TCP over wireless links

- TCP widely used in the Internet
- Desire to support Internet applications in a wireless setting
- Current systems: e.g., iMode
- 3G's promise to make this real
- TCP as is does not work

Main features of wireless connectivity

- Transmission errors
- Low bandwidth
- Variable (and possibly long) delays
- Time-varying and asymmetric behavior
- Heterogeneous environments

- These factors interact negatively with TCP operation

What is TCP anyway?

- Most Internet applications (including http, web, ftp, e-mail) are based on TCP/IP

application	
transport	(TCP)
network	(IP)
link	
physical	

Internet Protocol (IP)

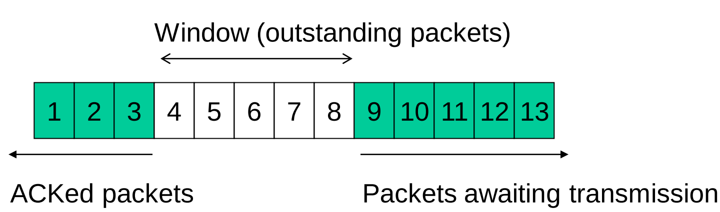
- Provides routing through the network
- Packet-based operation
  - Packets may get lost
  - Packets may get duplicated
  - Packets may get reordered
- Reliable delivery not guaranteed

Transmission Control Protocol (TCP)

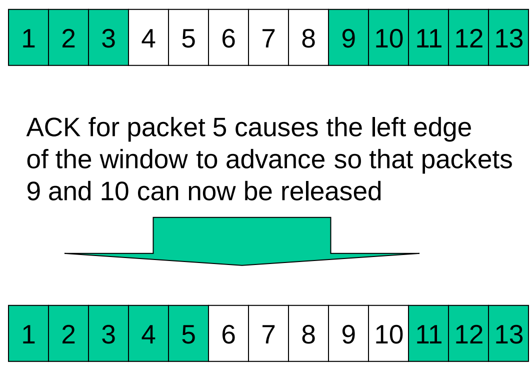
- Uses routing as provided by IP
- End-to-end operation
  - Confirmation via ACK and semantics
  - Loss recovery via retransmission
  - Reliable in-order delivery
- Congestion avoidance
- Congestion control

Sliding window mechanism

- At most W outstanding packets are allowed
- Window size adjusted dynamically based on network feedback
- Max window size is specified (flow control)



Sliding window mechanism



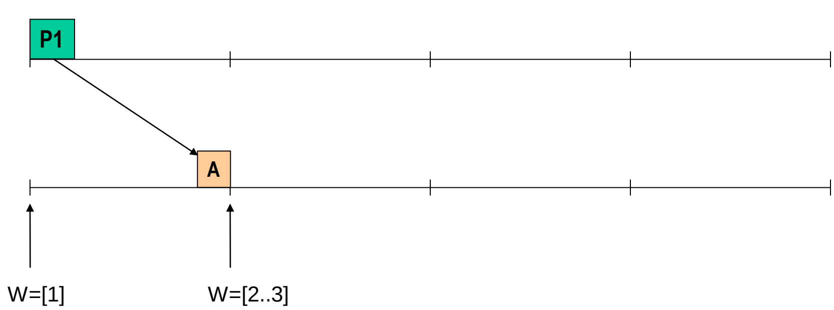
Window adaptation mechanism

- TCP tries to adapt the window size to the instantaneous network conditions
- Main idea: if congestion in the network, slow down transmission rate
  - Whenever a loss is detected, W is reduced
  - Upon correct ACKs, W is increased

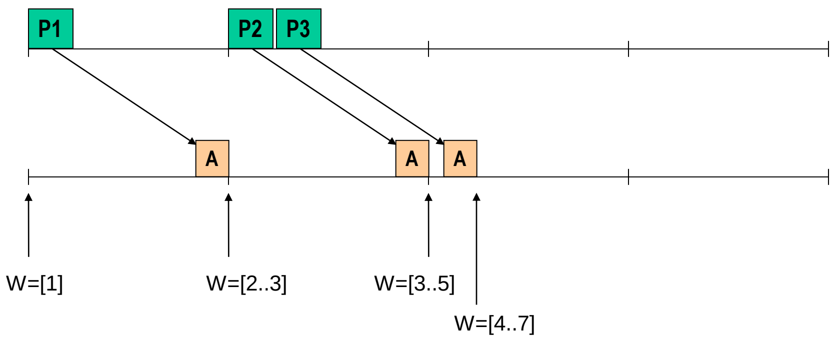
Window adaptation mechanisms

- Slow start
  - At the beginning, W grows by 1 for each ACK
  - Aggressive (exponential) increase for throughput
- Congestion avoidance
  - After reaching a threshold, W increases by 1 for each RTT (1/W for each ACK)
  - More gentle (linear) increase to probe for bandwidth

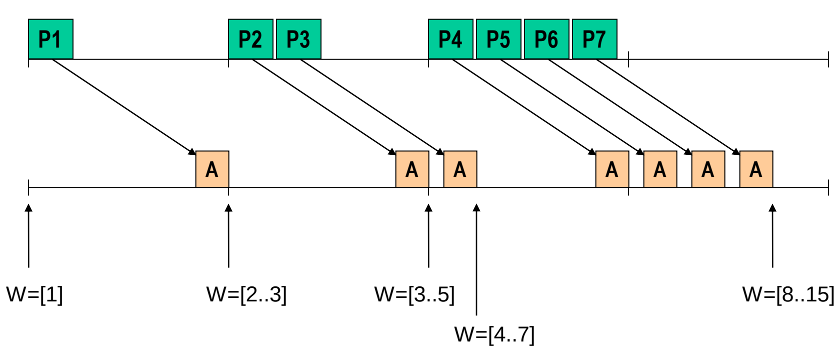
Window evolution example: RTT1



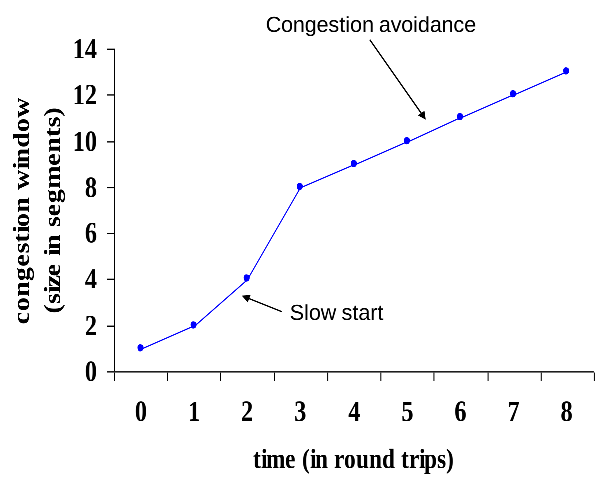
Window evolution example: RTT2



Window evolution example: RTT3



Window evolution example

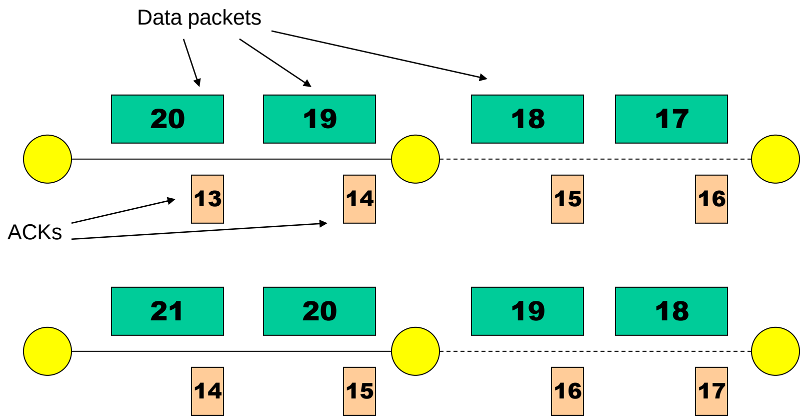


Bandwidth-delay product

- How large should the window be?
- Bandwidth-delay product: BDP = rate x RTT
- It is the amount of data which could be sent before a complete round-trip
- If W is smaller than BDP, sender forced idle
- In typical situations, BDP may be fairly large; however, in WLANs it may be very small

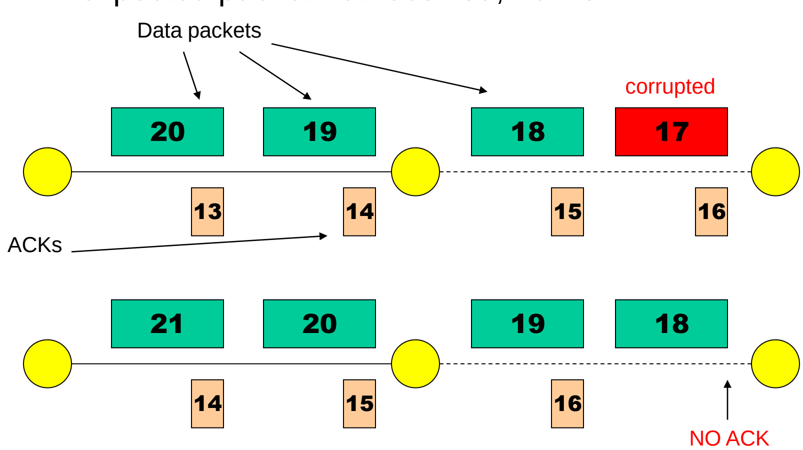
Cumulative ACK mechanism

- Each new packet generates an ACK



Packet losses

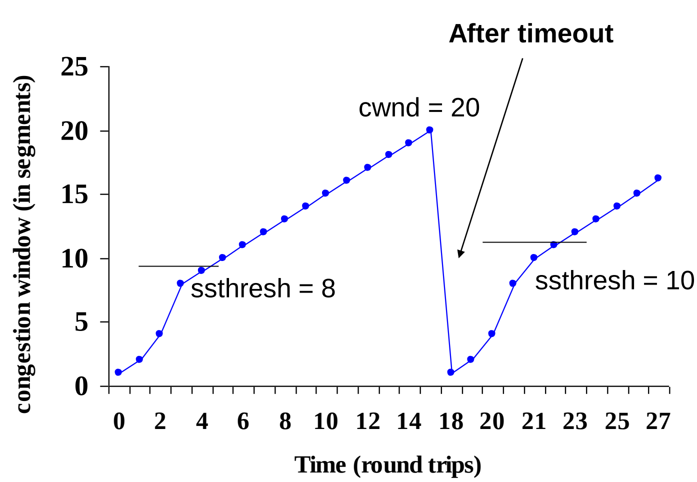
- If expected packet not received, no ACK



Loss detection - timeout

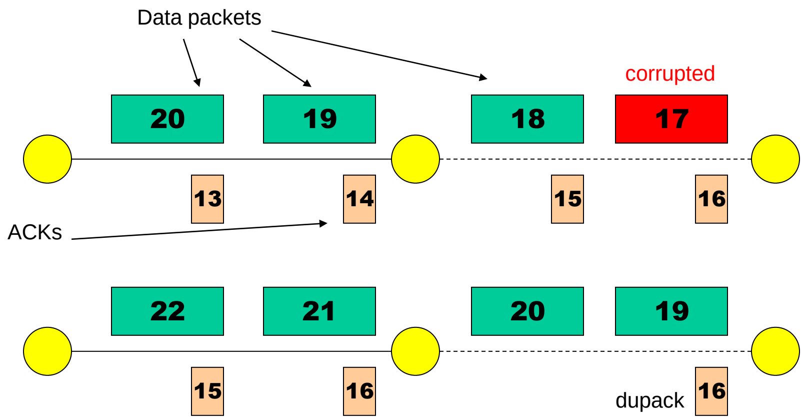
- If an ACK not received after a given time, conclude that packet was lost
- Shrink window down to 1 and initiate slow start
- Retransmission from oldest missing packet
- Problems:
  - Time taken to detect loss (TO granularity)
  - Dramatic action

Window evolution – example



Duplicate ACKs

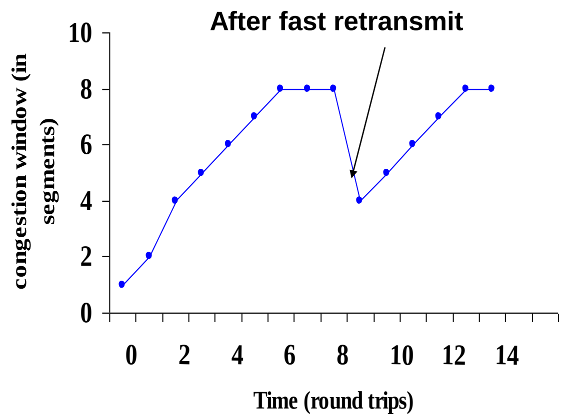
- Out-of-order packets generate duplicate ACKs



Loss detection – Fast retransmit

- Timeout too long
- dupACKs are an indication that a packet may be lost
  – When too many, go ahead and retransmit
  – Typical threshold: K=3
- Originally, update window as after timeout
- If Fast Recovery, divide window in half
  – Pipe is not choked, throughput is higher

Window evolution - example



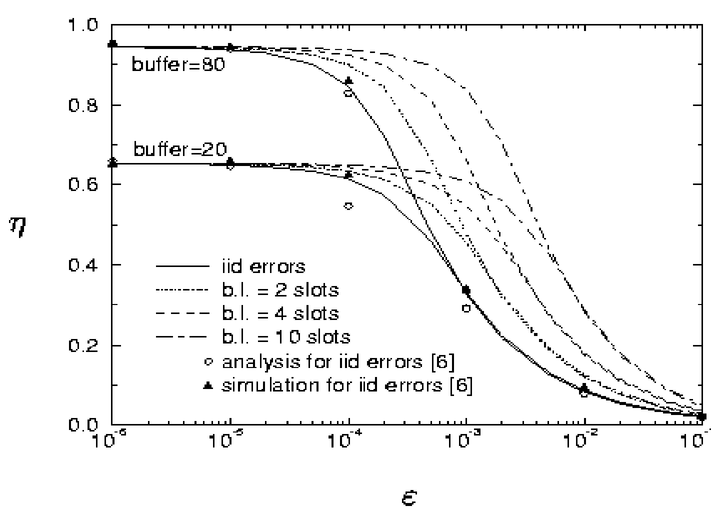
Summary of TCP main features

- Congestion and flow control via sliding window
- Window dynamically adapted: slow start, congestion avoidance
- Reliable and in-order delivery
- Loss detection via ACK traffic
- Loss recovery via retx

Impact of transmission errors

- Basic assumption in TCP: packets are lost because of congestion
- Obviously not true in wireless systems
- When a packet is lost on the wireless channel, **TCP does the wrong thing**
  - It chokes the pipe instead of retransmitting
- This leads to poor throughput, especially when errors are frequent
  - Too much time spent in SS&CA

Throughput vs. Error rates



Analytical study of TCP over Markov channels

- We now examine how the theory of Markov processes can be used to study TCP performance
- We make the following assumptions:
  - Zero round-trip
  - No link layer
  - Two-state Markov channel
  - Perfect feedback
  - Standard end to end TCP (various versions)
- Our goal is to evaluate the throughput performance, and the energy efficiency of the scheme
- See Zorzi et al. JSAC Jul. 2000

Analytical study of TCP over Markov channels

IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 18, NO. 7, JULY 2000 1289

Throughput Analysis of TCP on Channels with Memory

Michele Zorzi, *Senior Member, IEEE*, A. Chockalingam, *Senior Member, IEEE*, and Ramesh R. Rao, *Senior Member, IEEE*

Slow start and congestion avoidance

1. If $W(t) < W_{\mathrm{th}}(t)$, each ACK causes $W(t)$ to be incremented by 1. This is the *slow start* phase.
2. If $W(t) \geq W_{\mathrm{th}}(t)$, each ACK causes $W(t)$ to be incremented by $\frac{1}{W(t)}$. This is the *congestion avoidance* phase.
3. If timeout occurs at the transmitter at time t, $W(t^+)$ is set to 1, $W_{\mathrm{th}}(t^+)$ is set to $\lceil \frac{W(t)}{2} \rceil$, and the transmitter begins retransmission from the next packet after the last acknowledged packet.

Analytical model

- The joint evolution of the window parameters and the channel state can be tracked by a random process $\mathcal{X}(t) = (C(t-1), W_{\mathrm{th}}(t), W(t))$
  - $\star$ $W(t)$ and $W_{\mathrm{th}}(t)$ are the window size and the slow start threshold in slot t
  - $\star$ $C(t-1)$ is the channel state in slot $t-1$.
- This process is **not** Markov; two alternatives:
  - $\star$ make the model more complex
  - $\star$ **consider a sampled version which is Markov**

Sampling the random process - 1

- choose t_k immediately after timeout expiration
  - ⋆ window size shrinks to 1 and all timers are reset
  - ⋆ at this instant knowledge of $(C(t-1), W_{\mathrm{th}}(t), W(t))$ fully determines the future window/channel evolution
- choose t_k immediately after successful completion of a loss recovery phase
  - ⋆ all outstanding packets have been acknowledged, no timeouts

Sampling the random process - 2

- we pick t_k as before and define $X(k) \triangleq \mathcal{X}(t_k)$
- at these instants knowledge of $(C(t-1), W_{\mathrm{th}}(t), W(t))$ fully determines the future window/channel evolution
- sampled process is Markov!

State space of X(k)

- state space of the process $X(k)$:

$$\Omega_X = \left\{(C, W_{\mathrm{th}}, 1), C = B, G, 1 \le W_{\mathrm{th}} \le \left\lceil \frac{W_{\max}}{2} \right\rceil\right\} \cup \left\{(G, W_{\mathrm{th}}, W_{\mathrm{th}}), 1 \le W_{\mathrm{th}} \le \left\lceil \frac{W_{\max}}{2} \right\rceil\right\}$$

  where the first set corresponds to timeout and the second set corresponds to successful recovery phase

- total number of states is $3\lceil W_{\max}/2 \rceil - 1$.

Semi-Markov formulation

- the above information does not track transmissions, time and successes
- consider a semi-Markov process with $X(k)$ as its embedded Markov chain
- transitions are labeled with *transition metrics*: N_d slots, N_t transmissions, N_s successes
- these metrics must only depend on the transition
  - ⋆ if not, bounds with this property must be used

Semi-Markov analysis

- evolution of the process during a generic cycle k, from t_k to t_{k+1}
- all system variables are conditioned on $X(k) = (C(t_k - 1), W_{\text{th}}(t_k), W(t_k)) \in \Omega_X$
- let $n \geq 1$ be the first error in the cycle

$$\alpha_C(n) = P[\text{first error at } t = n | C(0) = C] = \begin{cases} p_{CB} & n = 1 \\ p_{CG} p_{GG}^{n-2} p_{GB} & n > 1 \end{cases}$$

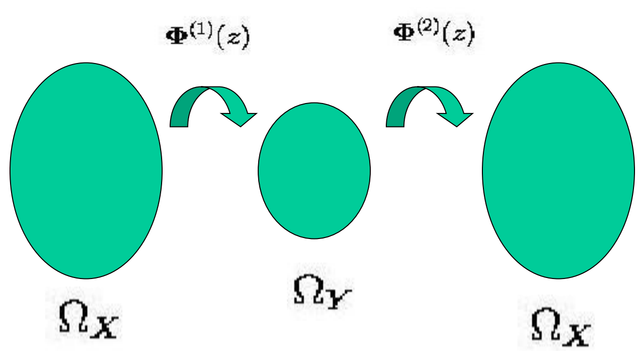
System evolution

- let $Y(k)$ be the system state at time n in cycle k
  - ⋆ no outstanding packets except for the last transmitted
  - ⋆ channel state at time n is B
  - ⋆ $W_{\text{th}}(n)$ has no role in future evolution
- cycle can be separated into two parts
  - ⋆ transition from a state $X(k) \in \Omega_X$ to a state $Y(k) \in \Omega_Y$
  - ⋆ transition from a state $Y(k) \in \Omega_Y$ to a state $X(k+1) \in \Omega_X$

Transition structure

- $\Phi^{(1)}(z)$: matrix of transition functions from $i \in \Omega_X$ to $j \in \Omega_Y$
- $\Phi^{(2)}(z)$: matrix of transition functions from $j \in \Omega_Y$ to $\ell \in \Omega_X$
- system evolution during a cycle is characterized by the matrix
$$\Phi(z) = \Phi^{(1)}(z)\Phi^{(2)}(z),$$
of transition functions from Ω_X to itself

System evolution



- No feedback transitions: consider the two parts separately

Transition functions

- let $\xi_{ij}(N_d, N_t, N_s)$ be the probability that the system makes a transition to state j in exactly N_d slots, and that in $\{1, 2, \ldots, N_d\}$ N_t transmission attempts are performed and N_s transmission successes are counted, given that the system was in state i at time 0. Then, we have

$$\Phi_{ij}(z_d, z_t, z_s) = \sum_{N_d, N_t, N_s} \xi_{ij}(N_d, N_t, N_s) z_d^{N_d} z_t^{N_t} z_s^{N_s}.$$

Performance analysis - 1

- we compute steady-state performance according to renewal theory:

$$\text{throughput} = \frac{\sum_{i \in \Omega_X} \pi_i \sum_{j \in \Omega_X} P_{ij} S_{ij}}{\sum_{i \in \Omega_X} \pi_i \sum_{j \in \Omega_X} P_{ij} D_{ij}},$$

- the transition matrix of the embedded Markov chain is $\boldsymbol{P} = \boldsymbol{\Phi}(1, 1, 1)$
- knowledge of $\boldsymbol{P} = \boldsymbol{\Phi}(1, 1, 1)$ and of $\boldsymbol{D}, \boldsymbol{T}$ and $\boldsymbol{S}$ is sufficient

Performance analysis - 2

- the matrix of average delays can be found as

$$\boldsymbol{D} = \left.\frac{\partial \boldsymbol{\Phi}(z_d, z_t, z_s)}{\partial z_d}\right|_{z_d, z_t, z_s = 1} = \boldsymbol{D}_1 \boldsymbol{\Phi}^{(2)}(1,1,1) + \boldsymbol{\Phi}^{(1)}(1,1,1)\boldsymbol{D}_2,$$

where

$$\boldsymbol{D}_i = \left.\frac{\partial \boldsymbol{\Phi}^{(i)}(z_d, z_t, z_s)}{\partial z_d}\right|_{z_d, z_t, z_s = 1}, \quad i = 1, 2$$

The averages of the other quantities, $\boldsymbol{T}$, $\boldsymbol{S}$, can be found similarly.

Computation of $\boldsymbol{\Phi}^{(1)}(z)$ and $\boldsymbol{\Phi}^{(2)}(z)$

1. a set of mutually exclusive events is identified, exhausting all possibilities
2. for each of those events, based on the origin state and on the protocol rules:
   (a) the destination of the corresponding transition is identified
   (b) the transition function is computed
3. transitions corresponding to distinct events but leading to the same destination state are combined (i.e., the corresponding transition functions are added), to obtain $\boldsymbol{\Phi}^{(1)}(z)$ and $\boldsymbol{\Phi}^{(2)}(z)$.

Computation of $\Phi^{(1)}(z)$

- Let $X = X(k) = (C, W_{\text{th}}, W)$.
- for a given n, the window at time n is deterministically found:

$$Y = W(n) = w(n, W, W_{\text{th}}).$$

- transition function:

$$\Phi^{(1)}_{XY}(z_d, z_t, z_s) = \sum_{n \in \mathcal{C}(X,Y)} \alpha_C(n) z_d^n z_t^n z_s^{n-1}$$

where $\mathcal{C}(X, Y) = \{n : w(n, W, W_{\text{th}}) = Y\}$.

Besides running the basic window adaptation algorithm, the transmitter performs the following tasks which are related to packet losses:

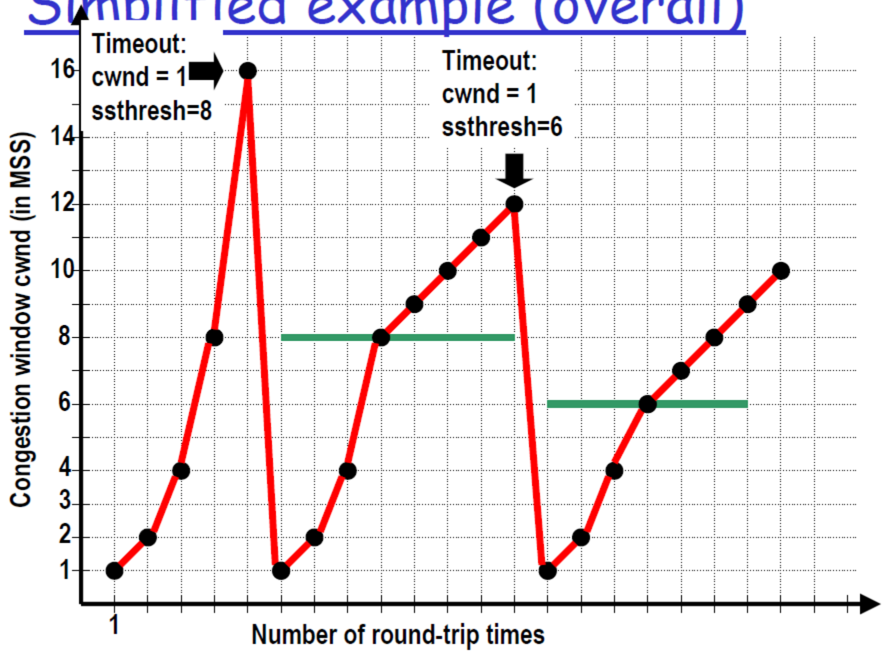
- *loss detection:* a mechanism by which the transmitter concludes (correctly or incorrectly) that a packet was lost
- *loss recovery phase:* a mechanism which allows the protocol to recover lost packets through retransmission
- *window adaptation during loss recovery:* the way window adaptation is handled while lost packets are being recovered (different than the basic window adaptation in general).

The above procedures are implemented in TCP OldTahoe, Tahoe, Reno, and NewReno as follows.

- In the case of OldTahoe, loss detection and recovery is performed only through timeout and retransmission. Window adaptation during loss recovery follows the basic algorithm.
- In the case of Tahoe, in addition to the regular timeout mechanism, a *fast retransmit* procedure is implemented for loss detection. If subsequent to a packet loss, the transmitter receives the Kth duplicate ACK at time t, before the timer expires, then the transmitter behaves as if a timeout has occured and begins retransmission, with $W(t^+)$ and $W_{th}(t^+)$ as given in the basic window adaptation algorithm.

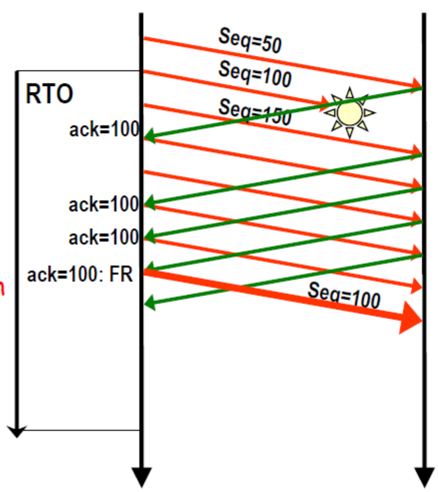
- In the case of Reno also, the fast retransmit procedure following a packet loss is implemented. However, the subsequent recovery procedure is different. If the Kth duplicate ACK is received at time t, then $W_{th}(t^+)$ is set to $\lceil W(t)/2 \rceil$, and $W(t^+)$ is set to $W_{th}(t^+) + K$ instead of 1 (the addition of K accounts for the K packets that have successfully left the network). The Reno transmitter then retransmits only the first lost packet. As the transmitter waits for the ACK for the first lost packet retransmission, it may get duplicate ACK's for the outstanding packets. The receipt of each of such duplicate ACK causes $W(t)$ to be incremented by 1. If there was only a single packet loss in the loss window, then the ACK for its retransmission will complete the loss recovery; W at this time would be set to W_{th}, and the transmission resumes according to the basic window control algorithm. If there are multiple packet losses in the loss window, then the ACK for the first lost packet retransmission will advance the left edge of the window, A, by an amount equal to 1 plus the number of good packets between the first lost packet and the next one. In this case, if the loss recovery is not successful due to lack of the duplicate ACK's necessary to trigger multiple fast retransmits, then a timeout has to be waited for.

Simplified example (overall)

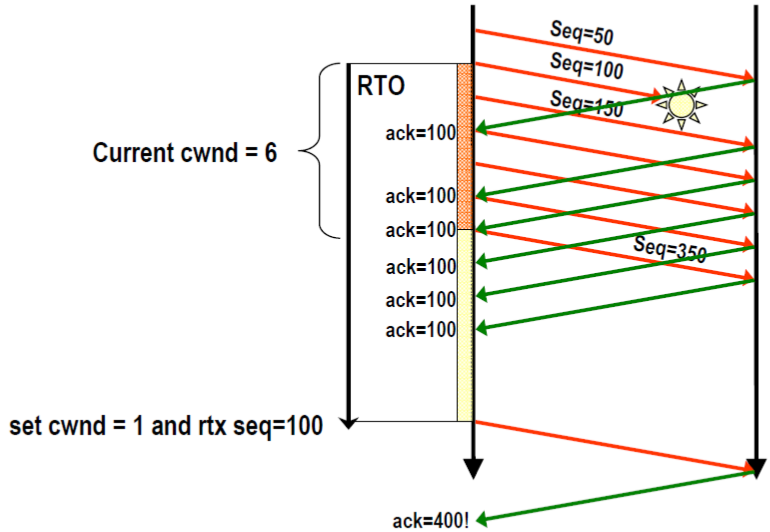


The Fast Retransmit Algorithm

- ➔ Idea: use duplicate ACKs!
  - ⇨ Receiver responds with an ACK every time it receives an out-of-order segment
  - ⇨ ACK value = last correctly received segment
- ➔ FAST RETRANSMIT algorithm:
  - ⇨ if 3 duplicate acks are received for the same segment, assume that the next segment has been lost. Retransmit it right away.
  - ⇨ Helps if single packet lost. Not very effective with multiple losses
- ➔ And then? A congestion control issue…

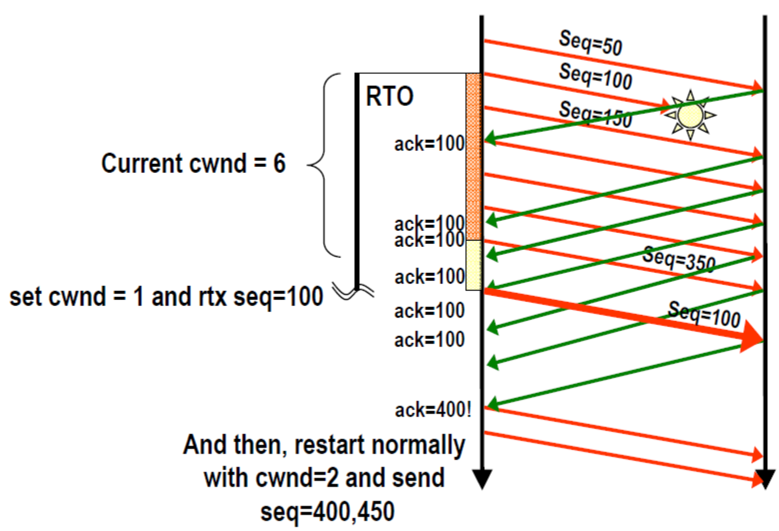


What happens AFTER RTO?
(without fast retransmit)



And then, restart normally with cwnd=2 and send seq=400,450

TCP TAHOE
(with fast retransmit)



Same as before, but shorter time to recover packet loss!

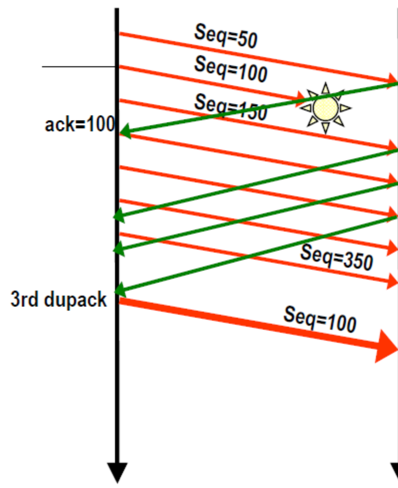
Motivations for fast recovery

FAST RECOVERY:

- The phase following fast retransmit (3 duplicate acks received)

- TAHOE approach: slow start, to protect network after congestion

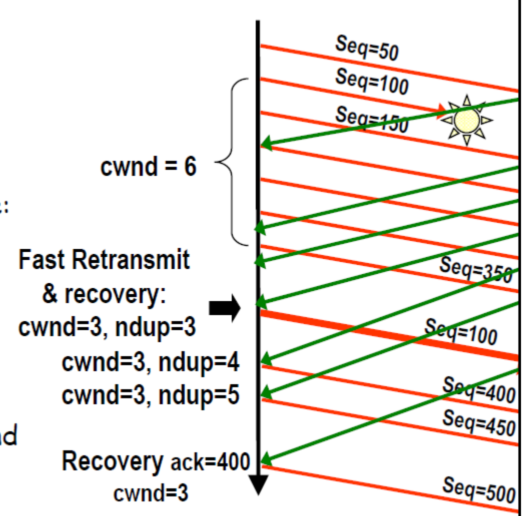
- However, since subsequent acks have been received, no hard congestion situation should be present in the network: slow start is a too conservative restart!



TCP RENO: Fast recovery rules

FAST RECOVERY RULES:

- Retransmit lost segment
- **Set cwnd = cwnd/2**
- **Restart with congestion avoidance (linear)**
- start fast recovery phase:
  - Set counter for duplicate packets ndup=3
  - Use "inflated" window: w = cwnd+ndup
  - Upon new dup_acks, increase ndup, not cwnd (and send new data)
  - Upon recovery ack, "deflate" window setting ndup=0



E. Computation of $\mathbf{\Phi}^{(2)}(z)$ for TCP Reno

The second part of the cycle can also be fully characterized by appropriately labeling transitions and counting events. Unlike in the previous case, $\mathbf{\Phi}^{(2)}(z)$ does depend on the way the different TCP versions handle packet loss recovery, and therefore it must be computed separately in the various cases. We address the case of TCP Reno in this subsection.

For simplicity of notation, in what follows we let $n = 0$, so that the first slot in the second phase corresponds to time 1. Define $\varphi_{ij}(k, x)$ as the probability that there are k successes in slots 1 through x and that the channel is in state j at time x, given that the channel was in state i at time 0. Both a recursive technique and explicit expressions for these probabilities are given in [24]. Note that the functions ϕ in that paper are defined in a slightly different way. However, it is straightforward to relate them by noting that $\varphi_{BG}(j, x) = \phi_{10}(j-1, x)$ and $\varphi_{GB}(j, x) = \phi_{01}(j+1, x)$, whereas in the other two cases they are the same.

1) The Case of $\lfloor Y \rfloor \leq K$: If $\lfloor Y \rfloor \leq K$, fast retransmit cannot be triggered, since after the packet lost at time 0, only $\lfloor Y \rfloor - 1 < K$ more packets can be transmitted, and K duplicate ACK's will never be received. In this case, timeout timer will expire and the lost packet will be retransmitted in slot $t_{k+1} = T_o$. Note that the value of the window size at timeout will still be equal to Y (recall that duplicate ACK's do not advance the window), so that after timeout the algorithm will set $W_{th} = \lceil Y/2 \rceil, W = 1$. This event will therefore lead to state $X(k+1) = (C, \lceil Y/2 \rceil, 1)$ with transition function

$$p_{BC}(T_o - 1) z_d^{T_o - 1} z_t^{\lfloor Y \rfloor - 1} z_s^{N_s}. \qquad (11)$$

$0 \leq E[N_s] \leq \sum_{i=1}^{\lfloor Y \rfloor -1} p_{BG}(i)$, where $p_{BG}(i)$ is the i-step transition probability of the Markov channel, computed as

2) The Case of $\lfloor Y \rfloor > K$: Let us now assume that $\lfloor Y \rfloor > K$. The following two cases can occur.

Case 1—Fast Retransmit is Not Triggered: If fewer than K slots in $\{1, 2, \cdots, \lfloor Y \rfloor - 1\}$ are successful, fast retransmit will not be triggered. The destination values of W_{th} and W will be as in the previous case. Let $\mathcal{A}(C, j)$ be the event that there are j successful slots in $\{1, 2, \cdots, \lfloor Y \rfloor - 1\}$ and that the channel state in slot $\lfloor Y \rfloor - 1$ is C. From the Markov channel characterization, the following probabilities can be derived: $P[\mathcal{A}(G, j)] = \varphi_{BG}(j, \lfloor Y \rfloor - 1)$ and $P[\mathcal{A}(B, j)] = \varphi_{BB}(j, \lfloor Y \rfloor - 1)$. The transition function leading from Y to state $X(k+1) = (C, \lceil Y/2 \rceil, 1)$ is then given by

$$p_{BC}(T_o - \lfloor Y \rfloor) z_d^{T_o - 1} z_t^{\lfloor Y \rfloor - 1} \sum_{j=0}^{K-1} P[\mathcal{A}(B, j)] z_s^{N_s(B,j)}$$
$$+ p_{GC}(T_o - \lfloor Y \rfloor) z_d^{T_o - 1} z_t^{\lfloor Y \rfloor - 1} \sum_{j=1}^{K-1} P[\mathcal{A}(G, j)] z_s^{N_s(G,j)} \tag{13}$$

where $0 \le N_s(B, j), N_s(G, j) \le j$ and the two terms account for the two possibilities for the channel state at time $\lfloor Y \rfloor - 1$. The sums are limited to $K - 1$ rather than $\lfloor Y \rfloor - 1$ since the number of successes must be less than K for the considered case of fast retransmit not triggered.

Case 2—Fast Retransmit is Triggered: If the Kth duplicate ACK is received, fast retransmit is triggered right after the Kth successful slot in $\{1, 2, \cdots, \lfloor Y \rfloor - 1\}$. Let $\mathcal{B}(K)$ be the event that the packet failure at time 0 is followed by K consecutive successes, and let $\mathcal{B}(i, \ell_1), K < i < \lfloor Y \rfloor, 0 < \ell_1 \le K$ be the event that the Kth success occurs at time i and the first loss after the loss in 0 occurs at time ℓ_1 (note that since $i > K$, there must be a packet loss before the Kth success). The probabilities of these events are given as follows:

$$P[\mathcal{B}(K)] = p_{BG} p_{GG}^{K-1} \tag{14}$$

$$P[\mathcal{B}(i, \ell_1)] = \begin{cases} p_{BB} \varphi_{BG}(K, i-1), \\ \quad \ell_1 = 1;\ i = K+1, \cdots, \lfloor Y \rfloor - 1 \\ p_{BG} p_{GG}^{\ell_1 - 2} p_{GB} \varphi_{BG}(K - \ell_1 + 1, i - \ell_1), \\ \quad \ell_1 = 2, \cdots, K;\ i = K+1, \cdots, \lfloor Y \rfloor - 1. \end{cases} \tag{15}$$

Case 2a: Consider first the occurrence of the event $\mathcal{B}(K)$. Since at time K the Kth duplicate ACK is received, retransmission of that packet is performed in slot $K+1$.

- If this retransmission is successful, the loss recovery phase is successfully completed, and a new cycle starts at time $K+2$. In this case, the destination state is $X(k+1) = (G, \lceil Y/2 \rceil, \lceil Y/2 \rceil)$[5] and the transition function is given by

$$P[\mathcal{B}(K)] p_{GG} z_d^{K+1} z_t^{K+1} z_s^{K+1}. \tag{16}$$

- If, on the other hand, the retransmission is a failure, the protocol will stop and wait for an ACK which will never be transmitted, and timeout will eventually resolve the deadlock. In this case, according to the TCP Reno rules, upon receiving the Kth duplicate ACK the window size will be updated to $W' = \min\{\lceil Y/2 \rceil + K, W_{\max}\}$, so that the new state after timeout is $X(k+1) = (C, \lceil W'/2 \rceil, 1)$, with transition function

$$P[\mathcal{B}(K)] p_{GB} p_{BC} (T_o - K - 2) z_d^{T_o - 1} z_t^{K+1} z_s^{K}. \tag{17}$$

Case 2b: Consider the occurrence of the event $\mathcal{B}(i, \ell_1)$. Successful loss recovery is not possible here, since in TCP Reno, multiple losses in a congestion window lead to deadlock and consequent timeout.

- If the retransmission at time $i+1$ is a failure, a behavior similar to the previous case can be observed, i.e., the next cycle will start in state $X(k+1) = (C, \lceil W'/2 \rceil, 1)$, with transition function given by

$$P[\mathcal{B}(i, \ell_1)] p_{GB} p_{BC} (T_o - i - 2) z_d^{T_o - 1} z_t^{i+1} z_s^{N_s} \tag{18}$$

where $\ell_1 - 1 \leq N_s \leq K$. Note, in fact, that the $\ell_1 - 1$ successful packets consecutively transmitted after the loss at time 0 will be acknowledged (without ever being retransmitted) when that lost packet is eventually received successfully, so that $\ell_1 - 1 \leq N_s$. Also, since K packets were successfully transmitted, in the best case they will all be acknowledged without being retransmitted, i.e., $N_s \leq K$.

- On the other hand, if the retransmission at time $i+1$ is successful, all packets preceding the one transmitted at time ℓ_1 are acknowledged, and the system will timeout at the end of slot $T_o + \ell_1 - 1$. In this case, the window size at that time will be $W'' = \min\{\lceil Y/2 \rceil + K + 1, W_{\max}\}$ (the ACK for the successful retransmission causes the window to be further increased by one with respect to the previous case), so that the next cycle will start in state $X(k+1) = (C, \lceil W''/2 \rceil, 1)$ with transition function

$$P[\mathcal{B}(i, \ell_1)] p_{GG} p_{GC}(T_o + \ell_1 - i - 2) \times z_d^{\ell_1 + T_o - 1} z_t^{i+1} z_s^{N_s} \quad (19)$$

where $\ell_1 \leq N_s \leq K + 1$. Note in fact that, in this case, the number of successes to be counted is at least one more than before, accounting for the successful retransmission at time $i+1$, but cannot be larger than $K+1$.

Consider OldTahoe first. $\Phi^{(1)}(z)$ is found as for TCP Reno. For $\Phi^{(2)}(z)$, note that the second part of the cycle, initiated by the loss at time 0, has duration which is deterministically equal to T_o, since every loss can only be recovered by timeout. The next cycle then starts in state $X(k+1) = (C, \lceil Y/2 \rceil, 1)$ with transition function

$$p_{BC}(T_o - 1) z_d^{T_o - 1} z_t^{\lfloor Y \rfloor - 1} z_s^{N_s} \quad (21)$$

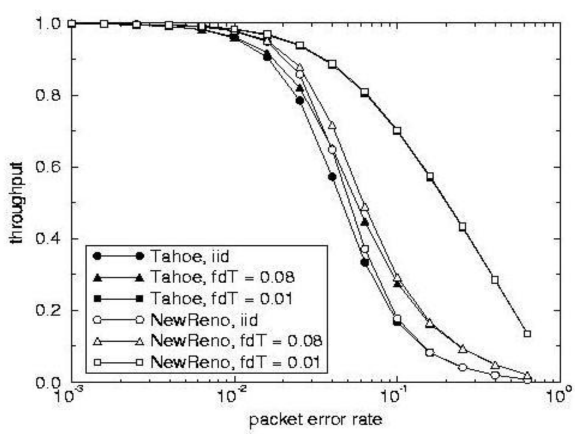
where $N_s \geq 0$ is upper-bounded by the number of successful slots in $\{1, 2, \cdots, \lfloor Y \rfloor - 1\}$, so that, in particular, $0 \leq E[N_s] \leq \sum_{i=1}^{\lfloor Y \rfloor - 1} p_{BG}(i)$.

Let us now focus on Tahoe. In this case, again, $\Phi^{(1)}(z)$ is the same as before. Regarding the computation of $\Phi^{(2)}(z)$, we first observe that all the events considered for TCP Reno and corresponding to no fast retransmit still apply in the case (note, in fact, that Tahoe and Reno differ only *after* fast retransmit is triggered). Therefore, we only need consider here the case in which fast retransmit is triggered, i.e., the case in which the Kth successful transmission occurs at time $i < \lfloor Y \rfloor$. The next cycle then starts in slot $i+1$ and in state $X(k+1) = (G, \lceil Y/2 \rceil, 1)$, with transition function

$$\varphi_{BG}(K,i) z_d^i z_t^i z_s^{N_s} \tag{22}$$

where $0 \leq N_s \leq K$.

Example of performance



Throughput comparison of TCP Tahoe and NewReno at different values of $f_D T$. $W_{max} = 24$. $K = 3$. $MTO = 100$.

References

- Howard, Dynamic probabilistic systems, Wiley 1971
- Zorzi-Rao-Milstein, IEEE Trans. Comm., Nov. 1998
- Zorzi & Rao, IEEE Trans. Comm., Sep. 1996
- Zorzi & Rao, IEEE Trans. Computer, Mar. 1997
- **M. Zorzi, A. Chockalingam, R.R. Rao, "Throughput Analysis of TCP on Channels with Memory," IEEE J. Selected Areas Comm., Jul. 2000.**