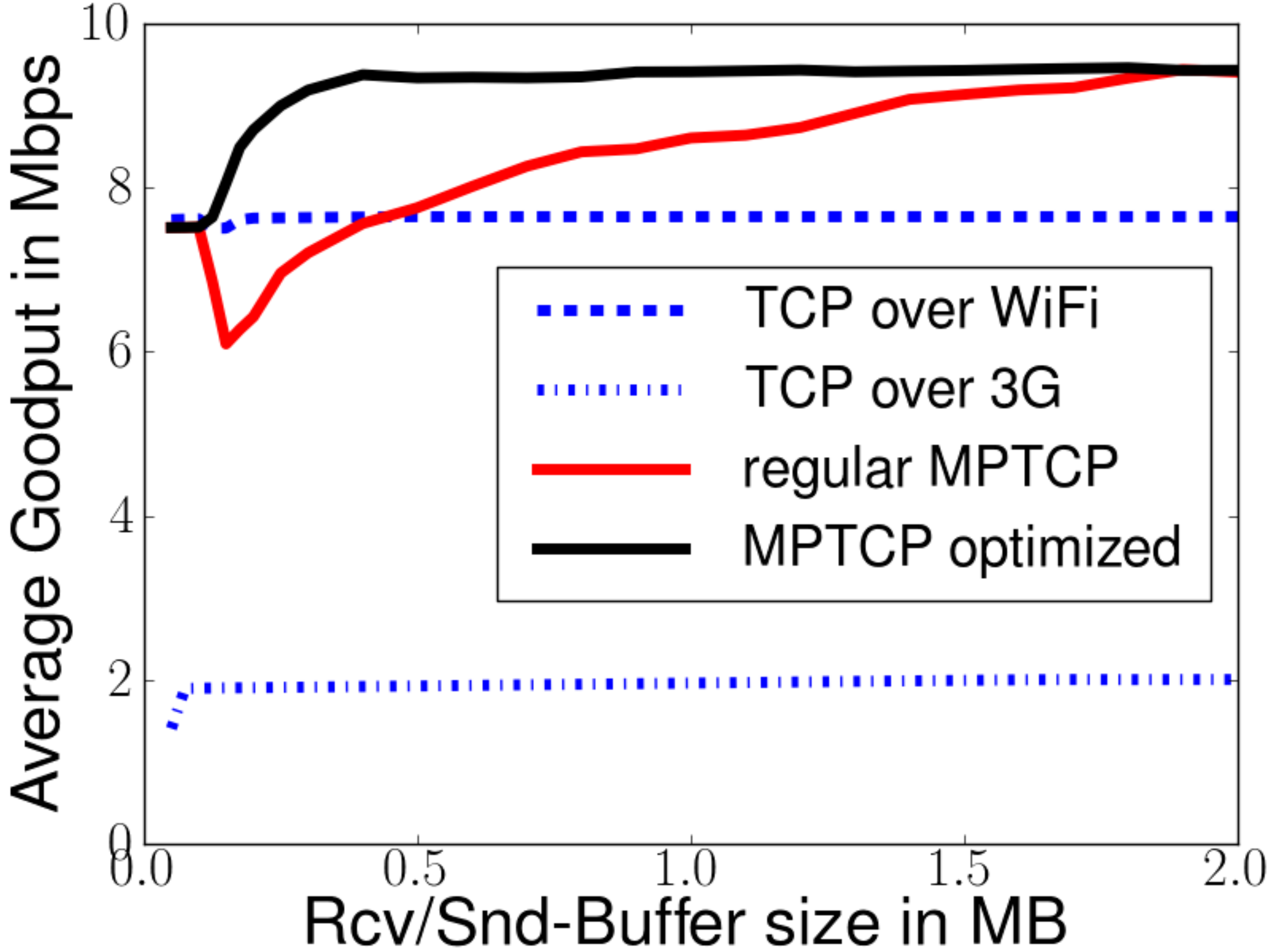
MPTCP over WiFi/3G



MPTCP over WiFi/3G



MPTCP over WiFi/3G



Demo

Conclusions

- Designing *a* Multipath TCP isn't difficult.
- Designing a *deployable* Multipath TCP is much harder.
  - Need to understand the evolving and undocumented Internet architecture.
  - Need defensive mechanisms to fall back to TCP behaviour when all else fails.
- Most extensions to TCP now face the same hurdles.

Conclusions (2)

- Designing a performant MPTCP needs care.
  - Especially need careful management of buffering to avoid unwanted interactions between subflows.

MPTCP allows standard applications to reap the benefits of multipath networks

- It is deployable today
- Try out the code – *http://mptcp.info.ucl.ac.be/*