TCP

Tahoe, Reno, NewReno,
SACK, and Vegas

cwnd: congestion window
swnd: usable sending window
rwnd: advertised receiver's window
ssthresh: slow-start threshold

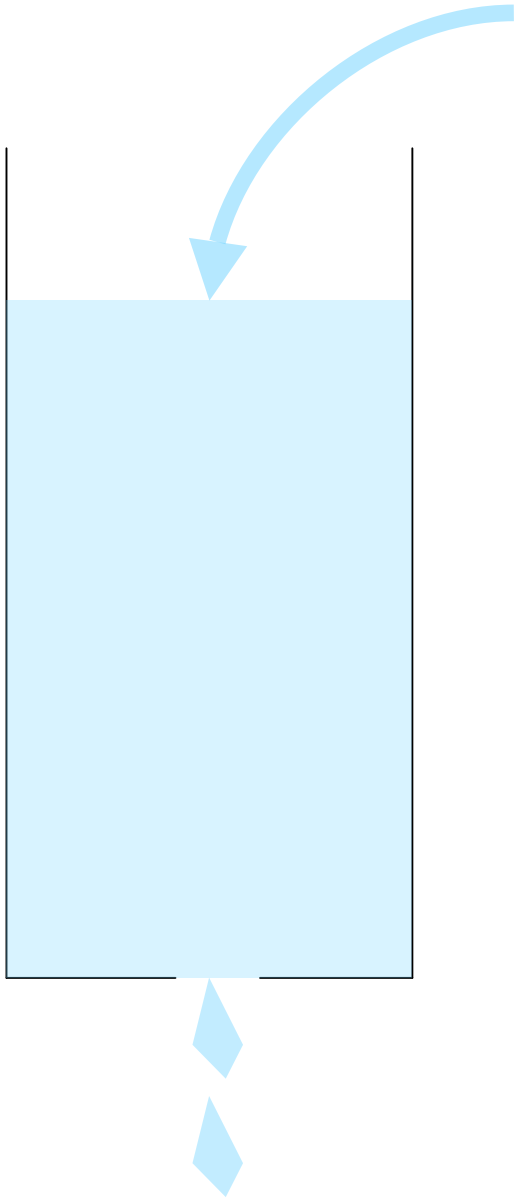
RFC793

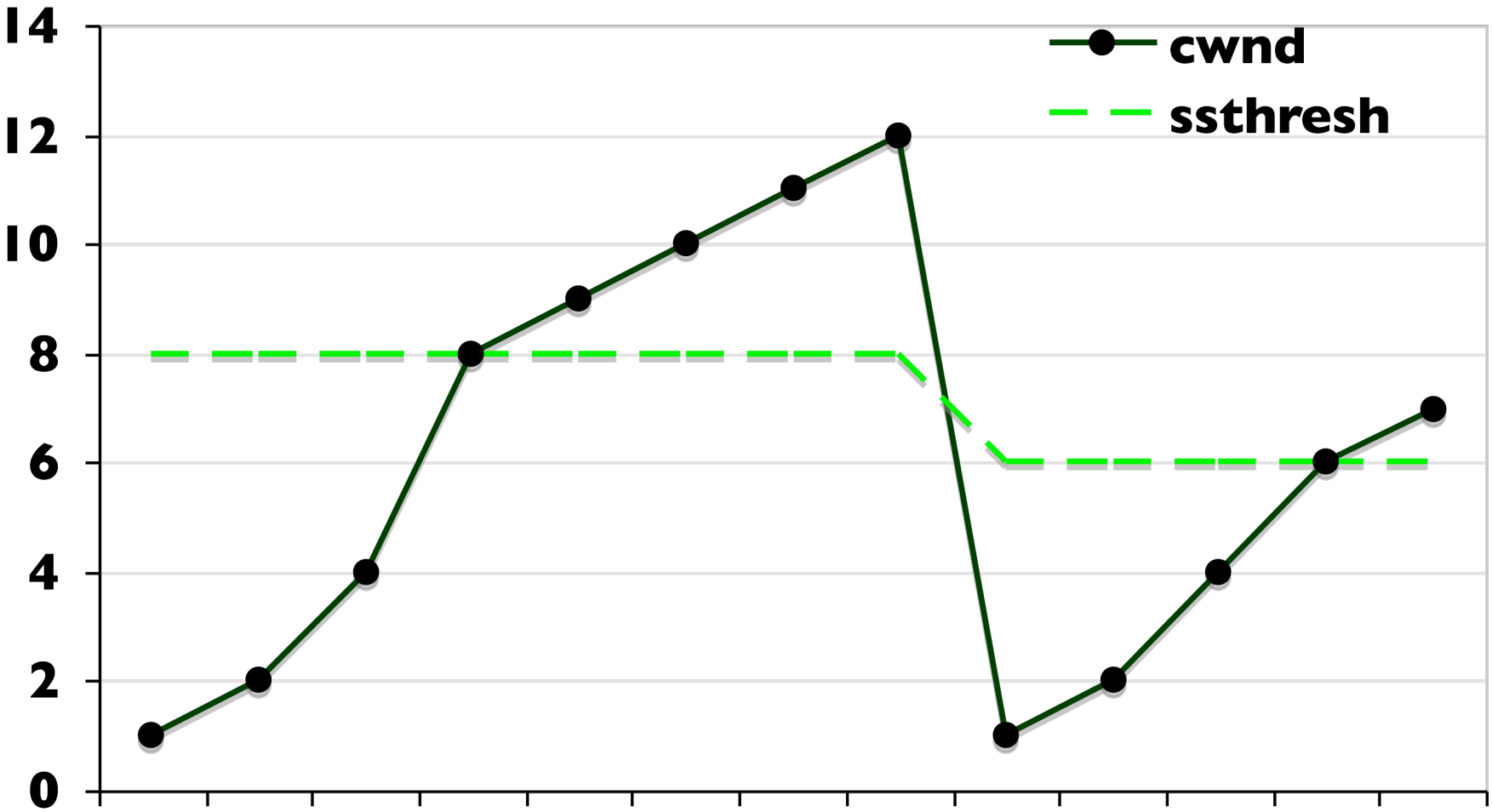
No cwnd

On timeout: retransmit

swnd = rwnd

TCP Tahoe




14
12
10
8
6
4
2
0
cwnd
ssthresh

new ack:

```
if (cwnd < sstresh)
    cwnd += 1
else
    cwnd += 1/cwnd
```

timeout/3rd dup ack:

retransmit all unacked

ssthresh = cwnd/2

cwnd = 1

Improving TCP Tahoe:

Packets still getting through in dup ack -- no need to reset the clock!

TCP Reno

new ack:

```
if (cwnd < sstresh)
    cwnd += 1
else
    cwnd += 1/cwnd
```

timeout:

retransmit 1st unacked

ssthresh = cwnd/2

cwnd = 1

3rd dup ack:

retransmit 1st unacked

ssthresh = cwnd/2

cwnd = cwnd/2 + 3

Fast Recovery:

the pipe is still almost full -- no need to restart

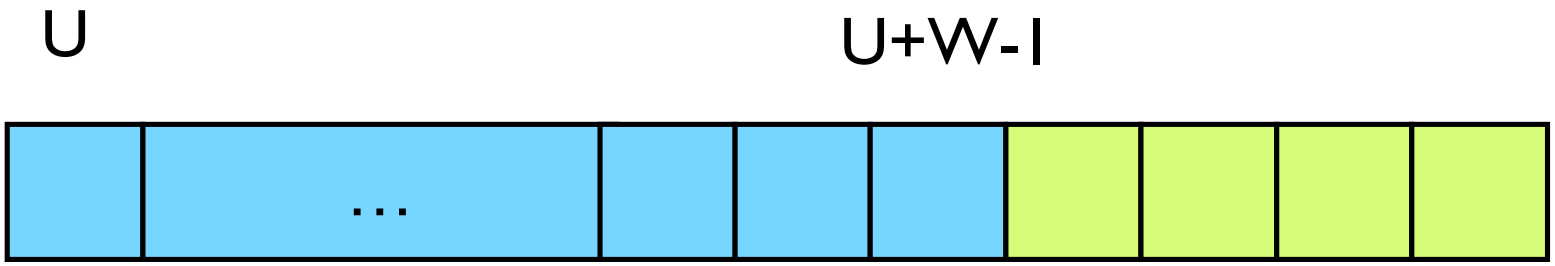
subsequent dup ack:

cwnd++

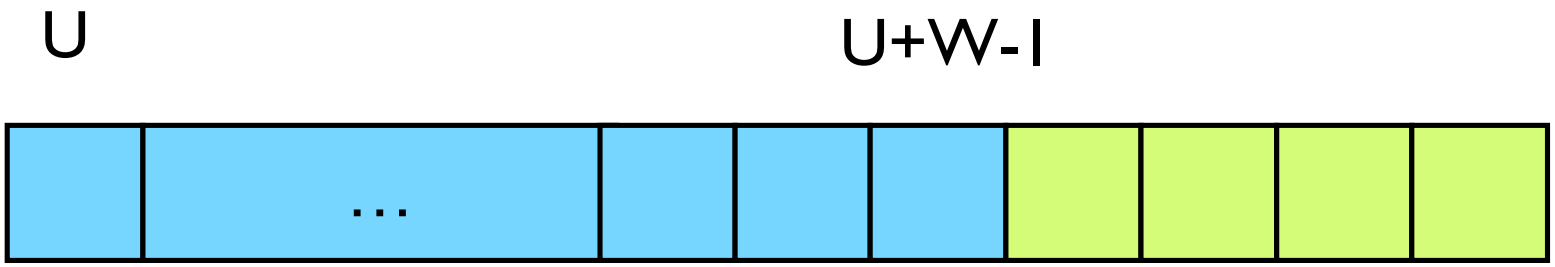
new ack:

cwnd = ssthresh

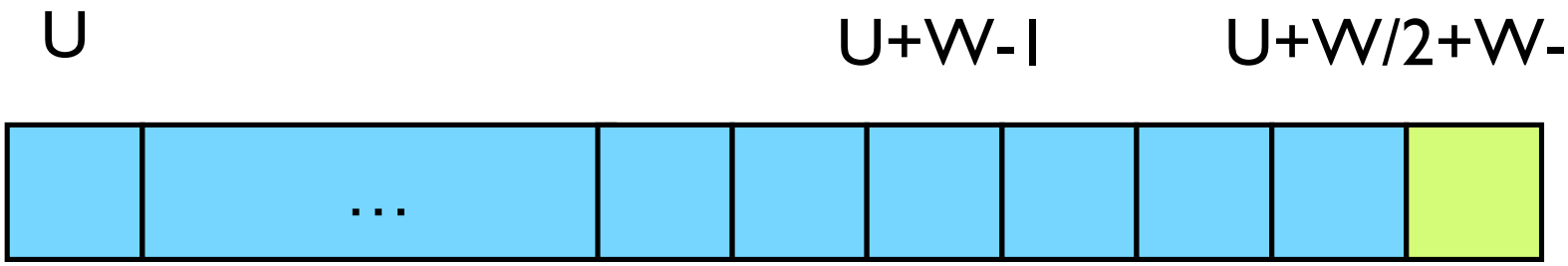
Suppose U is lost (oldest unacked) and all other packets are not. At time t, cwnd is W, and packets [U, U+W-1] are in the pipe.



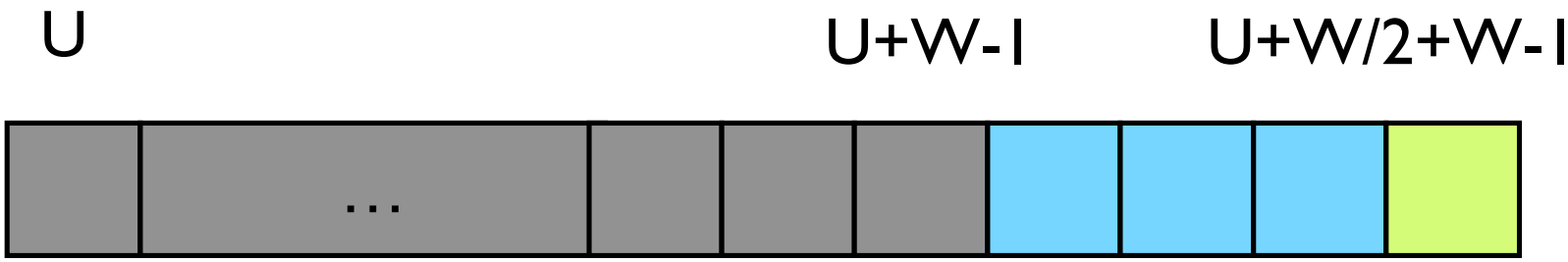
Between time t and t+RTT, we would have retransmitted U and received W-1 duplicate ACK.

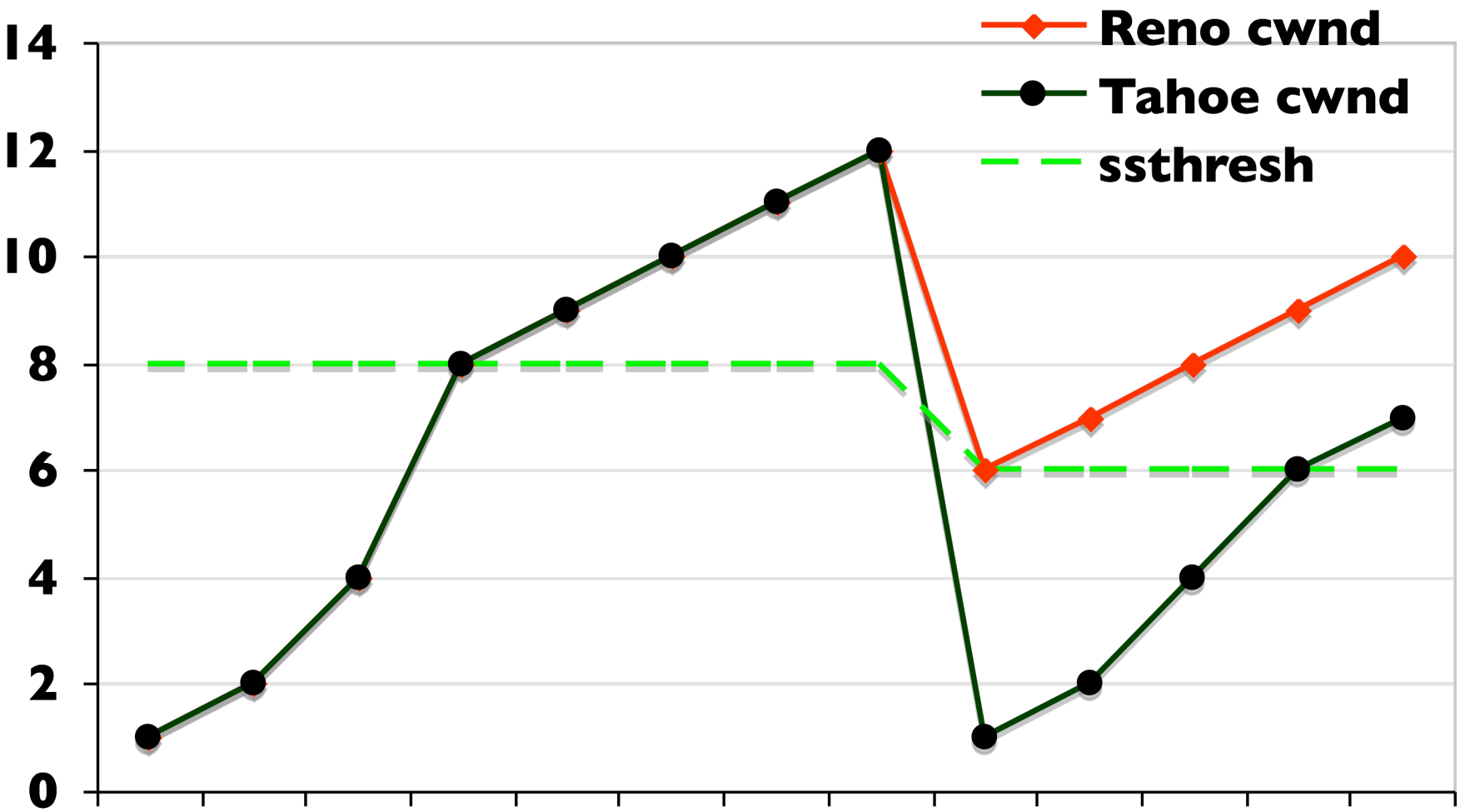


Between time t and t+RTT, the cwnd becomes W/2 + W-1. So we get to send W/2 new packets during the time. (Soon cwnd is going to become W/2 anyway..)

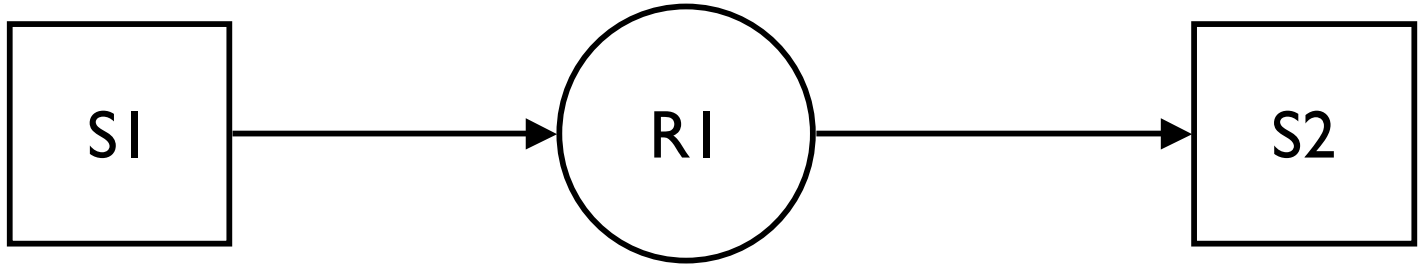


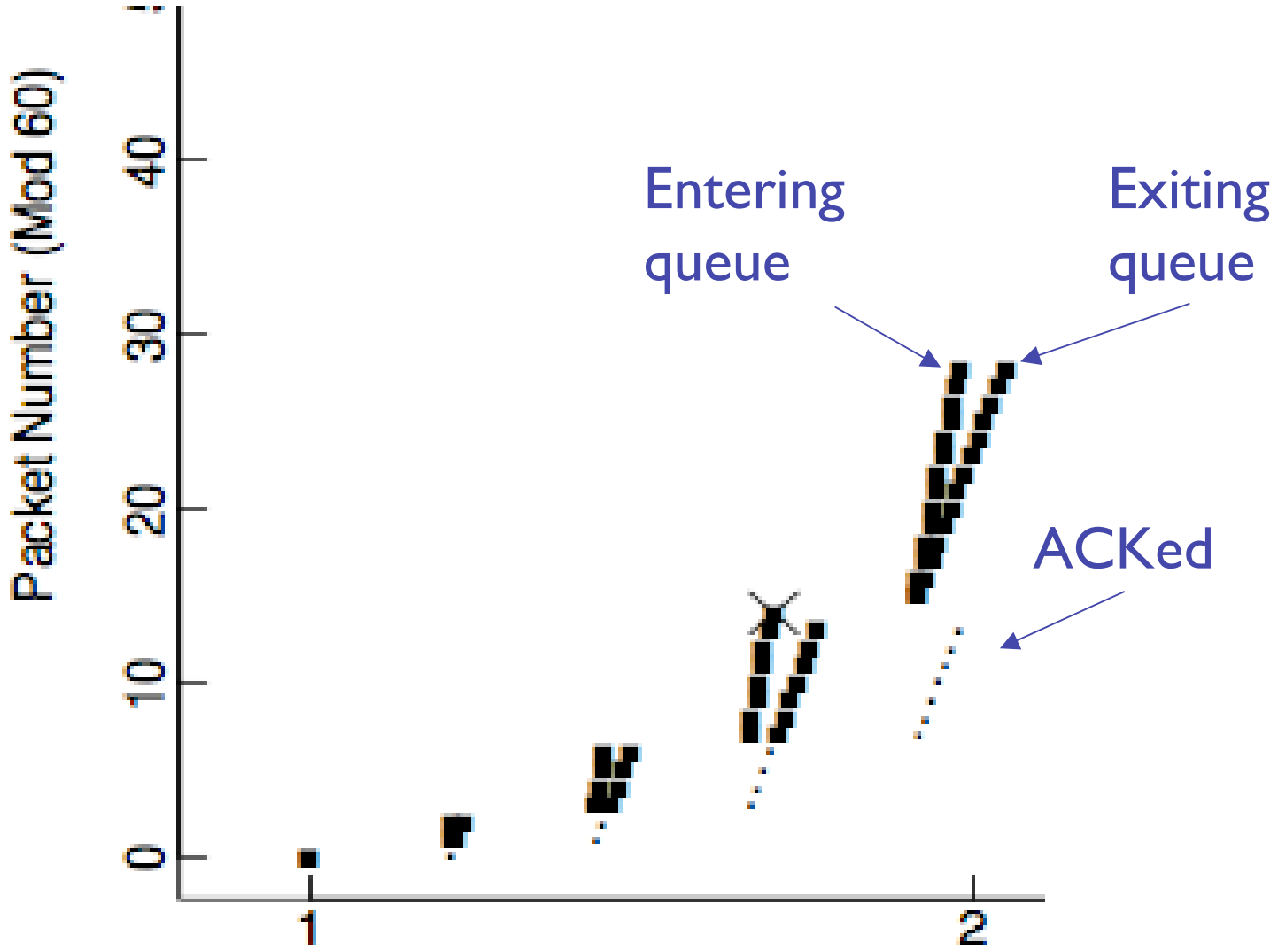
At time t+RTT, we receive ACK for packets [U,U+W-1], set cwnd to W/2.

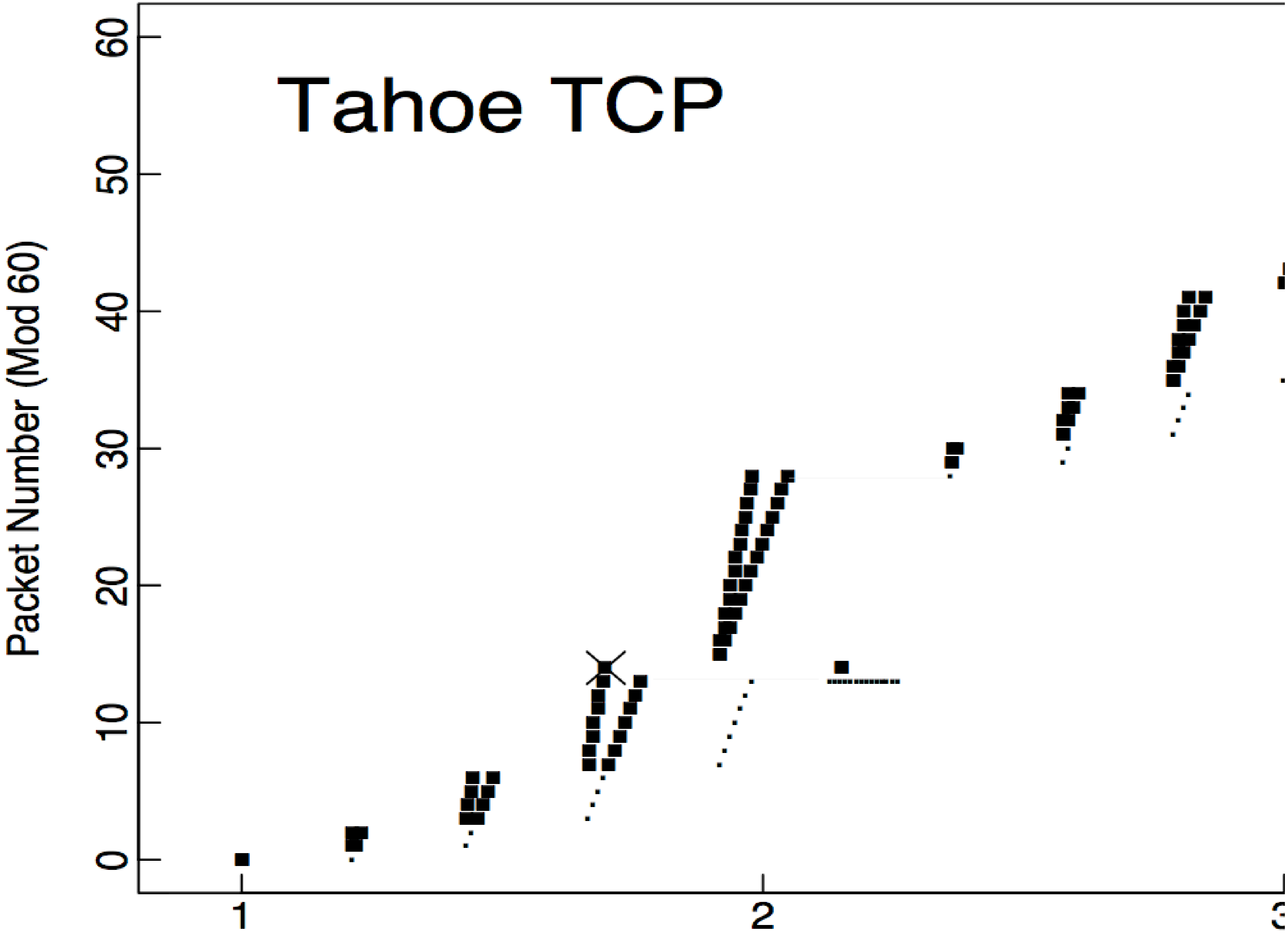



14
12
10
8
6
4
2
0
Reno cwnd
Tahoe cwnd
ssthresh

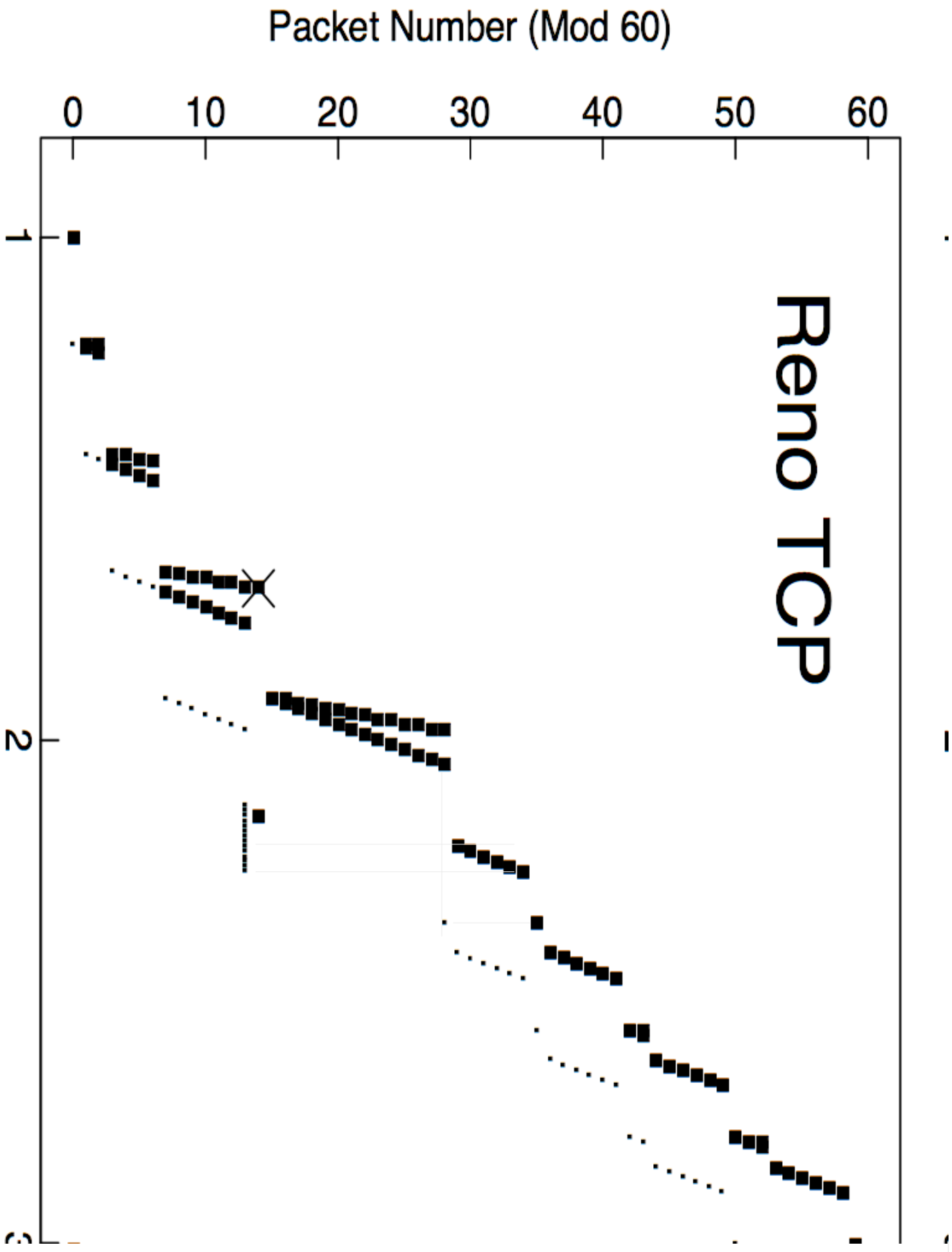
Simulation of TCP Tahoe/Reno

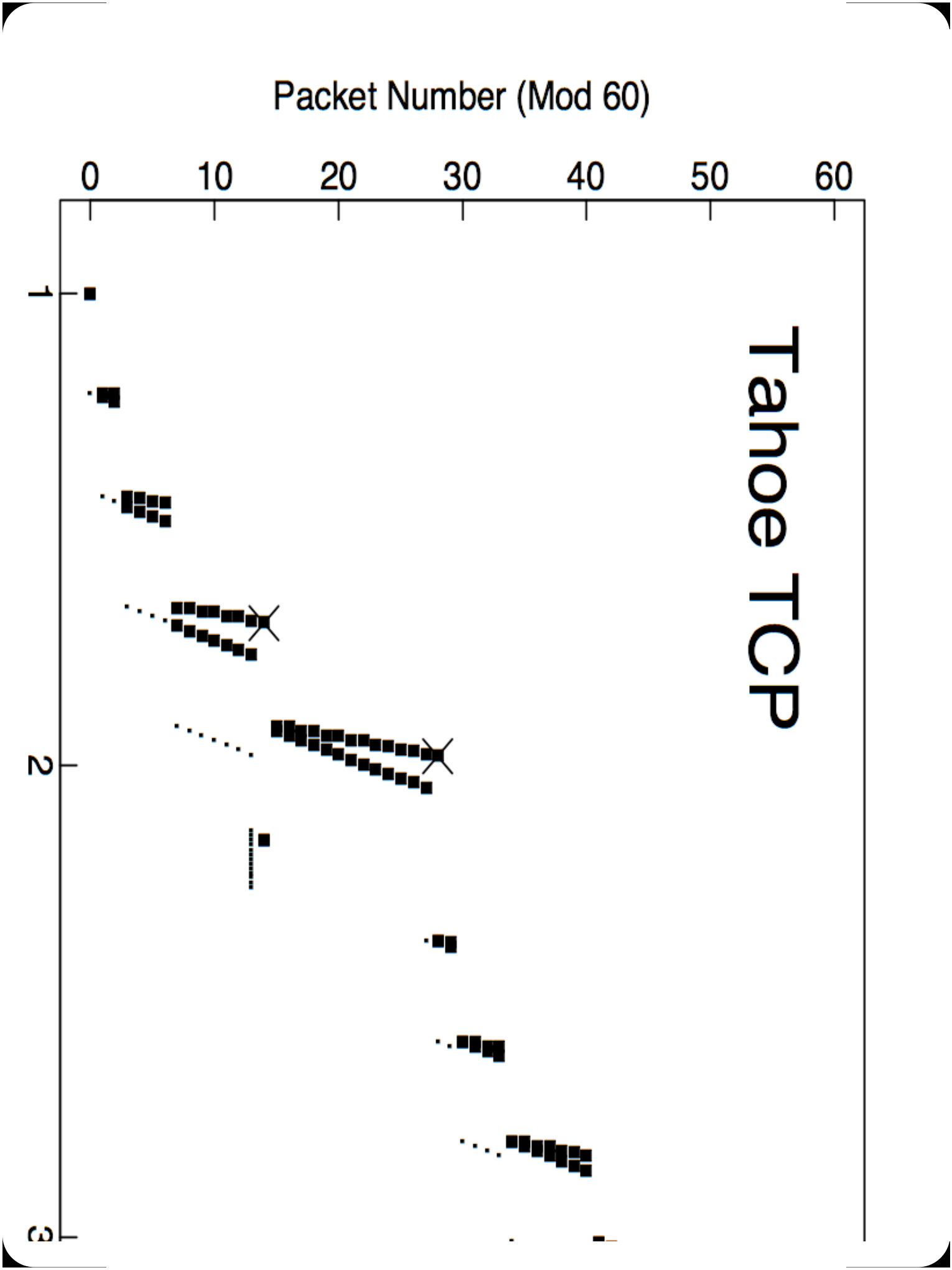

S1
R1
S2

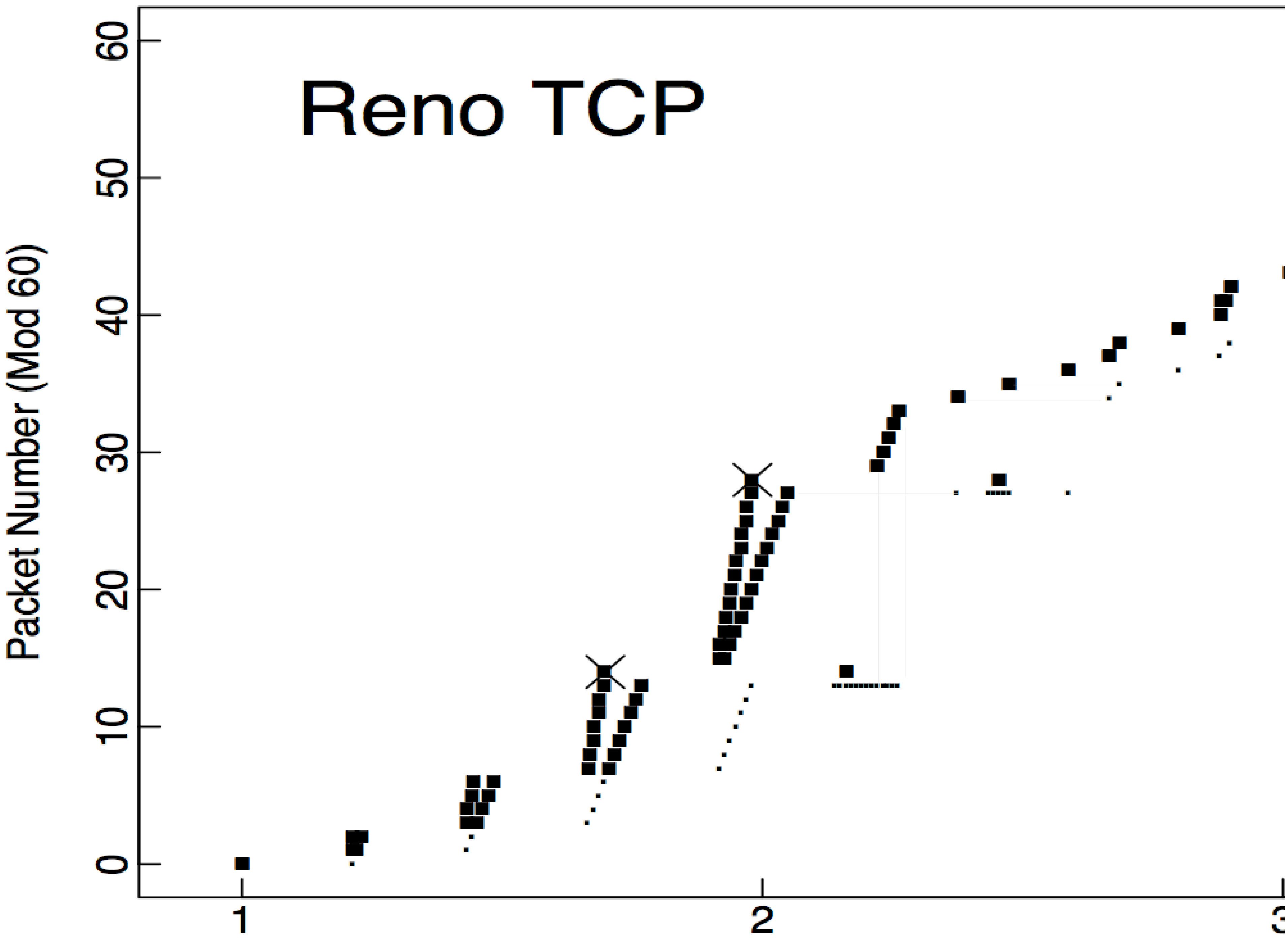

Packet Number (Mod 60)
0
10
20
30
40
1
2
Entering queue
Exiting queue
ACKed



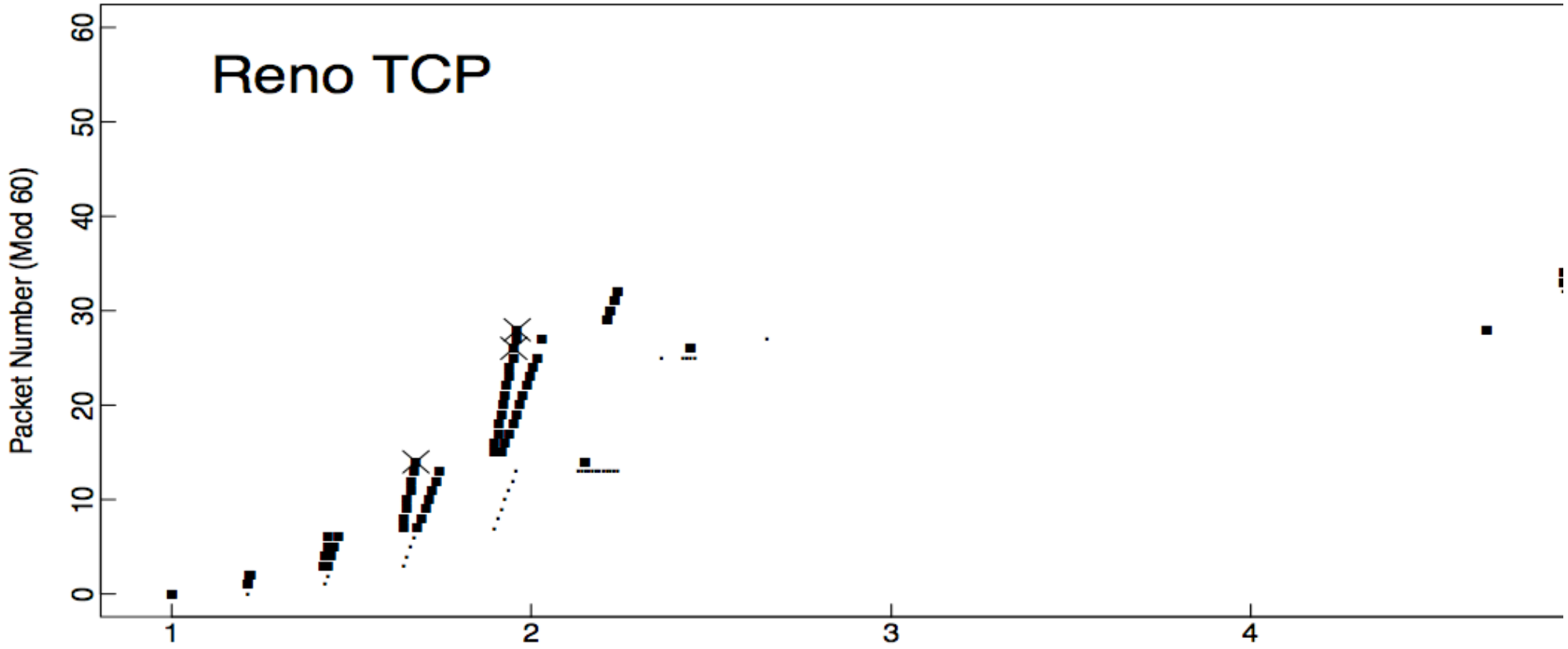
Tahoe TCP
Packet Number (Mod 60)
0
10
20
30
40
50
60
1
2
3

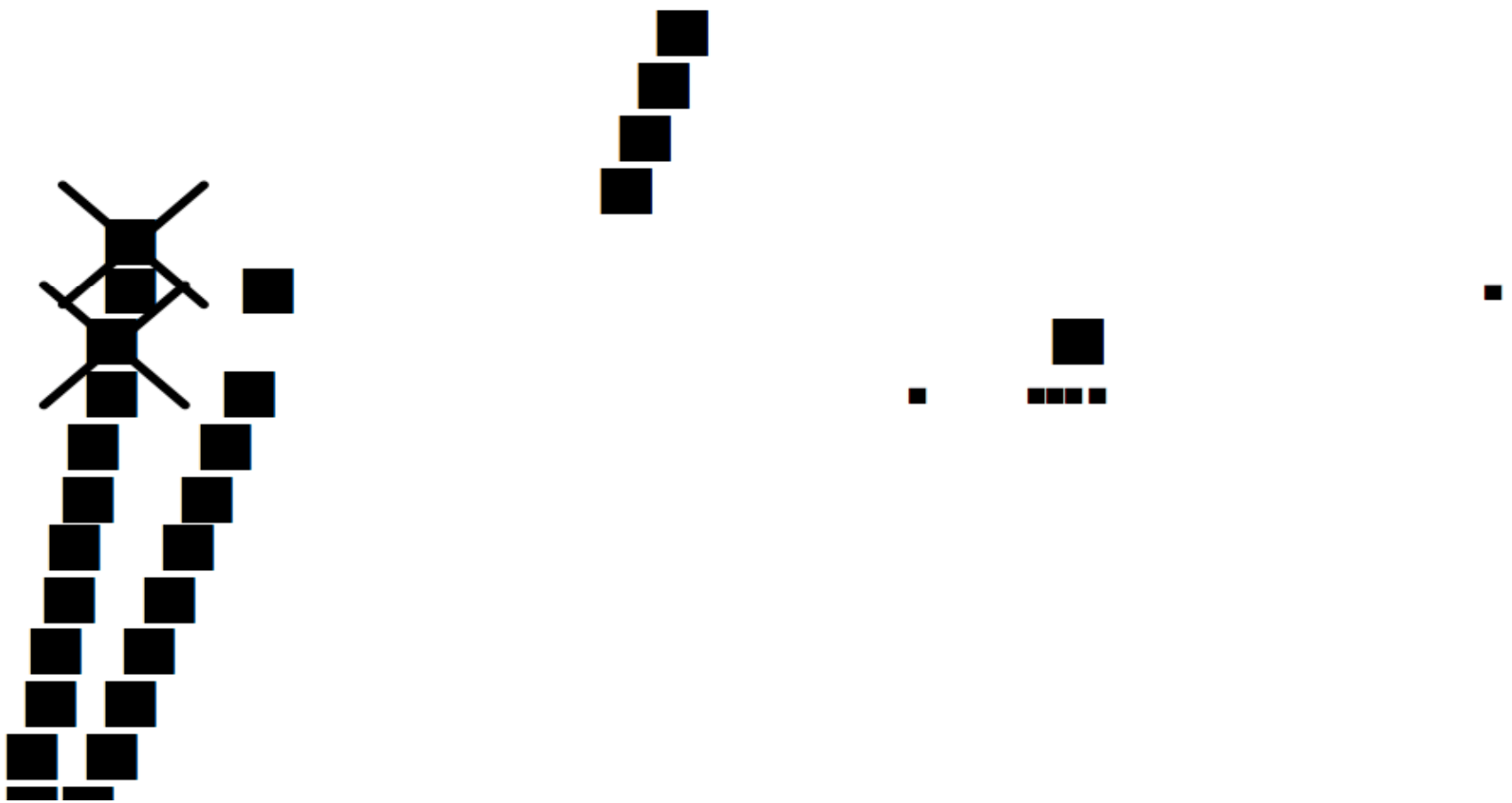

Packet Number (Mod 60)
0
10
20
30
40
50
60
1
2
Reno TCP


Packet Number (Mod 60)
0
10
20
30
40
50
60
1
2
3
Tahoe TCP



Reno TCP
Packet Number (Mod 60)
0
10
20
30
40
50
60
1
2


Reno TCP
Packet Number (Mod 60)
0
10
20
30
40
50
60
1
2
3
4

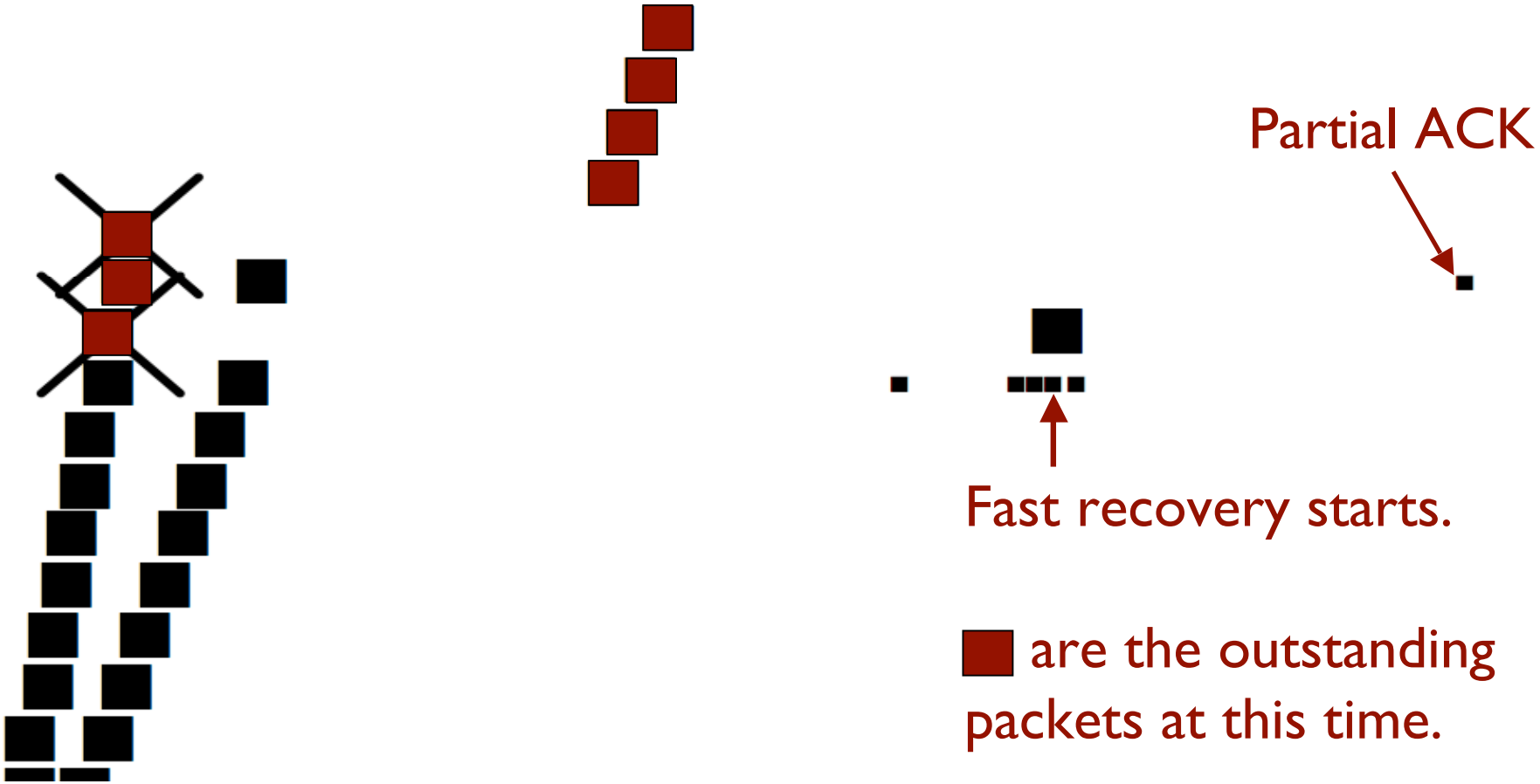


Improving TCP Reno:

Timeout if multiple losses in a window

TCP NewReno

Idea: stays in fast recovery until all ■ have been ACKed.




Perhaps the next
packet is lost?

3rd dup ack:

retransmit 1st unacked

ssthresh = cwnd/2

cwnd = cwnd/2 + 3

remember highest 

subsequent dup ack:

cwnd++

“complete” ack:

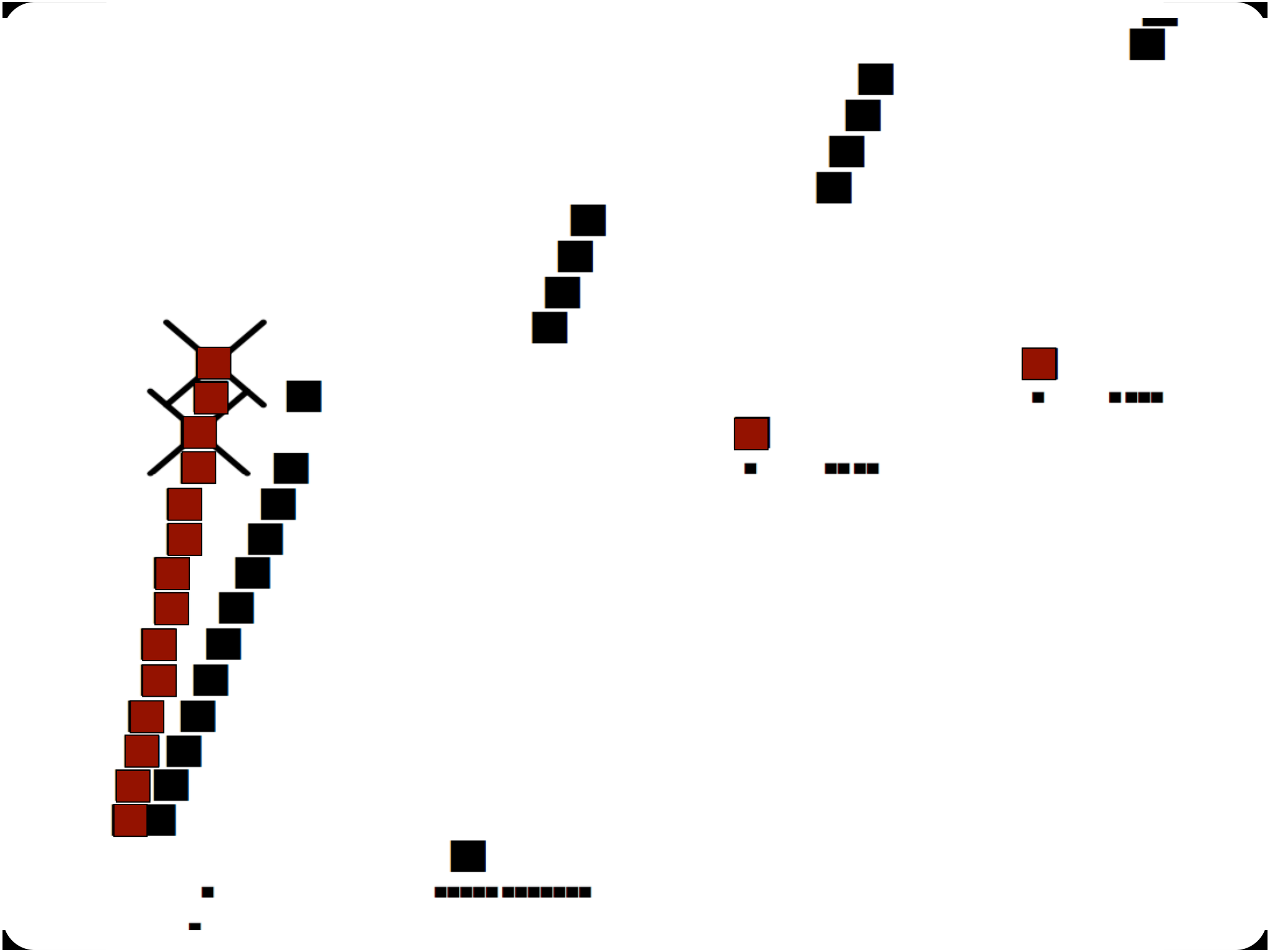
(all  are acked)

cwnd = ssthresh

“partial” ack:

retransmit

cwnd = ssthresh (?)



Note: RFC2581/RFC2582 give the accurate/gory details. Simplified version is presented here (eg. cwnd vs FlightSize, update of cwnd upon partial ACK).

TCP SACK

Coarse Feedback

Go-Back-N
vs
Selective Repeat

Use TCP header options to report received segments.

SACK Blocks:

1st block - report most recently received segments

subsequent blocks - repeat most recent previous blocks

pipe: num of outstanding packets in the path.

send only if pipe < cwnd

scoreboard: which packets have been received?

3rd dup ack:

pipe = cwnd - 3

retransmit 1st unacked

ssthresh = cwnd/2

cwnd = cwnd/2 + 3

subsequent dup ack:

cwnd++

pipe--

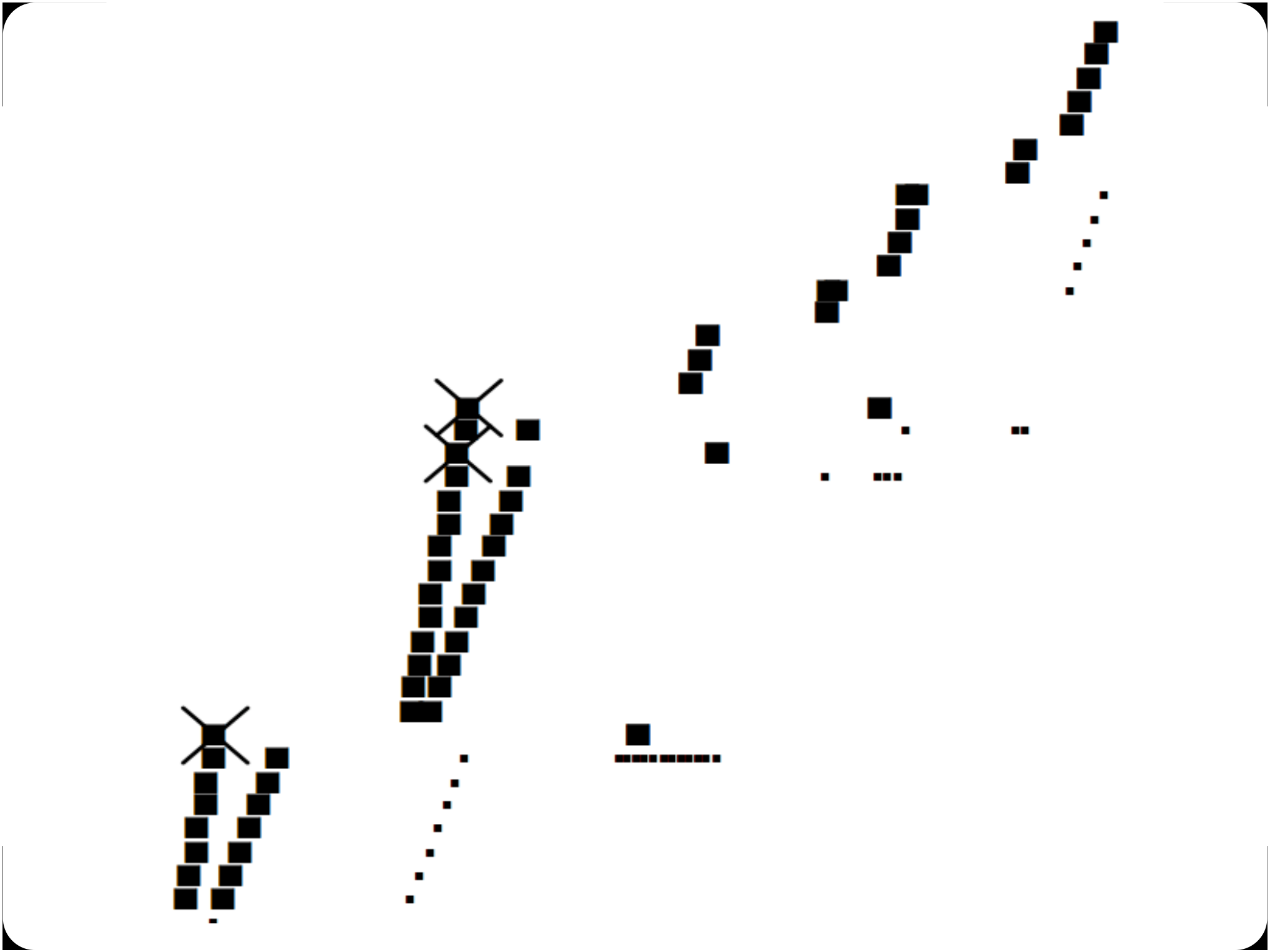
(if send new packet, pipe++)

“partial” ack:

retransmit

cwnd = ssthresh

pipe -= 2



Power of SACK:

Which packet has left the network?

Where is the gap?

Decouple *when* to send and *what* to send.

TCP Vegas

So far,

packet loss as

signal of congestion.

But, already **over congested** when packets are dropped

What other signals are there?


RTT
Load

Expected Sending Rate

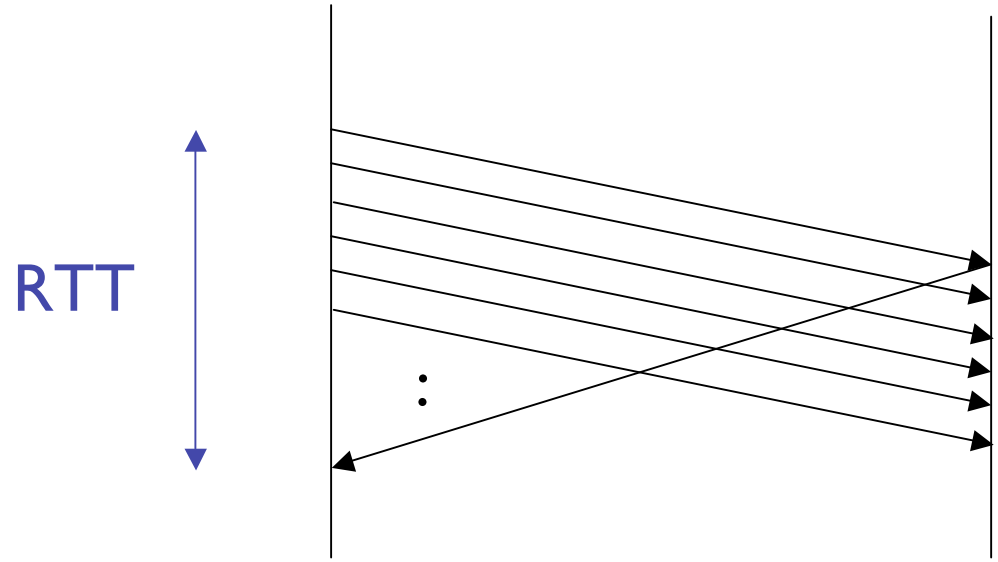
$$E = cwnd/BaseRTT$$

BaseRTT: RTT when no congestion

(take min measured RTT in practice)

Actual Sending Rate

$$A = cwnd/RTT$$



If (E-A) < alpha

cwnd++

else if (E-A) > beta

cwnd--

Intuition:
(E-A) x BaseRTT represents extra buffers occupied in the network

Picking alpha/beta

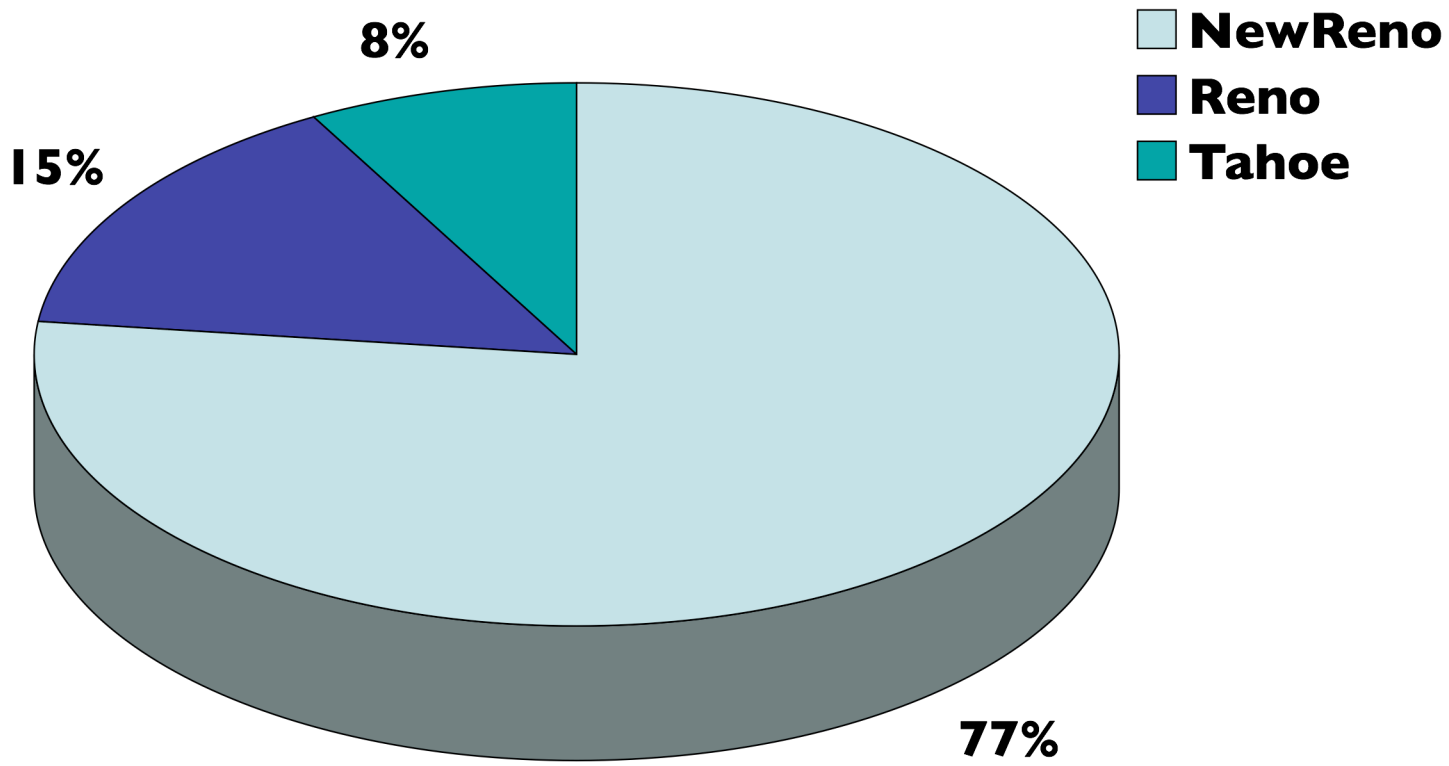
alpha: small but non-zero to take advantage of available bandwidth immediately. (= 1/BaseRTT)

Picking alpha/beta

beta: beta-alpha should not be too small to prevent oscillation.
(= 3/BaseRTT)

Deployment

Feb 2004



70%

SACK capable

Where is TCP Vegas?

Problem 1.

Can't compete with TCP Reno.

Problem 2.

Sensitive to RTT estimation.

TCP BIC/CUBIC

Linux 2.6.x

Compound **TCP**

MS Windows Vista