TCP over Wireless

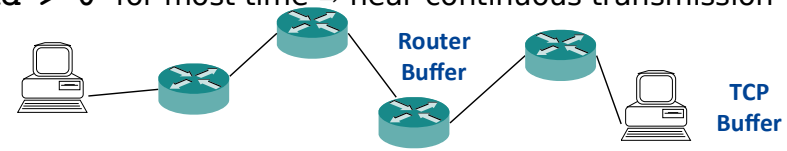
Classic TCP-Reno

Ideal operation

`in-flight` segments = **`cwnd`** (send **`cwnd`** without stopping)

Cumulative **`ACK`** for each 2 or 3 segments

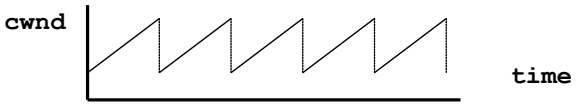
`cwnd > 0` for most time ⇒ near-continuous transmission



Finite buffers in routers

Collision avoidance

`cwnd` ← **`cwnd + 1`** until buffers fill ⇒ **`cwnd`** ← **`cwnd / 2`**



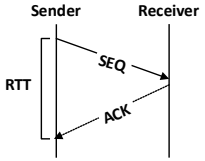
RTT and Buffer Errors

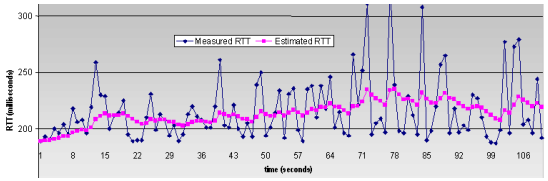
Round Trip Time

RTT = data transmit time + send buffer times
+ ACK transmit time + ACK buffer times

Buffer time ~ typical service time × buffer level

RTT = random variable (rise / fall sharply)





Buffer time in TCP

Timeout

Buffer level ⟶ RTT > RTO (retransmit timeout)

Packet considered lost

Out-of-order packet

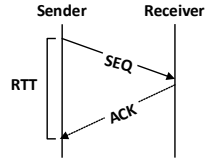
Buffer level ⟶ RTT(packet k) > RTT(packet k+1)

Receiver will send cumulative ACK if OOO packet not lost

RTT and Congestion

Buffer error condition

Buffer level ⟶ RTT > RTO for time T_{error}

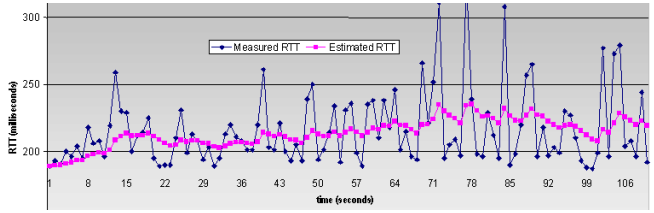


Isolated error

T_{error} < time between packets

Buffer empties before next packet

No need to lower transmission rate



Congestion

T_{error} > time between packets

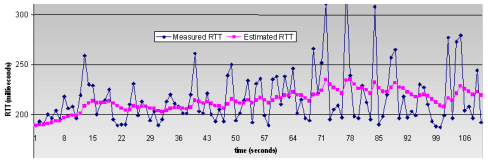
Multiple buffer errors

Lower transmission rate to prevent buffer errors

Congestion Control

Average transmission rate

$$R = \left\langle \frac{\text{packets sent}}{\text{time to ACK}} \right\rangle = \left\langle \frac{\text{cwnd}}{\text{RTT}} \right\rangle$$



R ↓ if cwnd ↓ or RTT ↑

Isolated buffer error

T_{error} < time between packets

RTT ↑ ⇒ R ↓ for T_{error}

Buffer corrects itself

No need to change cwnd

Congestion

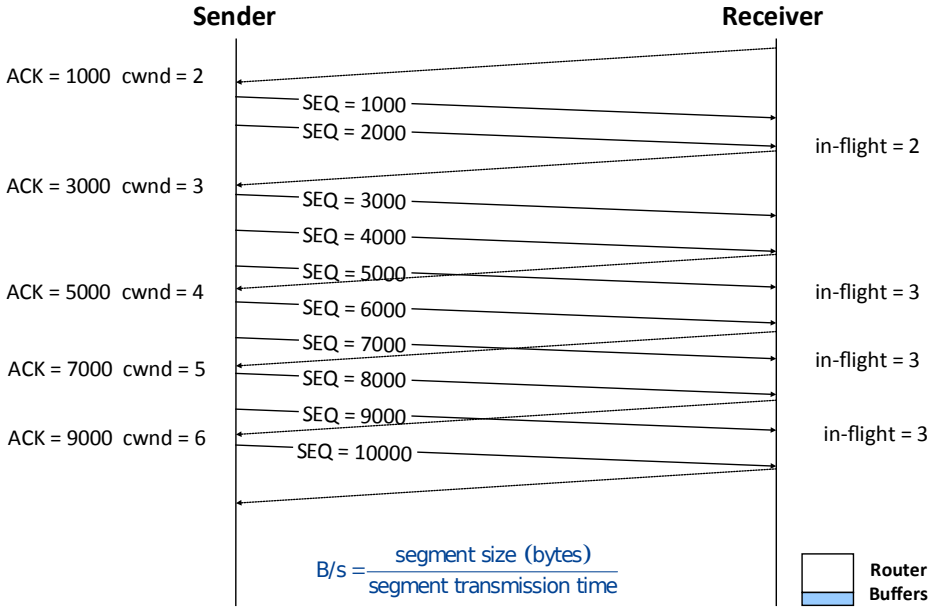
T_{error} > time between packets

RTT ↑ ⇒ R ↓

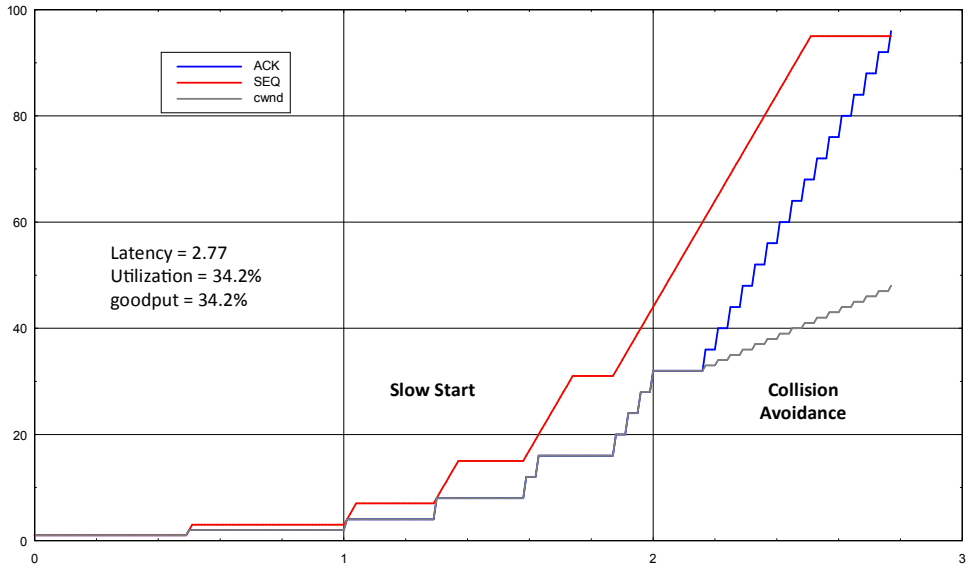
Buffer still too full

Senders: cwnd ↓ ⇒ R ↓

Classic TCP-Reno — No Congestion in Collision Avoidance



Error-Free Transmission



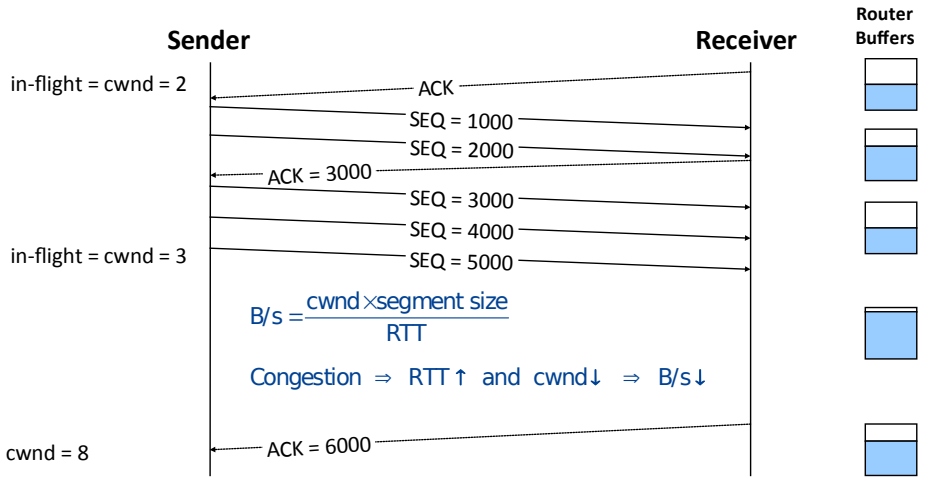
Classic TCP-Reno — Steady State in Collision Avoidance

Self-Clocking

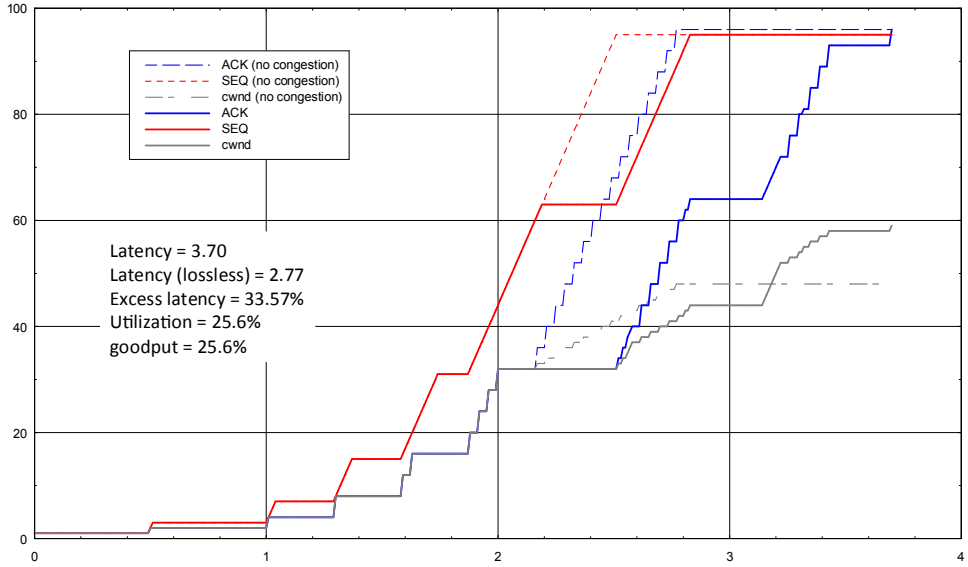


in-flight ≈ cwnd

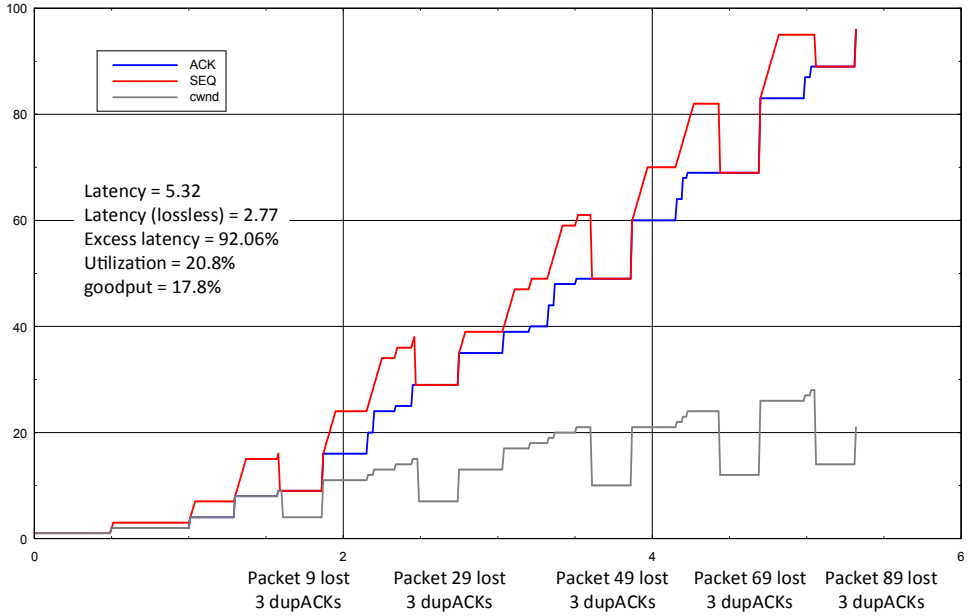
n new ACKs ⇒ { **in-flight ← in-flight - n**; **cwnw ← cwnd + 1** } ⇒ **n+1** new segments



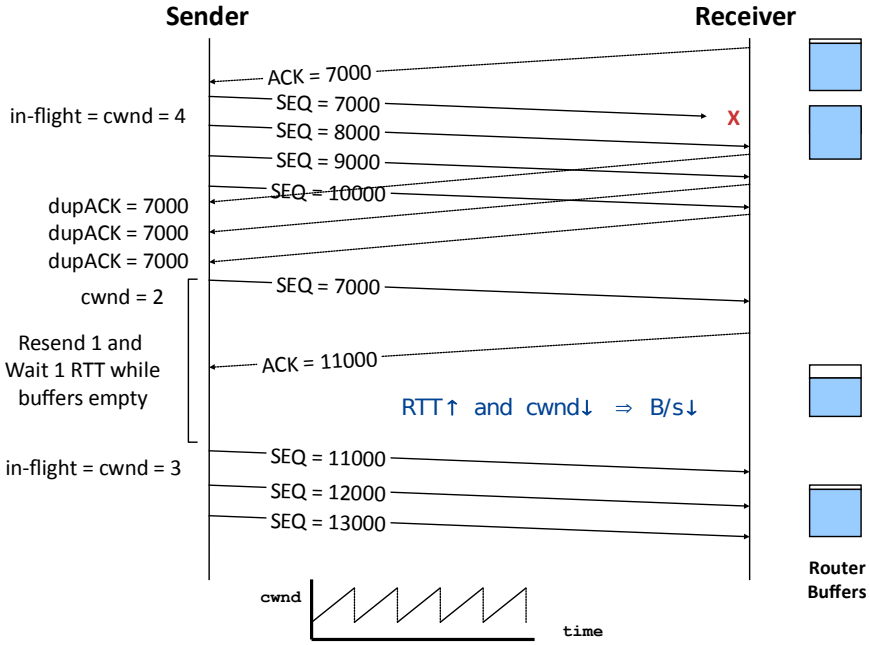
Congestion ⟶ Longer RTT



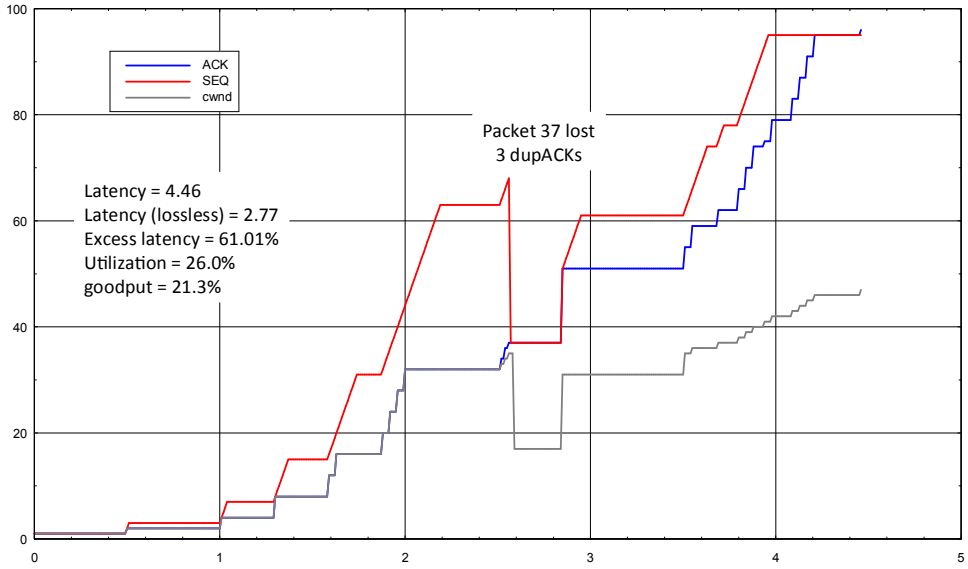
Steady State



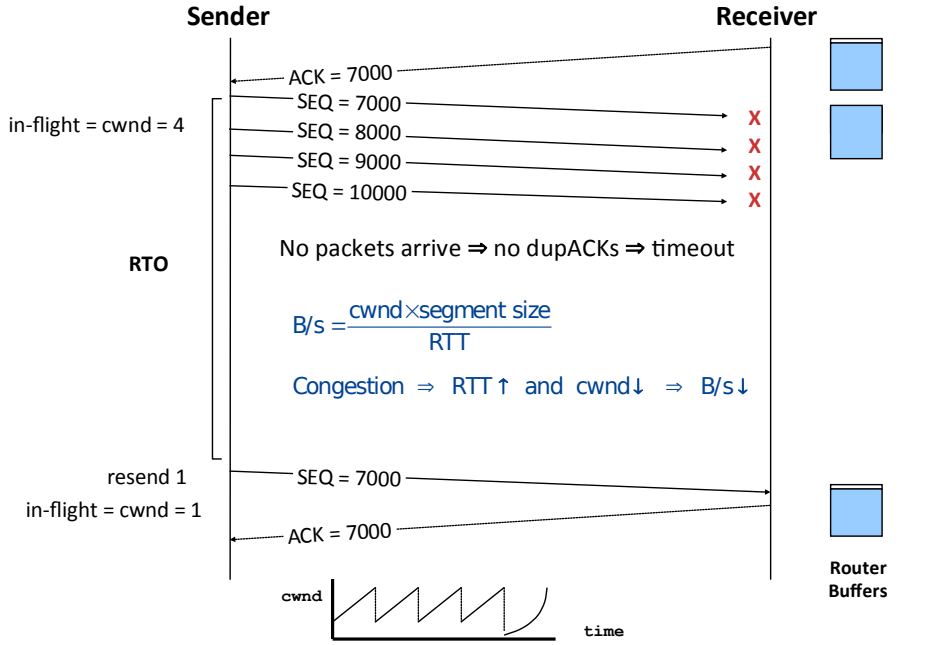
Classic TCP-Reno — 1 Lost Packet



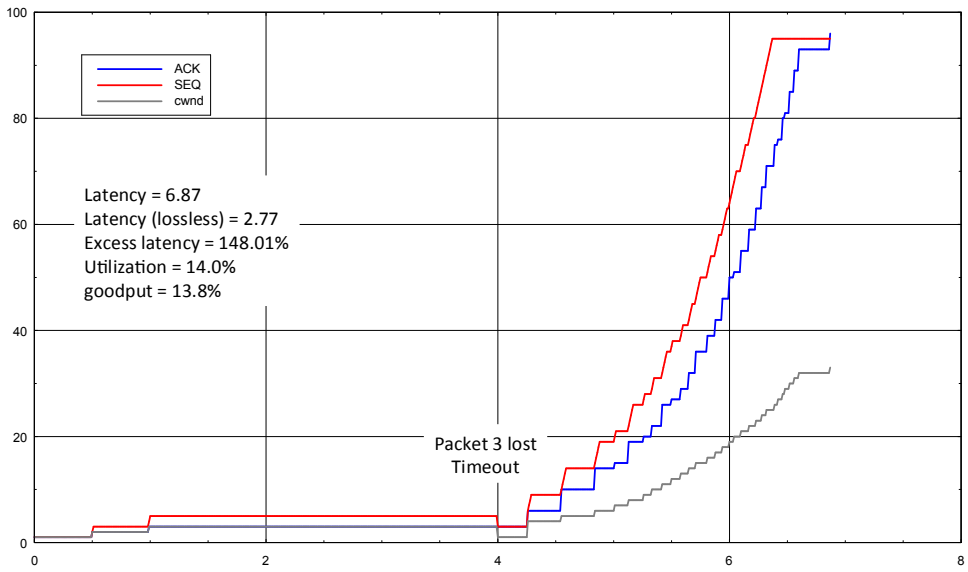
1 Lost Packet



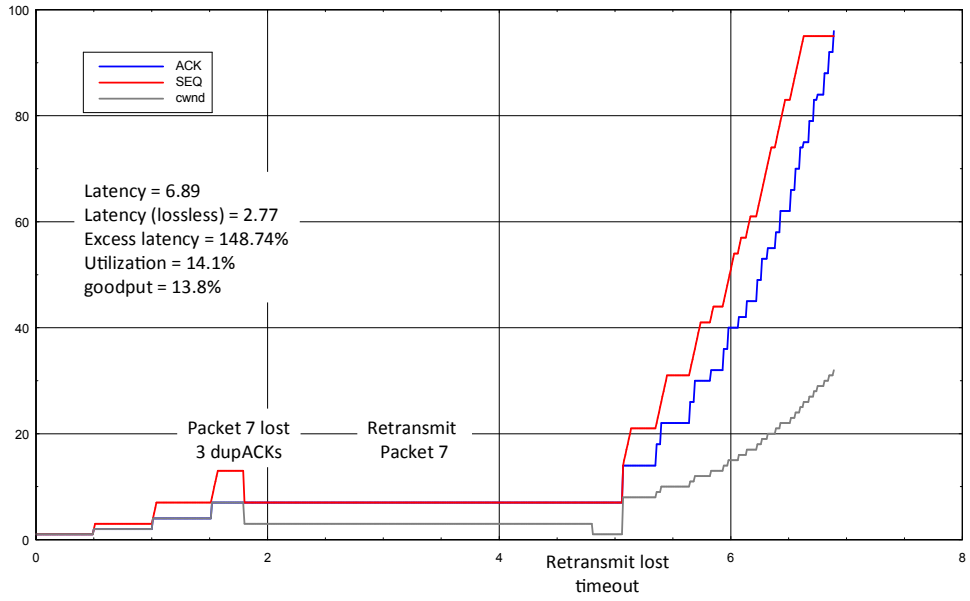
Classic TCP-Reno — Serious Congestion



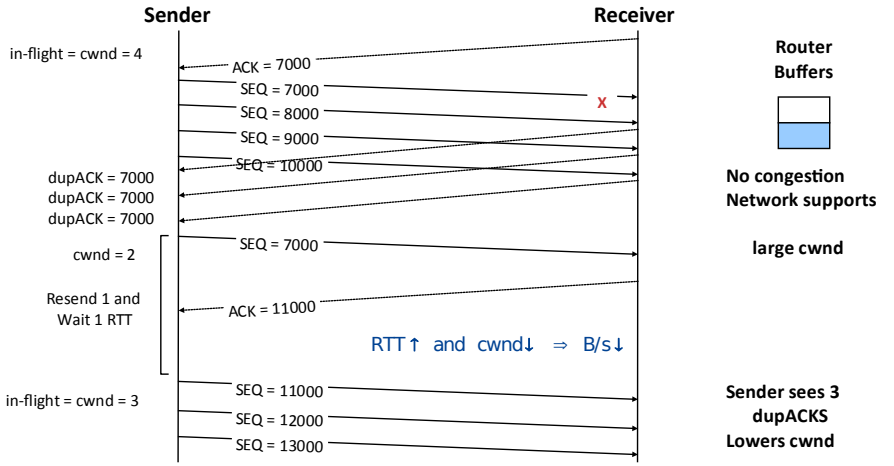
1 Lost Packet — Early



Lost Packet + Lost Retransmit



Classic TCP-Reno — 1 Packet Error



Reno assumes lost packet ⇒ congestion

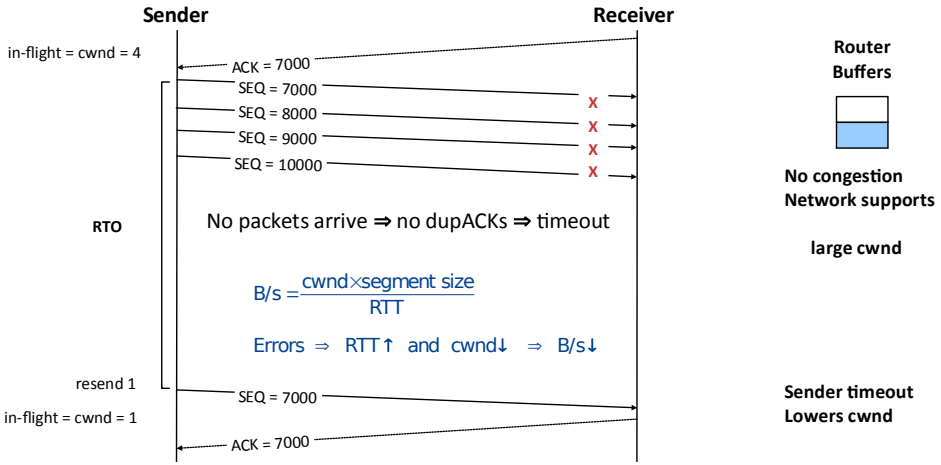
Low packet error on wired Ethernet — BER ≈ 10^{-5}

Moderate response

Lose 1 RTT + time to recover cwnd



Classic TCP-Reno — Multiple Packet Errors

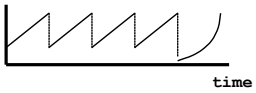


Reno assumes multiple lost packets ⇒ serious congestion

P(k consecutive packet errors on Ethernet) ≈ $(10^{-5})^k$

Extreme response

Lose RTO + time to recover cwnd in slow start



The Trouble with Wireless — 1

Variations in transmission medium

Multiple correlated packet losses + bit errors

TCP interprets as serious congestion — timeouts + slow start

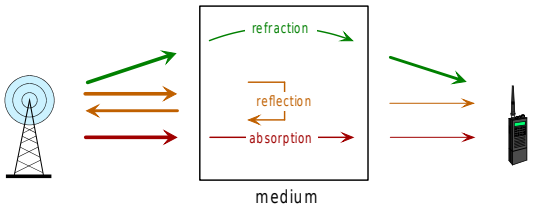
Fading channels

Refraction

Reflection

Absorption

Multipath



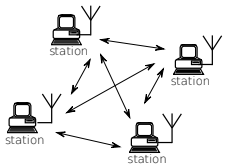
EMI (electromagnetic interference)

Other users

Other radio equipment

Other radiating equipment



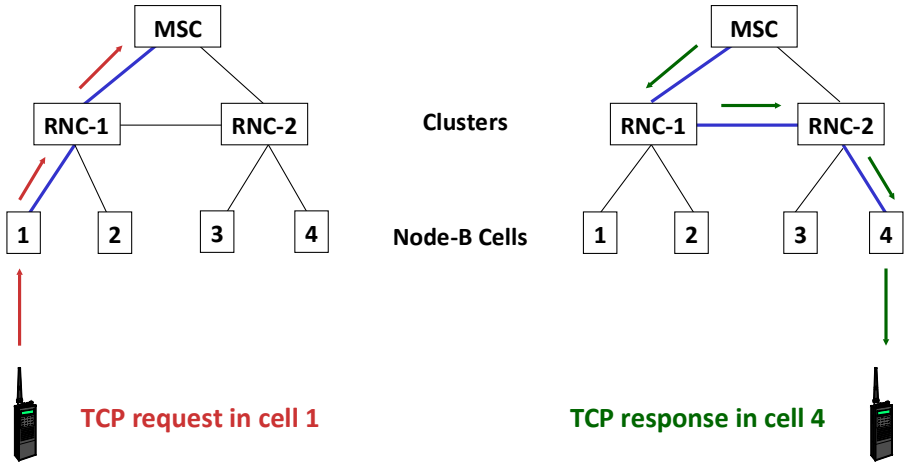


The Trouble with Wireless — 2

Mobility drops

User moves between wireless domains

Adds delays + buffers + dupACKs + timeouts



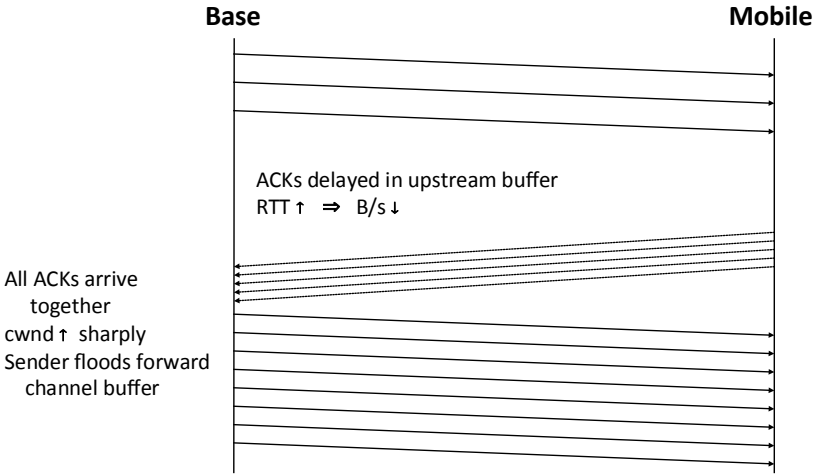
The Trouble with Wireless — 3

Link asymmetry

Upstream channel slower than downstream channel

Larger buffer ⇒ longer buffer delay ⇒ lower average B/s

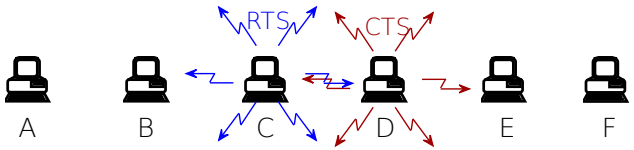
ACK compression

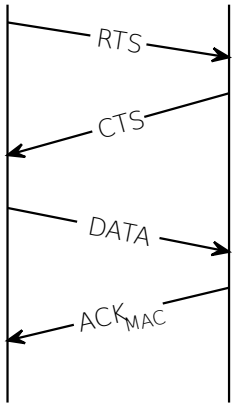


The Trouble with Wireless — 4

MACA in WiFi

Required to prevent hidden node problem





MACA overhead

RTS+CTS+ACK_{MAC} time ≈ TCP ACK time

TCP WiFi ACK delay = 2 × wired TCP ACK delay ⇒ RTT ↑

Timeouts ⇒ cwnd ↓

$$B/s = \frac{cwnd \times segment\ size}{RTT}$$

RTT ↑ and cwnd ↓ ⇒ B/s ↓

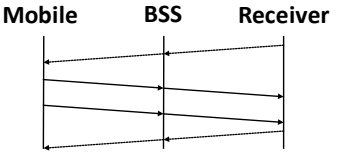
Typology of Approaches to Wireless TCP

Split mode

MS / BSS — specialized TCP

BSS / server — standard TCP

Proposed in 1990s



End-to-end

Refinements to TCP-Reno

- Better handling of non-congestion packet loss
- Example — TCP → TCP-Tahoe → TCP-Reno

Infrastructure-awareness

- Use IP and infrastructure layer information at TCP layer
- Example — ECN provides IP router information to TCP

Reactive congestion control

React to congestion loss

Example — Reno cuts cwnd after 3 dupACKs

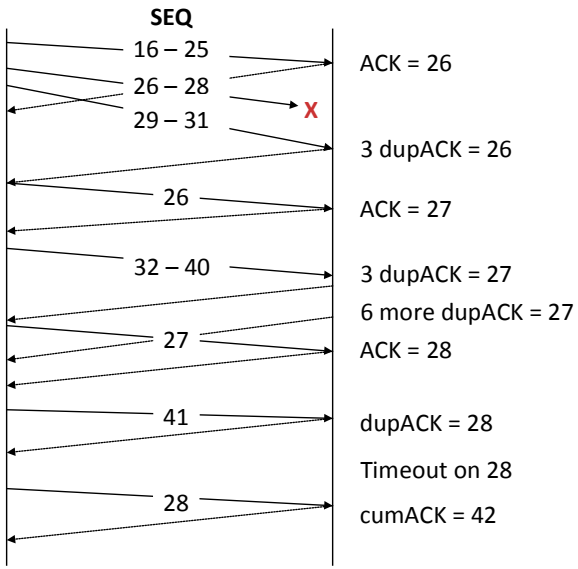
Proactive congestion control

Predict available bandwidth from ACK rates

Multiple Losses in Reno

Example — 3 consecutive lost packets of 16

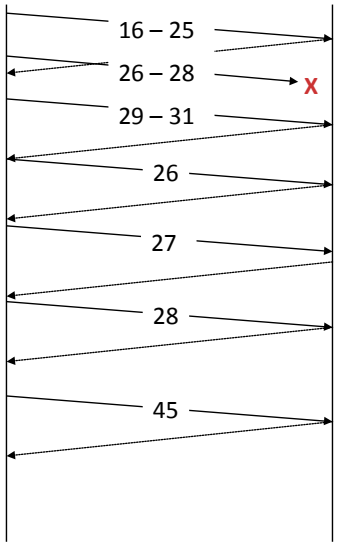
cwnd	in-flight
16	16
26	6
13	6
14	5
14	14
7	14
13	14
14	13
14	14
1	14
15	0



New Reno — RFC 6582

Maintain fast recovery until all in-flight ACKed

cwnd	in-flight
16	16
26	6
13	6
14	5
14	4
15	0
15	15
16	



ACK = 26

3 dupACK = 26

ACK = 27

ACK = 28

cumACK = 31

ACK = 46

Send next on partial ACK of resend

TCP-NCL (Non-Congestive Loss)

Heuristic

Timeout could be error

Retransmission timeout probably congestion

Enhancement to Reno

Treat first timeout as error

Treat long timeout as congestion

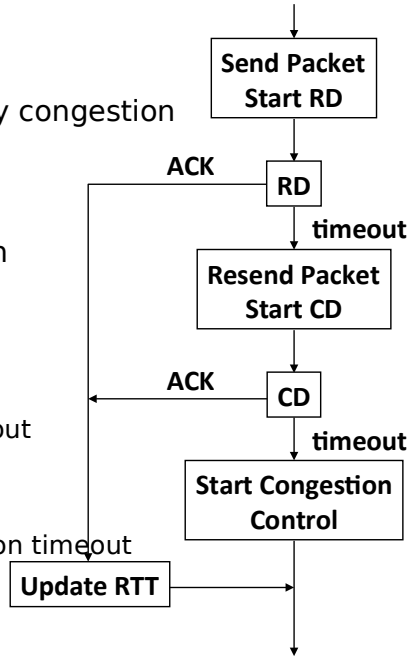
Two timeouts

Retransmission Decision (RD)

- Cancel on ACK
- Retransmit on timeout

Congestion Decision (CD)

- Cancel on ACK
- Congestion control on timeout



C. Lai, K. Leung, V.O.K. Li, "Enhancing Wireless TCP: A Serialized-Timer Approach", IEEE INFOCOM, 2010.

TCP Vegas

Standard Reno slow start

L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," IEEE JSAC, vol. 13, no. 8, Oct. 1995, pp. 1465-80.

`cwnd = 1`

`cwnd ← cwnd + size of data ACKed`

Base expectation

`expected throughput = cwnd / RTT`$_{min}$

Define thresholds α, β

Enhanced Reno congestion avoidance

Measure `RTT` for distinguished (special) segment of some size

`actual throughput = size of ACK received / RTT`

`diff = expected throughput - actual throughput`

if (`diff` < α) then `cwnd++` on each ACK

if (`diff` > β) then `cwnd--` on each ACK

dupACKs

Retransmit on 1 `dupACK` if `RTT >` `RTT`$_{Vegas\text{-}timeout}$

TCP — Modified Fast Retransmission (MFR)

Reno

Transmit cwnd packets

Packet n lost → k dupACKs → retransmit on k ≥ 3 dupACKs

Wait for ACK of n ⇒ no dupACKs if retransmit lost ⇒ timeout

Enhancement to Reno fast retransmit

k dupACKs ⇒ k packets arrived

- k dupACKs + 1 lost packet = k+1 packets accounted
- cwnd – (k+1) unaccounted

If (k > 3) cwnd = cwnd + (k – 3) → send more packets

Cumulative ACK ⇒ retransmitted packet + more arrived

If (dupACKs – 3 > unaccounted)

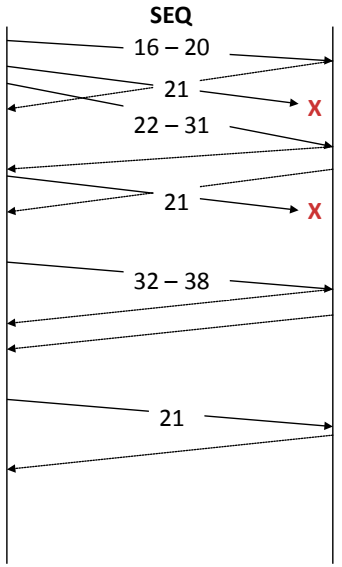
- New packets arrived but retransmitted n lost
- Retransmit n again — no timeout

S. Prasanthi, S. Chung, "An Efficient Algorithm for the Performance of TCP over Multi-hop Wireless Mesh Networks", Seventh International Conference on Information Technology, pp. 816 – 821, 2010.

Modified Fast Retransmission — Example

Prevents timeout on repeated loss

cwnd	in-flight
21	16
26	11
13	11
18	11
18	18
9	18
18	18
18	18
19	0



ACK = 21

3 dupACK = 21

5 more dupACK = 21 (8 of 10)

16 sent – 5 ACK – 8 dupACK – 1 lost = 2 ???

3 dupACK = 21 (late or new?)

6 more dupACK = 21

6 more dupACKS > 2 ???
Resend 21*

cumACK = 39

* If more dupACKS = 2 either:
21 lost again
Late dupACKs from 22 – 31

Modified Congestion Control (MCC)

Reno congestion control

ssthresh = cwnd

cwnd = cwnd / 2

Approach: Oops! Went to far. Cut back half.

Bandwidth Estimation (BWE)

$$BWE = \text{weighted average of } \frac{\text{bytes ACKed}}{\Delta RTT \text{ for bytes_to_ACK}}$$

$$BWE_s[k] = \frac{\text{packets ACKed} \times \text{bytes per packet}}{\text{now} - \text{last_ACK_time}}$$

$$BWE[k] = (1-\beta) \times \frac{BWE_s[k] + BWE_s[k-1]}{2} + \beta \times BWE[k-1]$$

L. Yongmei, J. Zhigang, Z. Ximan, "A New Protocol to Improve Wireless TCP Performance and Its Implementation", 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009.

Modified congestion control

3 dupACKs

ssthresh = (BWE * ΔRTTmin) / segment size

cwnd = ssthresh if (cwnd > ssthresh)

Timeout

cwnd = 1

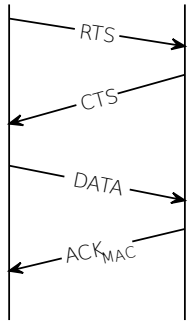
ssthresh = max{(BWE*ΔRTTmin) / segment size, 2}

Proposed Solutions for WiFi Overhead

Problem: MACA overhead

Each TCP ACK ⇒ RTS+CTS+ACK_{MAC}

RTS+CTS+ACK_{MAC} time ≈ TCP ACK time



Solution: reduce control traffic

Increase TCP ACK delay

Buffer more TCP ACKs in receiver

ACK 4 TCP segments instead of 2

Reduce ACK traffic by 50%

BWE not changed

Bits ACKed ↑ and RTT ↑

Problem: longer ACK delay ⇒ lower utilization

Sender waits longer for ACKs

Solution: infrastructure awareness

Sender writes cwnd in TCP option field

Immediate ACK if unACKed data in receiver ACK buffer ≈ cwnd

Freeze-TCP

Handles mobility drops

Handoff disconnect / reconnect

Temporary fading by obstacles

Mobile station monitors radio signal strength

Predicts impending disconnections

Freeze-TCP receiver proactively sets window = 0

TCP sender enters persist mode

On reconnect

Receiver sends multiple ACKs of last received packet

Prevents exponential back off of cwnd in sender

Infrastructure awareness

TCP layer exposed to details handled by lower layers

Roaming / handoff at layers 2 / 3

Radio signal quality at layers 1 and 2

TCP-Jersey

Available Bandwidth Estimation (ABE)

Similar to BWE expressed in segments

Congestion Warning (CW)

Requires explicit congestion notification (ECN) support in routers

IP routers mark packets on congested link

TCP receiver returns ECN warning to TCP sender

CW without loss

cwnd = ABE

3 dupACKs

Retransmit

If (CW in dupACK segment) cwnd = ABE

Else no change to cwnd

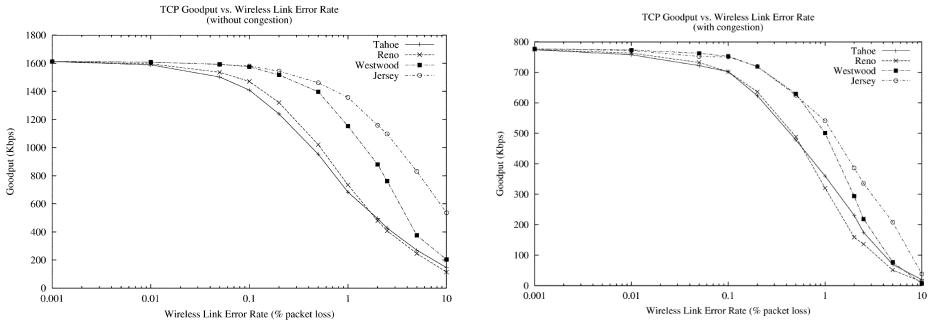
Timeout

Retransmit

If (CW in dupACK segment) cwnd = 1

Else no change to cwnd

TCP-Jersey Results



TCP Westwood

Modifies TCP sender

Not dependent on negotiation with TCP receiver

Not dependent on ECN support

Estimates available bandwidth

Counts ACKs and dupACKs as successful traffic

On packet loss set cwnd = available bandwidth

Improves on Reno cwnd = cwnd / 2

Performance almost as good as Jersey

Westwood Packet Counting

Reno enhancement

Standard slow start

Standard congestion avoidance

Calculate bandwidth estimate (BWE) on each ACK

$$BWE_k = \frac{d_k}{t_k - t_{k-1}} = \frac{\text{number of bytes ACKed at time } t_k}{\text{time of ACK - time of previous ACK}}$$

Bandwidth sample

$$BWE = \alpha_k BWE_k + (1 - \alpha_k)\frac{BWE_k + BWE_{k-1}}{2}$$

Bandwidth estimate

Handling isolated error

Out-of-order packets ⟶ dupACKs

dupACK ⇒ some packets **ARRIVED** at receiver

Count dupACKs in bandwidth estimate

On ACK of lost packet — treat dupACK as already counted

Serious congestion

Causes timeouts instead of dupACKs

Westwood Packet Counting Example

Packet transmission times

1	2	3	4	5	6	7	8	5	9
t_0	t_1		t_2				t_3	t_6	

Time	ACK	Arrived at Receiver	Packets for BWE	Counted dupACK	BWE
t_1	2	1	1	0	$1 / (t_1 - t_0)$
t_2	4	2 + 3	2	0	$2 / (t_2 - t_1)$
t_3	5	4	1	0	$1 / (t_3 - t_2)$
t_4	5	6	1	1	$1 / (t_4 - t_3)$
t_5	5	7	1	2	$1 / (t_5 - t_4)$
t_6	5	8	1	3	$1 / (t_6 - t_5)$
t_7	10	5 + 9	**2**	0	$2 / (t_7 - t_6)$

ACK jumps to 10 from 5 but 3 ACKs (6, 7, 8) already counted as dupACKs

Westwood ACK Counter (for BWE)

`newACK = ACK – prevACK ;`	Packets ACKed by new ACK
`// if (newACK = 1) do nothing`	No error condition — report 1 ACK

`if (newACK = 0)`
`  count++ ;`
`  newACK = 1 ;`
	ACK "stuck" on old value (dupACK)
	Increment dupACK counter
	Count 1 dupACK

`if (newACK > 1)`	ACK advances

`  if (count >= newACK)`
`  count = count – newACK + 1 ;`
`  newACK = 1 ;`
	Not all arrived packets ACKed
	Remove ACKed from dupACK count
	Count as additional dupACK

`  else if (count < newACK)`
`  newACK = newACK - count ;`
`  count = 0;`
	All packets ACKed in order
	Report ACKed – counted (new ACKs)
	Zero dupACK counter

`prevACK = ACK ;` `return(newACK);`	

Westwood Scenario 1 — No Errors

Receive 1 — 2,3 — 4,5,6,7 — 8,9,10,11,12,13,14,15

ACK 2 — 4 — 8 — 12 — 16 –→ newACK = 1 — 2 — 4 — 4 — 4

`newACK = ACK – prevACK ;`	1	2	4	4	4

`if (newACK = 0)`
`  count++ ;`
`  newACK = 1 ;`
	—	—	—	—	—

`if (newACK > 1)`					

`  if (count >= newACK)`
`  count = count – newACK + 1 ;`
`  newACK = 1 ;`
	—	—	—	—	—

`  else if (count < newACK)`
`  newACK = newACK - count ;`
`  count = 0;`
		2
		0
			4
			0
				4
				0
					4
					0

`prevACK = ACK ;`	2	4	8	12	16
`return(newACK);` Total = 15	1	2	4	4	4

Westwood Scenario 2 — 1 Packet Out-of-Order

Receive 1 — 2,3 — 5,4,6,7 — 8,9,10,11,12,13,14,15

ACK 2 — 4 — 4 — 6 — 10 — 14 — 16

`newACK = ACK – prevACK ;`	1	2	0	2	4	4	2

`if (newACK = 0)`
`  count++ ;`
`  newACK = 1 ;`
	—	—	1
			1
				—	—	—	—

`if (newACK > 1)`							

`  if (count >= newACK)`
`  count = count – newACK + 1;`
`  newACK = 1 ;`
	—	—	—	—	—	—	—

`  else if (count < newACK)`
`  newACK = newACK - count ;`
`  count = 0;`
		2
		0
				1
				0
					4
					0
						4
						0
							2
							0

`prevACK = ACK ;`	2	4	4	6	10	14	16
`return(newACK);` Total = 15	1	2	1	1	4	4	2

Westwood Scenario 3 — 1 Packet Very Out-of-Order

Receive 1 — 2,3 — 5,6,7,4 — 8,9,10,11,12,13,14,15

ACK 2 — 4 — 4 — 4 — 4 — 8 — 12 — 16

`newACK = ACK – prevACK ;`	1	2	0	0	0	4	4	4

`if (newACK = 0)`
`  count++ ;`
`  newACK = 1 ;`
	—	—	1
			1
				2
				1
					3
					1
						—	—	—

`if (newACK > 1)`								

`  if (count >= newACK)`
`  count = count – newACK + 1 ;`
`  newACK = 1 ;`
	—	—	—	—	—	—	—	—

`  else if (count < newACK)`
`  newACK = newACK - count ;`
`  count = 0;`
		2
		0
						1
						0
							4
							0
								4
								0

`prevACK = ACK ;`	2	4	4	4	4	8	12	16
`return(newACK);` Total = 15	1	2	1	1	1	1	4	4

Westwood Scenario 4 — Variation of Scenario 3

Receive 1 — 2,3 — 5,6,7,4 — 8,9,10,11,12,13,14,15
ACK 2 — 4 — 4 — 4 — 4 — 5 — 9 — 13 — 16

newACK = ACK − prevACK ;	1	2	0	0	0	1	4	4	3

if (newACK = 0)
  count++ ;
  newACK = 1 ;
	—	—	1
			1
				2
				1
					3
					1
						—	—	—	—

if (newACK > 1)									

if (count >= newACK)
  count = count − newACK + 1 ;
  newACK = 1 ;
	—	—	—	—	—	—	—	—	—

else if (count < newACK)
  newACK = newACK − count ;
  count = 0;
		2
		0
							1
							0
								4
								0
									3
									0

prevACK = ACK ;	2	4	4	4	4	5	9	13	16
return(newACK) ; Total = 15	1	2	1	1	1	1	1	4	3

Westwood Scenario 5 — Another Variation of Scenario 3

Receive 1 — 2,3 — 5,6,7,4 — 8,9,10,11,12,13,14,15
ACK 2 — 4 — 4 — 4 — 4 — 6 — 10 — 14 — 16

newACK = ACK − prevACK ;	1	2	0	0	0	2	4	4	2

if (newACK = 0)
  count++ ;
  newACK = 1 ;
	—	—	1
			1
				2
				1
					3
					1
						—	—	—	—

if (newACK > 1)									

if (count >= newACK)
  count = count − newACK + 1 ;
  newACK = 1 ;
	—	—	—	—	—	2
						1
							—	—	—

else if (count < newACK)
  newACK = newACK − count ;
  count = 0;
		2
		0
						—	2
							0
								4
								0
									2
									0

prevACK = ACK ;	2	4	4	4	4	5	10	14	16
return(newACK) ; Total = 15	1	2	1	1	1	1	2	4	2

Westwood Scenario 6 — 2 Packets Out-of-Order

Receive 9,10,11,13,14,15,8,12 (after ACK = 8)
ACK 8 — 8 — 8 — 8 — 8 — 8 — 12 — 16

newACK = ACK − prevACK ;	0	0	0	0	0	0	4	4

if (newACK = 0)
  count++ ;
  newACK = 1 ;
	1
	1
		2
		1
			3
			1
				4
				1
					5
					1
						6
						1
							—	—

if (newACK > 1)								

if (count >= newACK)
  count = count − newACK + 1 ;
  newACK = 1 ;
	—	—	—	—	—	—	3
							1
								—

else if (count < newACK)
  newACK = newACK − count ;
  count = 0;
							—	1
								0

prevACK = ACK ;	7	7	7	7	7	7	11	15
return(newACK) ; Total = 15	1	1	1	1	1	1	1	1

Westwind Congestion Control

Reno slow start

```
On (ACK && cwnd < ssthresh)
    cwnd ← cwnd + size of data ACKed
```

Reno congestion avoidance

```
On (ACK && cwnd > ssthresh)
    cwnd ← cwnd + 1
```

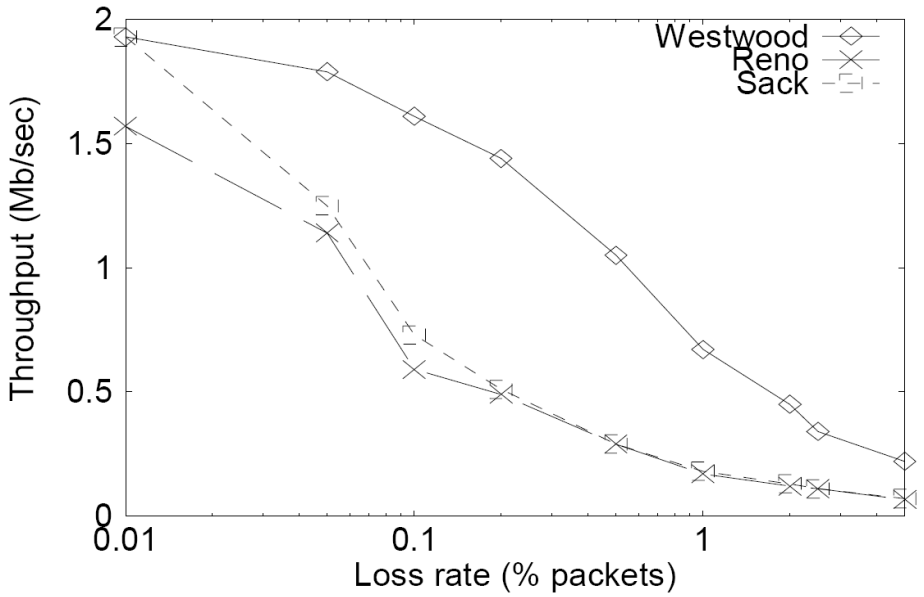
Modified fast recovery

```
On n dupACKs
    ssthresh = BWE * RTT_min / segment_size
    if (cwnd > ssthresh) cwnd = ssthresh
```

Modified timeout

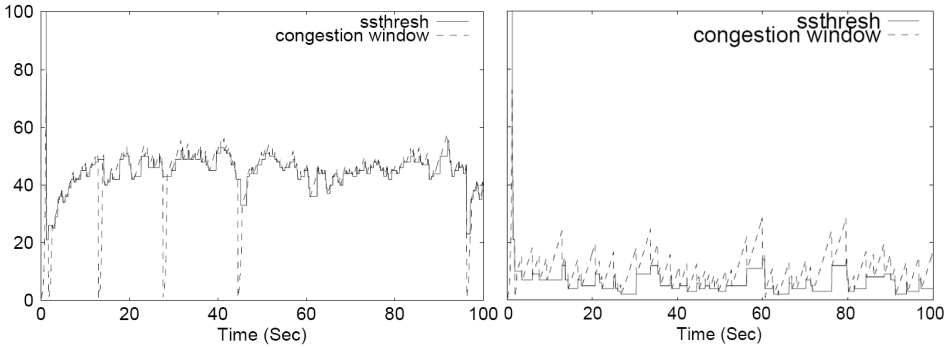
```
ssthresh = BWE * RTT_min / segment_size
if (ssthresh < 2) ssthresh = 2
cwnd = 1
```

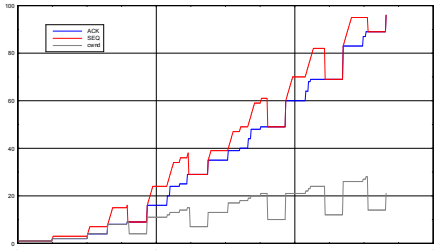
Westwood Performance — Throughput



Westwood — cwnd and ssthresh

Westwood versus Reno





Steady state cwnd (slide 10)