Mobilitate la nivel transport

- recapitulare TCP
- comportare TCP la delay și loss
- PEP (Performance Enhancing Proxies)
- MPTCP

Mobility and transport

1. Performance
   – TCP is averse to loss, delay

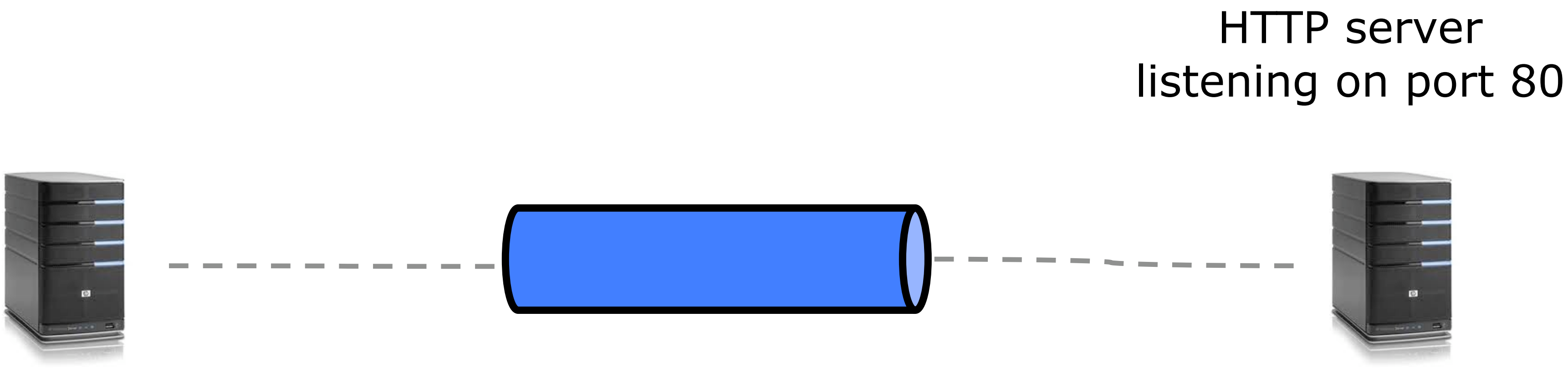
2. Functionality
   – TCP is tied to IP

TCP refresher

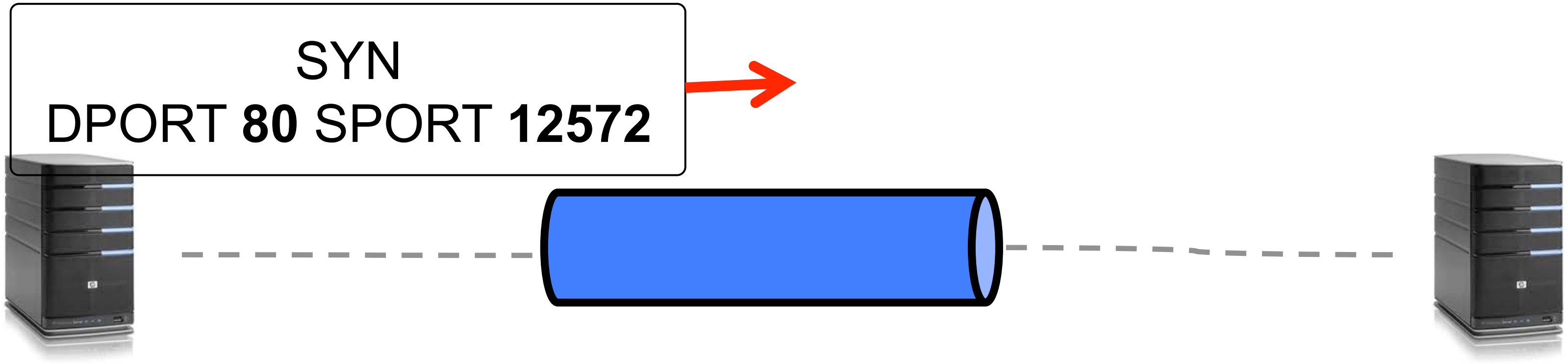
- TCP provides two functions:
  - reliable delivery
  - congestion control

- How does it react to
  - Loss?
  - Delay?
  - Mobility?

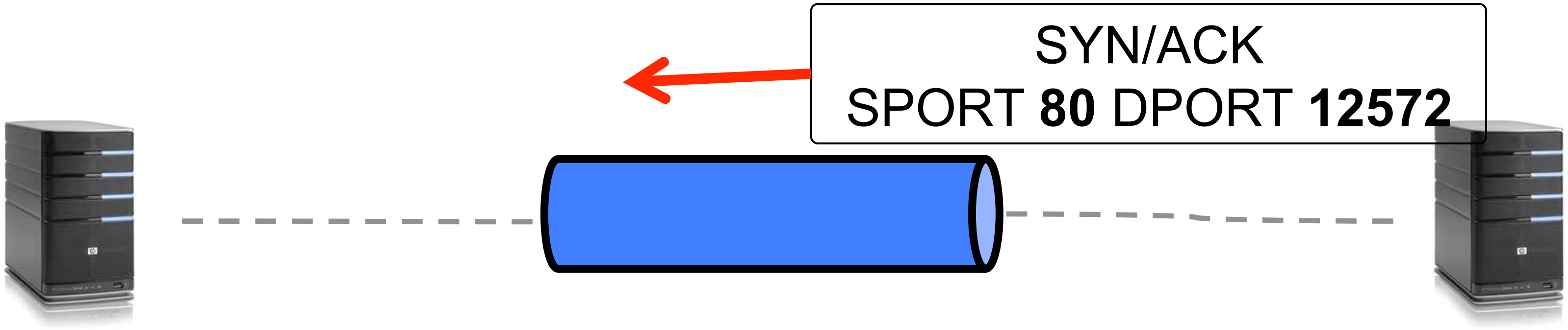
TCP Connection Setup



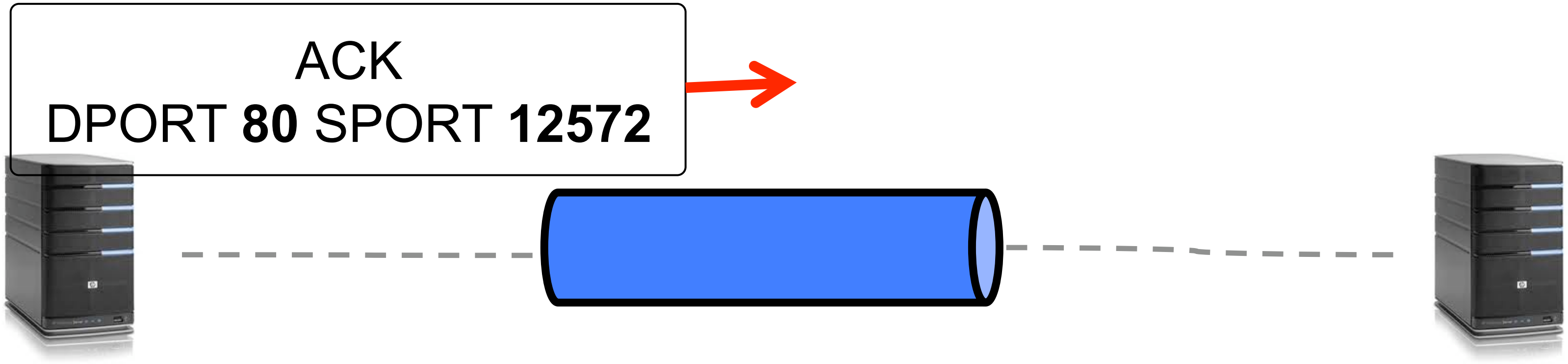
TCP Connection Setup



TCP Connection Setup


SYN/ACK
SPORT **80** DPORT **12572**

TCP Connection Setup

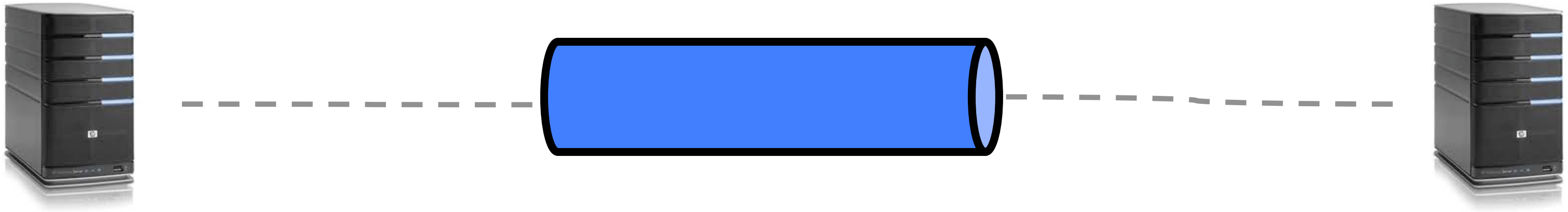


ACK
DPORT **80** SPORT **12572**

reliable in order byte stream delivery

- Apps send any numbers of bytes
  - Say 100.000B
- TCP split bytes into segments
  - Because network works with limited-size packets
- Sends them over the network
  - Segments can be lost/reordered
- TCP receiver MUST read data in order

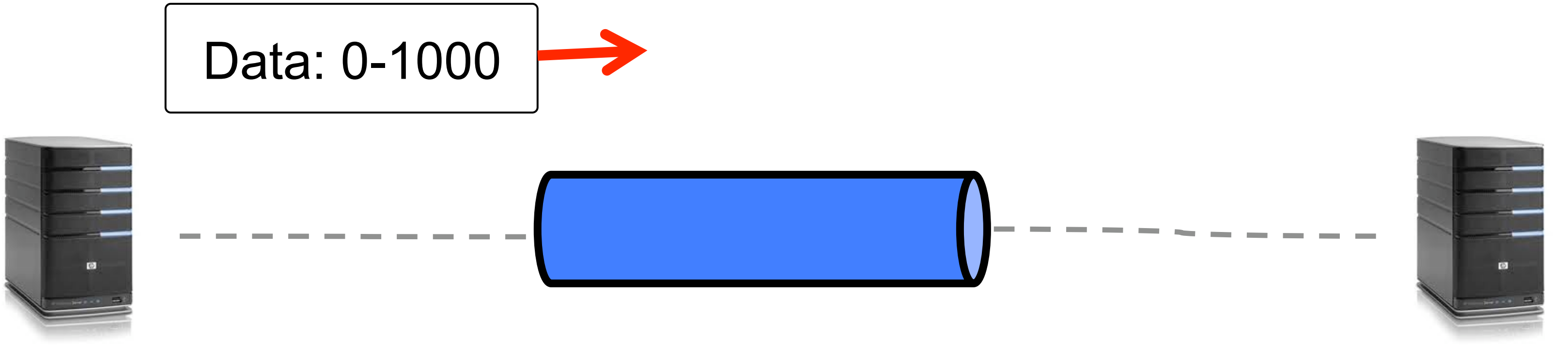
TCP Data Transmission



Browser: Send 3KB of data

HTTP Server:
Read Request

TCP Data Transmission

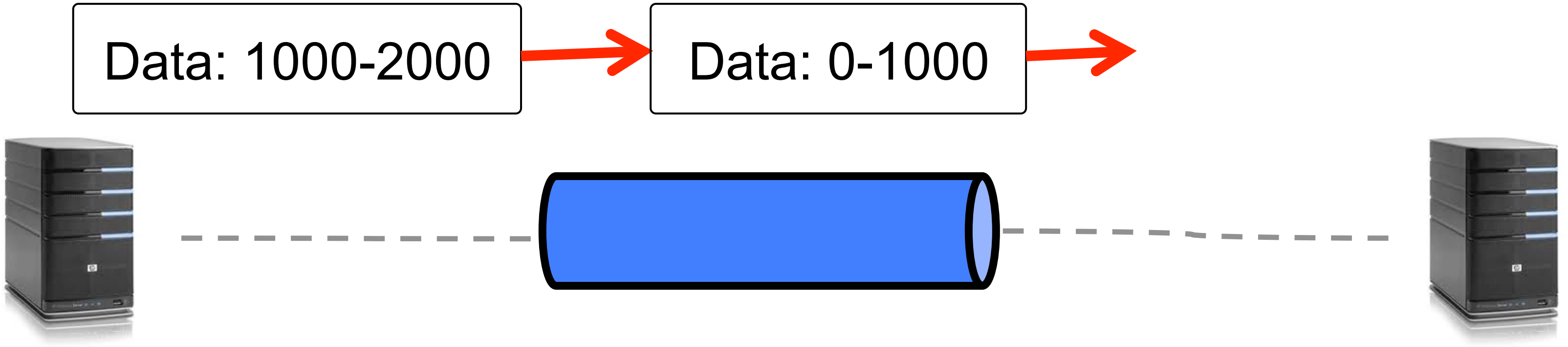


Data: 0-1000

Browser: Send 3KB of data

HTTP Server:
Read Request

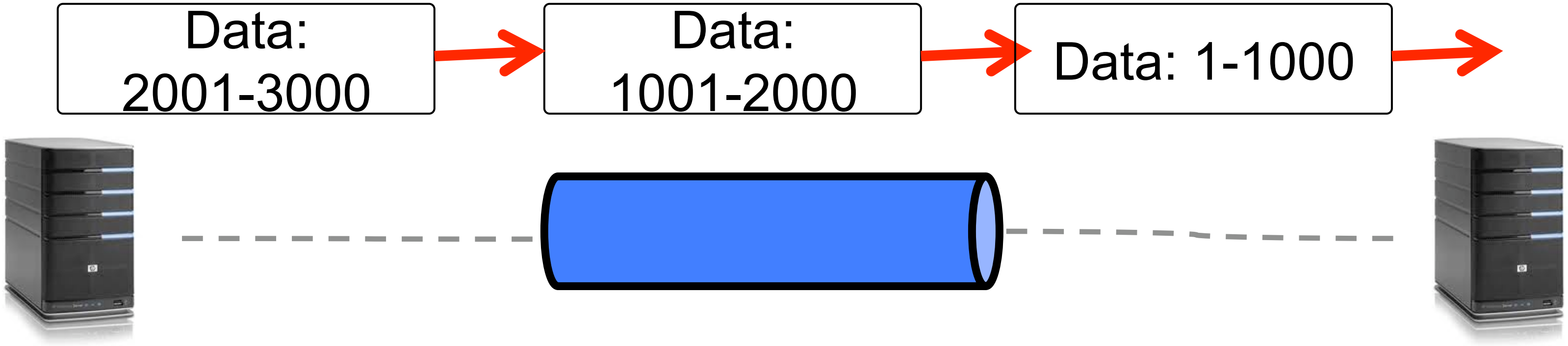
TCP Data Transmission



Browser: Send 3KB of data

HTTP Server:
Read Request

TCP Data Transmission



Data: 2001-3000

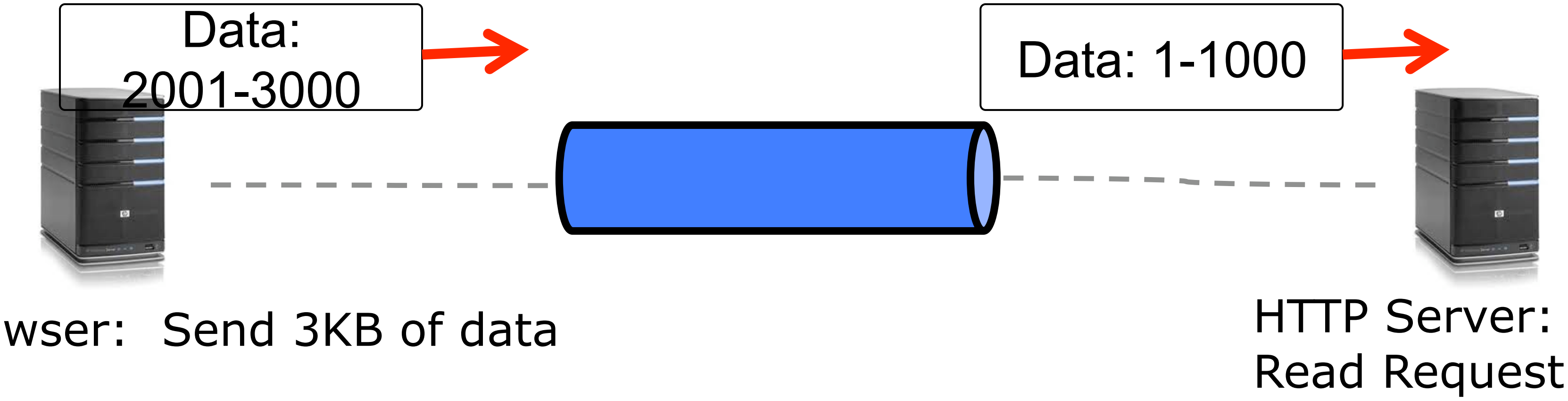
Data: 1001-2000

Data: 1-1000

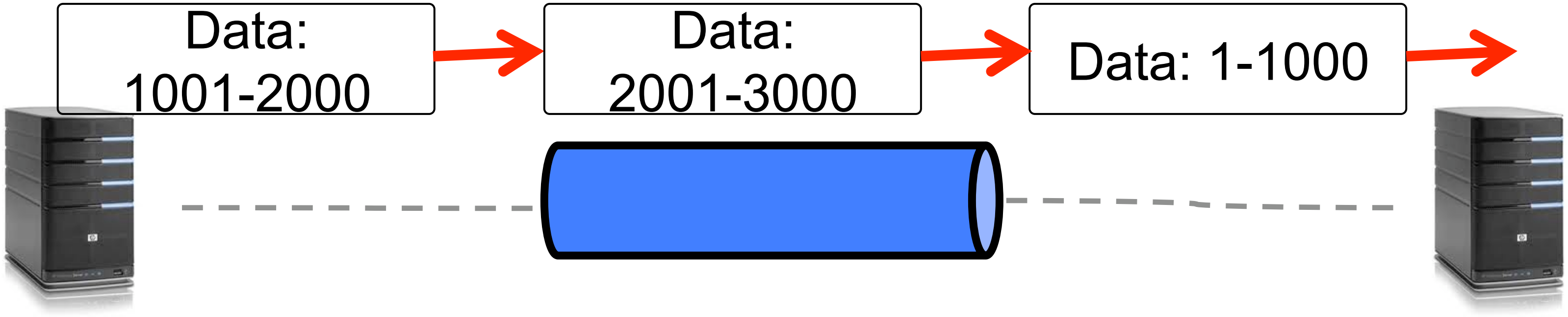
Browser: Send 3KB of data

HTTP Server: Read Request

TCP Data Transmission: Lost Packets



TCP Data Transmission: Reordering



Data: 1001-2000

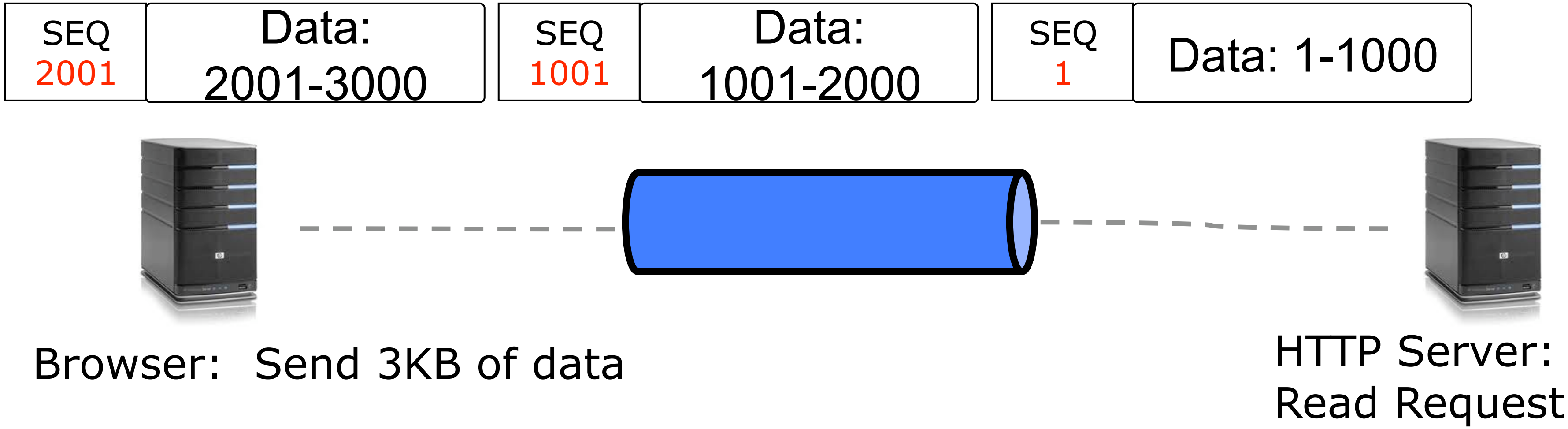
Data: 2001-3000

Data: 1-1000

Browser: Send 3KB of data

HTTP Server:
Read Request

Sequence Numbers and ACKs



SEQ 2001 | Data: 2001-3000

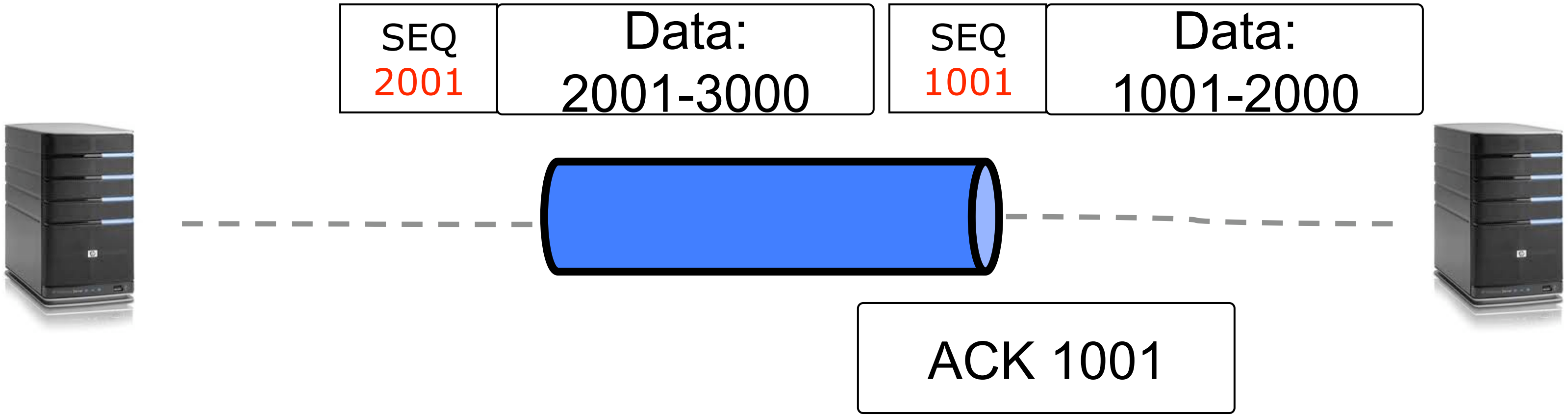
SEQ 1001 | Data: 1001-2000

SEQ 1 | Data: 1-1000

Browser: Send 3KB of data

HTTP Server:
Read Request

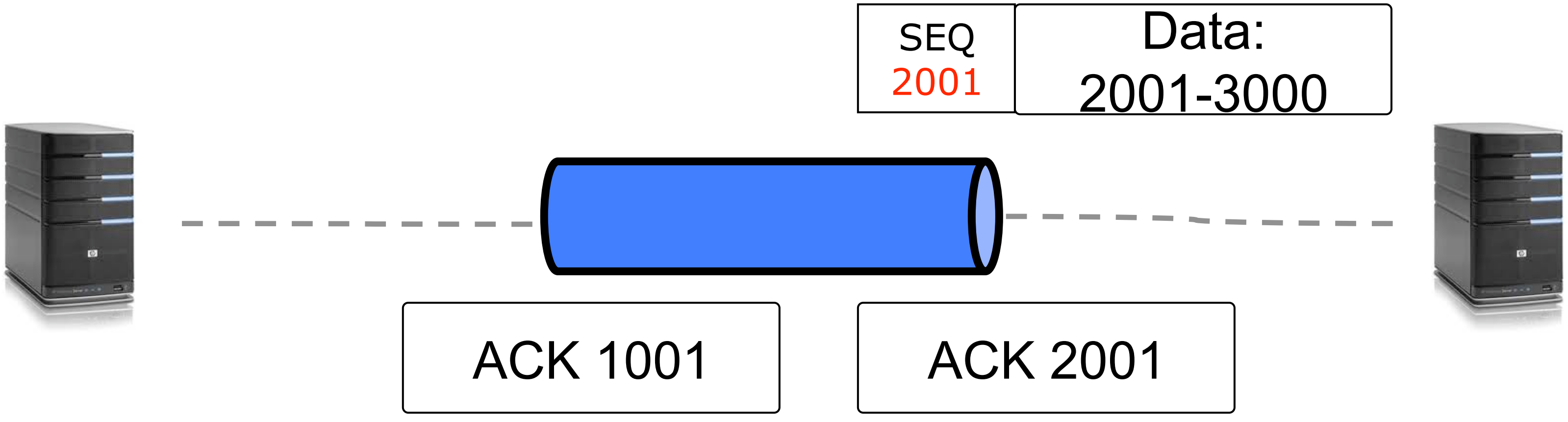
Sequence Numbers and ACKs



Browser: Send 3KB of data

HTTP Server:
Read Request

Sequence Numbers and ACKs



SEQ
2001

Data:
2001-3000

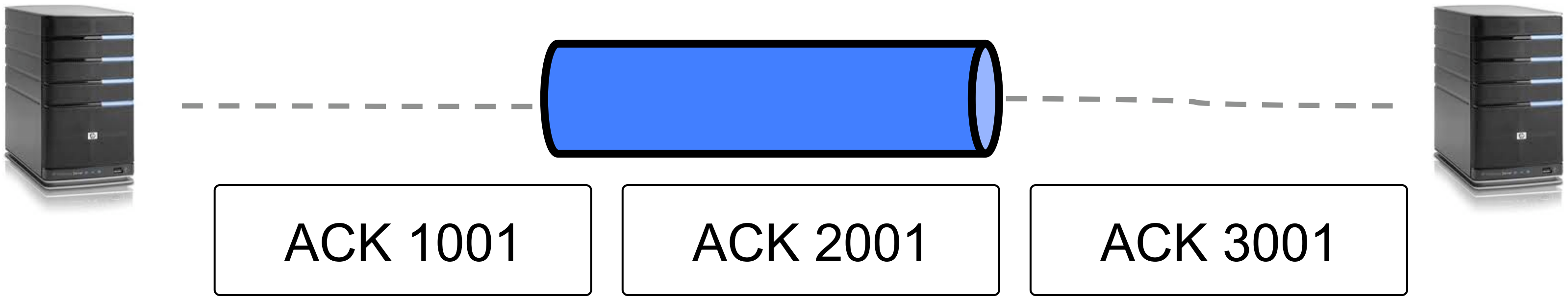
ACK 1001

ACK 2001

Browser: Send 3KB of data

HTTP Server:
Read Request

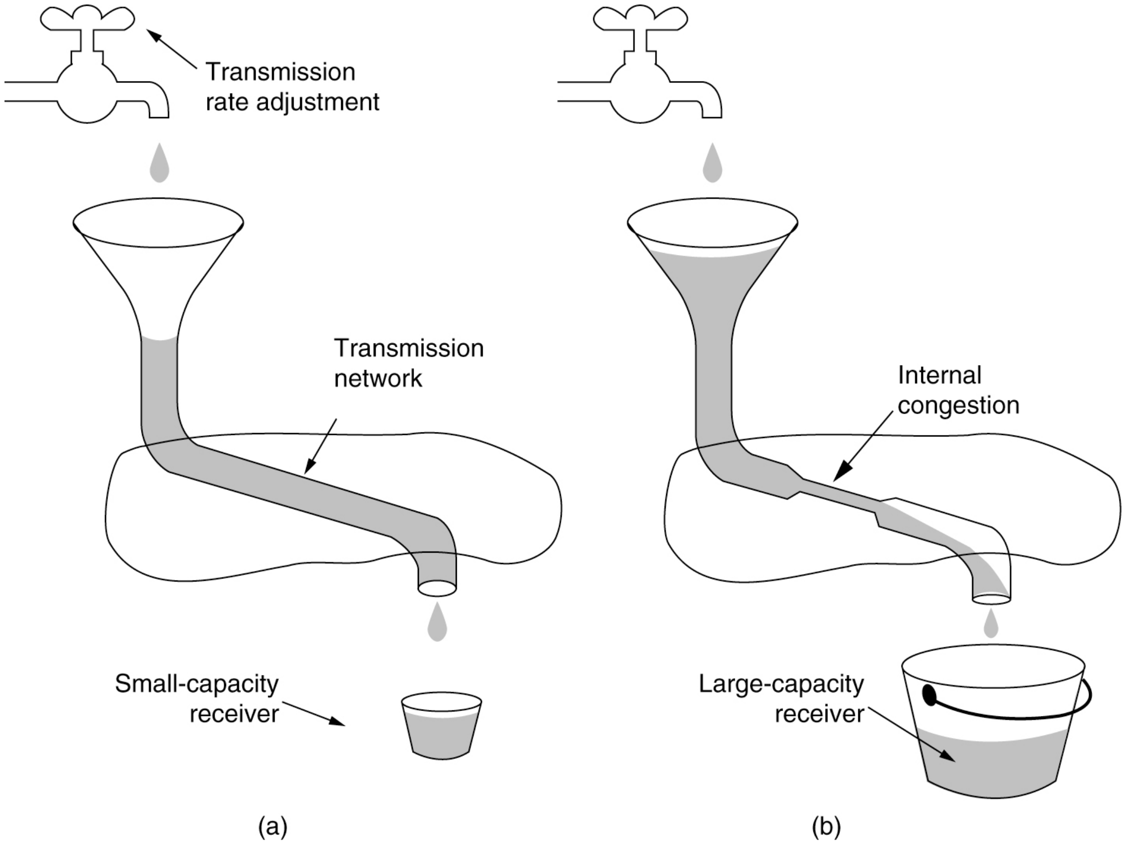
Sequence Numbers and ACKs



Browser: Send 3KB of data

HTTP Server:
Read Request

TCP congestion control



(a) Fast network feeds a slow receiver

(b) Slow network feeds a fast receiver.

What is the optimal window? …

… BDP = bandwidth delay product

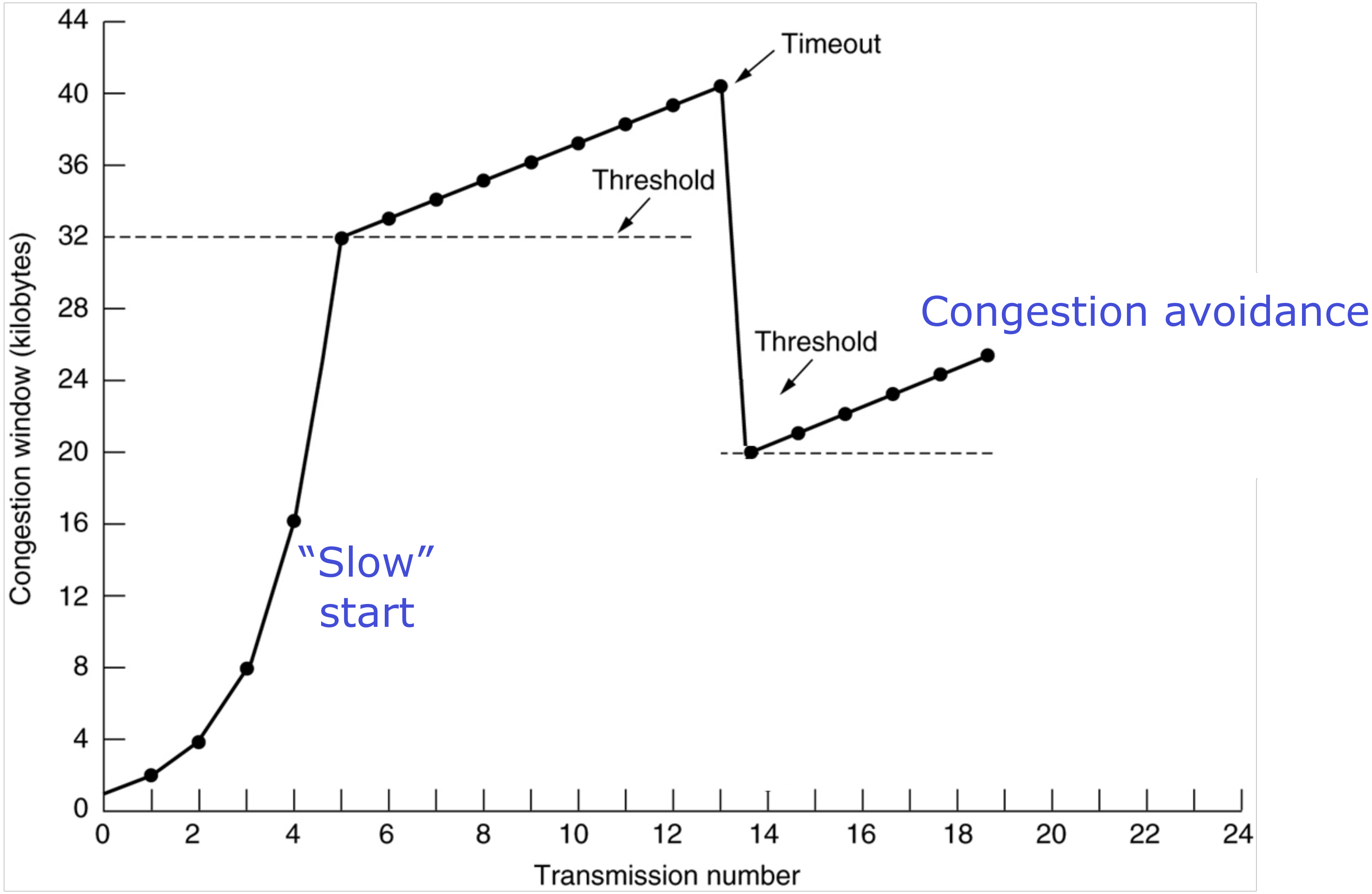
TCP congestion control

- Sender uses minimum of two windows
  - AW (advertised window) = receiver capacity, in each ACK segment
  - CW (congestion window) = network capacity, estimated by sender
  - CW crows in two phases
- “slow start”, actually exponential
  - Up to a threshold
- “congestion avoidance” (prevention), liniar increase
  - After threshold
- Threshold
  - Inițially 64K
  - Cut in half after a timeout
- fast recovery, fast retransmit

TCP slow start

- Used at the beginning, and after a timeout
- "slow" compared to selective repeat
- Used to discover capacity
  - Initially CW = 1 segment
  - After each ACK, CW increases by 1
  - Exponential!

TCP congestion control



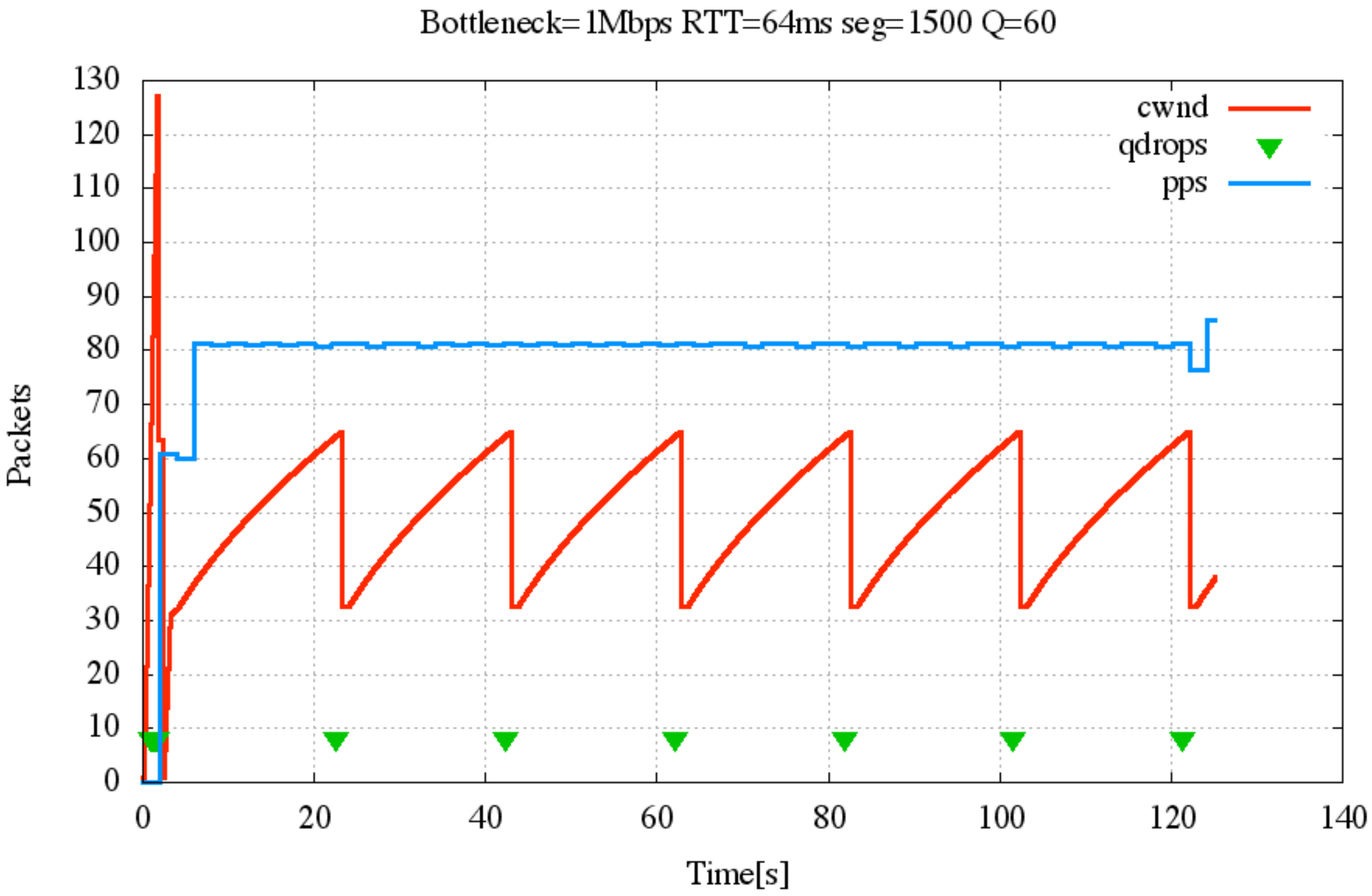
Example MSS=1024 bytes

TCP congestion avoidance

1. TCP slow start
   - Increment CW with each ACK
   - (exponențial)
2. TCP congestion avoidance
   - Increase CW with 1/CW at each ACK
   - Effect: grows with 1 segment per RTT, liniarly

ACK duplicate 3 times:

- CW = CW/2 (fast recovery)
- retransmit segment (fast retransmit)
- stay in congestion avoidance

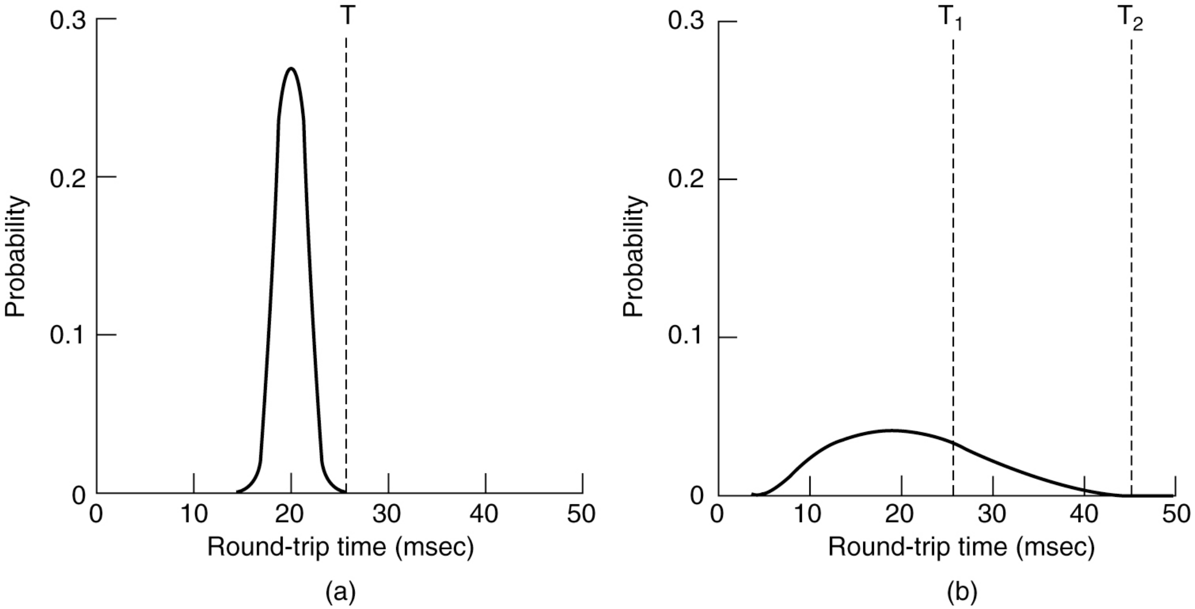


- CW oscillates around optimum
- Lost segment
    - repeat ACK
    - fast retransmit
- Throughput stays constant

TCP timers

Retransmission timer – started with each segment sent

- What is the expiration interval?
- RTT hard to estimate over internet



(a) ACK time distribution at layer 2

(b) ACK time distribution at layer 4

timer = T1 => useless retransmissions

timer = T2 => wait too much for most segments

TCP – dynamic timers

Jacobson: adjust RTT based on continuous measurements

- RTT = current estimate, M = last measurement
- $RTT = \alpha\, RTT + (1 - \alpha) M$, $\alpha = 7/8$

How long to wait?

- Estimate standard deviation of RTT (also called “jitter”)
- Estimate $D = \alpha\, D + (1 - \alpha)|RTT - M|$
- **Timeout= RTT + 4D**

TCP performance

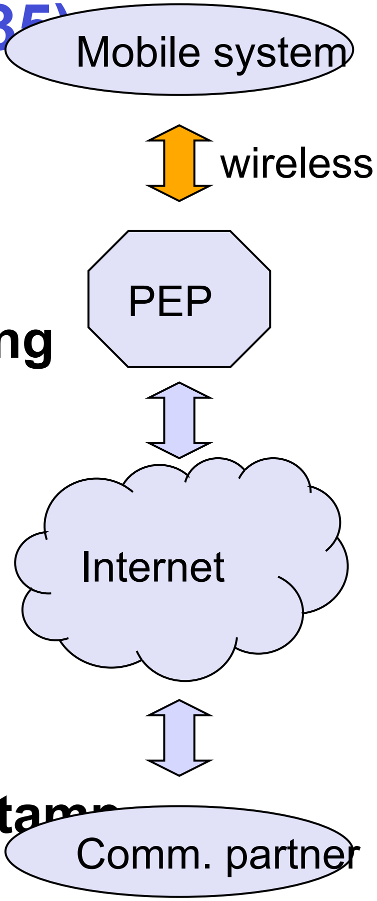
- How TCP reacts to
  - Loss
    - Covered by layer 2 (FEC, ARQ)? => delay
    - Uncovered? => fast retransmit, timeout :-(
  - Delay
    - Reduced throughput (high BDP)
    - timeout :-(
- TCP interprets loss as congestion

Design bug!

- “Solutions”
  - SACK = selective acknowledgments
  - ECN = Explicit congestion notification
  - PEP = Performance enhancing proxies

TCP “improvements”

- Performance enhancing proxies (PEP, RFC 3135)
  - Transport layer
    - Local retransmissions and acknowledgements
  - Additionally on the application layer
    - Content filtering, compression, picture downscaling
    - Web service gateways?
  - Big problem: breaks end-to-end semantics
    - Disables use of IP security
    - Choose between PEP and security!
- More open issues
  - RFC 3150 (slow links) header compression, no timestamp
  - RFC 3155 (links with errors)
    - States that explicit congestion notification cannot be used
  - In contrast to 2.5G/3G recommendations!



Mobility and transport

- Performance
  - TCP is averse to loss, delay

- Functionality
  - TCP is tied to IP

Mobile IP reminder

- IP address = location identifier & host identity
- MobileIP
  - Home Agent
  - Foreign Agent, Care of Address
  - tunneling, triangle routing
- MobileIP problems
  - Security – FA, HA authentication
  - Firewalls
    - needs reverse tunneling
  - NAT
- instead, would like to use multiple IPs….

MPTCP (multipath TCP)

- Why?
  - Most web servers are multi-homed
  - Smart phones have multiple wireless connections
  - Data centers have multiple paths between hosts

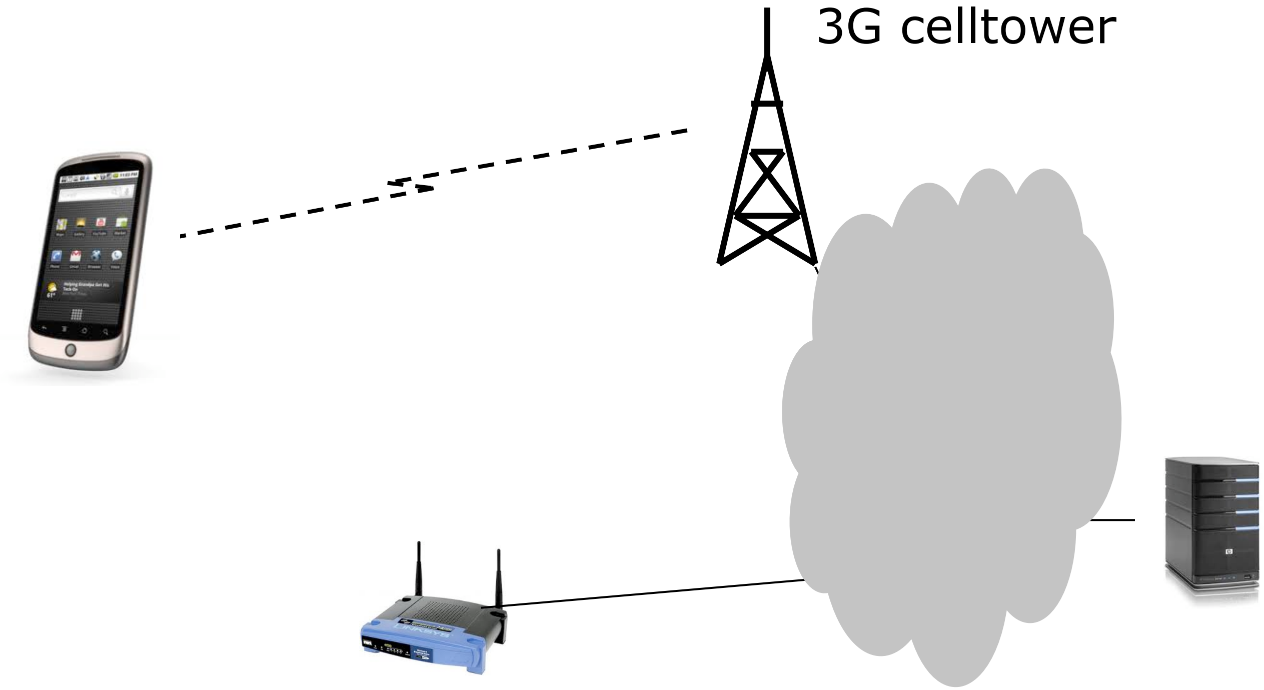
- Advanced adoption stage (2015)
  - RFC 6824
  - Linux kernel patch
  - Android kernel
  - Apple SIRI

- Costin Raiciu
  UPB

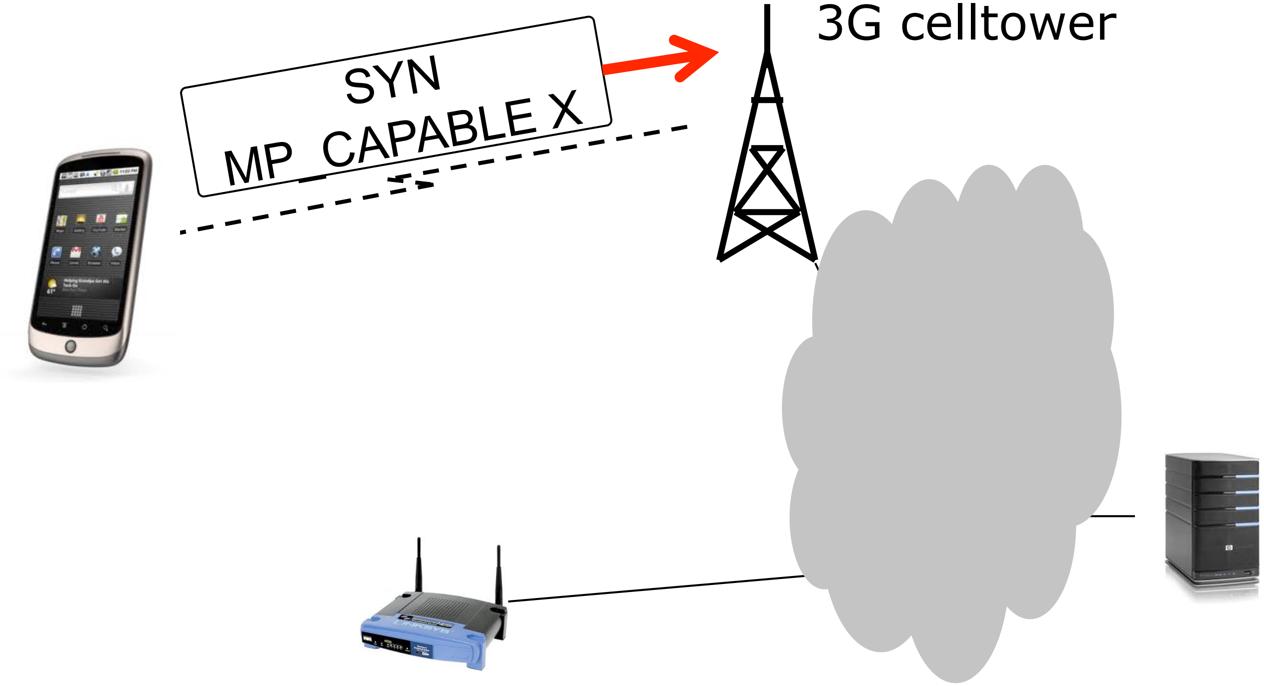
Octavian Purdilă
Intel/UPB

4/27/15

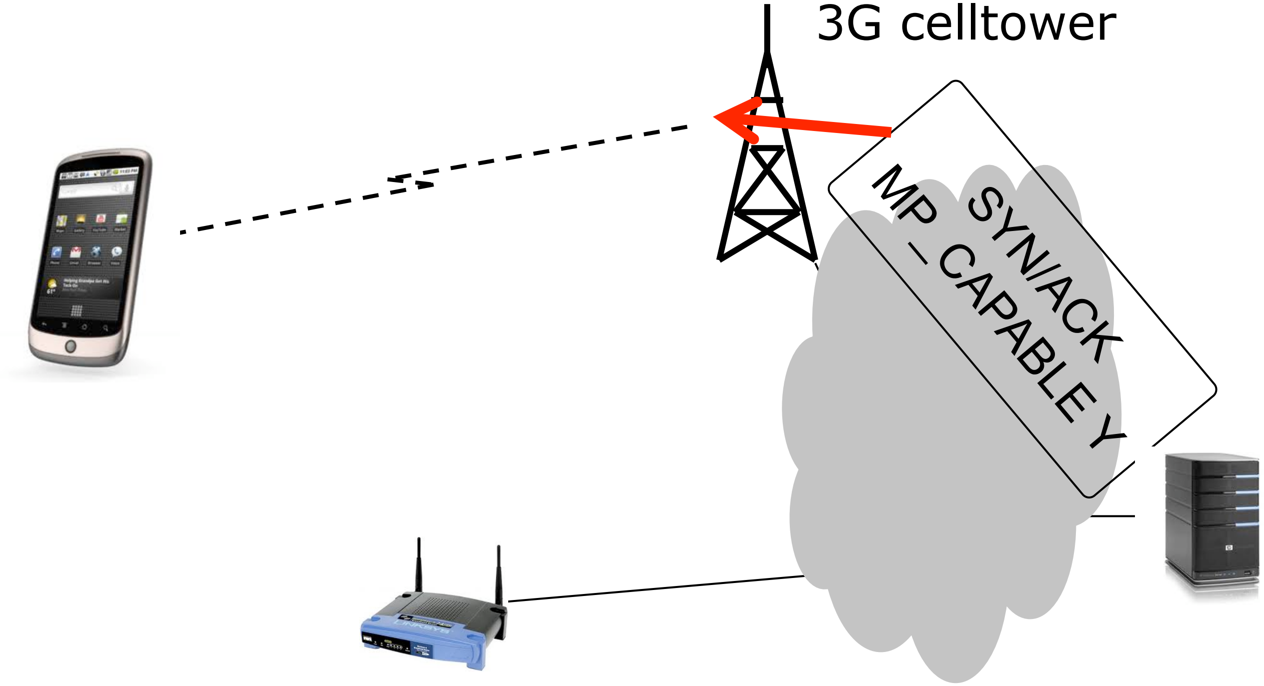
MPTCP Mobile Architecture



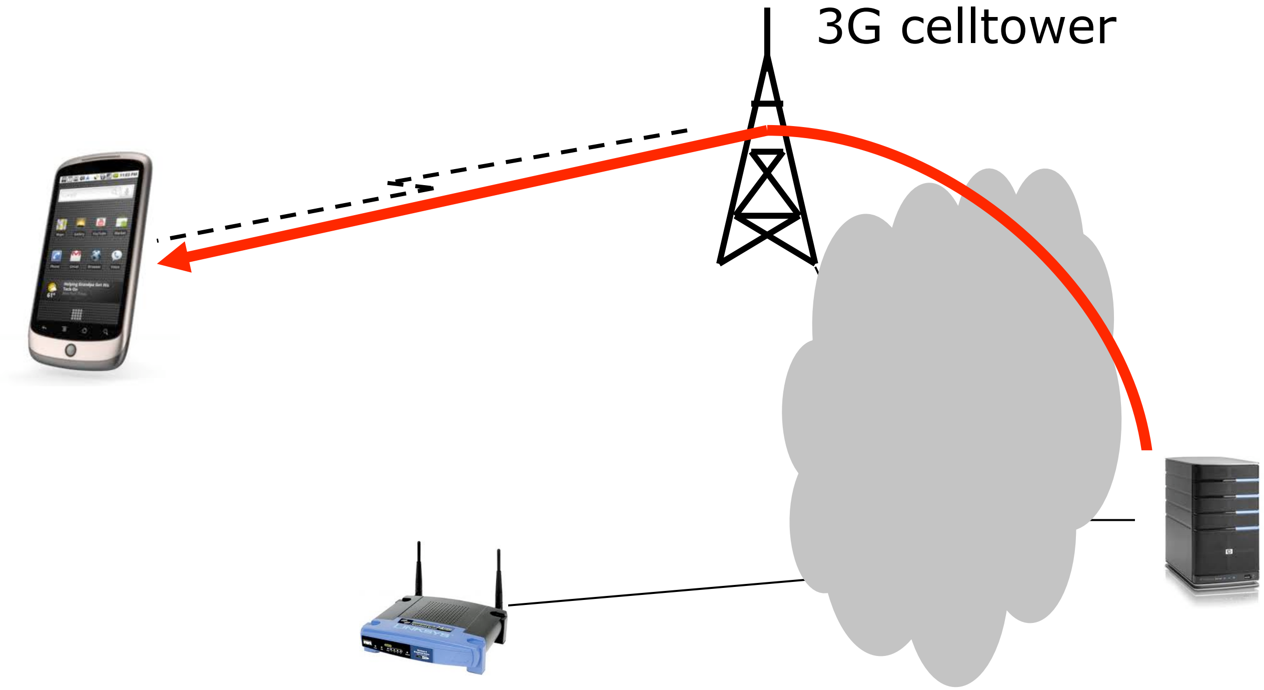
MPTCP Mobile Architecture



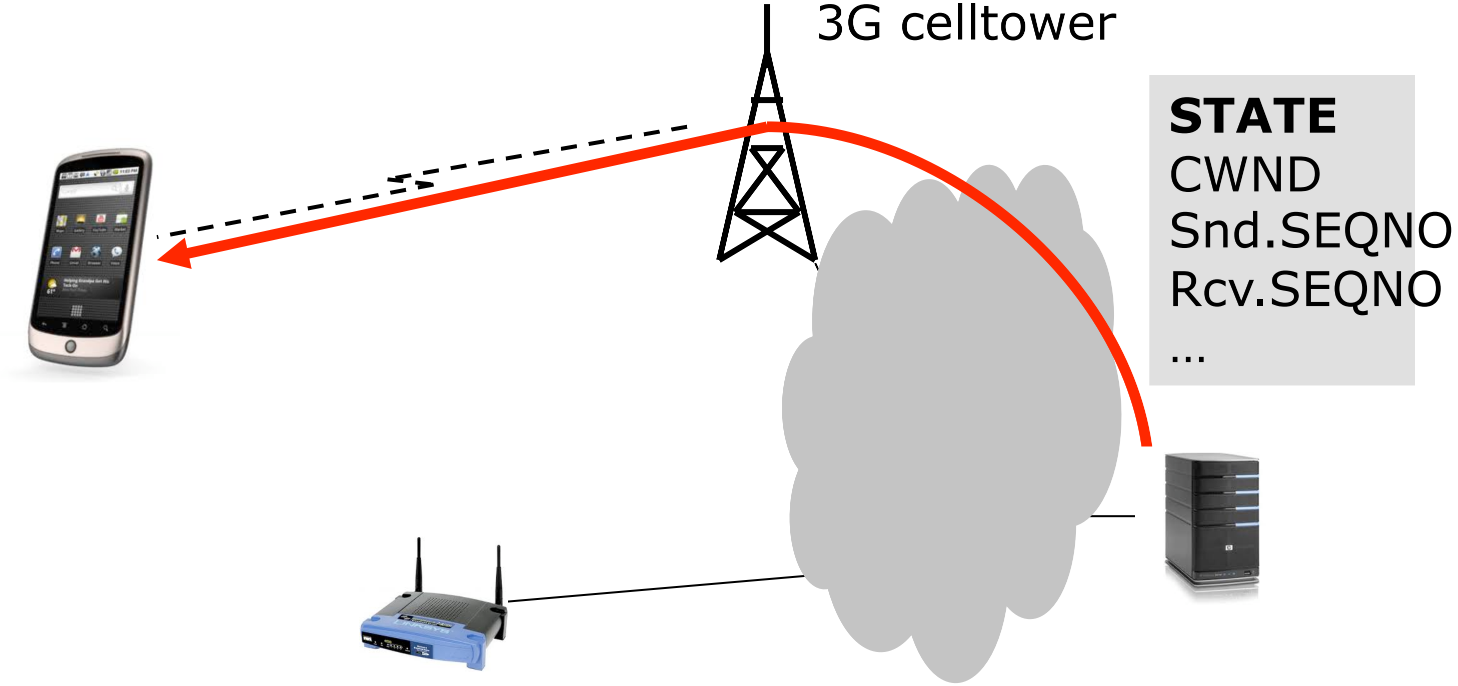
MPTCP Mobile Architecture



MPTCP Mobile Architecture

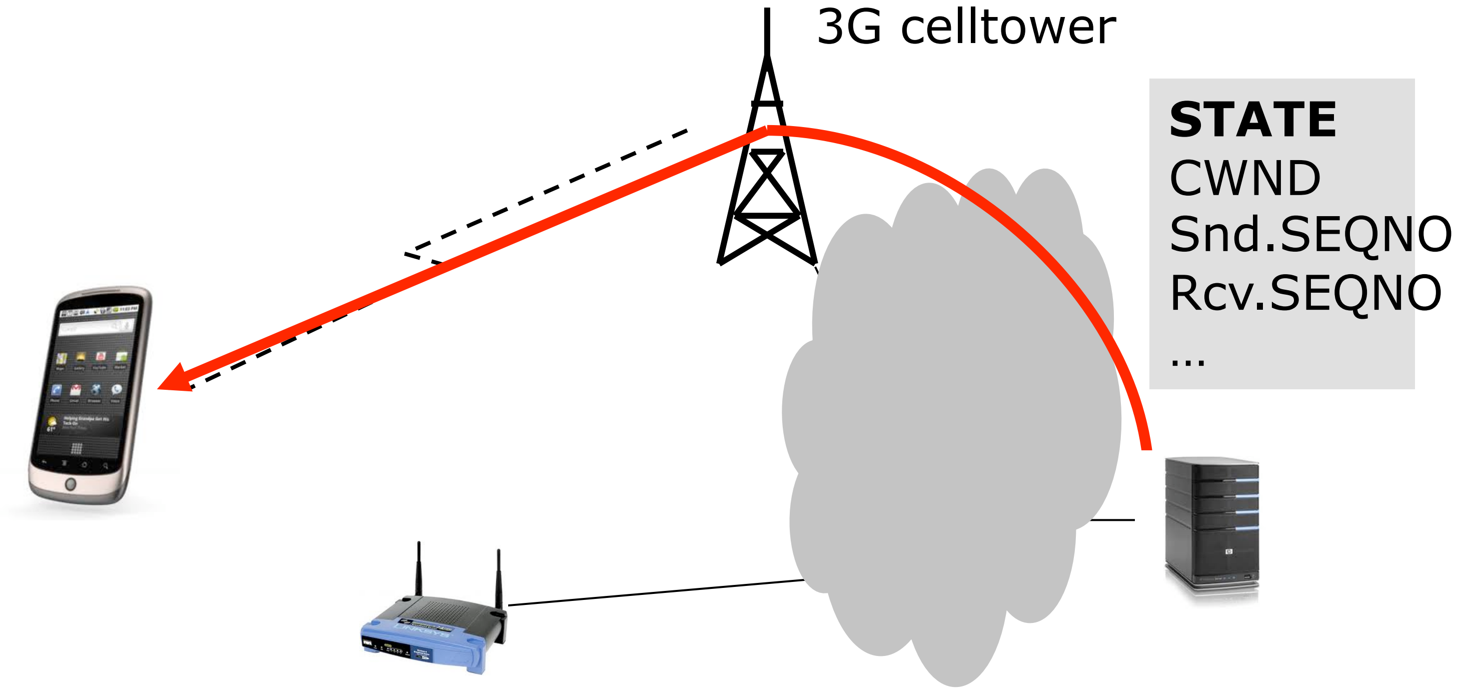


MPTCP Mobile Architecture



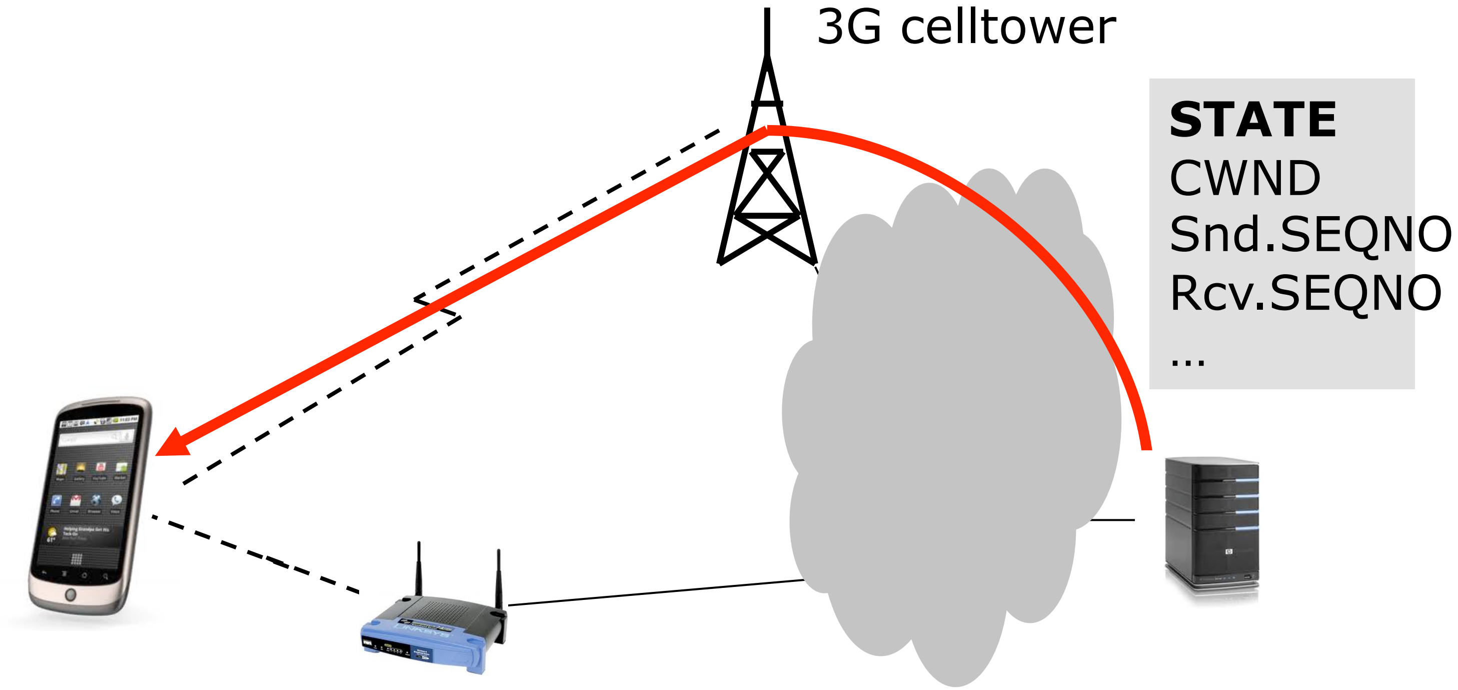
STATE
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture



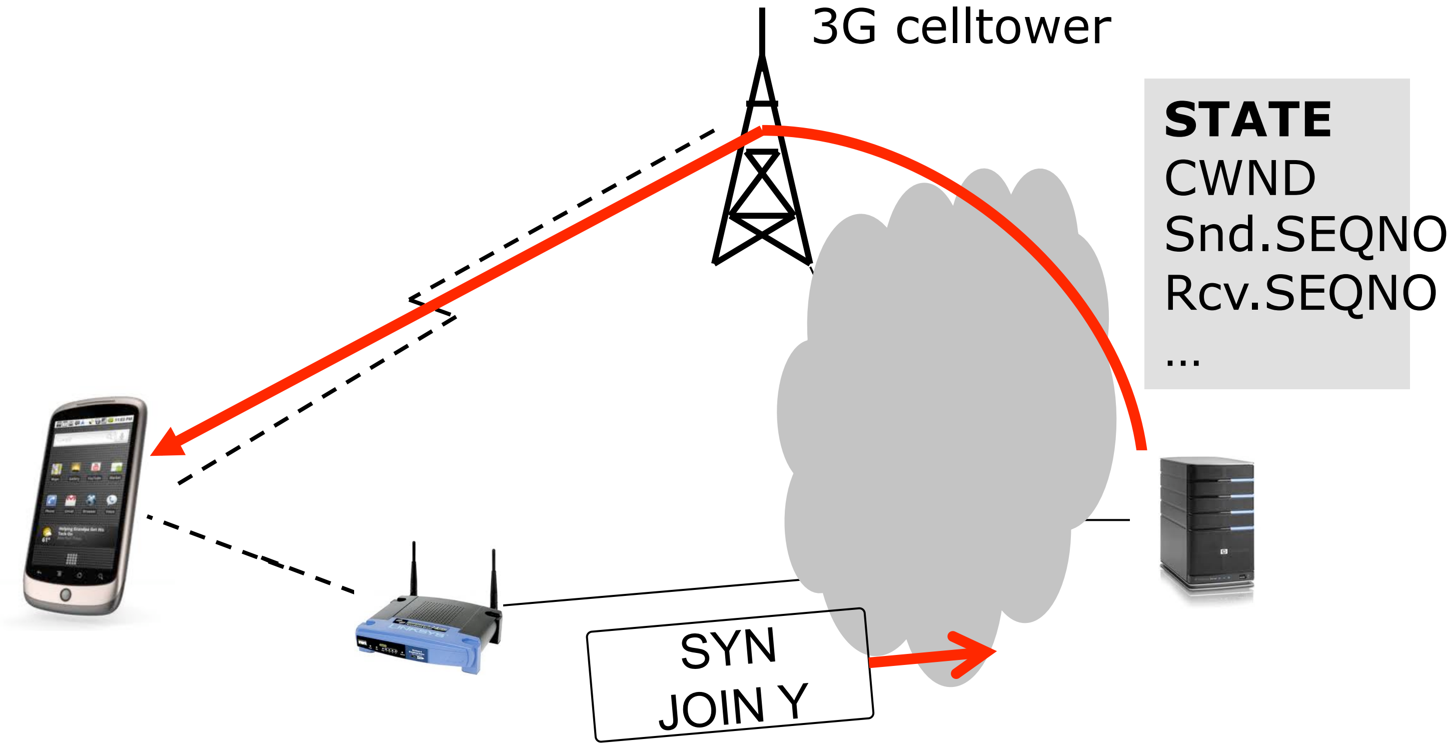
STATE
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture



STATE
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture

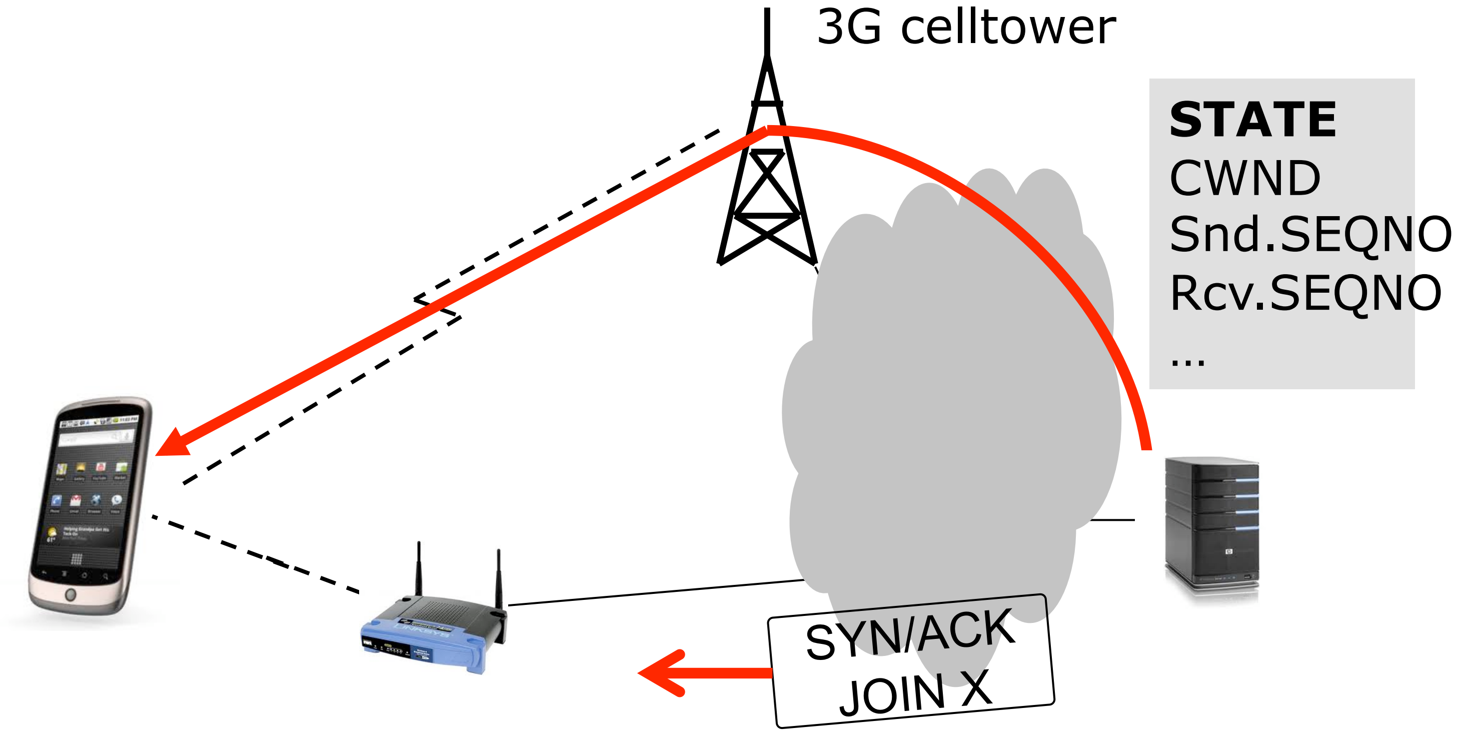


3G celltower

STATE
CWND
Snd.SEQNO
Rcv.SEQNO
...

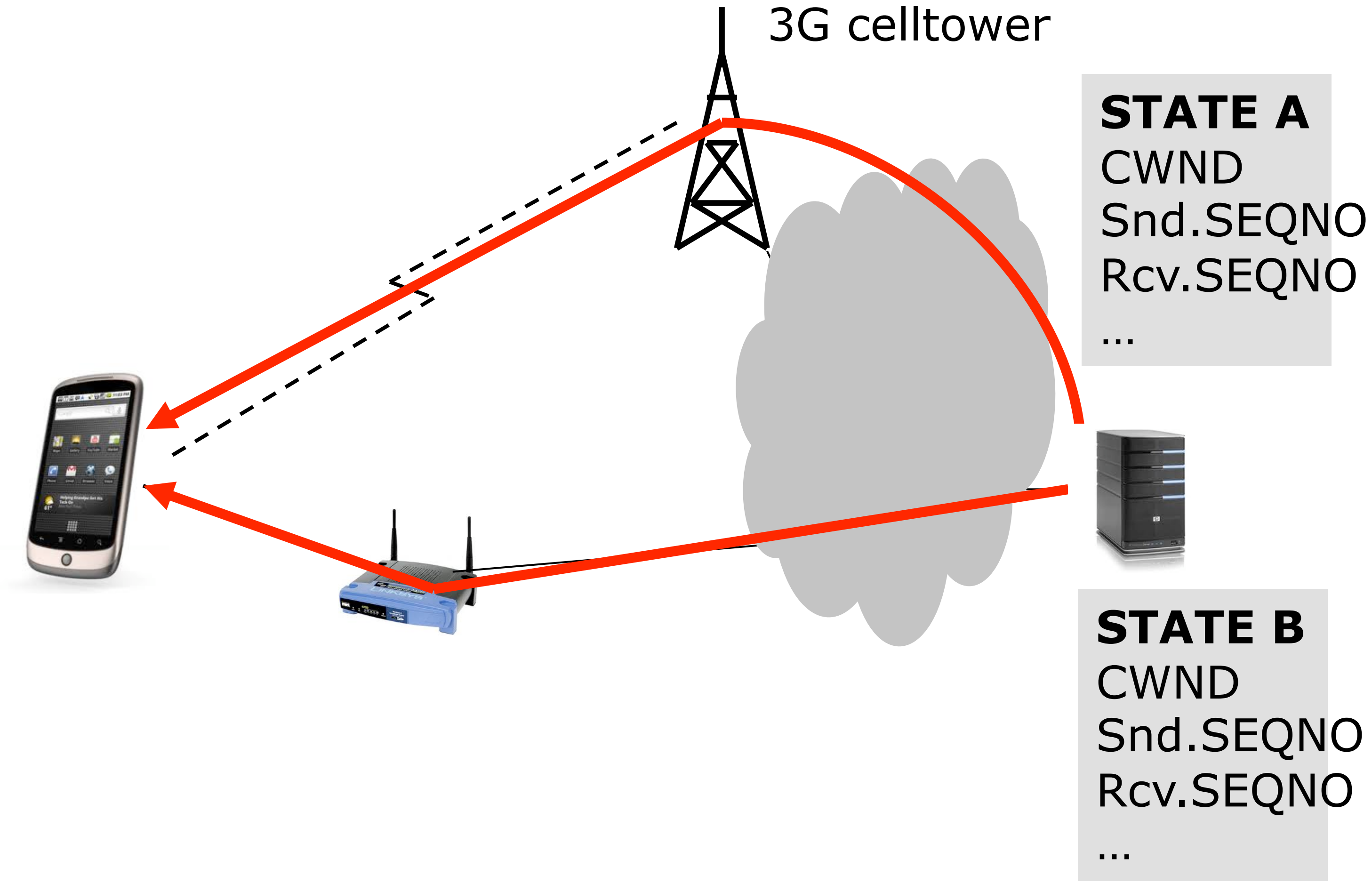
SYN
JOIN Y

MPTCP Mobile Architecture



STATE
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture

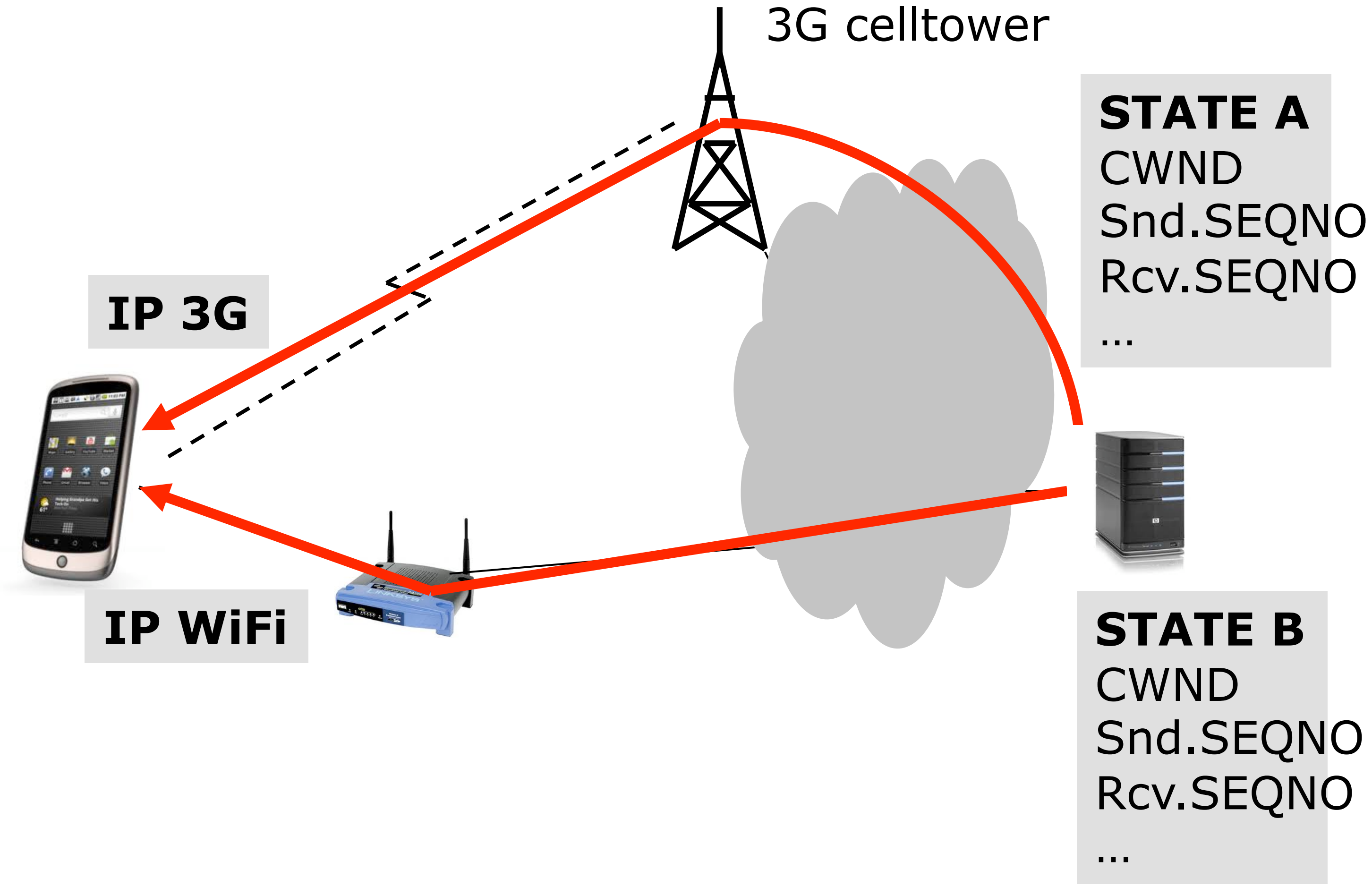


3G celltower

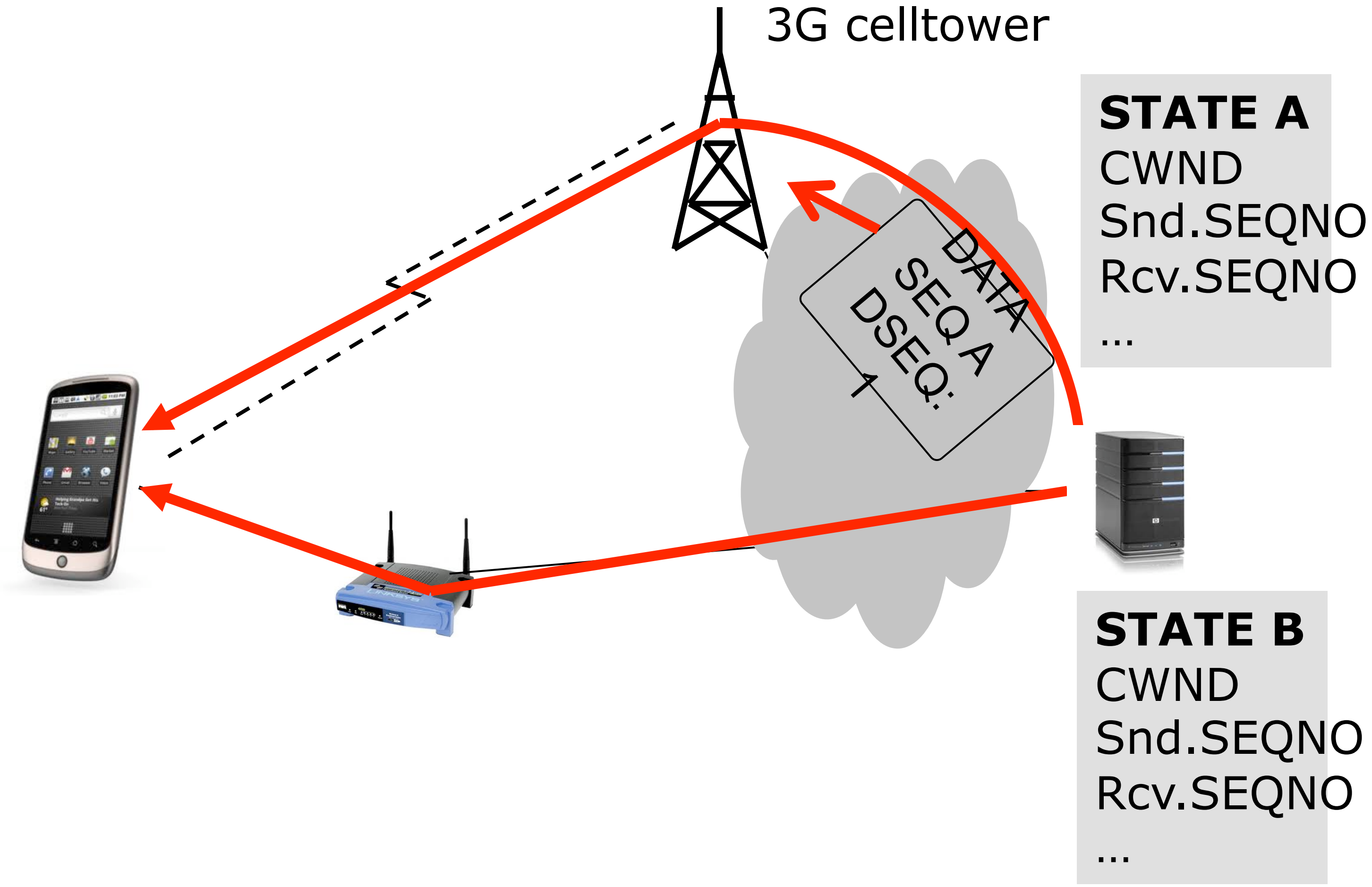
STATE A
CWND
Snd.SEQNO
Rcv.SEQNO
…

STATE B
CWND
Snd.SEQNO
Rcv.SEQNO
…

MPTCP Mobile Architecture



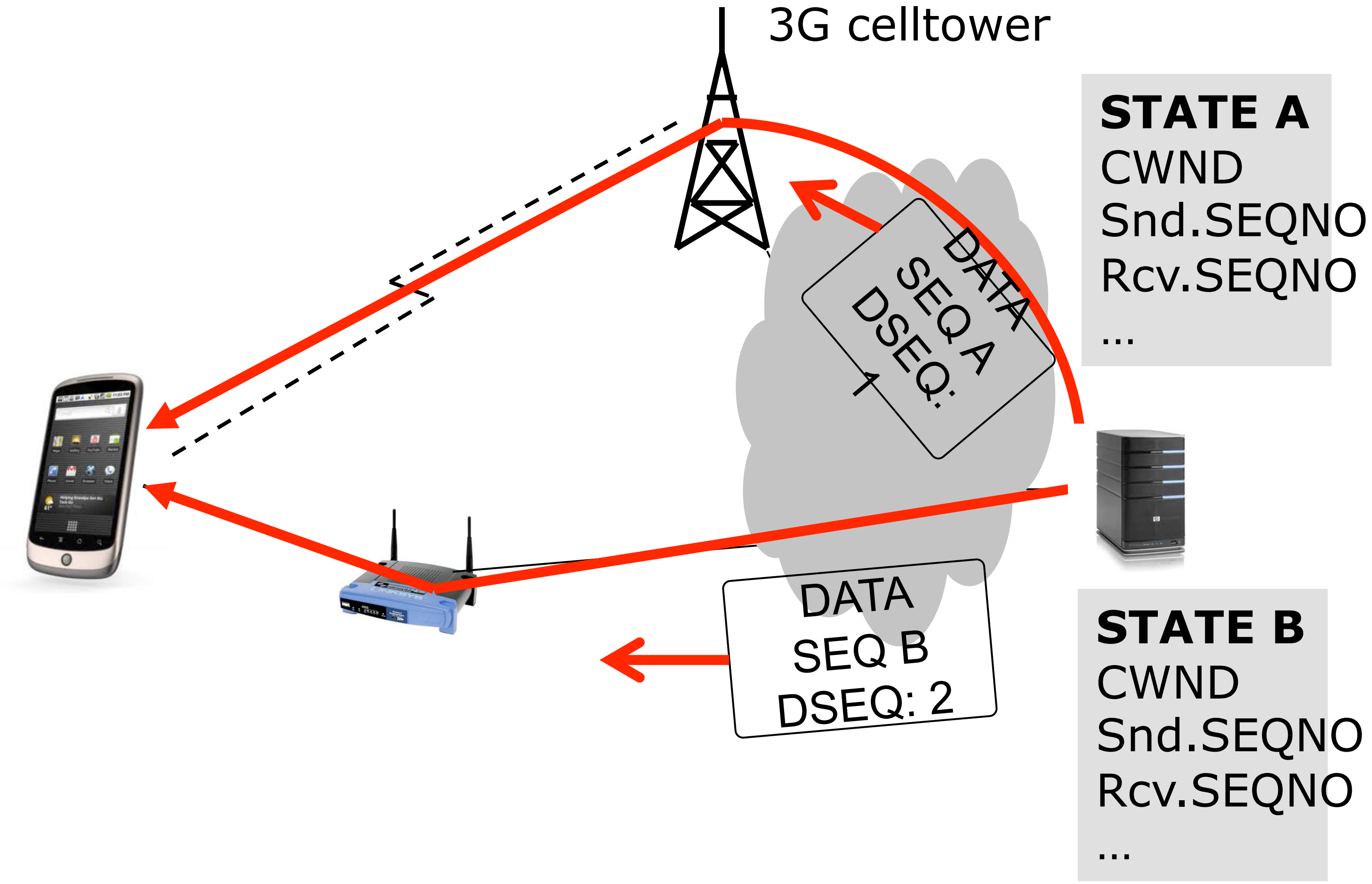
MPTCP Mobile Architecture



STATE A
CWND
Snd.SEQNO
Rcv.SEQNO
...

STATE B
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture



3G celltower

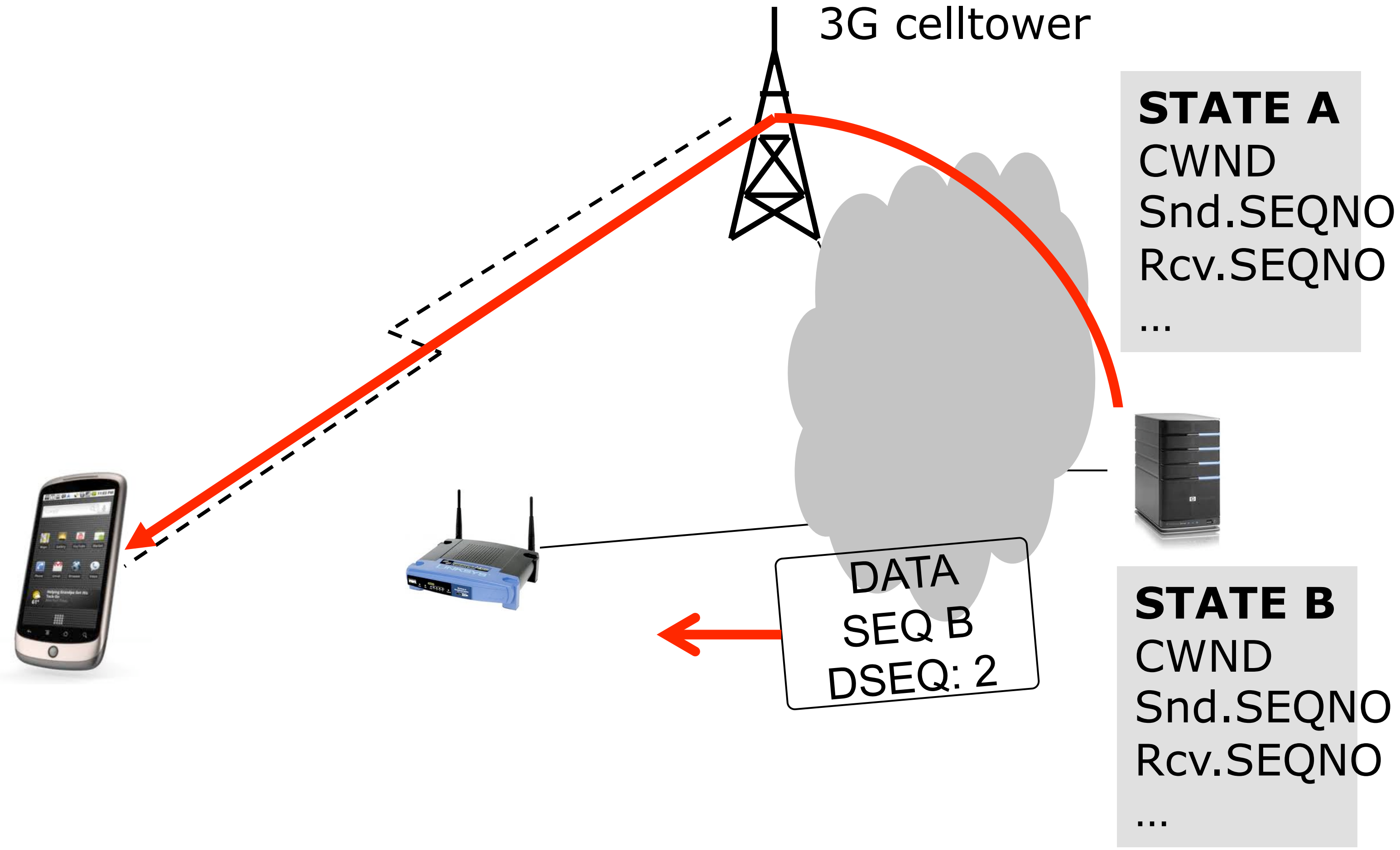
STATE A
CWND
Snd.SEQNO
Rcv.SEQNO
...

DATA
SEQ A
DSEQ: 1

DATA
SEQ B
DSEQ: 2

STATE B
CWND
Snd.SEQNO
Rcv.SEQNO
...

MPTCP Mobile Architecture



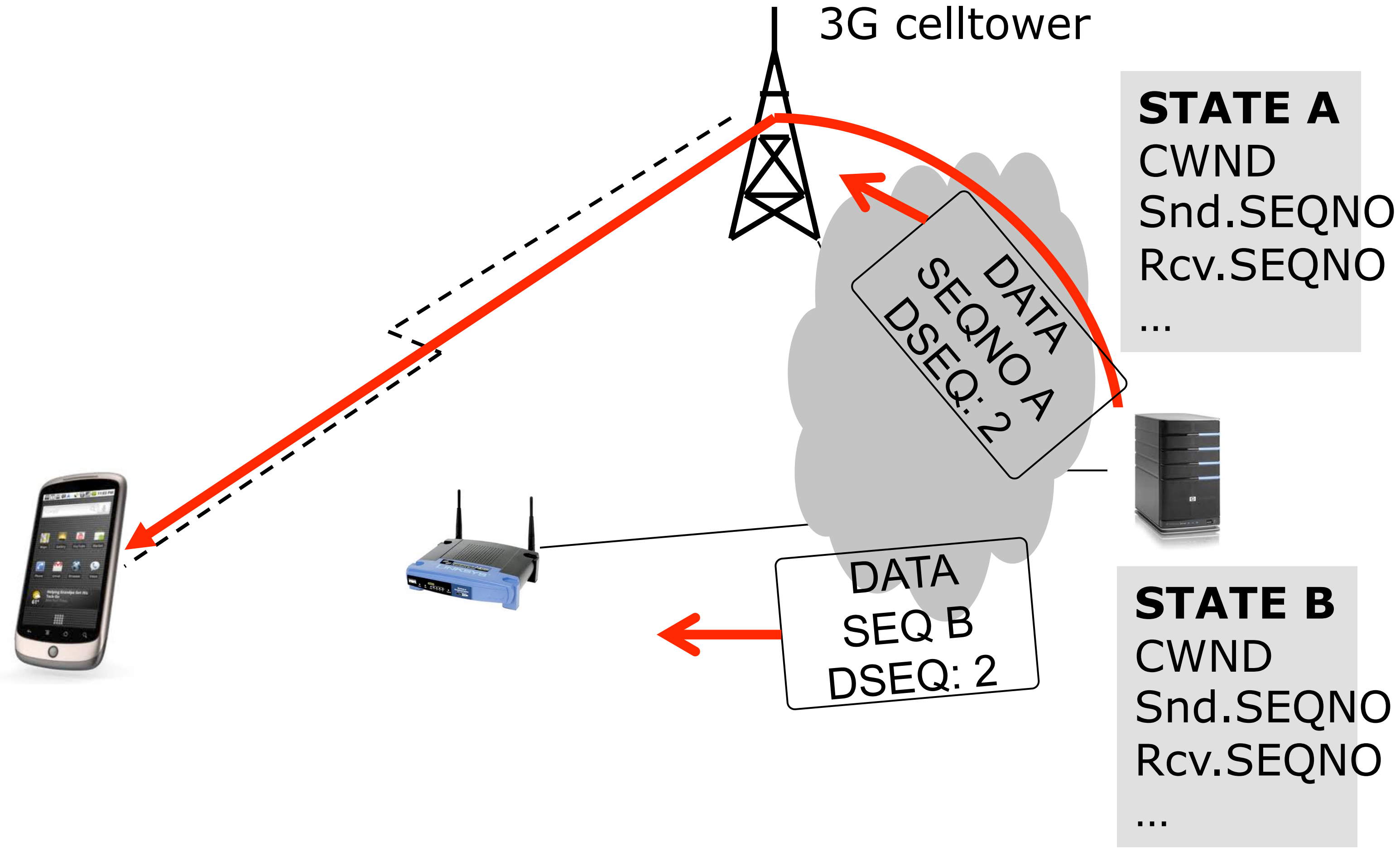
3G celltower

STATE A
CWND
Snd.SEQNO
Rcv.SEQNO
...

DATA
SEQ B
DSEQ: 2

STATE B
CWND
Snd.SEQNO
Rcv.SEQNO
...

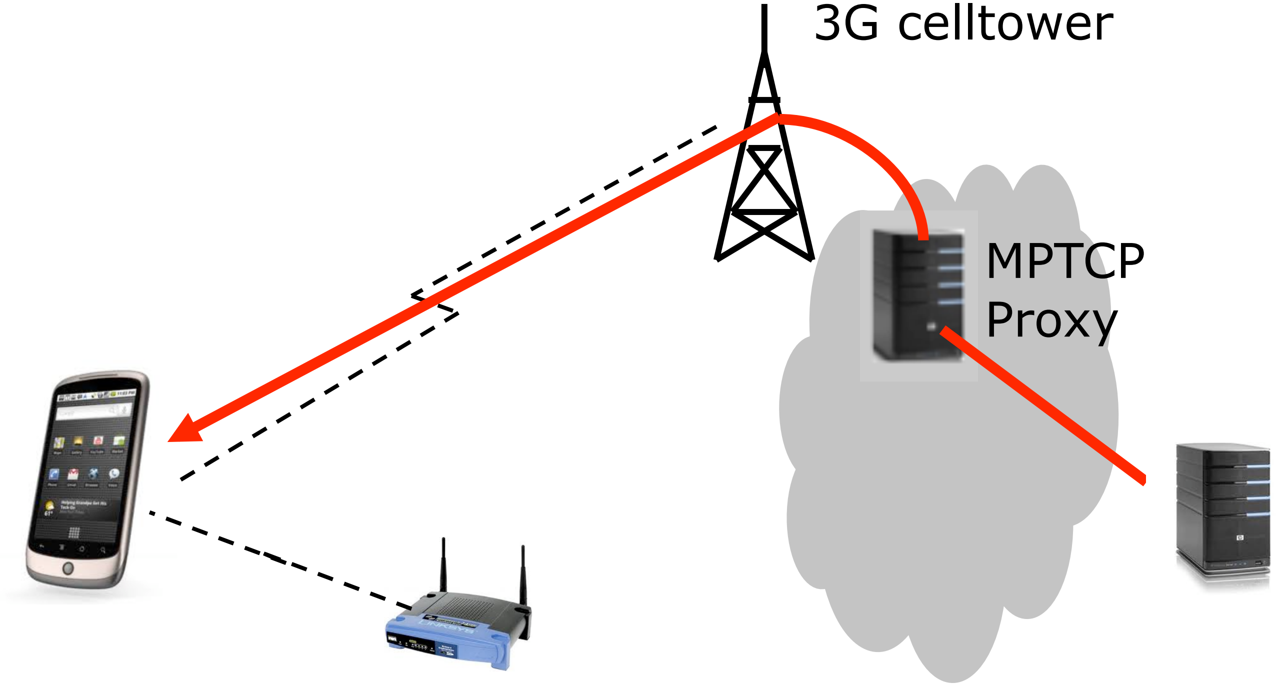
MPTCP Mobile Architecture



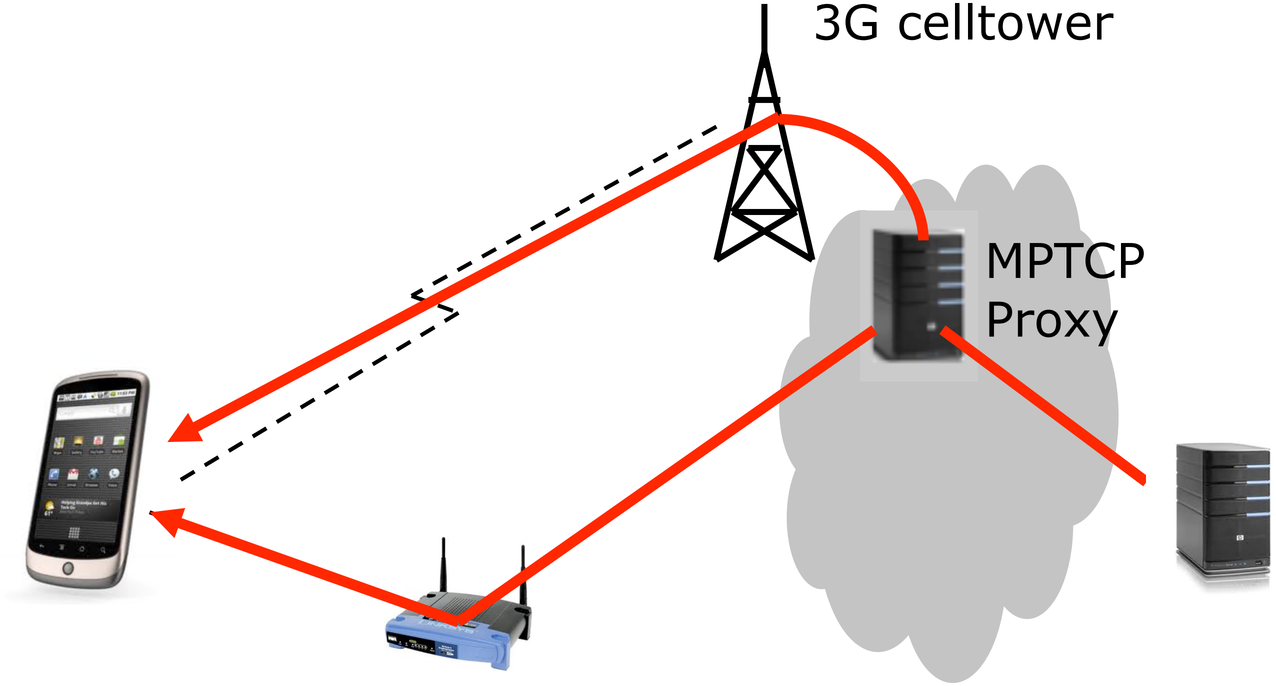
What if the remote server is not MPTCP enabled?

- This will be the case for initial deployment
- Solution: use an MPTCP proxy
  - Will be deployed by the mobile operator
  - Phones will be configured with the proxy's address via DHCP
- The proxy can also help with:
  - Simultaneous move
  - Peer to peer operation

Proxy: Server Does Not Speak MPTCP



Proxy: Server Does Not Speak MPTCP



Middleboxes: problems, problems, problems...

- TCP offload engines may re-segment TCP packets, replicating options on all segments.
- Firewalls may drop packets with options.
- Firewalls may remove options from packets.
- Proxies may ack data before it’s received by receiver.
- Proxies may report their window, not the receiver’s.
- Proxies/NATs may rewrite/extend/shrink payload and fix up sequence numbers accordingly.
- Normalizers may ensure retransmissions are consistent with the original data.
- Firewalls may rewrite sequence numbers in packets.

MPTCP is deployable

- Unmodified apps and network
  - Socket API not changed
- Protocol works at least as well as regular TCP
- Always works when a regular TCP would work
  - Falls back to TCP when path/endpoint not MPTCP capable
- Plays nicely with all the strange middleboxes out there
- Allows transport layer mobility!

Summary

- Network layer mobility
  - IP address = Identity & location ☹
  - Mobile IP, triangle routing, NAT ☹

- Transport layer mobility
  - TCP endpoint = IP address ☹
  - TCP sensitive to loss, delay ☹
  - Performance enhancing proxies 😐
  - MPTCP ☺