

Advanced Networks

Transport layer: TCP

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Dr. Wei Bao | Lecturer
School of Computer Science



THE UNIVERSITY OF
SYDNEY

Go-back-N:

- › sender can have up to N unacked packets in pipeline

- › receiver only sends

cumulative ack <https://powcoder.com>

- does not ack packet if there is a gap

- › sender has timer for oldest unacked packet

- when timer expires, retransmit *all* unacked packets

Selective Repeat:

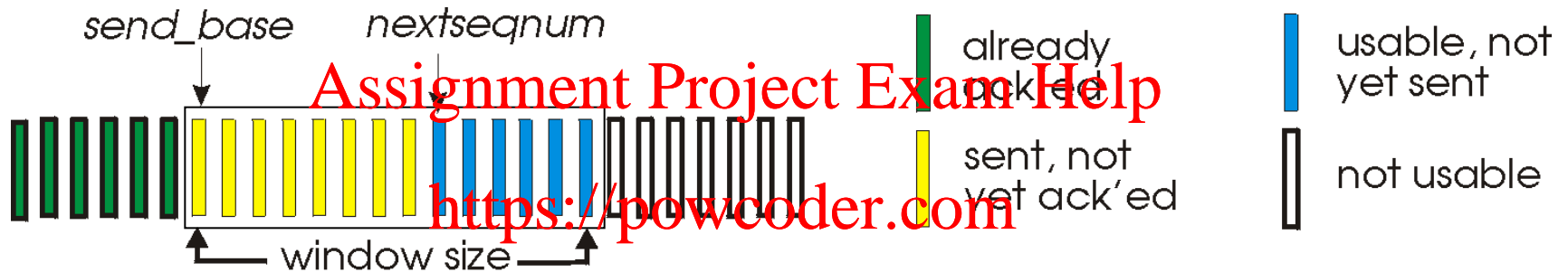
- › sender can have up to N unacked packets in pipeline

- › receiver sends *individual ack* for each packet

<https://powcoder.com>
sender maintains timer for each unacked packet

- when timer expires, retransmit only that unacked packet

- › “window” of up to N , consecutive unacked pkts allowed



Add WeChat powcoder

- ❖ ACK(n): ACKs all pkts up to, including seq # n - “*cumulative ACK*”
 - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖ $timeout(n)$: retransmit packet n and all higher seq # pkts in window

- › “window” of up to N, consecutive unacked pkts allowed

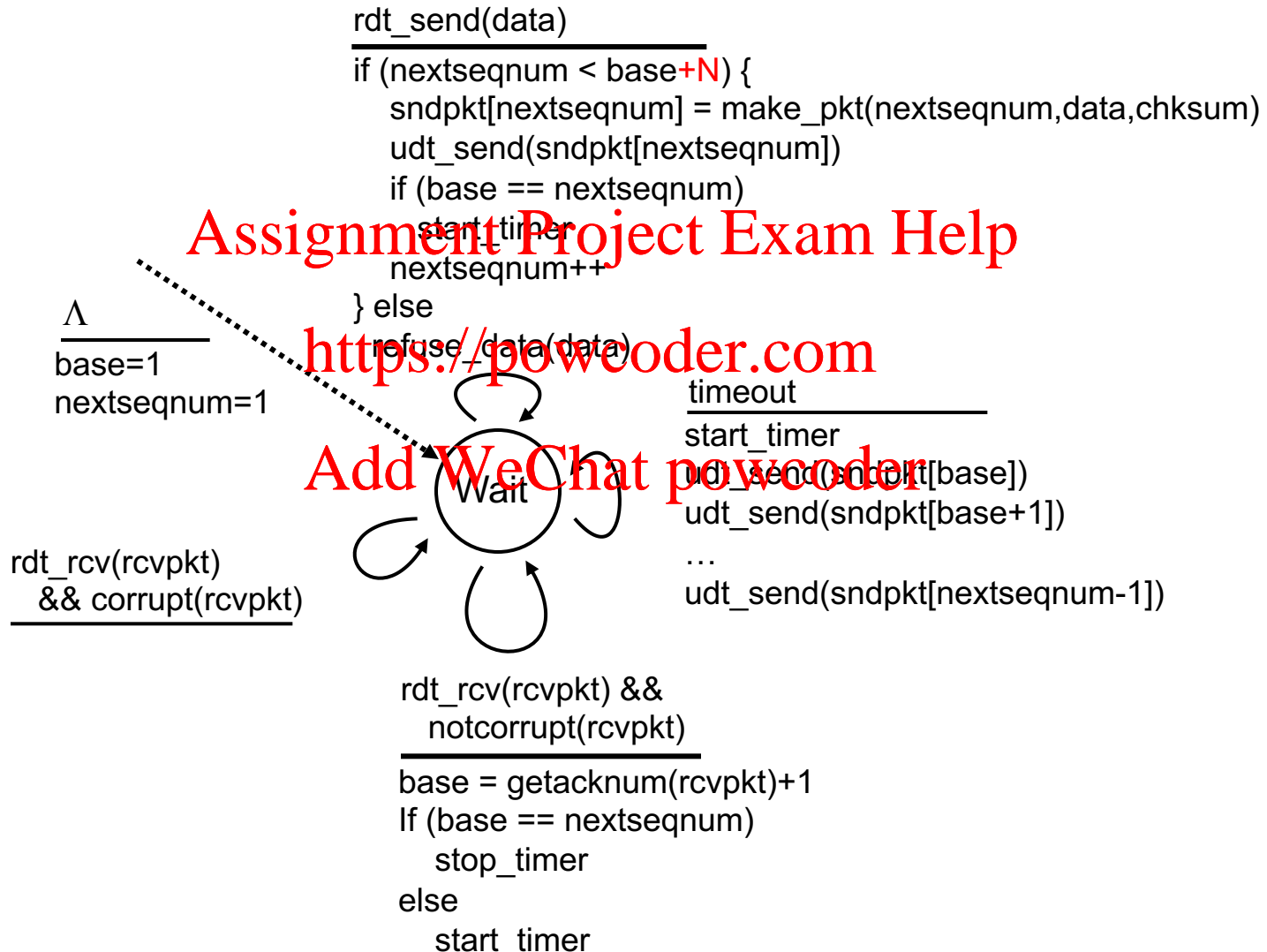


Add WeChat powcoder

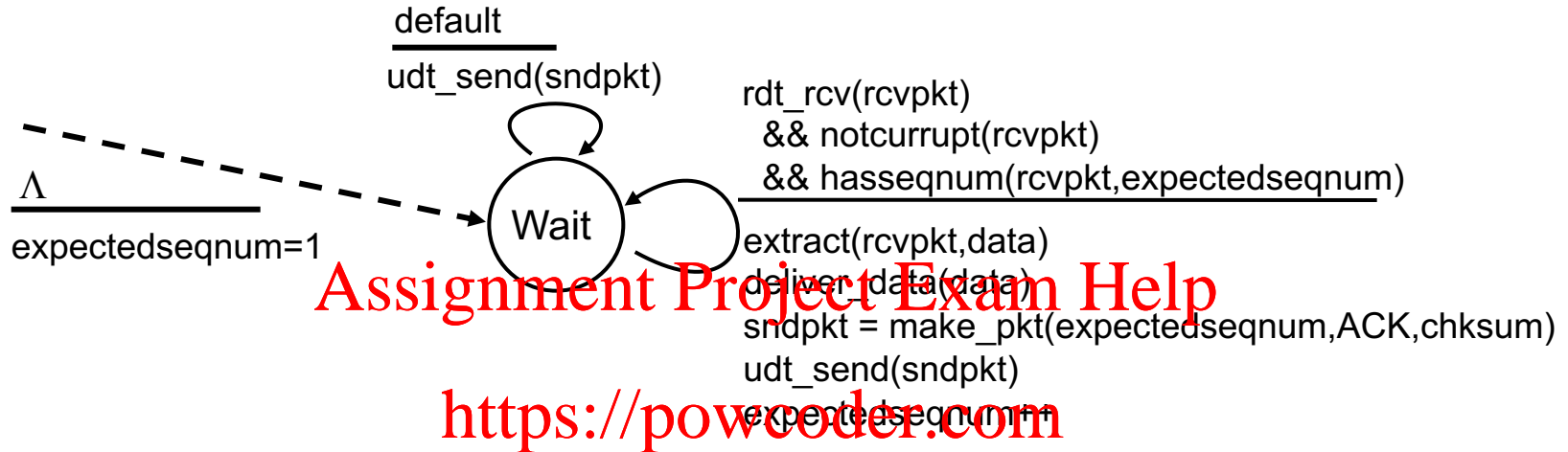
- ❖ ACK(n): ACKs all pkts up to, including seq # n - “*cumulative ACK*”
 - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖ *timeout(n)*: retransmit packet n and all higher seq # pkts in window



GBN: sender extended FSM



GBN: receiver extended FSM



ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **expectedseqnum**
- › out-of-order pkt:
 - discard (don't buffer): *no receiver buffering!*
 - re-ACK pkt with highest in-order seq #



GBN in action

sender window (N=4)

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

rcv ack0, send pkt4
rcv ack1, send pkt5

ignore duplicate ACK



pkt 2 timeout

send pkt2
send pkt3
send pkt4
send pkt5

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

receive pkt4, discard,
(re)send ack1

receive pkt5, discard,
(re)send ack1

rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

X/loss

- › receiver *individually* acknowledges all correctly received pkts
 - buffers pkts as needed for eventual in-order delivery to upper layer
- › sender only resends pkts for which ACK not received
 - sender timer for each unACKed pkt
- › sender window
- › receiver window

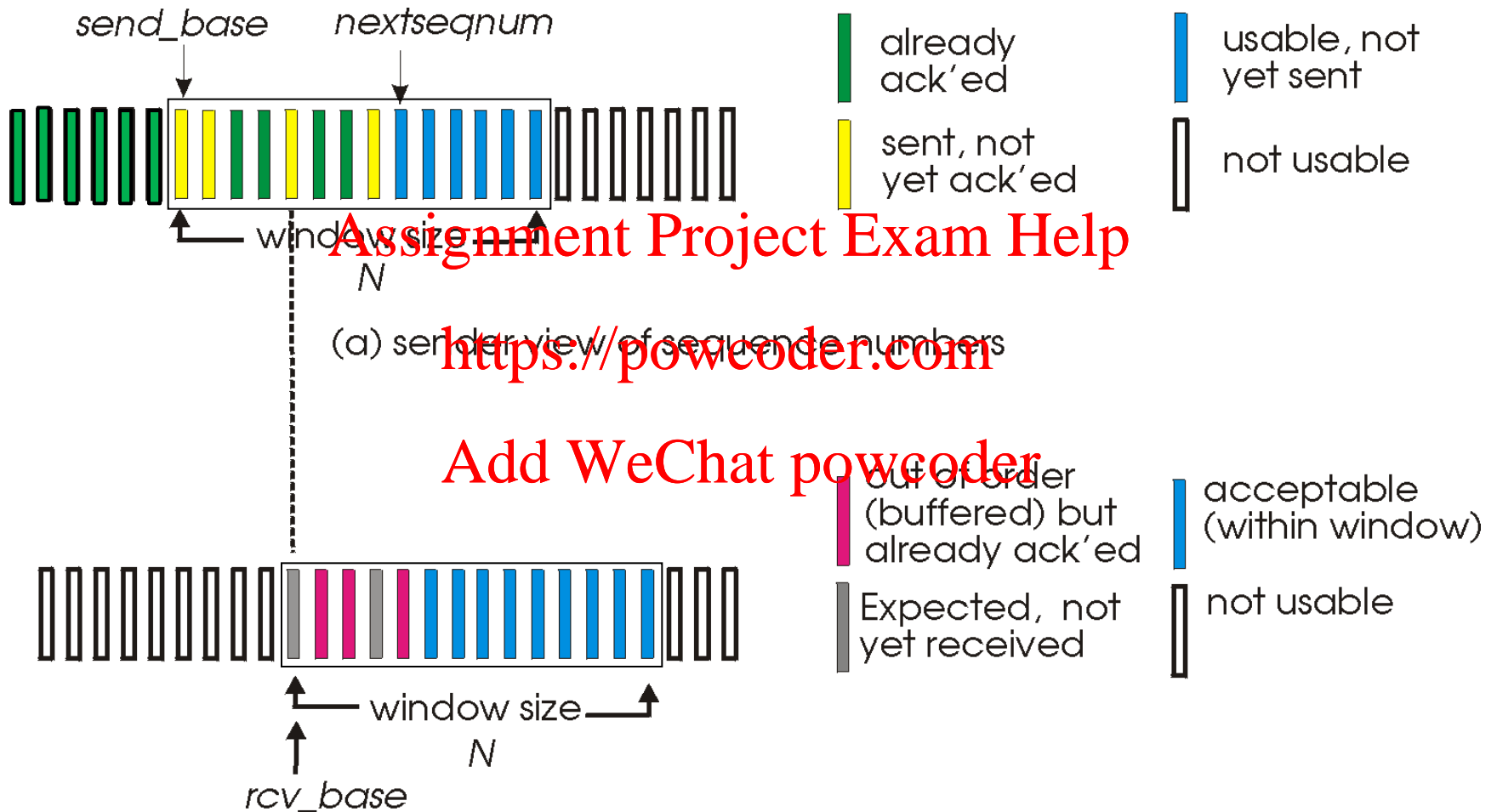
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Selective repeat: sender, receiver windows



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

sender

data from above:

- › if next available seq # in window, send pkt

timeout(n):

- › resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N-1]:

- › mark pkt n as received
- › if n is smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- ❖ send ACK(n)
- ❖ out-of-order: buffer
- ❖ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ❖ ACK(n)

otherwise:

- ❖ ignore



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Selective repeat in action

sender window (N=4)

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8 9

sender

send pkt0

send pkt1

send pkt2

send pkt3

(wait)

rcv ack0, send pkt4

rcv ack1, send pkt5

record ack3 arrived



pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

receiver

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4

receive pkt5, buffer,
send ack5

rcv pkt2; deliver pkt2,
pkt3, pkt4, pkt5; send ack2

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

X/loss

Q: what happens when ack2 arrives?



Connection-oriented Transport TCP

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

› point-to-point:

- one sender, one receiver

› reliable, in-order byte stream

› pipelined:

- TCP congestion and flow control set window size

› full duplex data:

- bi-directional data flow in same connection

› MSS: maximum segment size

› connection-oriented:

- handshaking (exchange of control msgs) inits sender, receiver state before data exchange

› flow controlled:

- sender will not overwhelm receiver

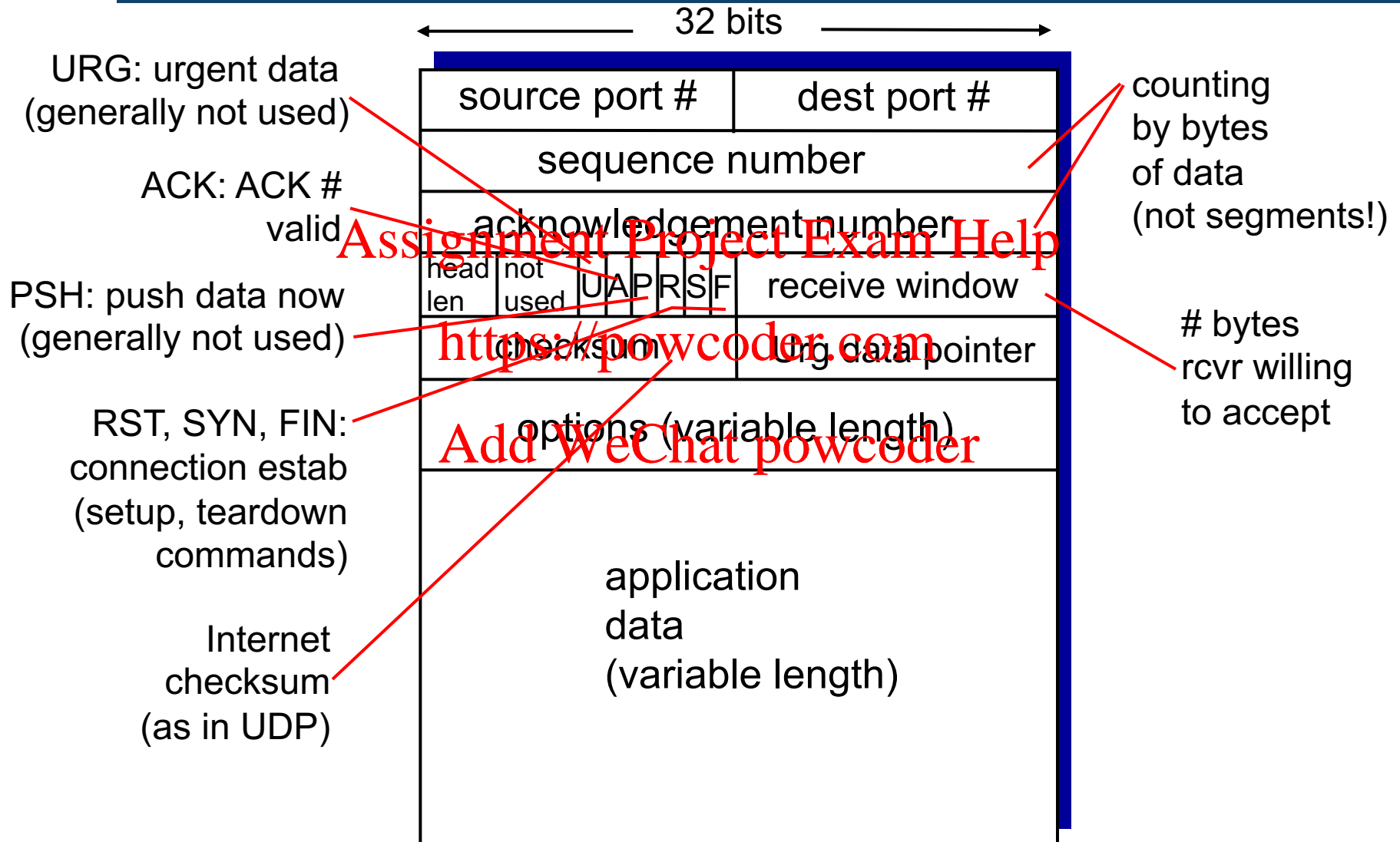
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

- “number” of first byte in segment’s data

acknowledgements:

- seq # of **next byte** expected from other side
- cumulative ACK

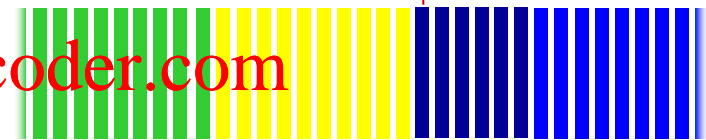
Q: how receiver handles out-of-order segments

- A: TCP spec doesn’t say,
- up to implementor
- Most will store, but still use cumulative ACK

outgoing segment from sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

window size
N



sender sequence number space

sent
ACKed

sent, not-
yet ACKed
("in-
flight")

usable
but not
yet sent

not
usable

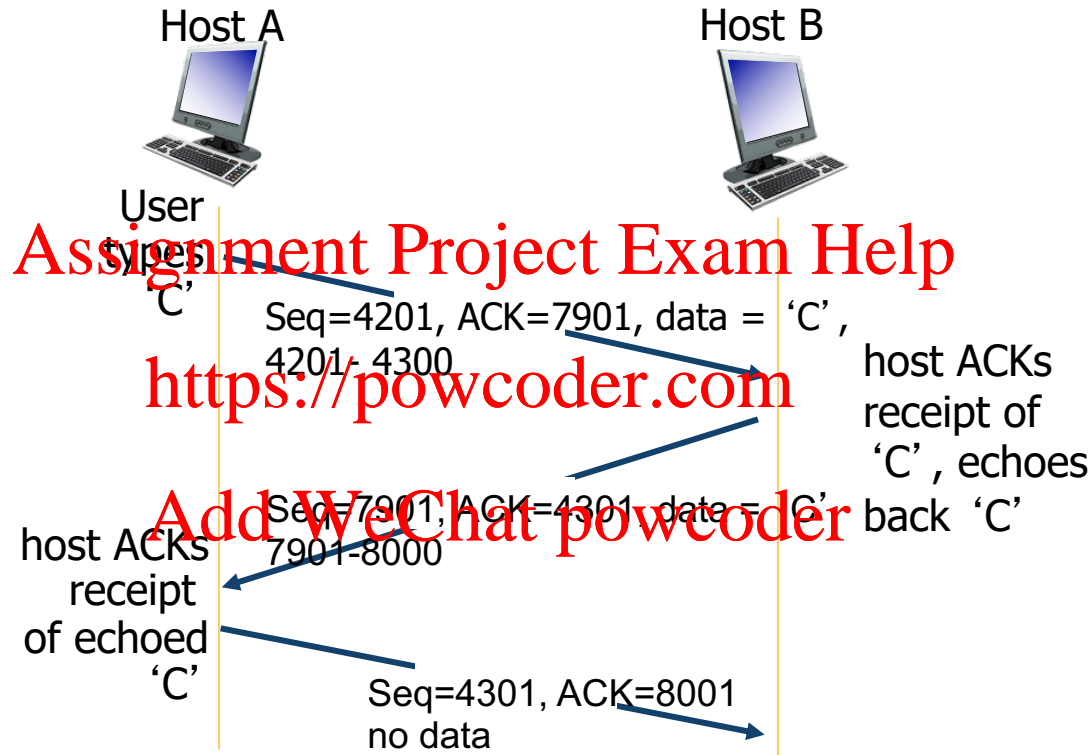
incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

A



TCP seq. numbers, ACKs



simple telnet scenario

Assignment Project Exam Help

<https://powcoder.com>

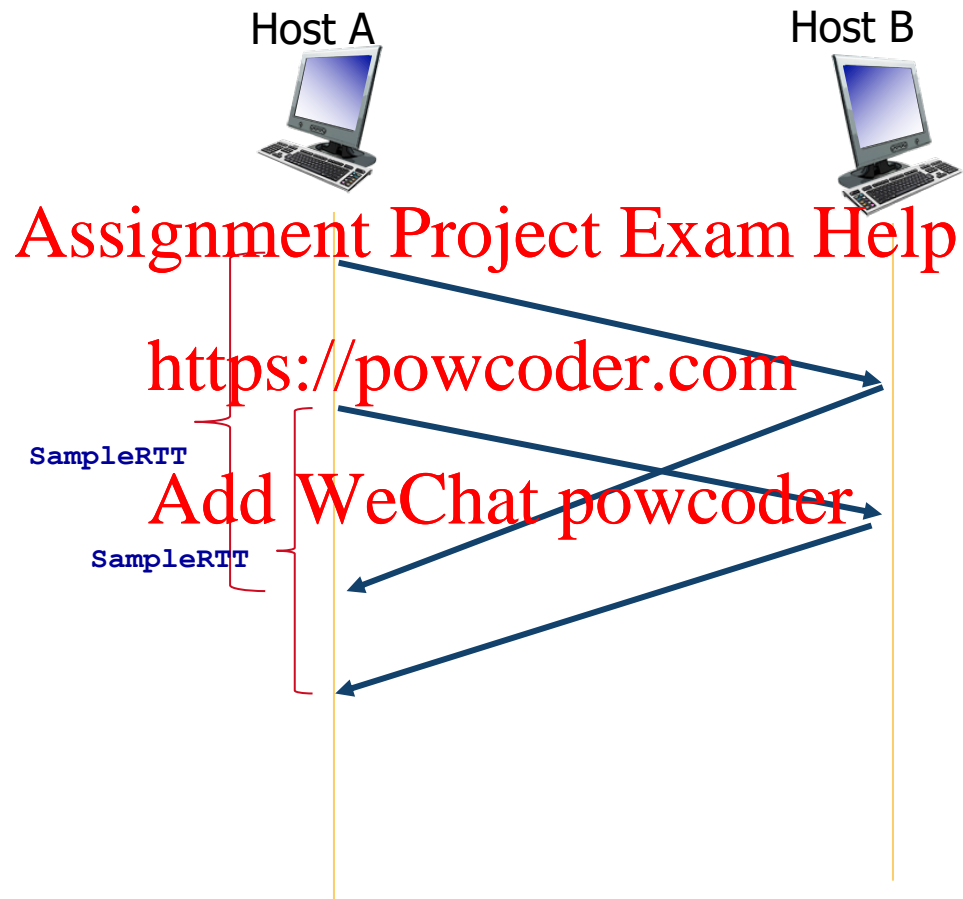
Add WeChat powcoder

Q: how to set TCP timeout value?

- › longer than RTT
 - but RTT varies
- › *too short*: premature timeout, unnecessary retransmissions
- › *too long*: slow reaction to segment loss

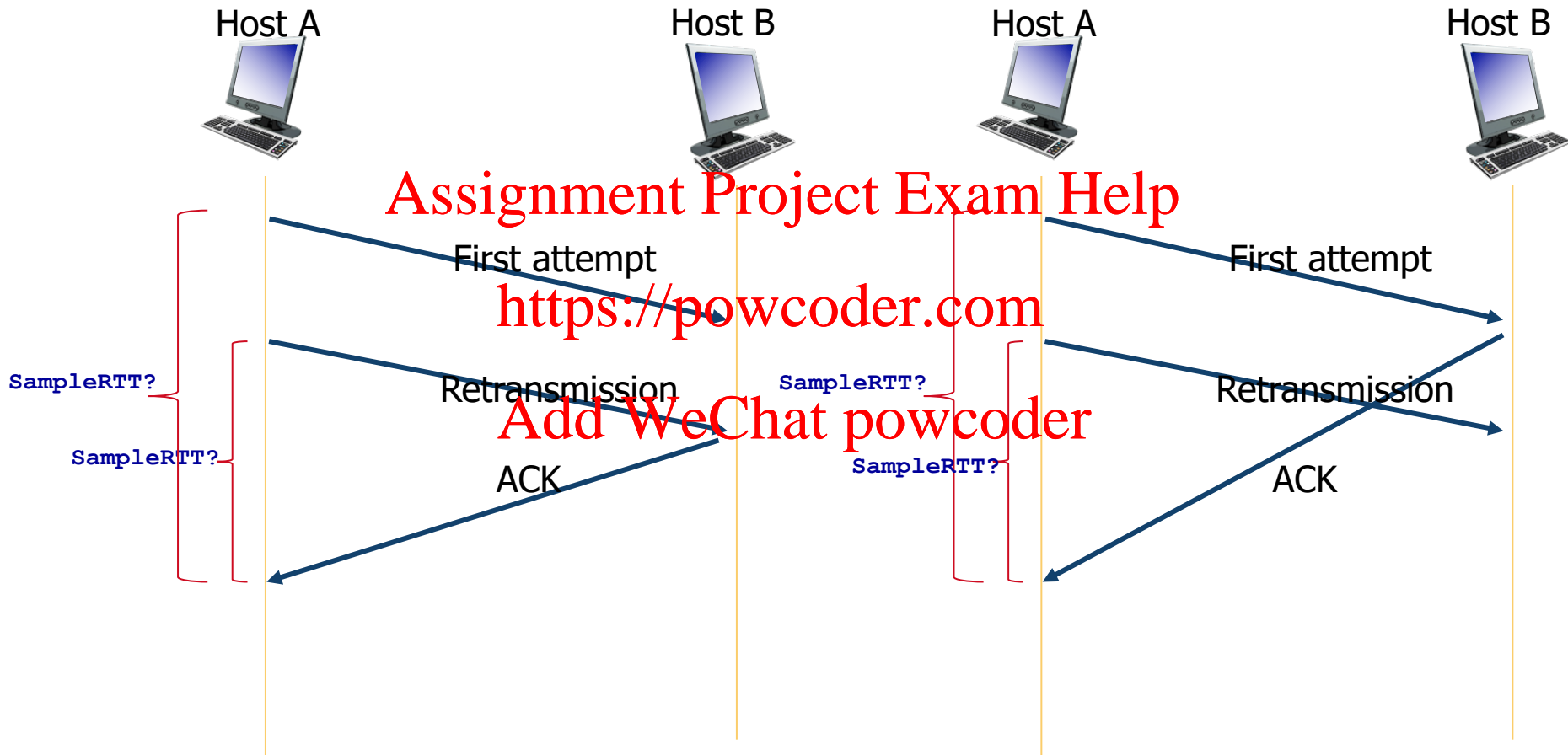
Q: how to estimate RTT?

- › **SampleRTT**: measured time from segment transmission until ACK receipt
 - ignore retransmissions
- › **SampleRTT** will vary, want estimated RTT “smoother”
 - weighted average of several recent measurements, not just current **SampleRTT**





Ignore retransmissions

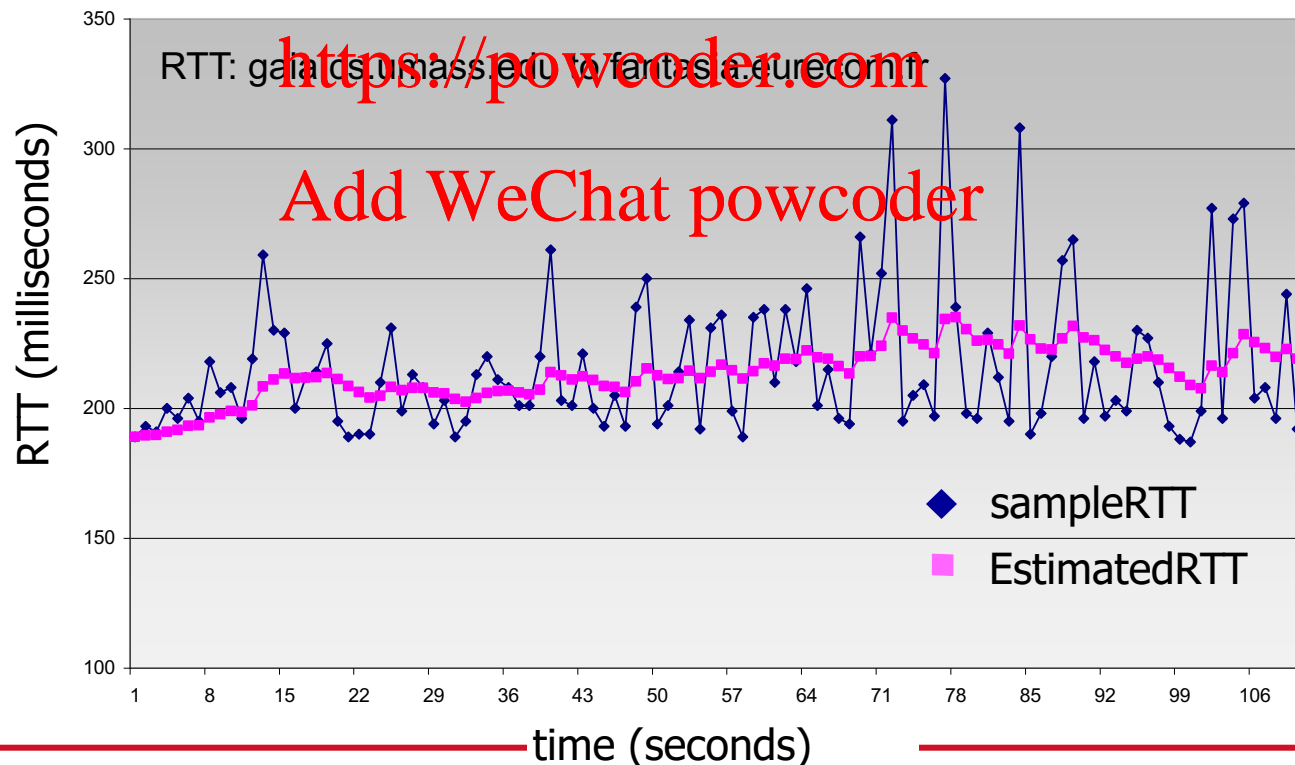


TCP round trip time, timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❖ exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical $\alpha = 0.25$

Assignment 0. Project Exam Help



› **timeout interval:** **EstimatedRTT** plus “safety margin”

- large variation in **EstimatedRTT** → larger safety margin

› estimate SampleRTT deviation from EstimatedRTT:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

Add WeChat powcoder

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



↑
estimated RTT

↑
“safety margin”



Reliable Data Transfer in TCP

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

- › TCP creates rdt service on top of IP's unreliable service

Assignment Project Exam Help

- pipelined segments
 - cumulative acks
 - single retransmission timer
- let's initially consider simplified TCP sender:
- ignore duplicate acks
 - ignore flow control, congestion control
- › retransmissions triggered by:
 - timeout events
 - duplicate acks

<https://powcoder.com>

Add WeChat powcoder

data rcvd from app:

- › create segment with seq #
- › seq # is byte-stream number of first data byte in segment
- › start timer if not already running
 - think of timer as for oldest unacked segment
 - expiration interval: `TimeoutInterval`

timeout:

- › retransmit segment that caused timeout

*restart timer**ack rcvd:*

- › if ack acknowledges previously unacked segments

- update what is known to be ACKed
- start timer if there are still unacked segments

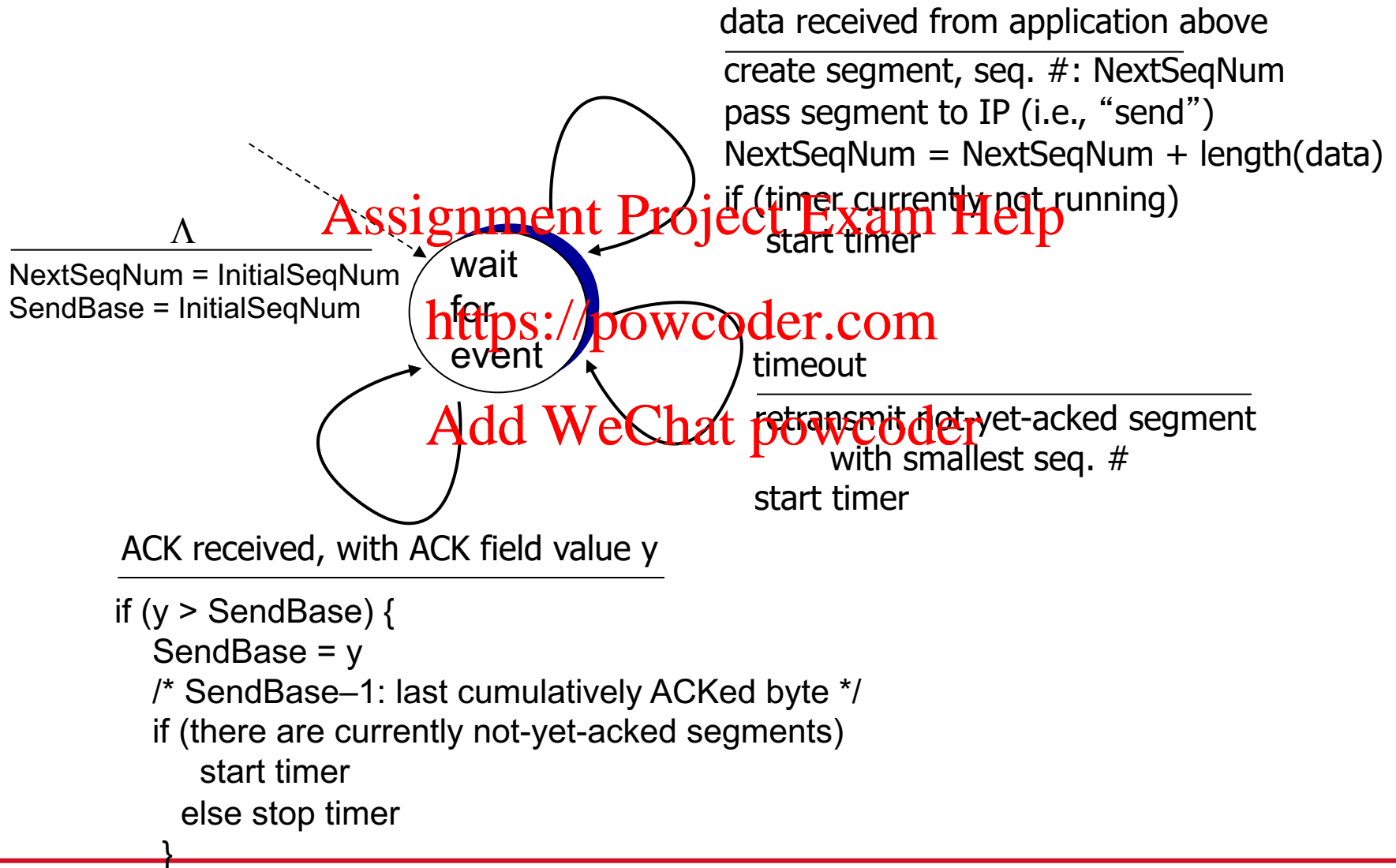
Assignment Project Exam Help

<https://powcoder.com>

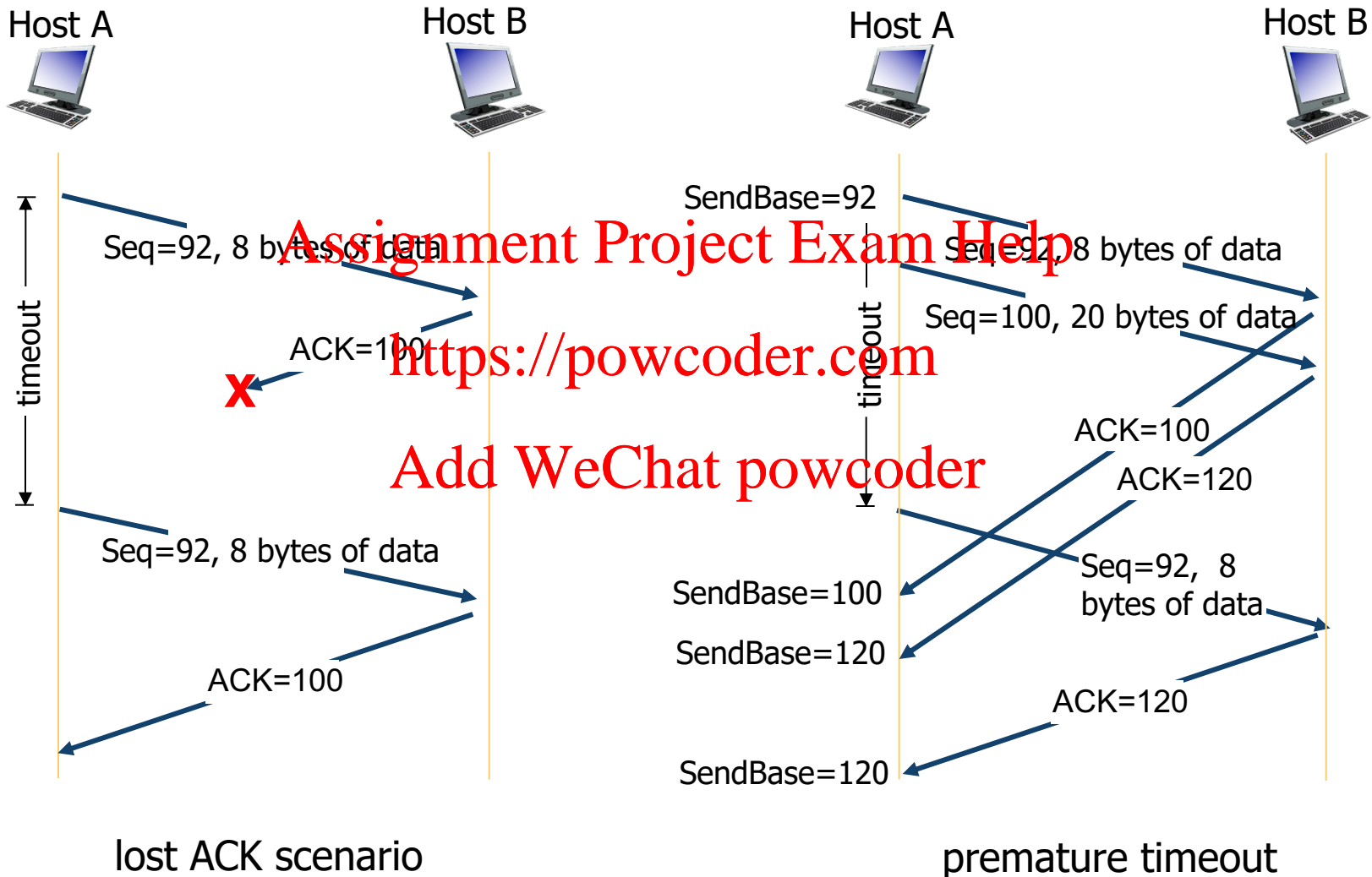
Add WeChat powcoder



TCP sender (simplified)



TCP: retransmission scenarios



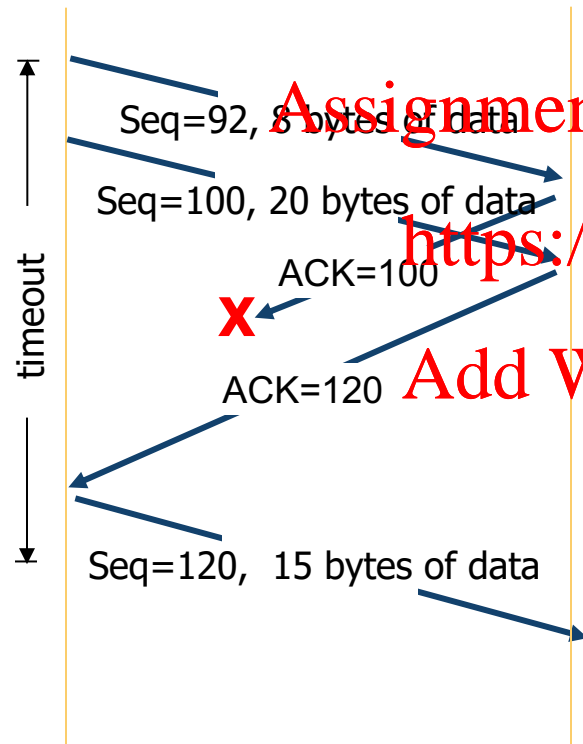


TCP: retransmission scenarios

Host A



Host B



cumulative ACK

Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder



TCP ACK generation [RFC 1122, RFC 2581]

event at receiver

TCP receiver action

arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed

delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK

arrival of in-order segment with expected seq #. One other segment has ACK pending

immediately send single cumulative ACK, ACKing both in-order segments

arrival of out-of-order segment higher-than-expect seq. # . Gap detected

immediately send *duplicate ACK*, indicating seq. # of next expected byte

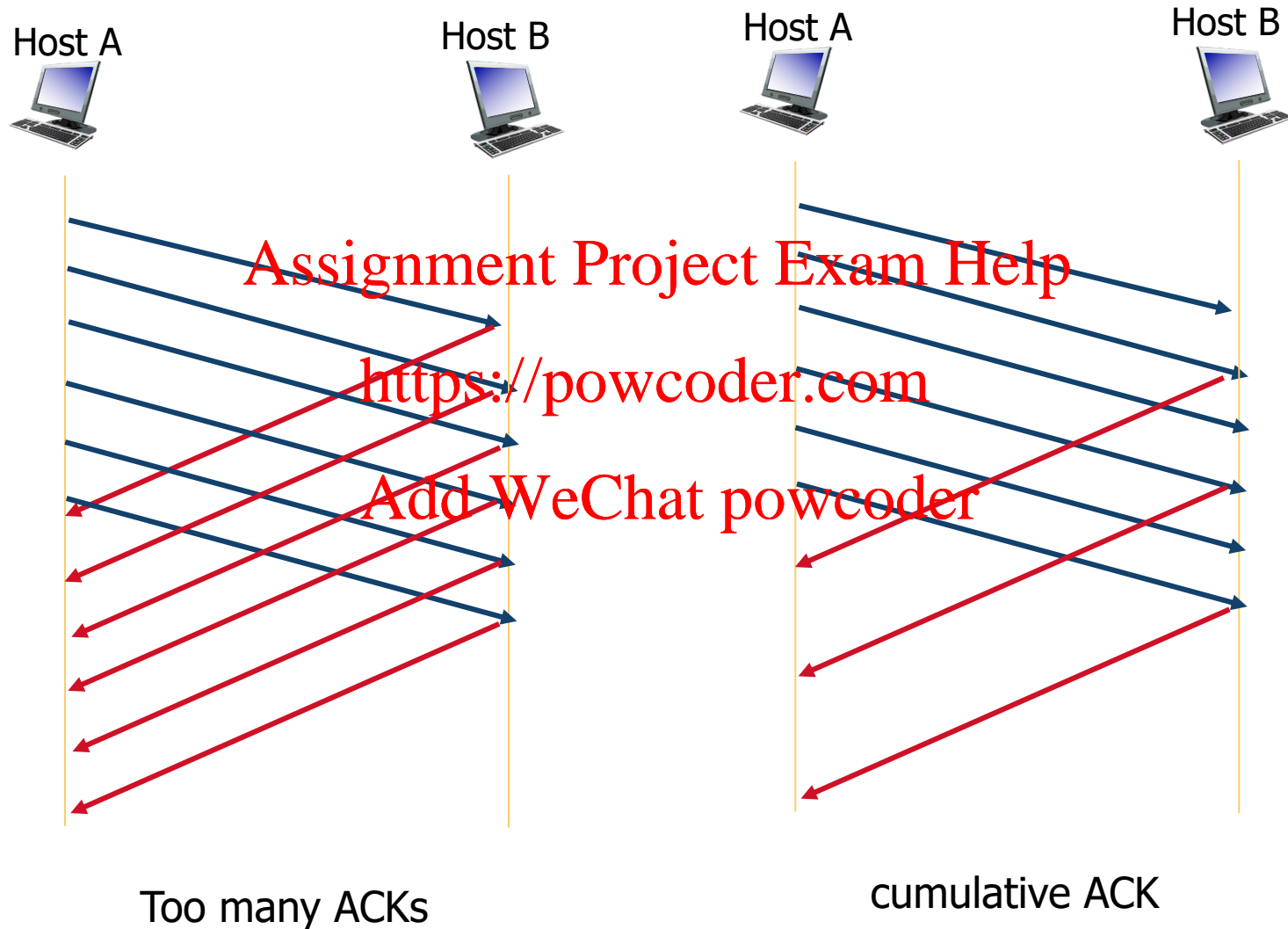
arrival of segment that partially or completely fills gap

immediate send ACK, provided that segment starts at lower end of gap

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder





Host A



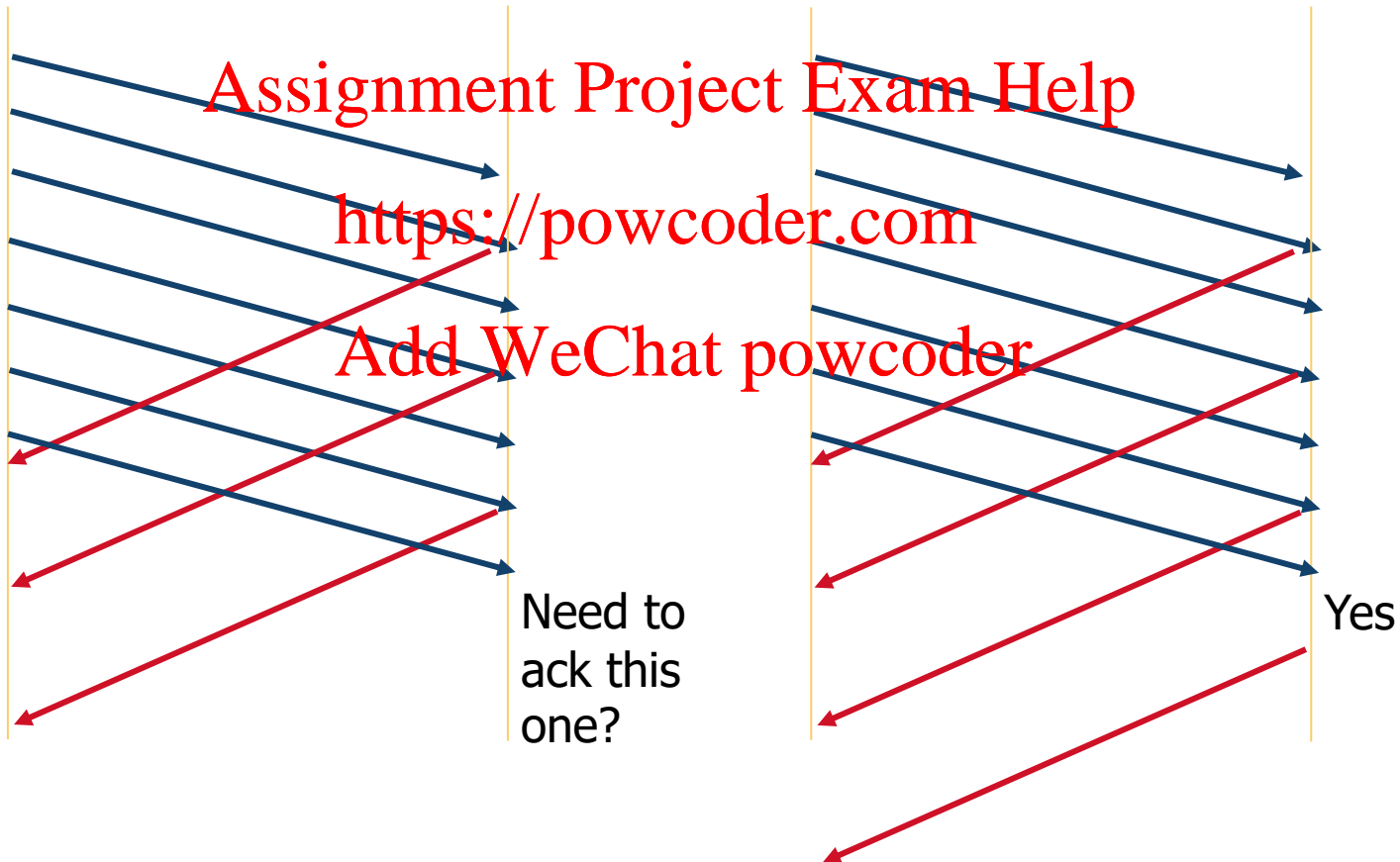
Host B



Host A



Host B



› time-out period often relatively long:

- long delay before resending lost packet

› detect lost segments via duplicate ACKs.

- sender often sends many segments back-to-back
- if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

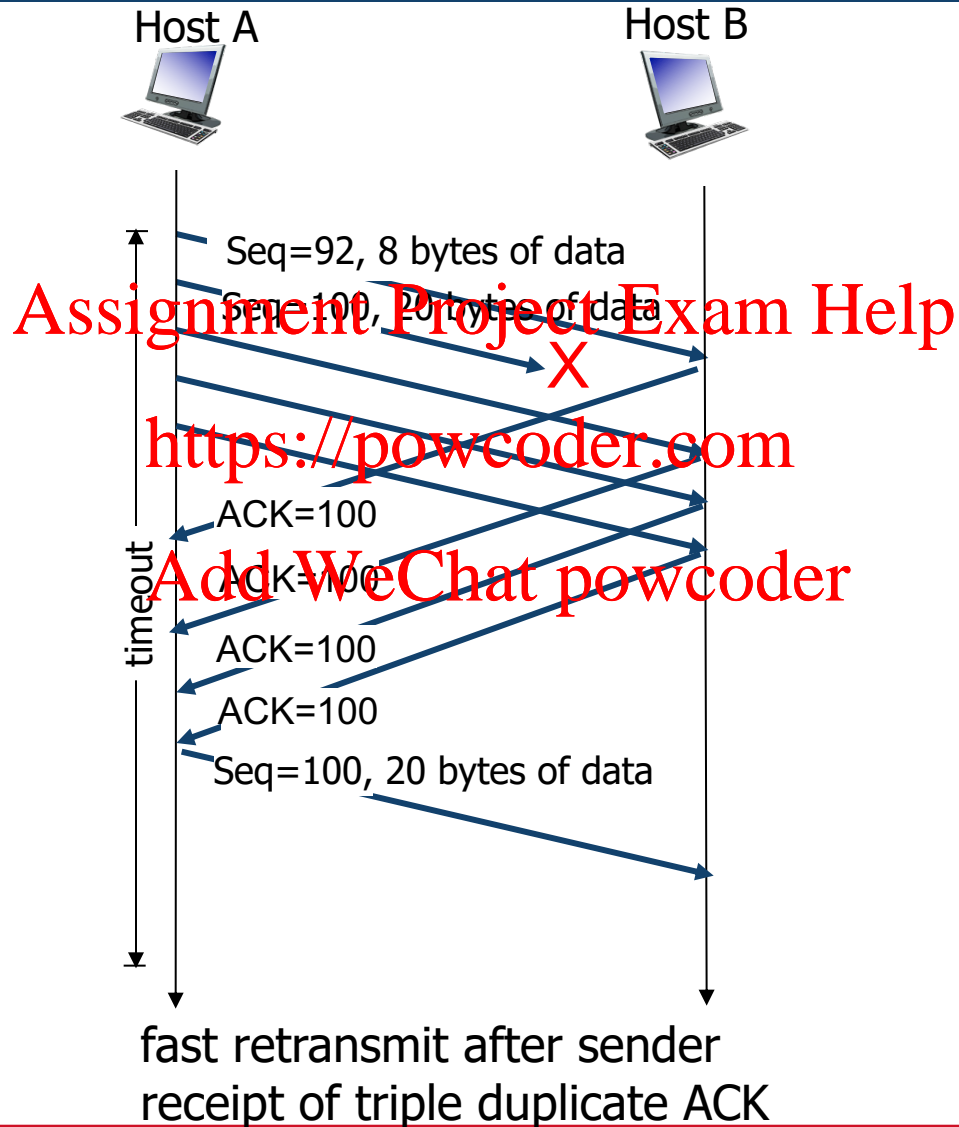
if sender receives 3 duplicate ACKs for same data

(“triple duplicate ACKs”), resend unacked segment with smallest seq #

- likely that unacked segment lost, so don't wait for timeout



TCP fast retransmit





Assignment Project Exam Help
Flow Control in TCP

<https://powcoder.com>

Add WeChat powcoder



TCP flow control

application may
remove data from
TCP socket buffers

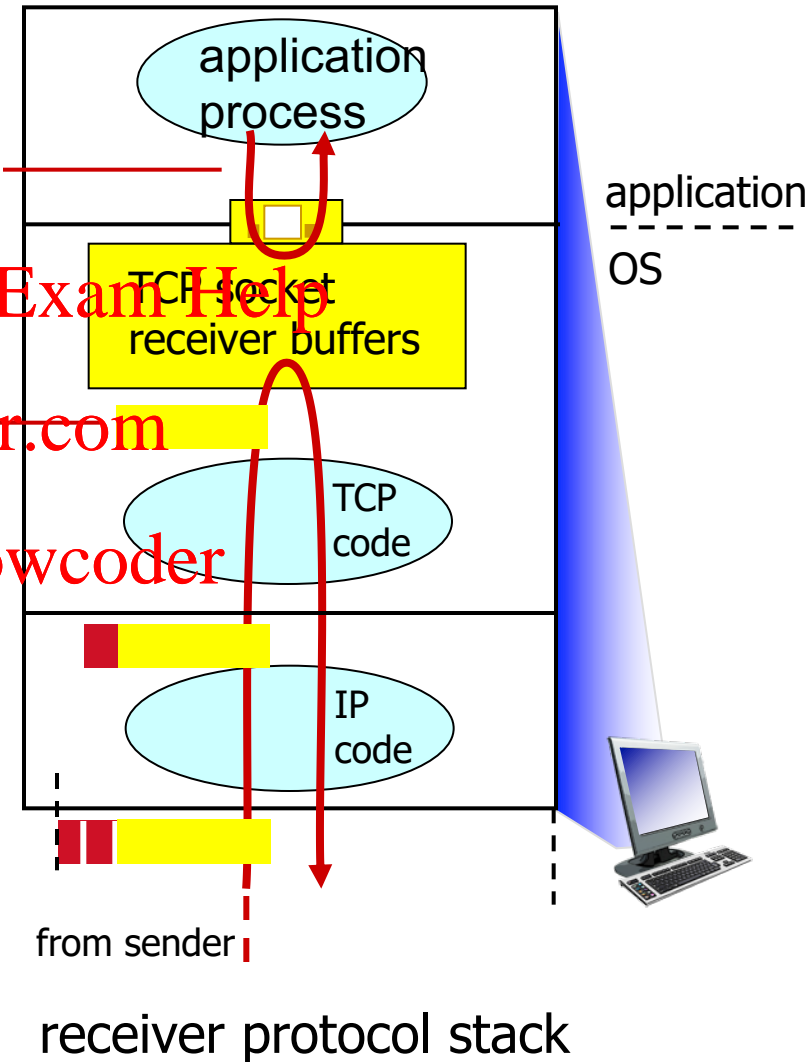
Assignment Project Exam Help

... slower than TCP
receiver is delivering
(sender is sending)

Add WeChat powcoder

flow control

receiver controls sender, so
sender won't overflow
receiver's buffer by transmitting
too much, too fast

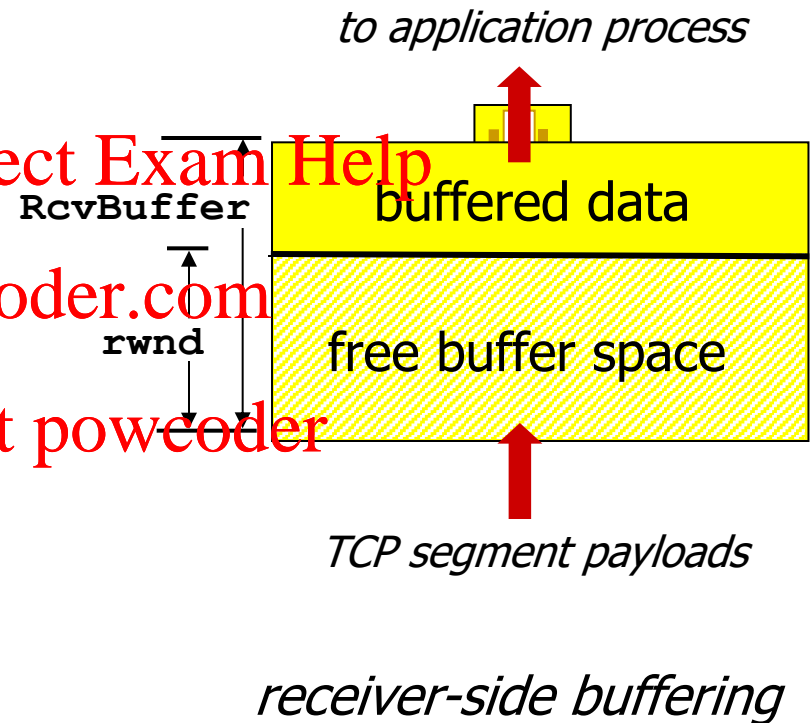


- › receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments

- **RcvBuffer** size set via socket options (typical default is 4096 bytes)

- many operating systems automatically adjust **RcvBuffer**

- › sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- › guarantees receive buffer will not overflow





Connection Management in TCP

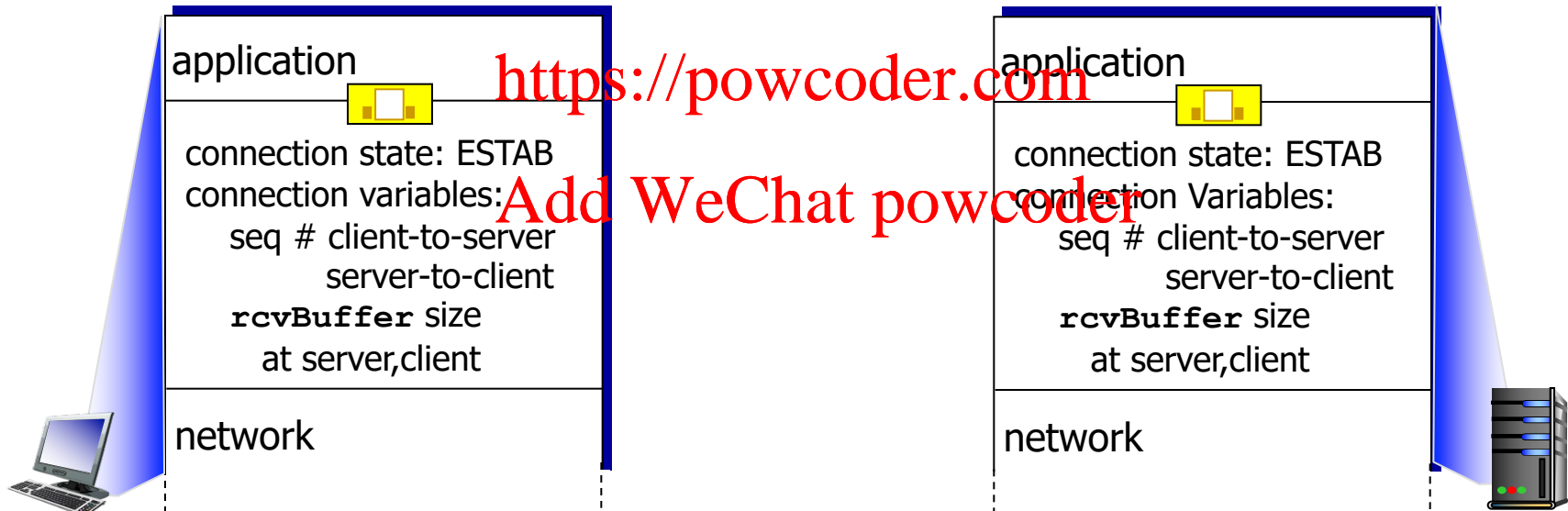
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

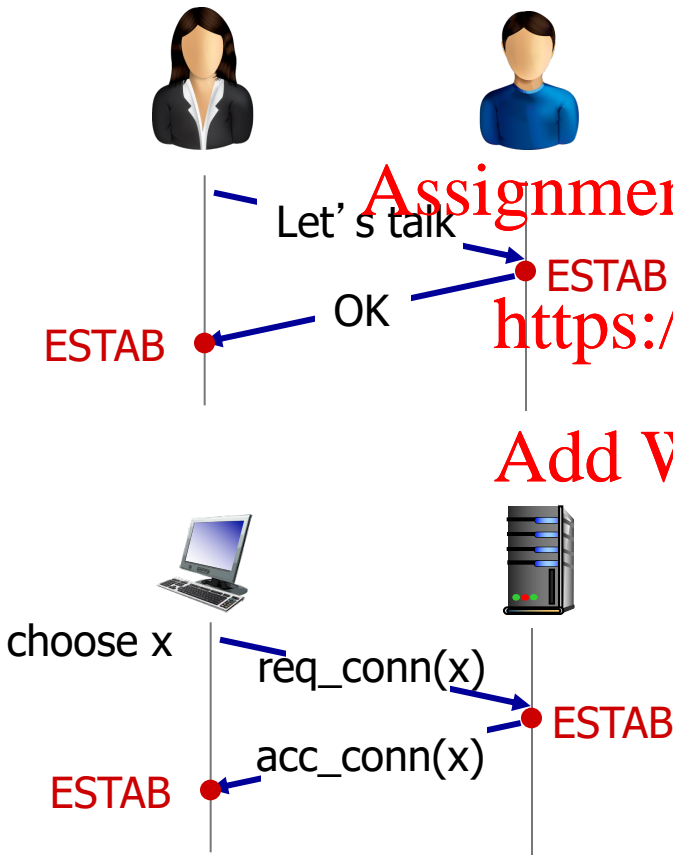
before exchanging data, sender/receiver “handshake”:

- › agree to establish connection (each knowing the other willing to establish connection)
- › agree on connection parameters



Agreeing to establish a connection

2-way handshake:



Q: will 2-way handshake always work in network?

- > variable delays
- > retransmitted messages (e.g. `req_conn(x)`) due to message loss
- > message reordering

can't "see" other side

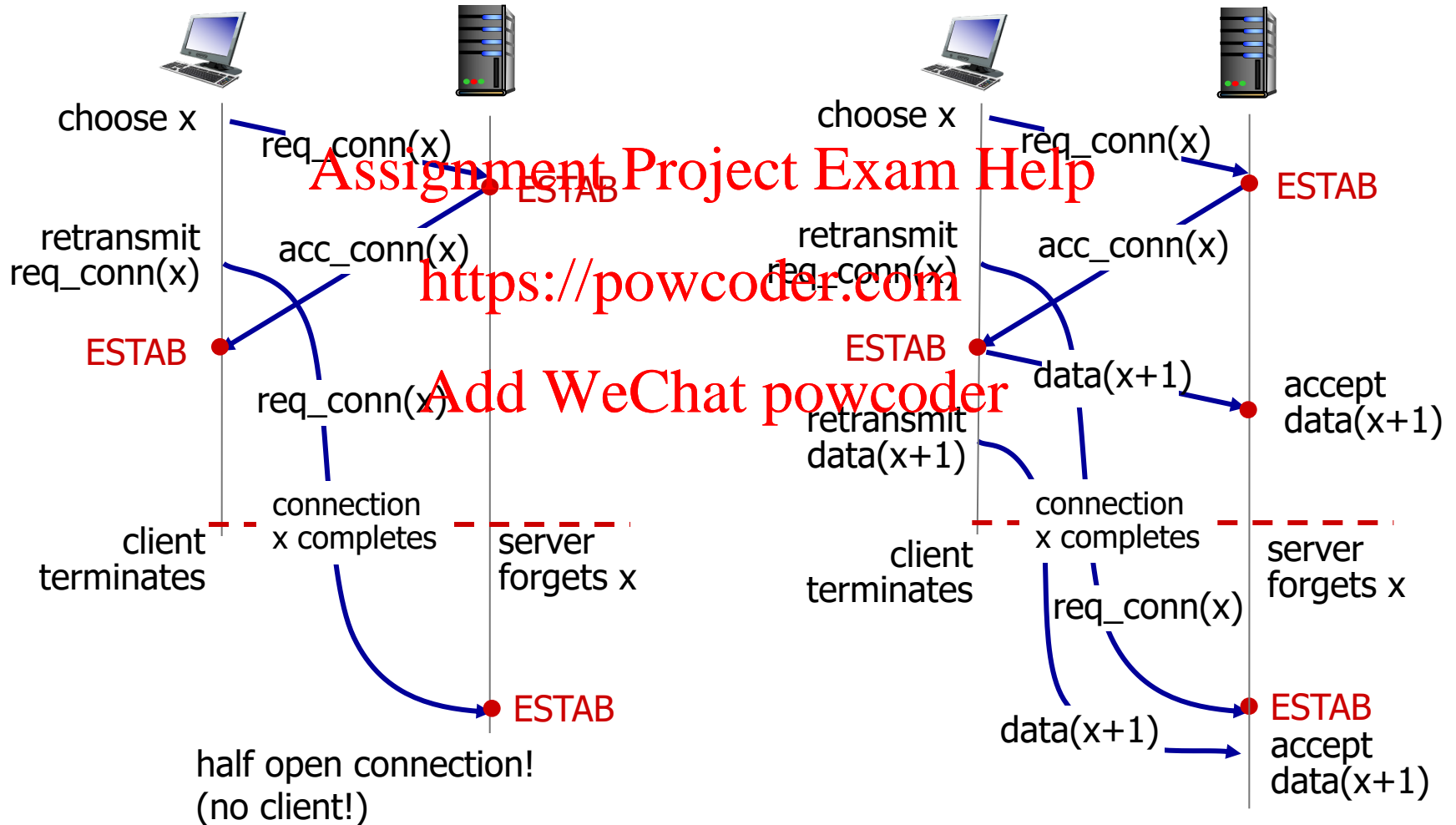
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Agreeing to establish a connection

2-way handshake failure scenarios:





TCP 3-way handshake

client state

LISTEN

SYNSENT

ESTAB

choose init seq num, x
send TCP SYN msg



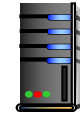
SYNbit=1, Seq=x

<https://powcoder.com>

Add WeChat powcoder

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ACKbit=1, ACKnum=y+1



server state

LISTEN

SYN RCVD

ESTAB

choose init seq num, y
send TCP SYNACK
msg, acking SYN

received ACK(y)
indicates client is live

› client, server each closes their side of connection

- send TCP segment with FIN bit = 1

› respond to received FIN with ACK

<https://powcoder.com>

Add WeChat powcoder

TCP: closing a connection

client state

ESTAB

`clientSocket.close()`

FIN_WAIT_1

can no longer
send but can
receive data

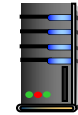
FIN_WAIT_2

wait for server
close

TIMED_WAIT

timed wait
for $2 * \text{max}$
segment lifetime

CLOSED



server state

ESTAB

CLOSE_WAIT

LAST_ACK

CLOSED

FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

can still
send data

FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

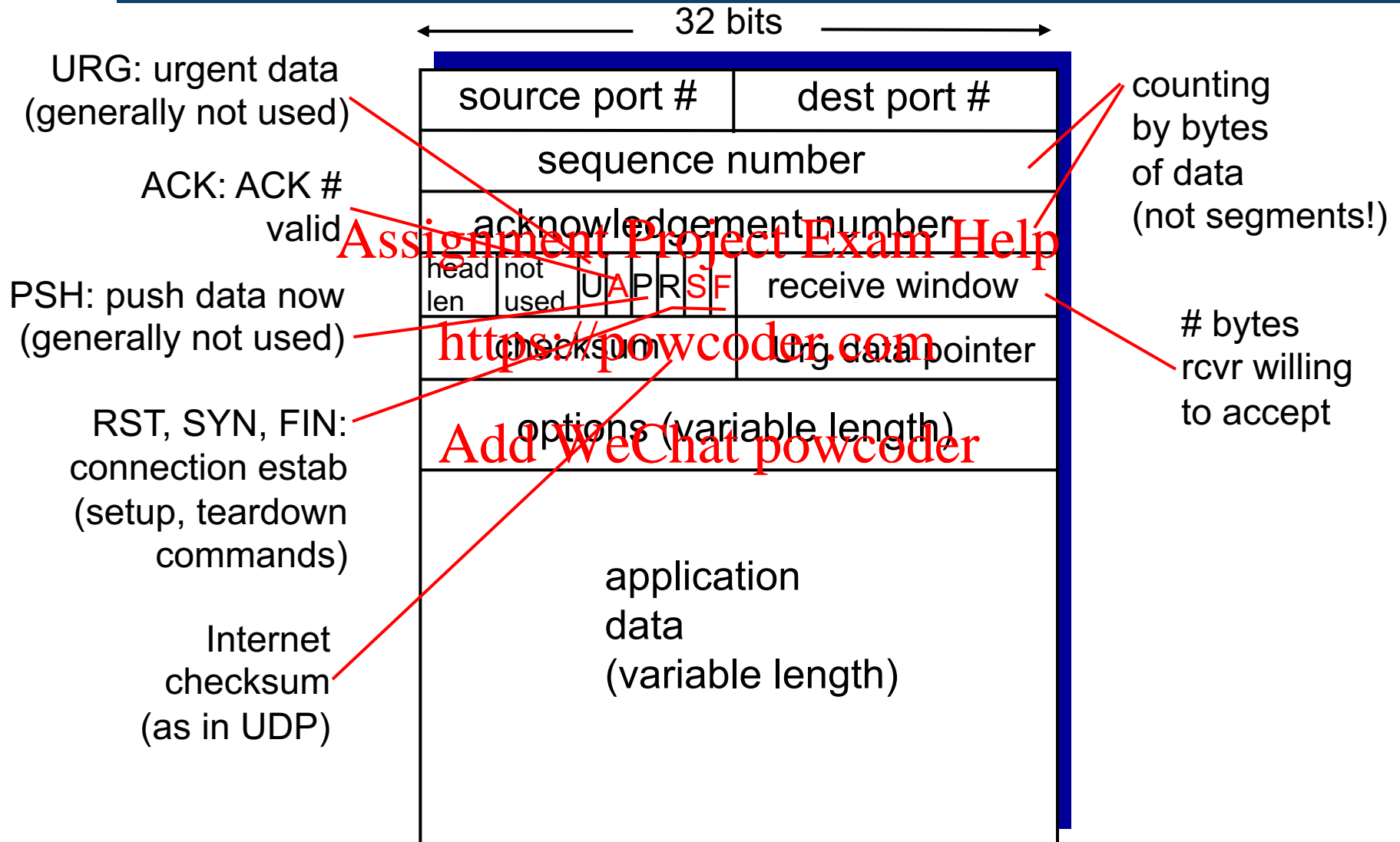
can no longer
send data

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

TCP segment structure





Principles of Congestion Control

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

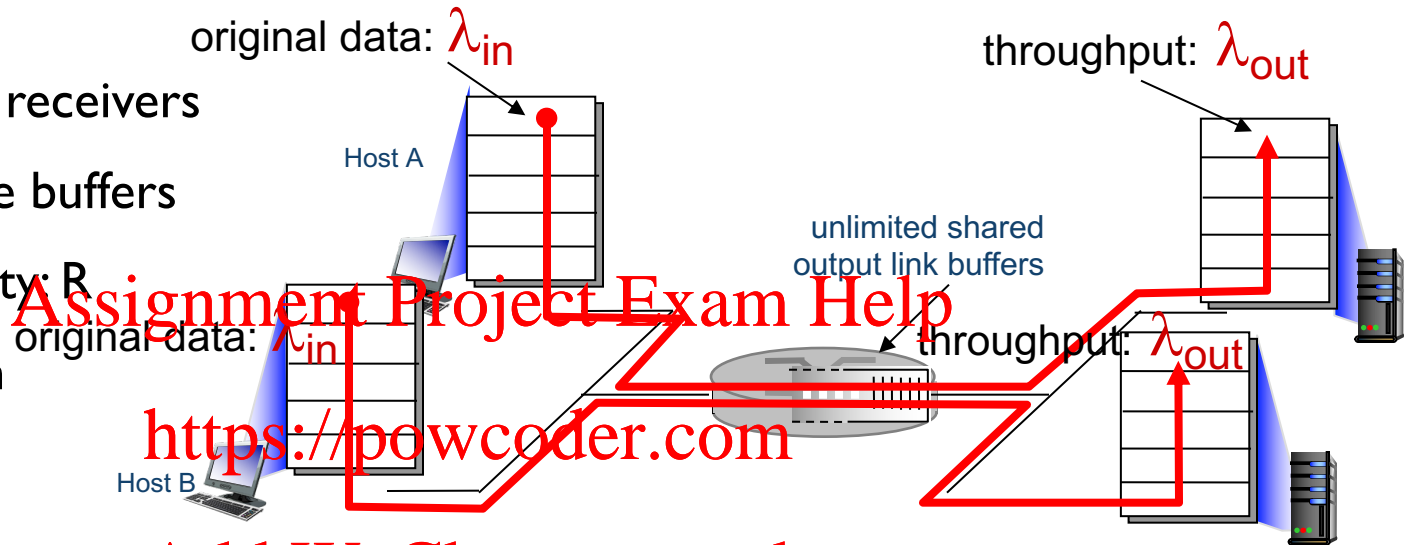
congestion:

- › informally: “too many sources sending too much data too fast for *network* to handle”
- › different from flow control!
- › manifestations:
 - lost packets (buffer overflow at routers)
 - long delays (queueing in router buffers)
- › a top-10 problem!

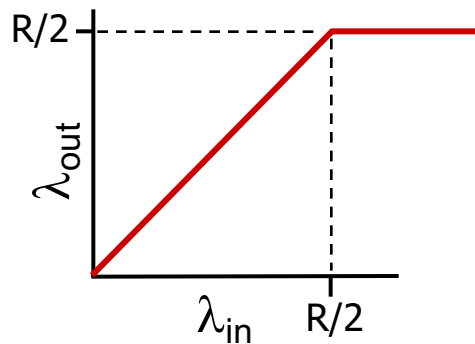


Causes/costs of congestion: scenario 1

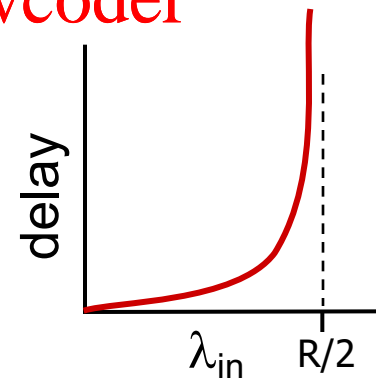
- › two senders, two receivers
- › one router, infinite buffers
- › output link capacity: R
- › no retransmission



Add WeChat powcoder



- › maximum per-connection throughput: $R/2$



- ❖ large delays as arrival rate, λ_{in} , approaches capacity

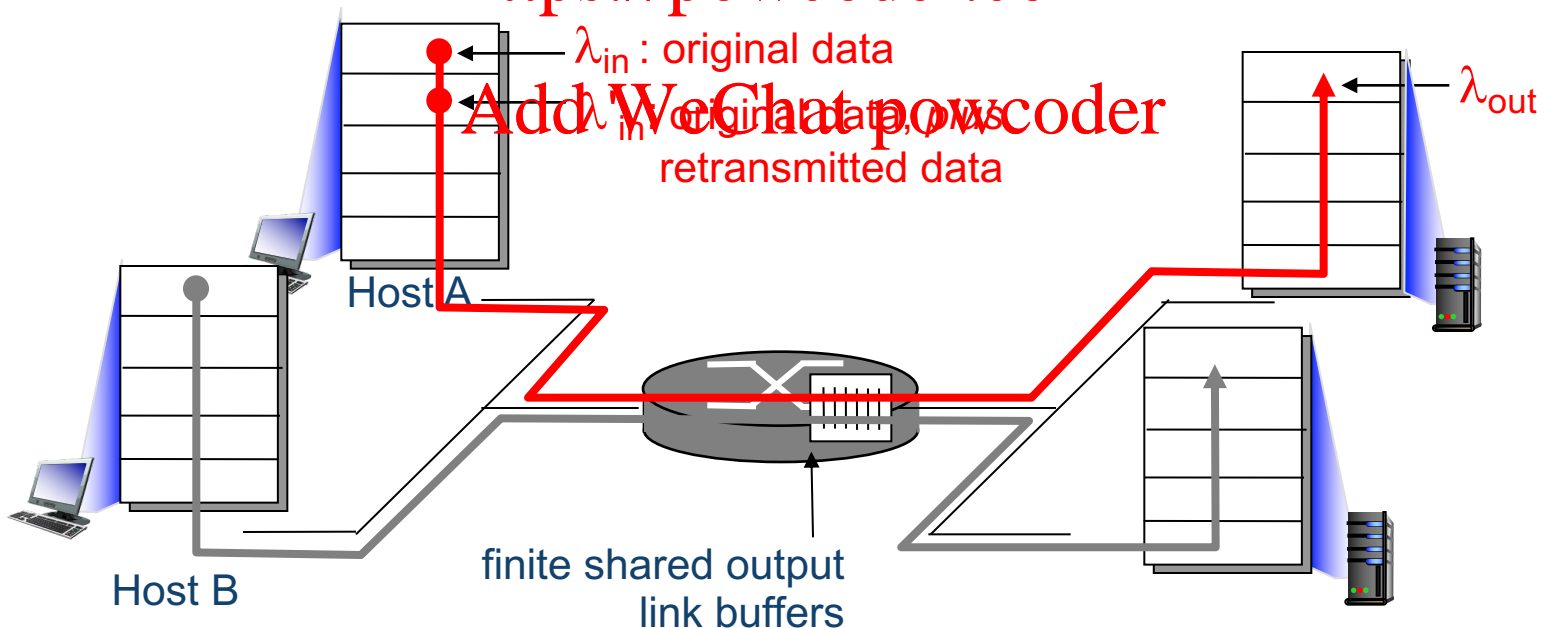
Causes/costs of congestion: scenario 2

- › one router, *finite* buffers
- › sender retransmission of timed-out packet
 - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$,
 - Goodput
 - transport-layer input includes retransmissions: $\lambda_{in} > \lambda_{out}$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat: powcoder
retransmitted data

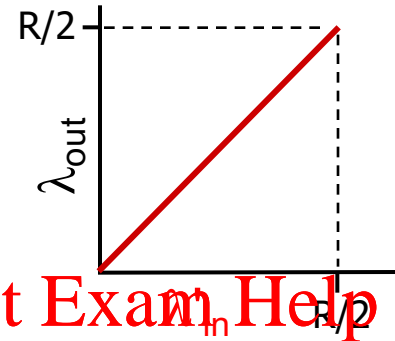




Causes/costs of congestion: scenario 2

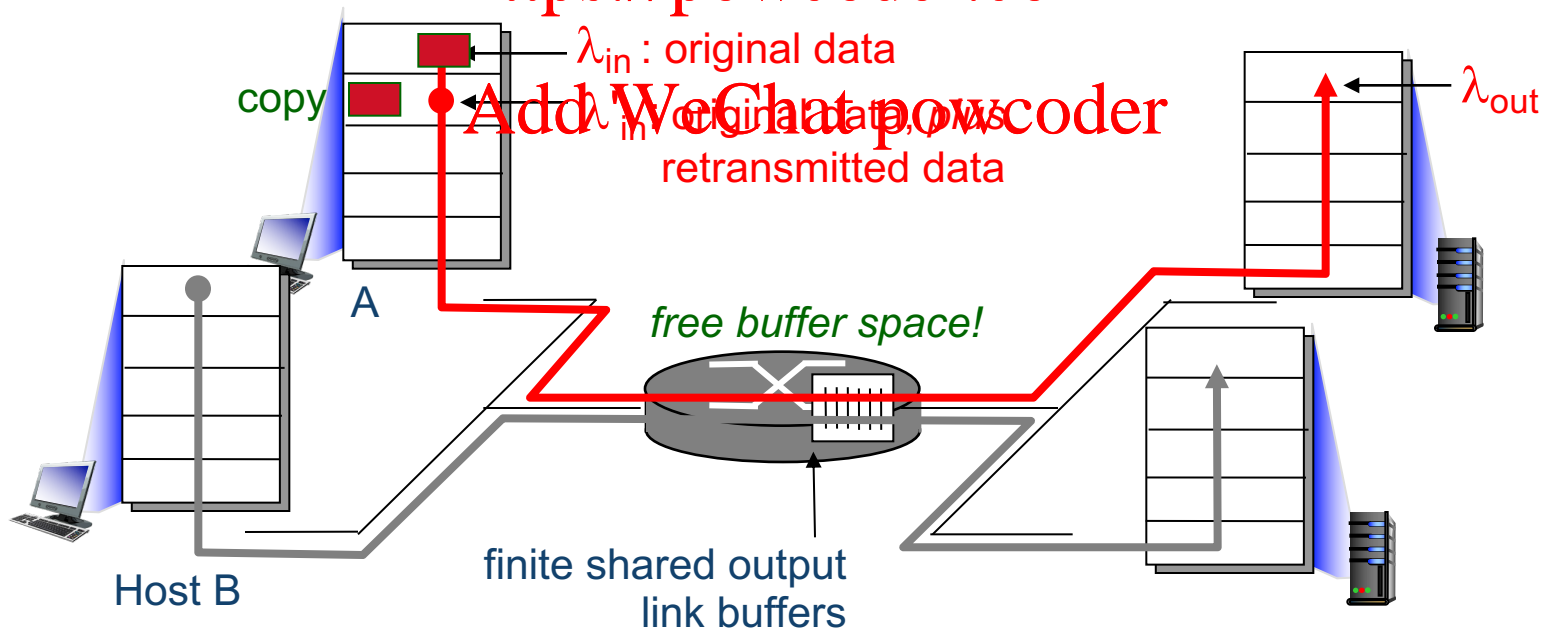
idealization: perfect
knowledge

- › sender sends only when
router buffers available



Assignment Project Exam Help

<https://powcoder.com>



Causes/costs of congestion: scenario 2

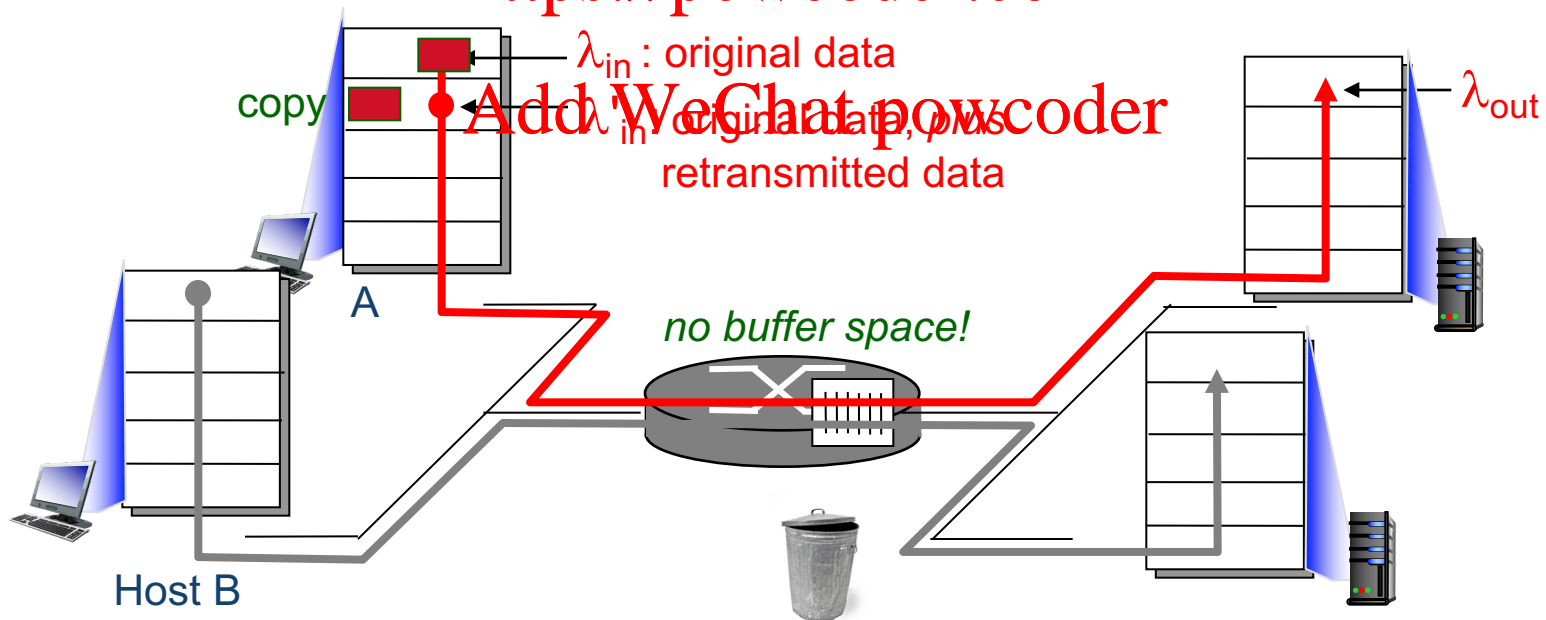
Idealization: *known loss*

packets can be lost, dropped
at router due to full buffers

- › sender only resends if
packet *known to be lost*

Assignment Project Exam Help

<https://powcoder.com>

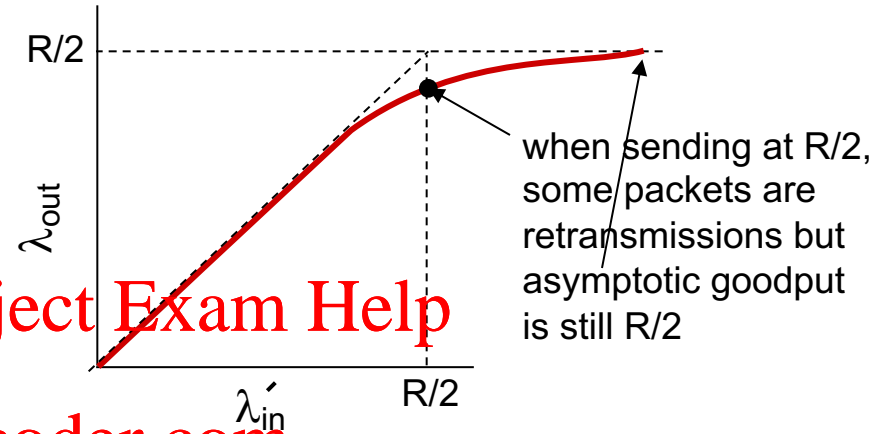


Causes/costs of congestion: scenario 2

Idealization: *known loss*

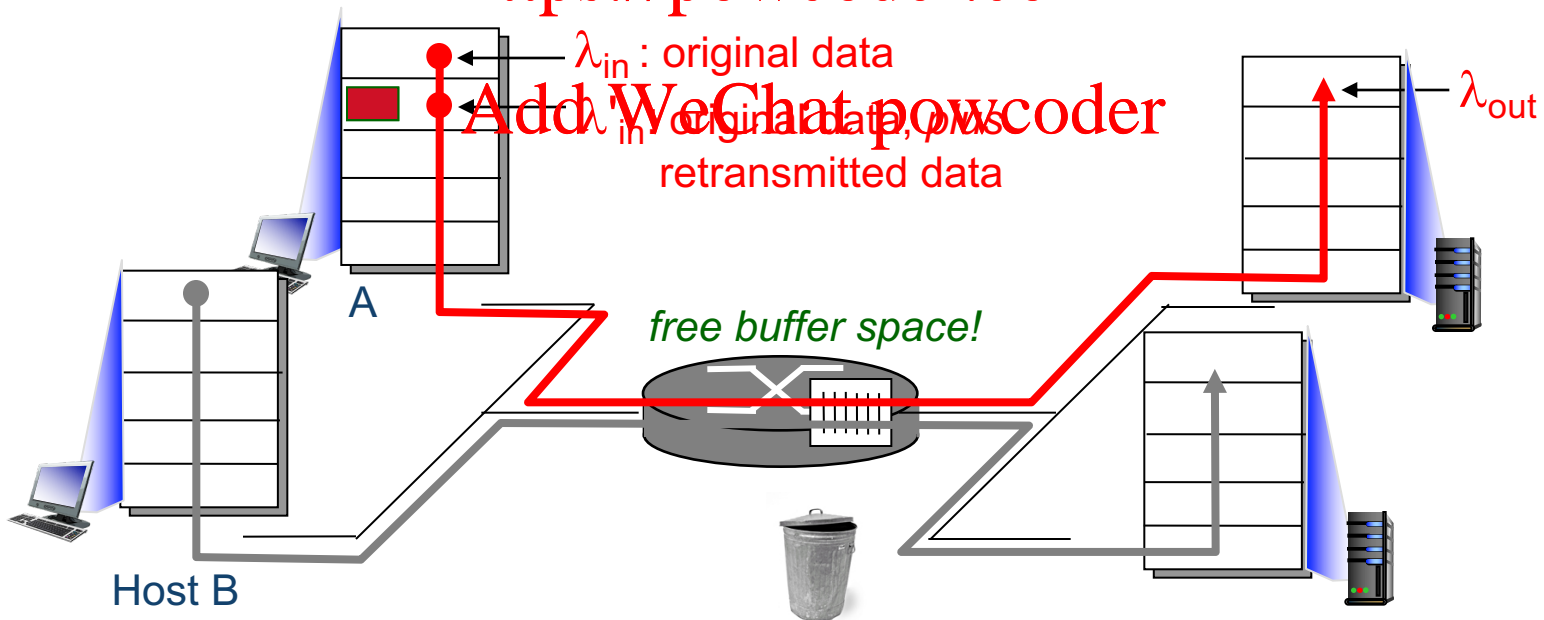
packets can be lost, dropped at router due to full buffers

- › sender only resends if packet known to be lost



<https://powcoder.com>

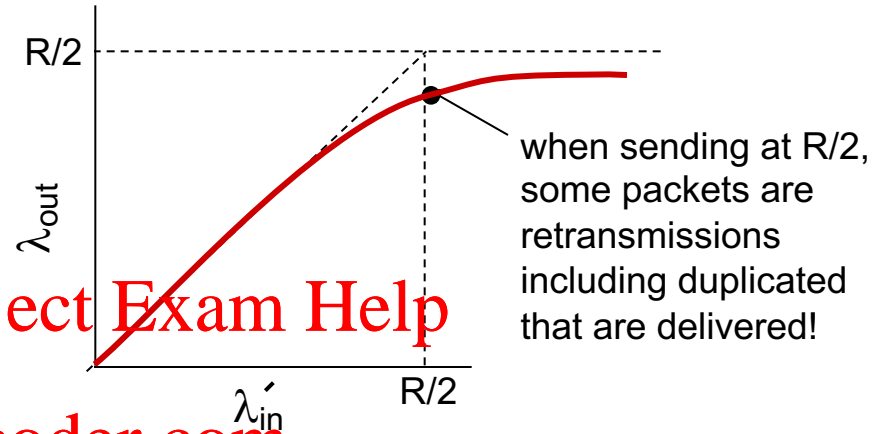
Add WeChat powcoder
retransmitted data



Causes/costs of congestion: scenario 2

Realistic: *duplicates*

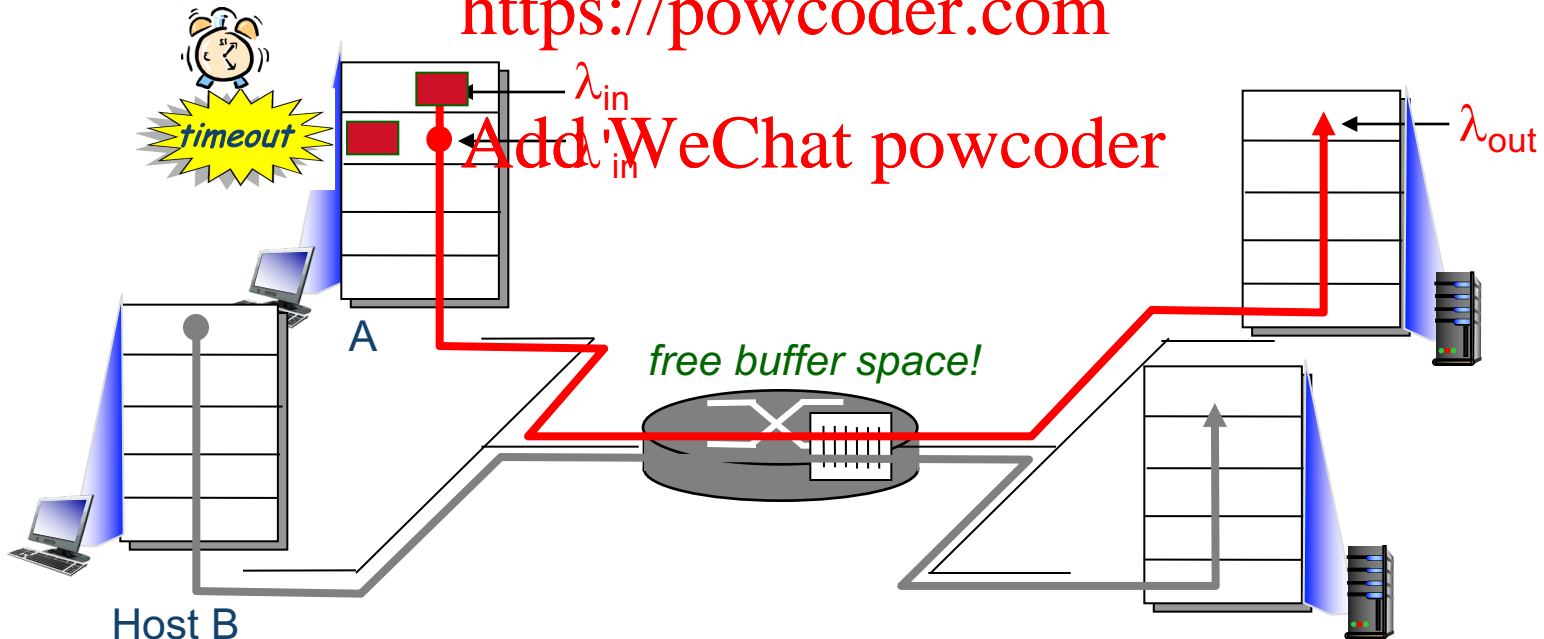
- ❖ packets can be lost, dropped at router due to full buffers
- ❖ sender times out prematurely, sending *two* copies, both of which are delivered



Assignment Project Exam Help

<https://powcoder.com>

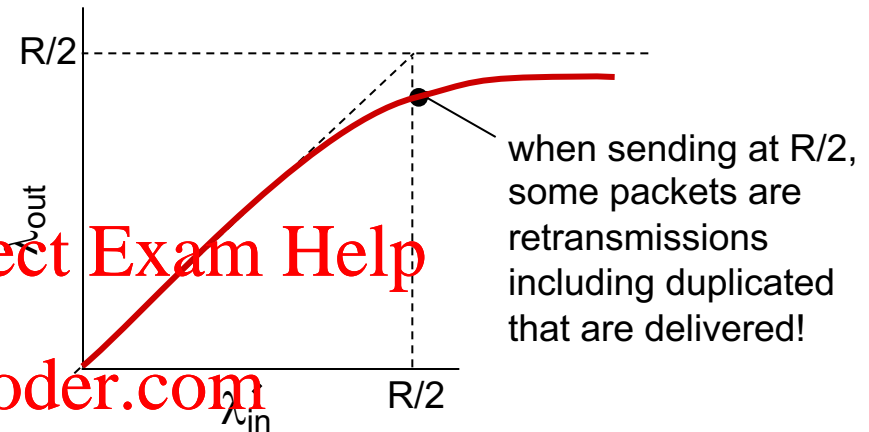
Add WeChat powcoder



Causes/costs of congestion: scenario 2

Realistic: *duplicates*

- ❖ packets can be lost, dropped at router due to full buffers
- ❖ sender times out prematurely, sending *two* copies, both of which are delivered



“costs” of congestion: Add WeChat powcoder

- ❖ more work (retrans) for given “goodput”
- ❖ unneeded retransmissions: link carries multiple copies of pkt
 - decreasing goodput



Causes/costs of congestion: scenario 3

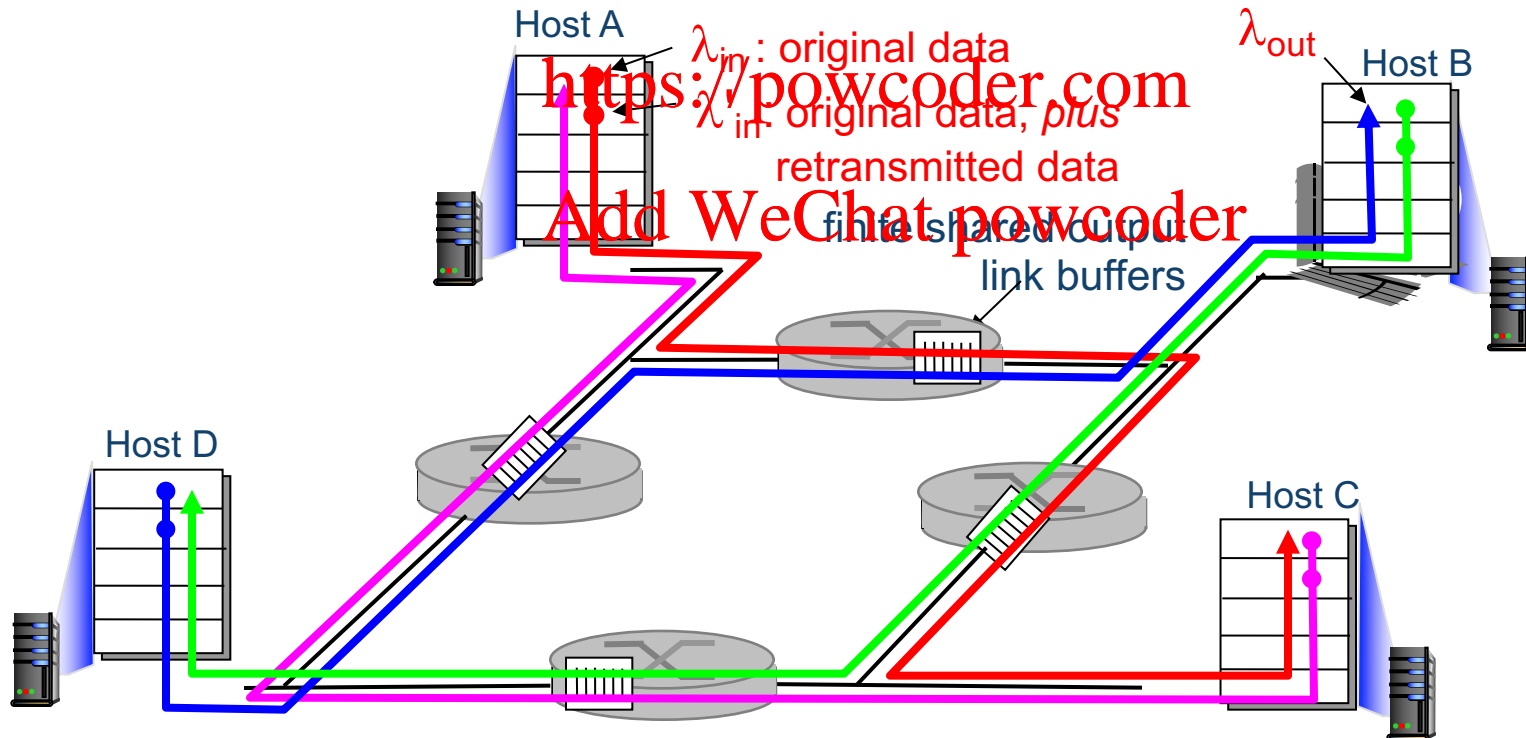
- › four senders
- › multihop paths
- › timeout/retransmit

Q: what happens as λ_{in}' increases ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput $\rightarrow 0$

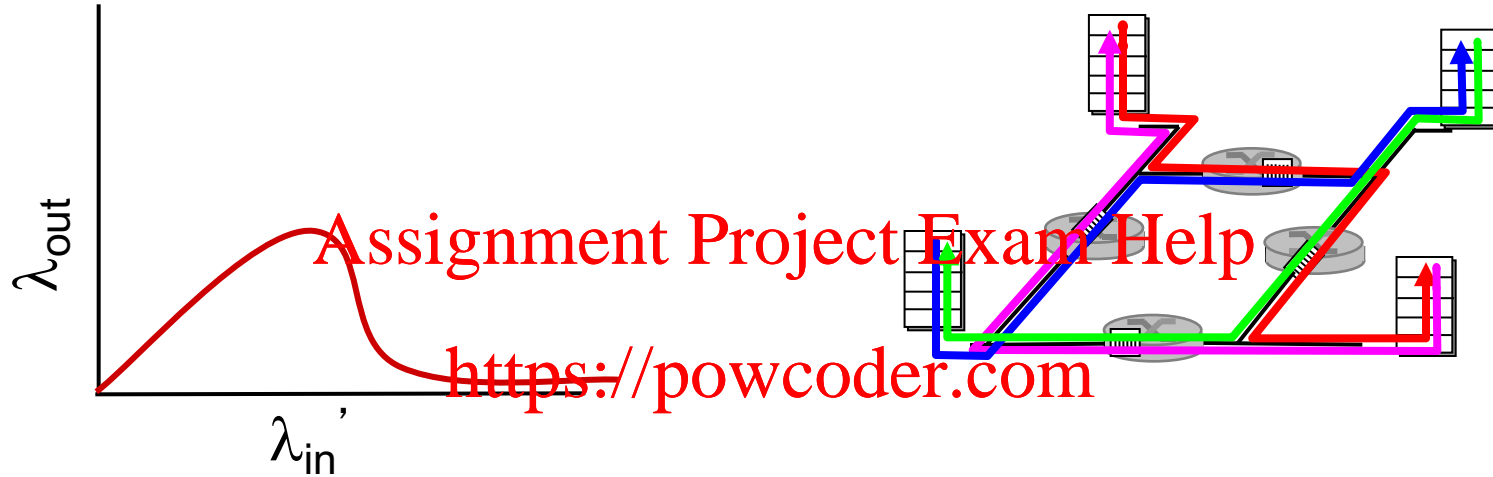
Assignment Project Exam Help

<https://powcoder.com>
Add WeChat powcoder





Causes/costs of congestion: scenario 3



Add WeChat powcoder

another “cost” of congestion:

- ❖ when packet dropped, any “upstream transmission capacity used for that packet was wasted!

two broad approaches towards congestion control:

end-end congestion control:

- › no explicit feedback from network
- › congestion inferred from end-system observed loss, delay
- › approach taken by TCP

network-assisted congestion control:

- › routers provide feedback to end systems
 - single bit indicating congestion
 - explicit rate for sender to send at

<https://powcoder.com>

Add WeChat powcoder



TCP Congestion Control

Assignment Project Exam Help

<https://powcoder.com>

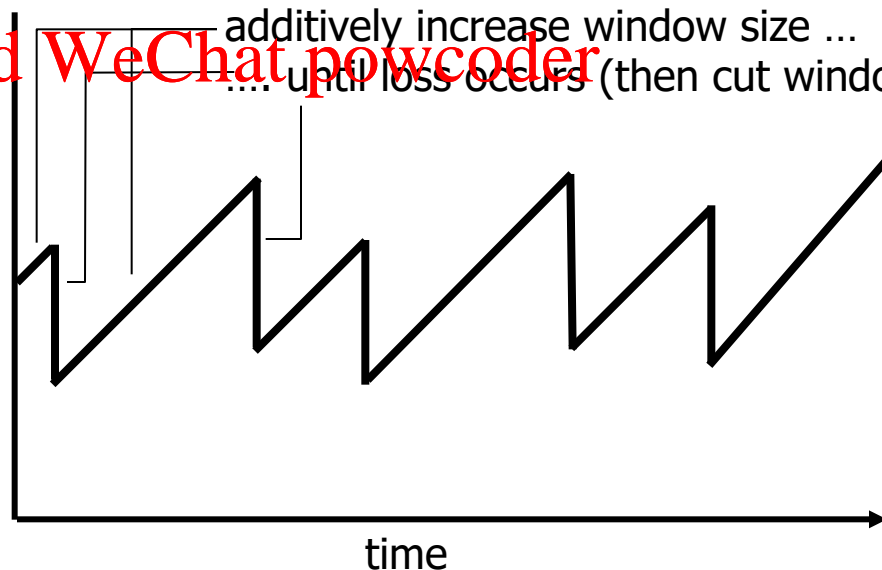
Add WeChat powcoder

Additive increase multiplicative decrease (AIMD)

- ❖ *approach*: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
 - *additive increase*: increase $cwnd$ by 1 MSS (maximum segment size) every RTT until loss detected
 - *multiplicative decrease*: cut $cwnd$ in half after loss

additively increase window size ...
... until loss occurs (then cut window in half)

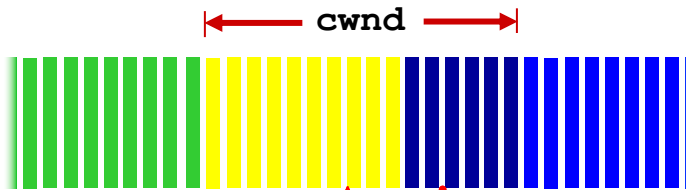
cwnd: TCP sender
congestion window size



AIMD saw tooth
behavior: probing
for bandwidth

TCP Congestion Control: details

sender sequence number space



TCP sending rate:

- › roughly: send cwnd bytes, wait RTT for ACKS, then send more bytes

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

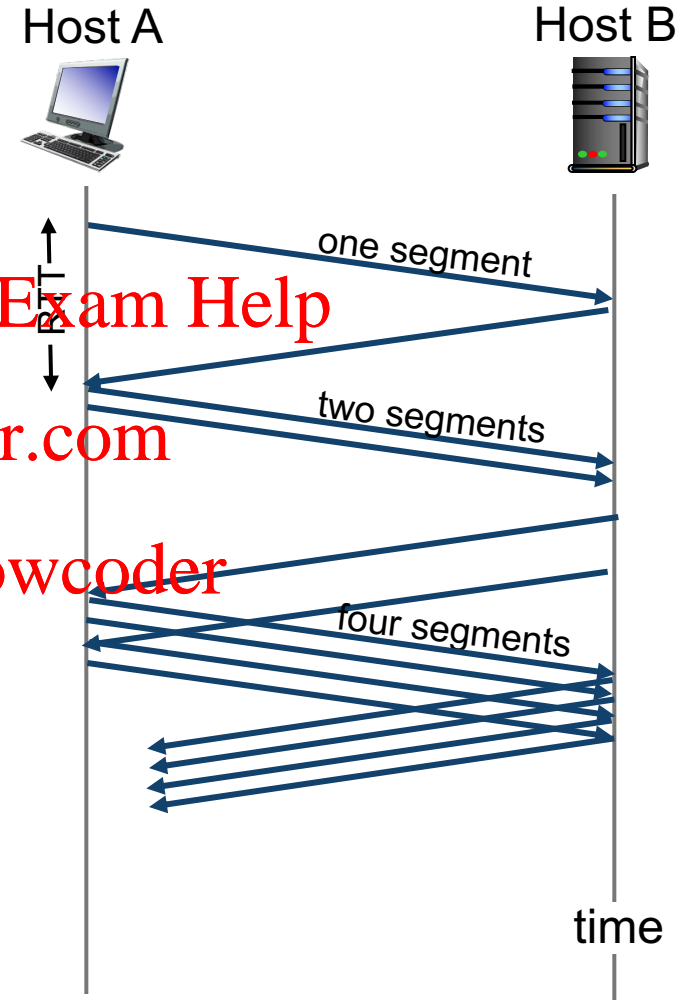
- › sender limits transmission:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

- › **cwnd** is dynamic, function of perceived network congestion

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- › when connection begins, increase rate exponentially:
 - initially `cwnd` = 1 MSS
 - double `cwnd` every RTT
 - done by incrementing `cwnd` for every ACK received
- › summary: initial rate is slow but ramps up exponentially fast
- › when should the exponential increase switch to linear (additive increase)?

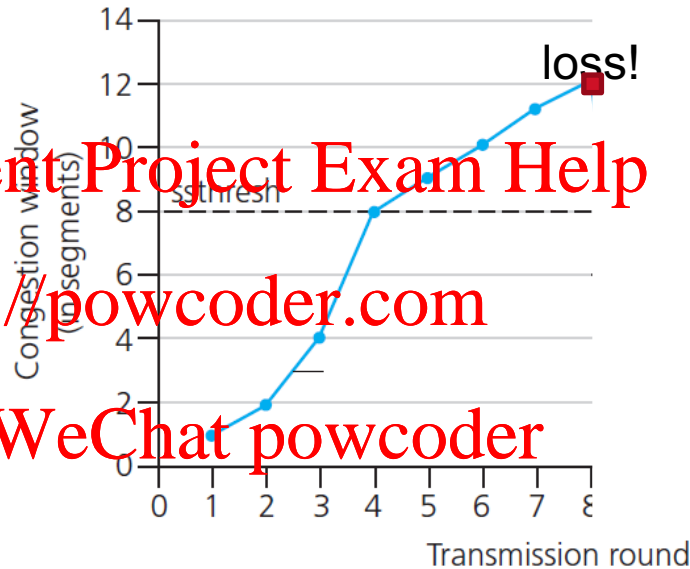


TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: cwnd reaches **ssthresh**

cwnd=12 ssthresh=6



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Implementation:

- › At beginning **ssthresh**, specified in different versions of TCP
- › (In this example **ssthresh=8 segment**)
- › on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

› loss indicated by timeout:

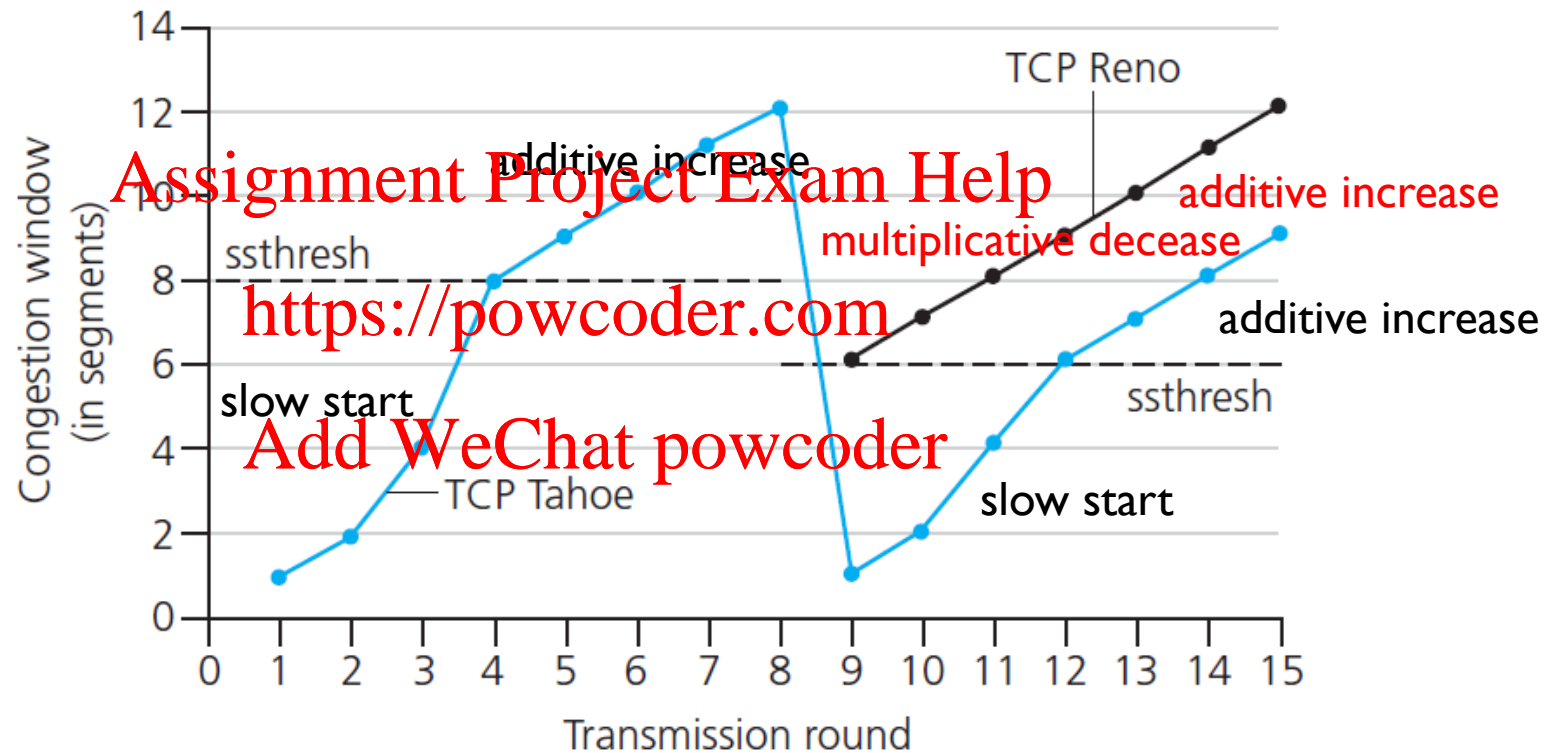
- **cwnd** set to 1 MSS;
- window then grows exponentially (as in slow start) to **ssthresh**, then grows linearly

› loss indicated by 3 duplicate ACKs:

- › TCP Tahoe, same as loss indicated by timeout, always sets **cwnd** to 1 (timeout or 3 duplicate acks)
- › TCP RENO
 - **cwnd** is cut in half window then grows linearly (additive increase)
 - fast recovery

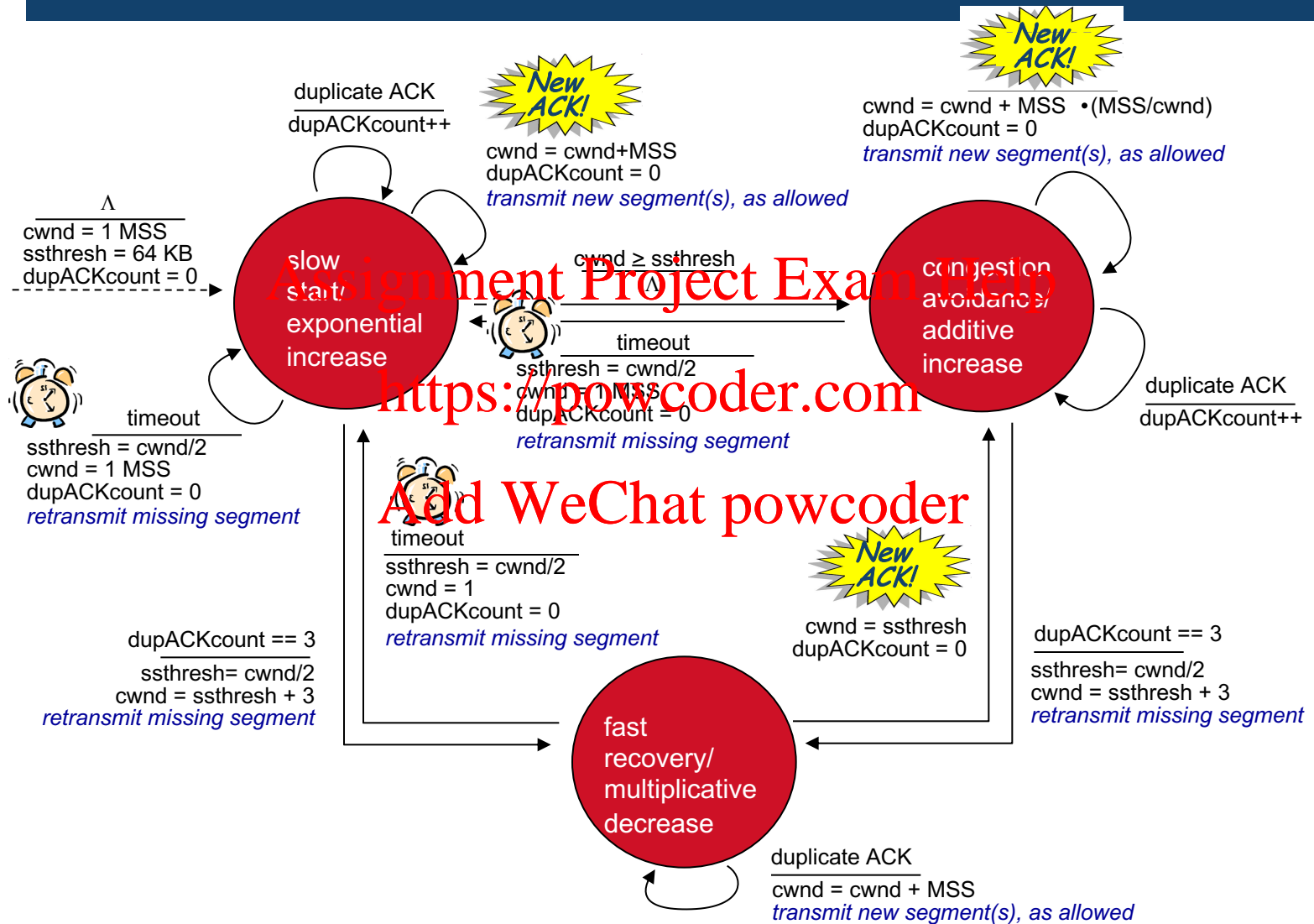


TCP: switching from slow start to CA





Summary: TCP Reno Congestion Control



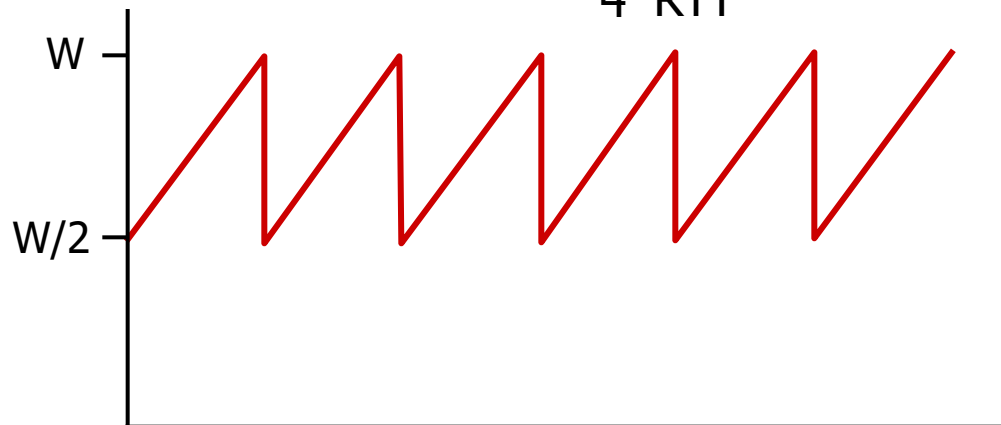
› avg. TCP thruput as function of window size, RTT?

- ignore slow start, assume always data to send

› W: window size (measured in bytes) when loss occurs

- avg. window size (# in-flight bytes) is $\frac{3}{4}W$
- avg. thruput is $\frac{3}{4}W$ per RTT

avg TCP thruput = $\frac{3}{4} \frac{W}{RTT}$ bytes/sec



TCP Futures: TCP over “long, fat pipes”

- › example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- › requires $W = 83,333$ in-flight segments
- › throughput in terms of segment loss probability, L [Mathis 1997]:

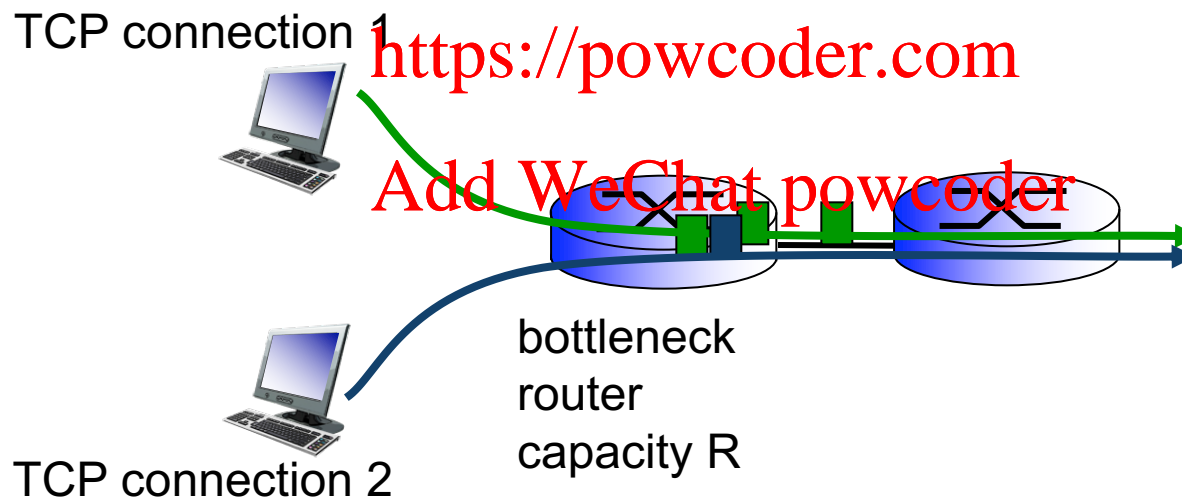
<https://powcoder.com>

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

- ➔ to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$
– *a very small loss rate!*
- › new versions of TCP for high-speed
 - › Vegas, Westwood, CUBIC, etc.

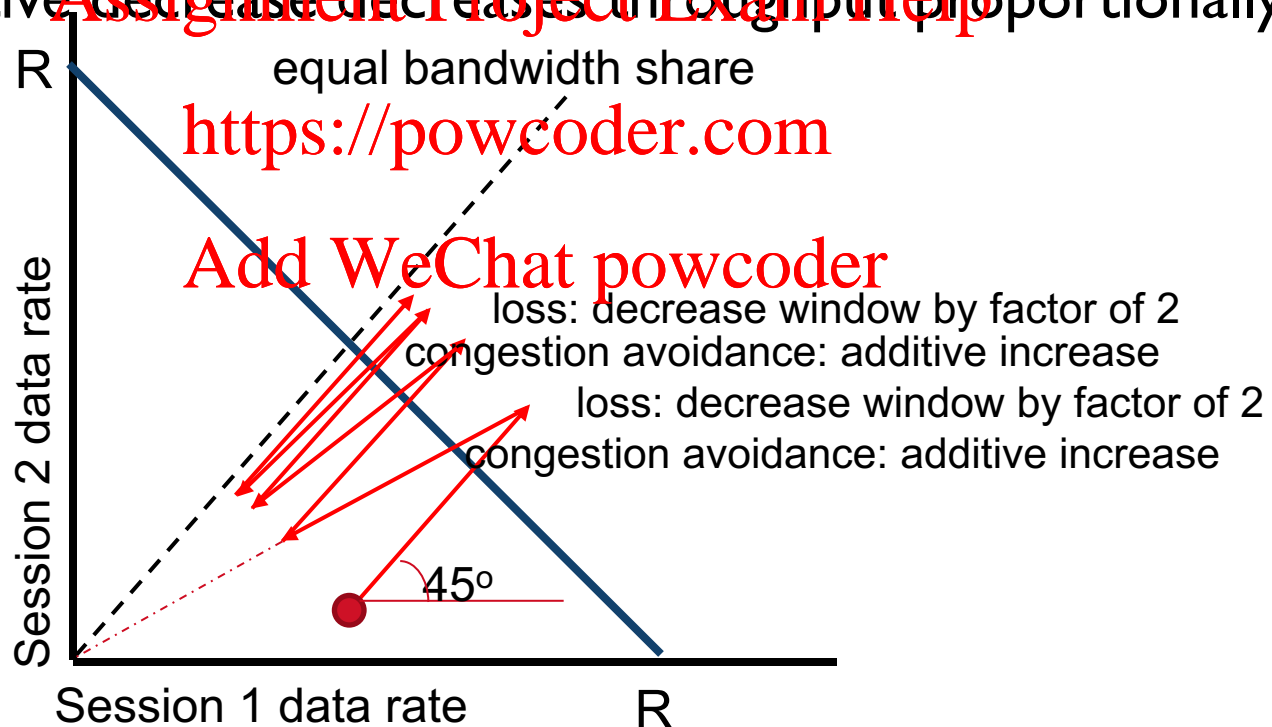
Fairness: K TCP sessions share same bottleneck link of bandwidth R , each has average rate of R/K

Assignment Project Exam Help



two competing sessions:

- › additive increase gives slope of 1, as throughput increases
- › multiplicative decrease decreases throughput proportionally



Fairness and UDP

- › multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
- › instead use UDP:
 - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- › application can open multiple parallel connections between two hosts
 - › e.g. link of rate R
 - App 1 asks for 1 TCP, gets $0.1R$
 - App 2 asks for 9 TCPs, gets $0.9R$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Window size = min (rwnd, cwnd)

Assignment Project Exam Help

<https://powcoder.com>

receive window *congestion window*

flow control

congestion control

- › principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - connection setup, teardown
 - flow control <https://powcoder.com>
 - congestion control
- › instantiation, implementation in the Internet
 - UDP
 - TCP

Assignment Project Exam Help

Add WeChat powcoder