


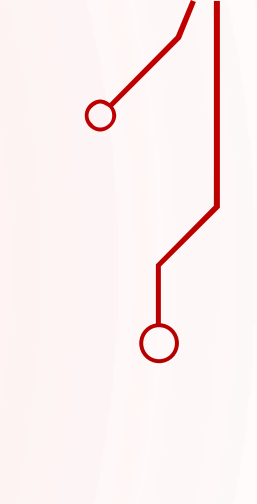
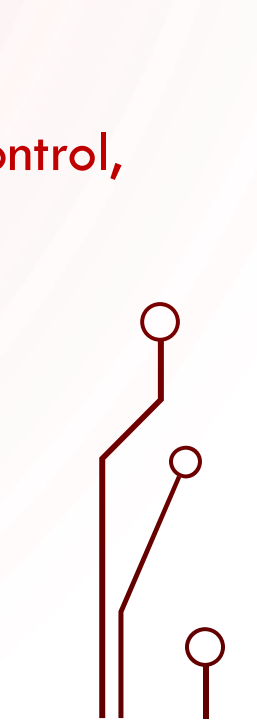
A decorative graphic on the left side of the slide consisting of a network of thin, dark red lines. These lines form a complex, branching pattern that resembles a circuit board or a stylized tree. Small circles are placed at various points where the lines intersect or terminate, adding to the technical or digital aesthetic of the design.

CHAPTER 3

TRANSPORT LAYER



ROADMAP

1. Transport-layer services
 2. Multiplexing and Demultiplexing
 3. Connectionless transport: **UDP**
 4. Principles of reliable data transfer
 5. Connection-oriented transport: **TCP** (segment structure, reliable data transfer, flow control, connection management)
 6. Principles of congestion control
 7. **TCP** congestion control
- 
- 
- 

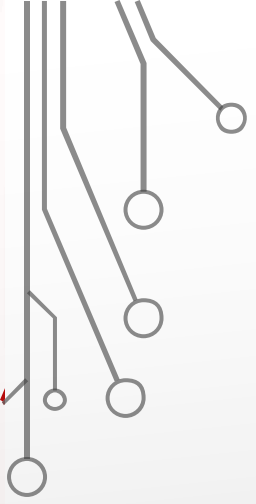


TRANSPORT-LAYER SERVICES

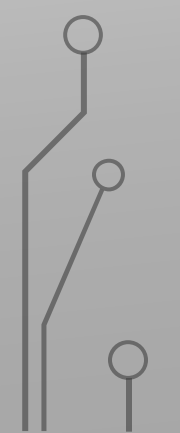
SERVICES AND PROTOCOLS

TRANSPORT LAYER SERVICES AND PROTOCOLS

- ال **OS** يقدم **Basic Networking Services** بتكون هي نفسها **Transport Layer Services**
- ال **Transport Layer** زيه زي اي **Layer** ثاني بيكون ليه **Logical Connection** بينه و بين ال **Transport Layer** اللي عند ال **Receiver**
- ال **Transport Protocols** بتشتغل علي ال **End-Systems** اللي هم اجهزة ال **End-User**
- لما تيجي تبعت رسالة ال **Transport Layer** بيستلم الرسالة من **Application Layer** و يقسم الداتا علي هيئة **Segments** وال **Transport Layer** اللي عند **Receiver** بيجمع ال **Segments** ده ثاني ويبعتها لل **Application Layer**
- ال **Network application** يقدر يستخدم اكثر من **Transport Layer Protocol**
- ال **Internet** بيستخدم اتنين **Transport Layer Protocols** مهمين هم ال **TCP** وال **UDP**

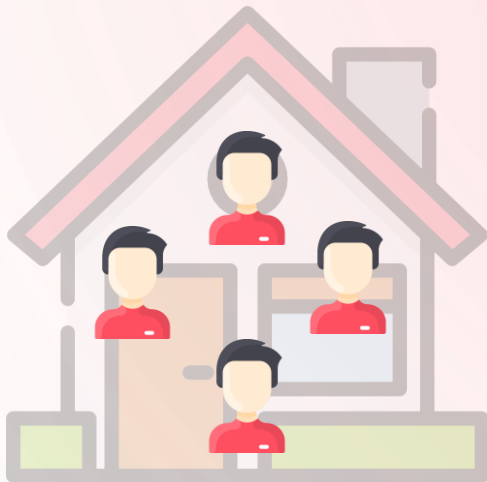


TRANSPORT LAYER **VS** NETWORK LAYER

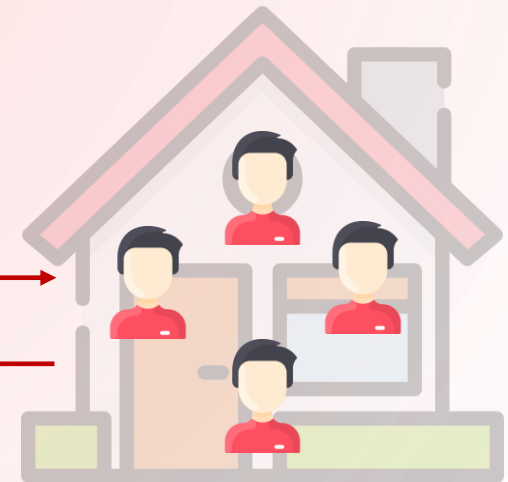
- 
- **NETWORK LAYER** : LOGICAL COMMUNICATION BETWEEN **HOSTS**
 - **TRANSPORT LAYER** : LOGICAL COMMUNICATION BETWEEN **PROCESSES**
- RELIES ON NETWORK LAYER **AND** ENHANCE NETWORK LAYER SERVICES
- 

HOUSEHOLD ANALOGY

12 OF KIDS SENDING LETTERS TO 12 KIDS



- **Processes = kids**
- **app messages = letters in envelopes**
- **hosts = houses**
- **transport protocol = Ann and Bill**
- **network-layer protocol = postal service**



INTERNET TRANSPORT LAYER PROTOCOLS

TCP & UDP

UDP & TCP doesn't Provide Delay guarantess or Bandwidth guarantess

TCP service:

- **connection-oriented:** setup required between client and server processes
- **reliable transport:** between sending and receiving process
- **flow control:** sender won't overwhelm receiver
- **congestion control:** throttle sender when network overloaded
- **does not provide:** timing, minimum throughput guarantees, security

UDP service:

- **unreliable data transfer** between sending and receiving process
- **does not provide:** connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security
- علي الرغم من ان ال **UDP** مبيقدمش مميزات كثيرة زي ال **TCP** بس هو **خفيف** جدا و **سريع** اكثر في ال **Send** و ال **Receive** لانه مبميرش بمراحل كثيرة زي ال **TCP**



MULTIPLEXING & DEMULTIPLEXING

HOW IT WORKS ?

MULTIPLEXING

GATHERING DATA FROM MULTIPLE SOCKETS, ENVELOPING DATA WITH HEADER (LATER USED FOR DEMULTIPLEXING)

- وظيفة ال **Multiplexing** انه يجمع ال **Data** اللي هتتبع من اكر من **Process** ويبعتهم مرة واحدة
- زي مثلا لو عندنا **3 Processes** هم **Browser** و **Email** و **File Transfer** كلهم بيبيعو داتا في نفس الوقت , كل **Data** بتتبع من كل **Process** بتكون ليها **Source/Destination Port Number** و بيتضاف برضو ال **Transport Header** من **Transport Layer** (اللي هستخدمها في عملية ال **Demultiplexing** بعدين) وبعدها يعمل لكل ال **data** ده **Multiplexing**
- بحيث الناتج اللي يطلعي هو مجموعة من ال **data** بتتبع مرة واحدة لل **Receiver** وكل **data** بتكون ل **Process** معينة

DEMULTIPLEXING

DELIVERING RECEIVED SEGMENTS TO CORRECT SOCKET

- لما ال **Receiver** يستلم مجموعة ال **Data** التي اتعملهم **Multiplexing** , يفصلهم ثاني ل **data** منفصلين ويبدأ انه يرسل كل **data** لل **Socket** الخاص بيها
- وارسال ال **data** لكل **Process** بتعتمد علي **Destination Port Number** التي بيبقي موجود في ال **Transport Header**
- ال **Socket** هو ال **API** التي بيكون بين ال **Transport Layer** وال **Application Layer** وبعد ما ال **Data** بتوصل لل **Socket** , بتوصل لل **Process** التي عند ال **Receiver**
- وكذا تكون وظيفة ال **Demultiplexing** انه يفصل مجموعة ال **data** ويتأكد ان كل **data** توصل لل **Socket** و ال **Process** الخاص بيها

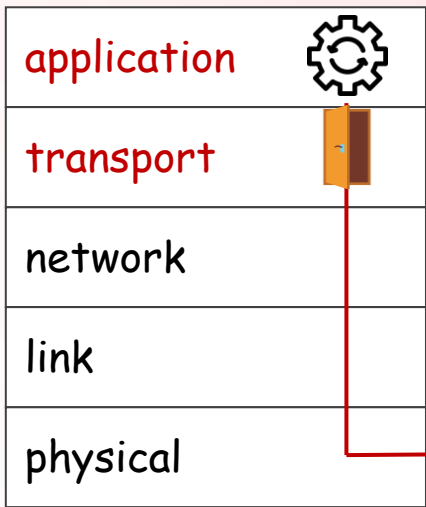
MULTIPLEXING/DEMULTIPLEXING EXAMPLE



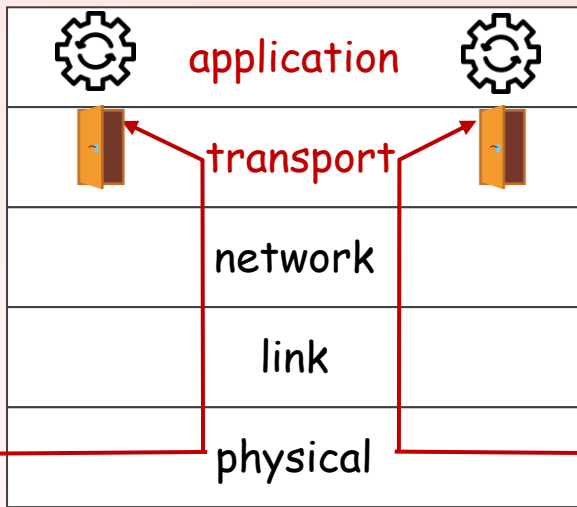
= socket



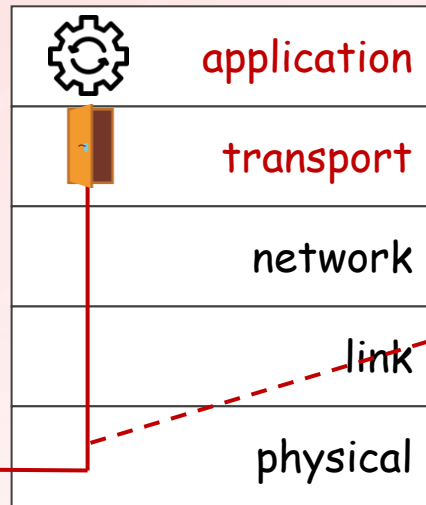
= process



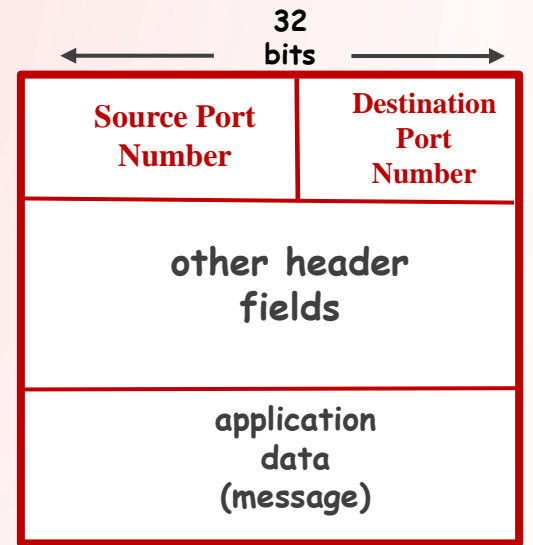
Host 1



Host 2



Host 3



TCP/UDP segment format

A decorative graphic on the left side of the slide, consisting of a network of thin, dark red lines and small circles, resembling a circuit board or a neural network structure.

CONNECTIONLESS TRANSPORT

UDP

UDP: USER DATAGRAM PROTOCOL [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
 - lost
 - delivered out of order to app
- **connectionless:**
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

Why is there a UDP ?

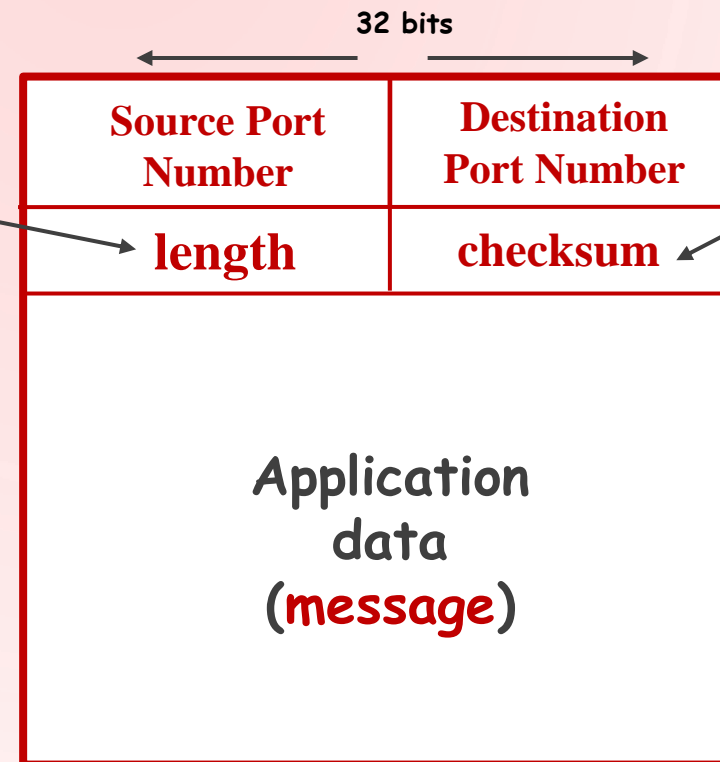
- no connection establishment (which can add delay)
- **simple:** no connection state at sender, receiver
- small segment header
- **no congestion control:** UDP can blast away as fast as desired

• ده تعريف لل UDP والمميزات والعيوب ونفس الكلام اللي اتقال قبل كدا

UDP: MORE

- often used for **streaming multimedia apps**
 - loss tolerant
 - rate sensitive
- other **UDP** uses
 - DNS
 - SNMP
- reliable transfer over **UDP**: add **reliability at application layer**
 - application-specific error recovery!

Length, in bytes of UDP segment, including header



TCP/UDP segment format

ال **Checksum** يكون رقم الهدف منه اني اتأكد ان الرسالة سليمة ومحصلة اي مشاكل لو الرسالة وصلت والرقم كان سليم الرسالة بتعملها **Accept** لو الرقم اتغير يتعمل **Decline** والرسالة مش بتتبعث تاني علشان احنا بنستخدم ال **UDP**

UDP CHECKSUM

GOAL : DETECT ERRORS IN TRANSMITTED SEGMENT

- ال **Segment** بيبقي عندها **Source Port , Destination Port, Length , Payload** كل دول بيتم تجزئتهم علي **16 bit** او **2 bytes** وبيتجمعهم **تاني** علشان بيطلع معايا ال **Sum** وده لو جبت منه ال **1's Complement** بيطلع معايا ال **Checksum**

- ال **Checksum** بيكون عبارة عن رقم باينري بيكون **16 bit** بينضاف في ال **Segment** عند ال **Sender**

- ال **Receiver** بيستلم ال **Segment** وبيعمل نفس عملية **التجزئة والجمع** الي بتتعمل عن ال **Sender** ولكن بيجمع معاهم ال **Checksum** فا بيطلع **رقم جديد** , **الرقم ده** لو كله كان **1** يبغي الرسالة محصلهاش **Corruption** وسليمة

- لو الرقم مطلعش كله **1** عند ال **Receiver** بيكون حصل **Error** والرسالة مبتتبعش تاني لاننا بنستخدم ال **UDP**

INTERNET CHECKSUM EXAMPLE

- Note

- When adding numbers, a **carryout** from the most significant bit needs to be added to the result

- **Example:** add two 16-bit integers

	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																	
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
<hr/>																	
sum	1	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

A decorative graphic on the left side of the slide, consisting of a network of thin red lines and small circles, resembling a circuit board or a stylized tree structure.

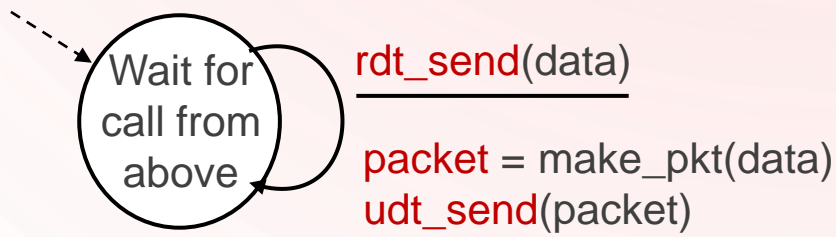
PRINCIPLES OF RELIABLE DATA TRANSFER

RDT & GO-BACK-N & SELECTIVE REPEAT

RDT 1.0

RELIABLE TRANSFER OVER A RELIABLE CHANNEL

- هنا بنفترض ال **Channel** بتكون في الحالة المثالية , انه ميكونش في **Packet Loss** او **Bit Errors**
- فابساطة ال **Receiver** هيستلم ال **Packet** من **Sender** منغير ما يعمل اي حاجة اضافية زي انه بيعت **ACKs(Acknowledgements)** علشان يعرف ان ال **Packet** وصلت سليمة ولا لا وده بتكون الفكرة اللي هتتبنى عليها ال **RDT 2.0**
- وجود **Reliable Channel** يعتبر حاجة صعبة جدا انها تبقي موجودة



sender



receiver

RDT 2.0

CHANNEL WITH BIT ERRORS

- هنا بيكون عندي **Unreliable Channel** وهنا يمكن يحصل **Bit Errors**
- ونقدر نكتشف ال **Bit Errors** عن طريق ال **Checksum**
- لو اكتشفنا ان ال **Packet** حصلها **Bit Error** او **Corruption** ساعتها بنبعث **NAKs (Negative Acknowledgements)** لل **Sender** , ولما يعرف ال **Sender** ان ال **Packet** حصلتها مشاكل بيبعثها تاني لل **Receiver**
- ولو ال **Packet** وصلت سليمة بنبعث **ACKs** لل **Sender** انه ال **Packet** وصلت منغير مشاكل
- فكرة ال **NAKs** و ال **ACKs** بتساعدني انه يبقي عندي **Error detection** و **Control Message**

RDT 2.0

HAS A FATAL FLOW

- برضو يمكن ال **NAKs** او **ACKs** يحصلهم **Bit Error** وده لانهم برضو يعتبرو **Message** , فا ايه اللي بيحصل لما واحد فيهم بيحصله **Corruption** ؟
- لو ال **NAKs** او ال **ACKs** حصلهم **Corruption** ال **Sender** مش هيعرف يفرق اذا كانت الرسالة اللي بتوصله من ال **Receiver** ده كانت **ACKs** ولا **NAKs**
- في كل مرة ال **Sender** بيبعت فيها **Segment** بيضيف معاها **Sequence Number** وده بيكون رقم ثابت و مبيتكرش لنفس ال **Segment**
- فا الحل ان ال **Sender** بيعت ال **Packet** تاني لل **Receiver** لو كان ال **Message** اللي وصلت **NAKs** فا ساعتها مش هتحصل مشكلة ال **Duplicate** لان ال **Packet** اللي اتبعتت في المرة الاولى مش هتكون موجودة
- بس لو كانت الرسالة **ACKs** فا هنا ال **Receiver** بيعمل **Check** علي ال **Sequence Number** لو كان نفس ال **Sequence Number** ال **Packet** اللي هو استلمه بيعمل **Discard** وده بهدف اننا نحل مشكلة ال **Duplicate**

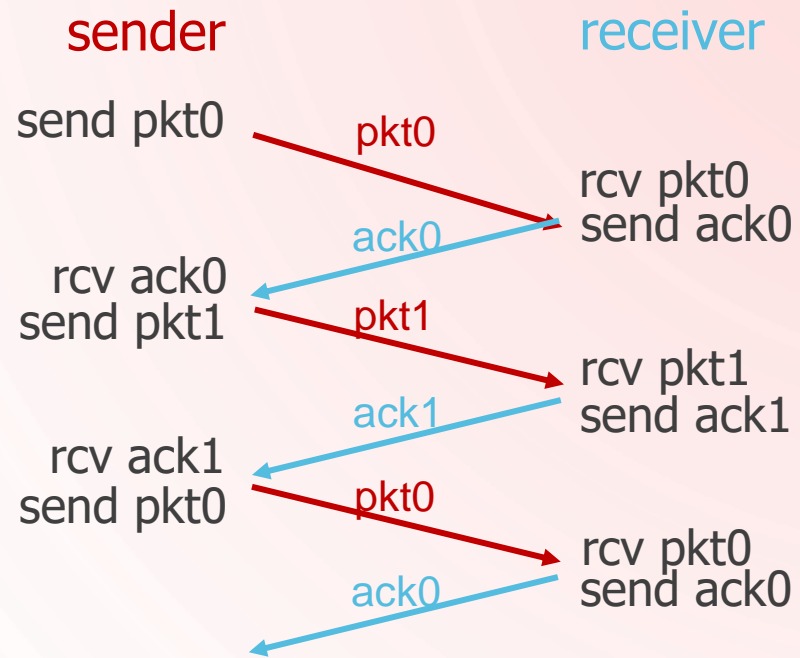
Sender sends one packet,
then waits for receiver
response

RDT 3.0

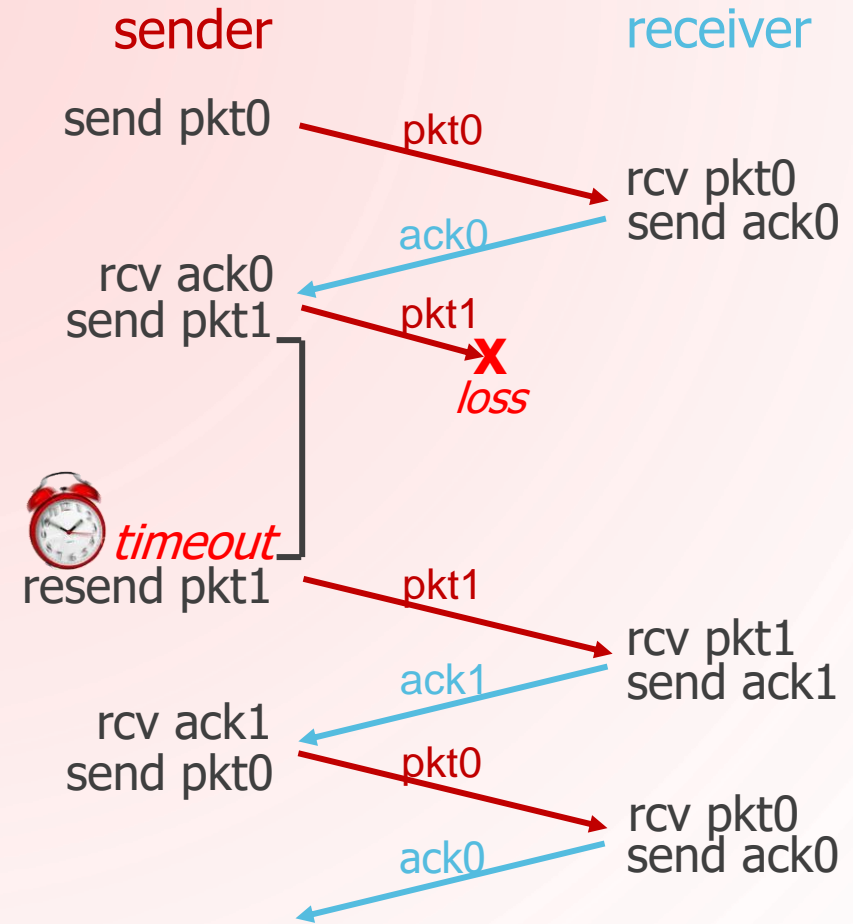
CHANNELS WITH ERRORS AND LOSS

- هنا يمكن يحصل **Packet Loss** لل **Packets** التي بتتبعث او ل **ACKs** التي بستمته من ال **Receiver**
- هنا بيكون عندي اضافة جديدة عند ال **Sender** وهي ال **Reasonable Time** وده بيكون زي **Countdown Timer** بيستاه ال **Sender** لحد ما ال **ACKs** توصله
- لو ال **Countdown Timer** بقا **Timeout** وموصلش **ACKs** لل **Sender** , ال **Sender** بيبعت ال **Packet** ثاني لل **Receiver**
- ممكن ال **ACKs** تتاخر نتيجة انه حصل **Delay** ومش انه حصلها **Packet Loss** وفي الحالتين ال **Sender** هيبعت ال **Packet** ثاني لو موصلش **ACKs** خلال فترة ال **Countdown Timer**
- وطبعاً في احتمالية انه يحصل **Duplication** وعلشان نحلها بنستخدم ال **Sequence Number** التي قولناها في ال **RDT 2.0**

RDT3.0 IN ACTION

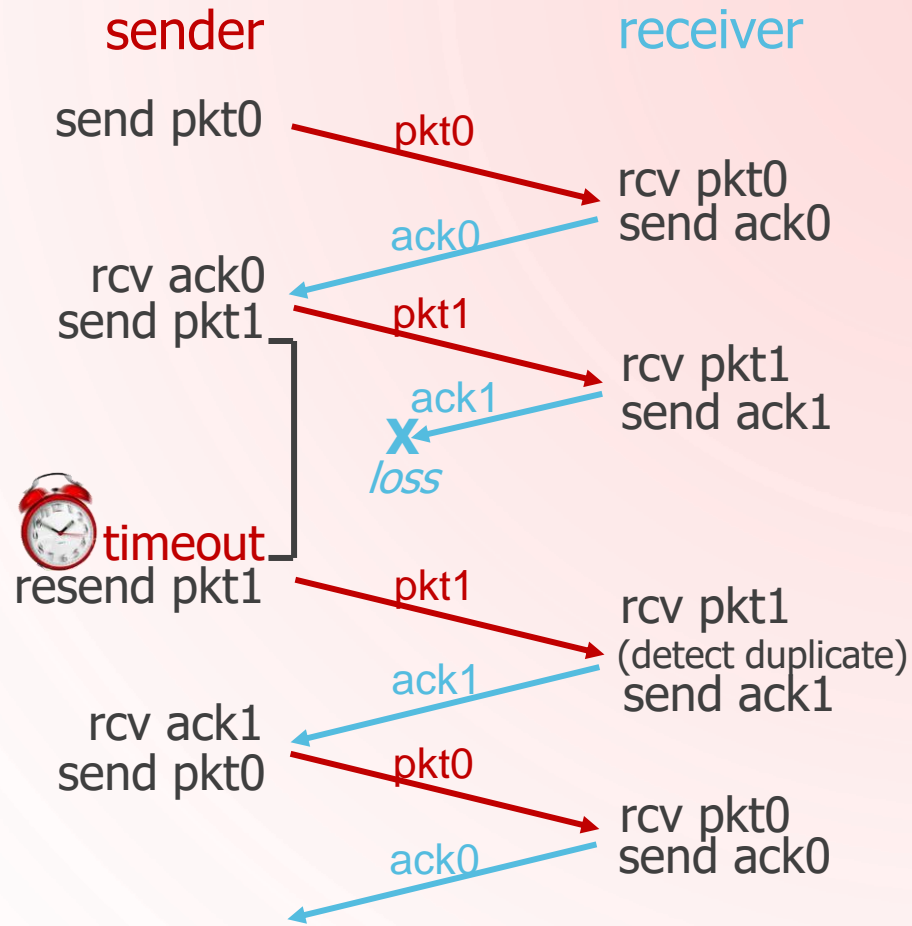


(a) no loss

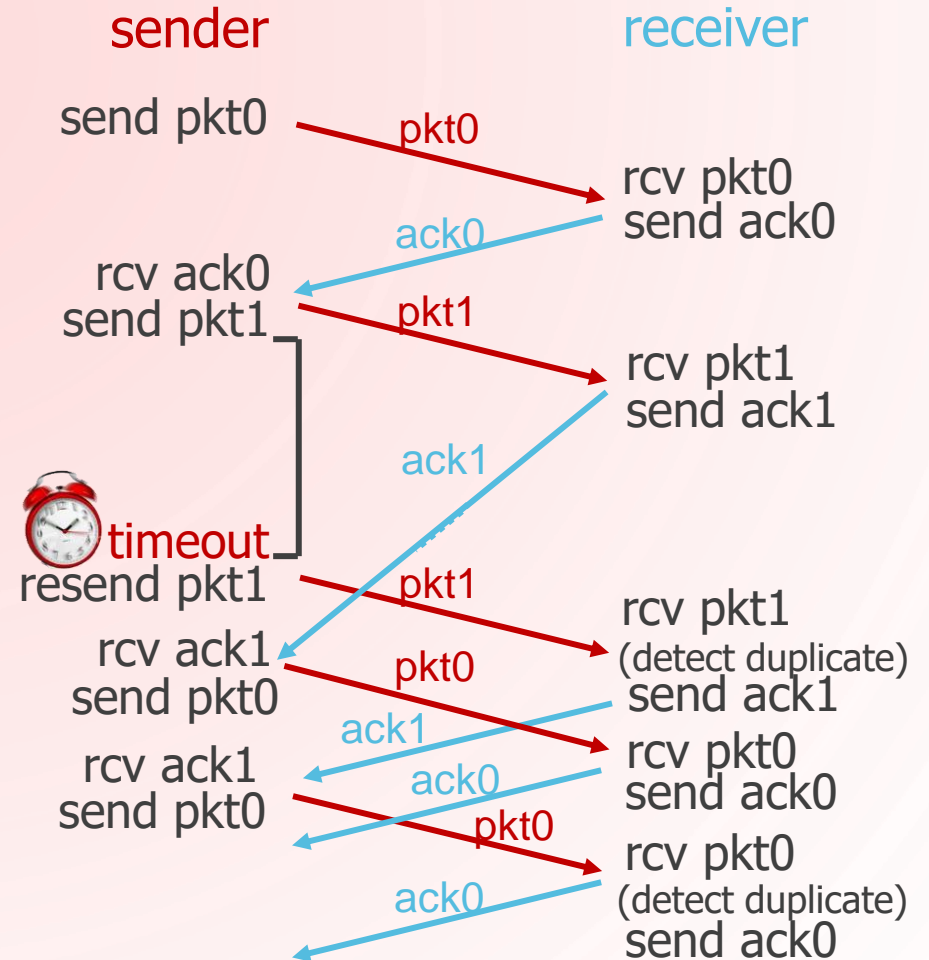


(b) packet loss

RDT3.0 IN ACTION



(c) ACK loss



(d) premature timeout/ delayed ACK

RDT 3.0

PERFORMANCE

- رغم ان ال **RDT 3.0** يقدم **Reliable Connection** ويؤكد ان ال **Packet** تكون وصلت سليمة ومنغير مشاكل
- الا ان ال **RDT 3.0** عنده مشكلة في ال **Performance** وهو انه بيستتي وقت طويل جدا علشان يبعث **Packet** تانية او ال **Packet** اللي بعدها
- ال **Countdown Timer** اللي بيستناه ال **Sender** بيكون علي اساس ال **RTT(Round Trip Time)**

PERFORMANCE OF RDT3.0

- **rdt3.0** is correct, but performance stinks
- e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

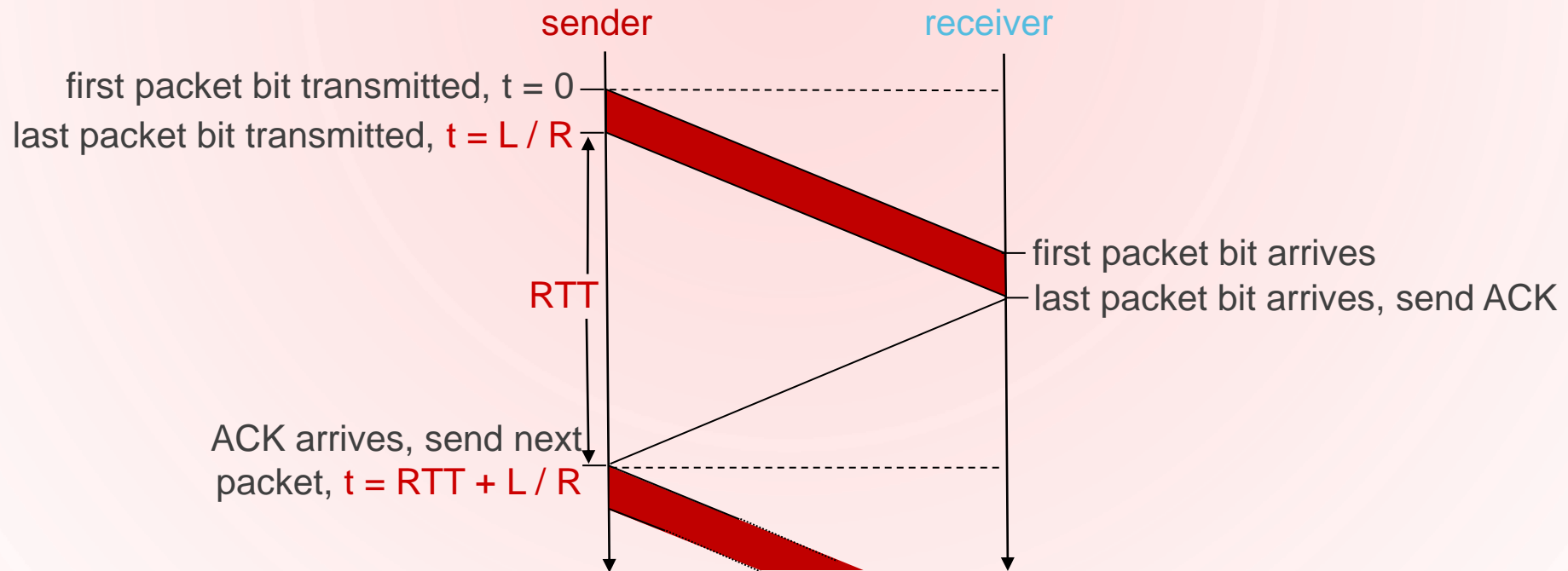
$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

- U_{sender} : **utilization** – fraction of time sender busy sending

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- if **RTT**=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- network protocol **limits use of physical resources!**

RDT3.0: STOP-AND-WAIT OPERATION



$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Utilization in networking refers to the degree to which a sender effectively employs available resources, such as bandwidth or processing power, to transmit data.

A decorative graphic on the left side of the slide, consisting of a network of thin red lines and small circles, resembling a circuit board or a stylized tree structure.

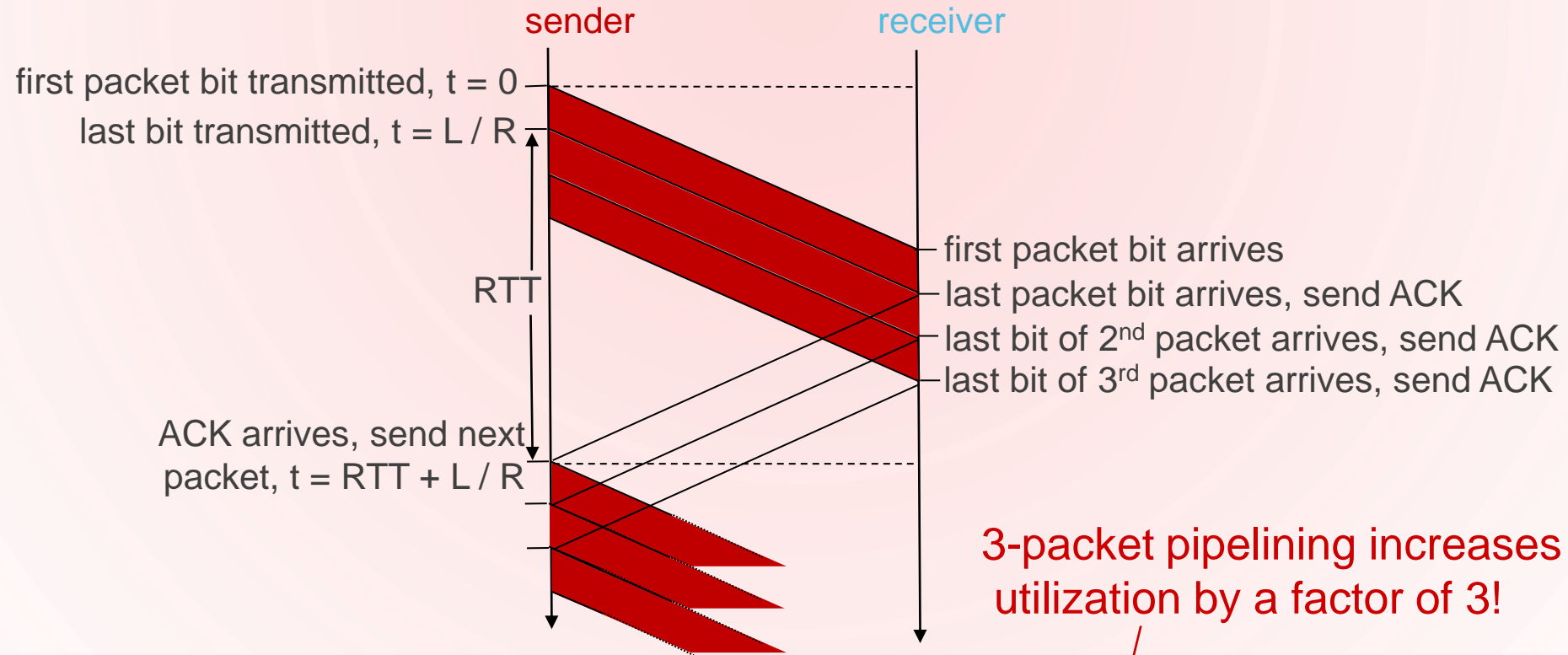
PIPELINED PROTOCOLS

GO-BACK-N & SELECTIVE REPEAT

PIPELINED PROTOCOLS

- مفهوم ال **Pipelining** انه ال **Sender** يبعث اكثر من **Packet** مع بعض وبعدها بيستلم ال **ACKs** ليهم كلهم , بدل ما كان بيبعث **Packet** ويستتي وقت لحد ما ال **ACKs** بتوصل ويبعث الثانية
- وهنا بيحل مشكلة ال **Performance** لانه ال **Sender** وهو بيبعث عدد **n** من ال **Packets** مثلا فا هو بيستفيد بجزء من ال **RTT** اللي هو كان بيستتي فيهم
- ولما بيكون بيبعث اخر **Packet** احتمال ال **ACKs** توصله وكدا يبدأ يبعث ال **Packets** اللي بعدها ويكون وفر وقت
- في طريقتين يمكن ننفذ بيهم ال **Pipelining** اللي هم : **Go-Back-N** وال **Selective Repeat**

PIPELINING: INCREASED UTILIZATION



$$U_{\text{sender}} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

PIPELINED PROTOCOLS: OVERVIEW

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
 - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
 - when timer expires, retransmit *all* unacked packets

Selective Repeat:

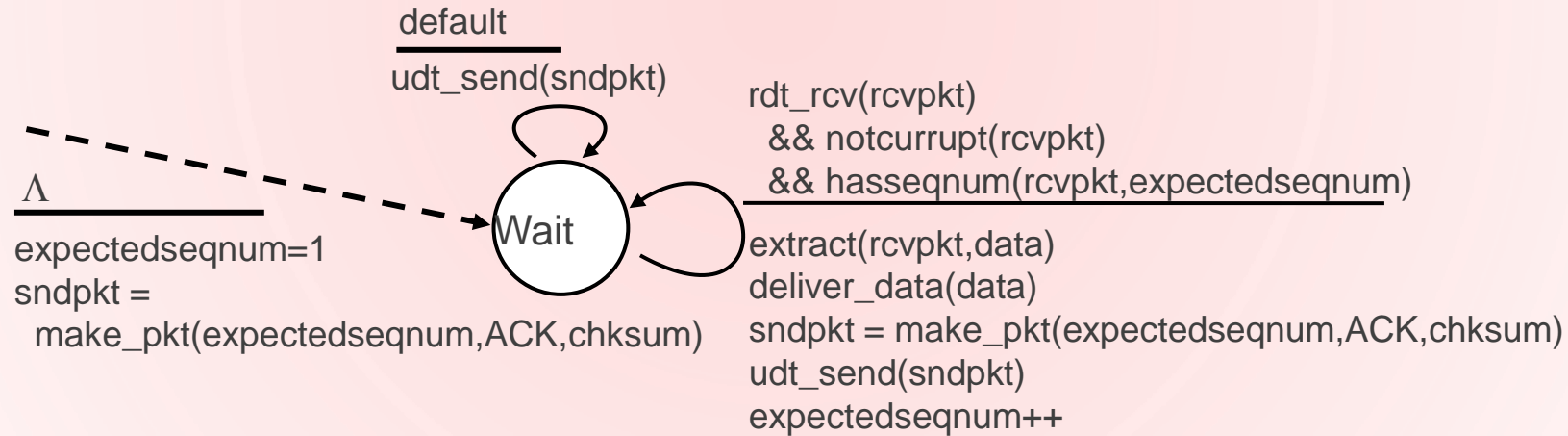
- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
 - when timer expires, retransmit only that unacked packet

GO - BACK - N

WHEN TIMER EXPIRES, RETRANSMIT ALL UNACKED PACKETS

- هنا ال **Sender** بيبيع ال **Packets** ويستتي ال **ACKs** لكل ال **Packets** انها توصله
- لو حصل **Timeout** وفي **ACKs** موصلتش ل **Packet** معينة , ال **Sender** بيبيع كل ال **Packets** بداية من اخر **ACK** وصلته
- يعني مثلا لو بعت **10 Packets** وحصل **Timeout** واخر **ACK** وصلي كان **ACK** ل **Packet** رقم **4** , فا ال **Sender** هيبعت كل ال **Packets** من **5** لحد **10**

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

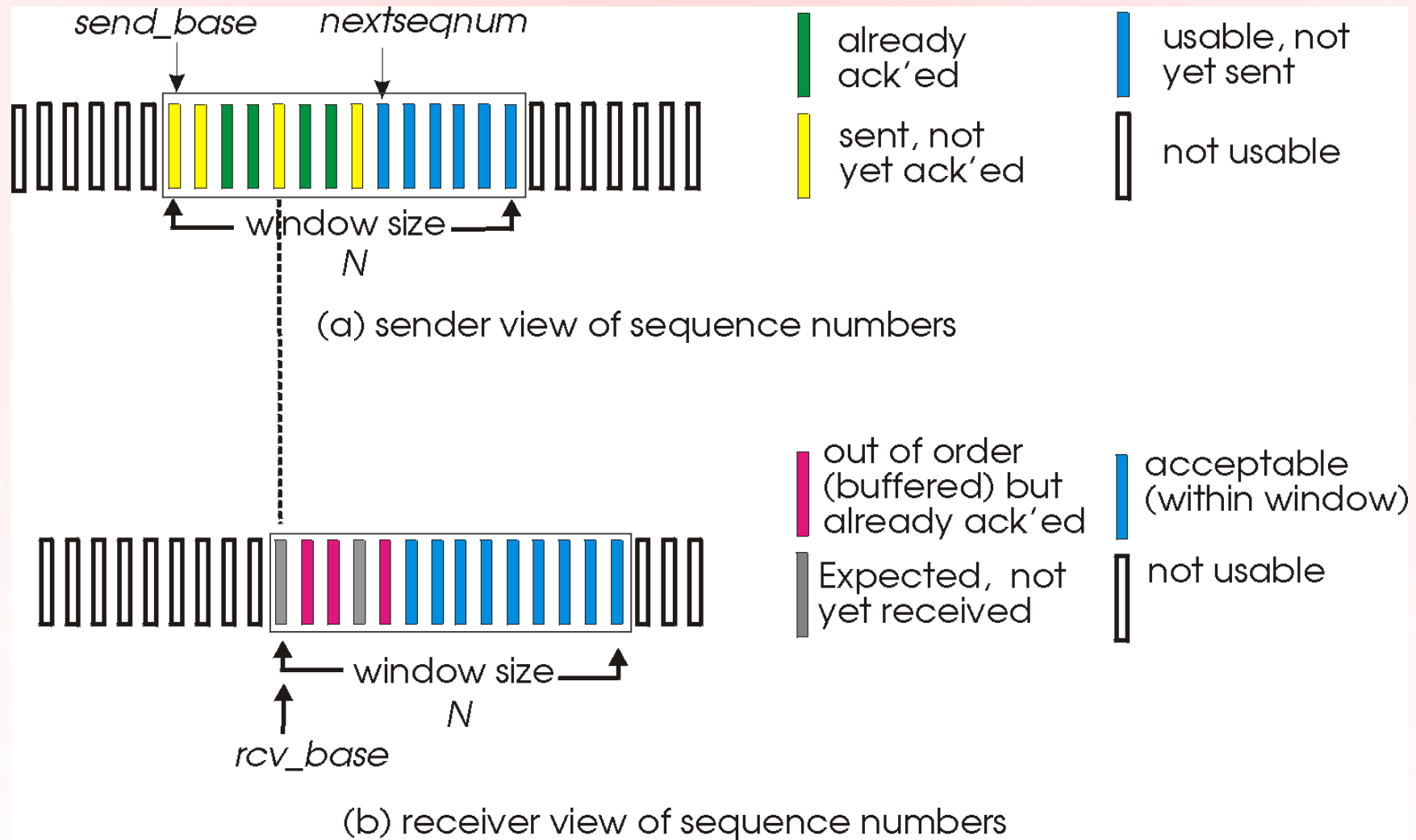
rcv pkt5, deliver, send ack5

SELECTIVE REPEAT

WHEN TIMER EXPIRES, RETRANSMIT ONLY THAT UNACKED PACKET

- هنا ال **Sender** برضو بيبيع ال **Packets** كلها ويستني ال **ACKs** ليهم انهم يتبعو
- ولما يحصل **Timeout** وفي **Packets** موصلش ال **ACK** الخاص بيها بيبيع ال **Packets** ده بس
- مثلا لو بعنا **10 Packets** , وحصل **Timeout** وموصلش **ACKs** غير ل **Packets 1,3,5,7** فا هبع ال **Packets** الي منغير **ACKs** وهيكونو **Packets 2,4,6,8,9,10**

SELECTIVE REPEAT: SENDER, RECEIVER WINDOWS



SELECTIVE REPEAT

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]:

- mark pkt n as received
- if n 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

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

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

Q: what happens when ack2 arrives?



CONNECTION-ORIENTED TRANSPORT:TCP

SEGMENT STRUCTURE & RELIABLE DATA TRANSFER &
CONTROL & CONNECTION MANAGEMENT

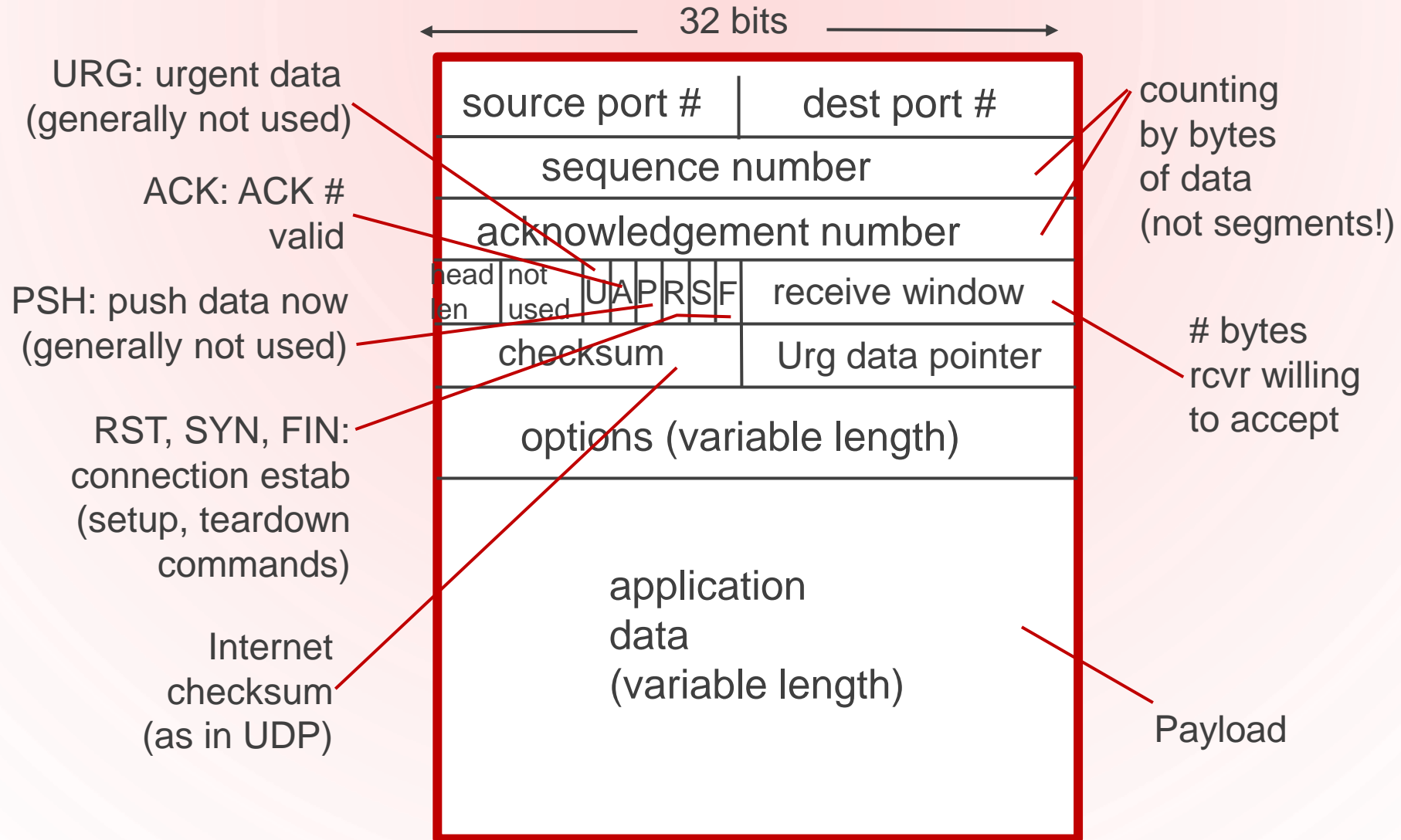
TCP: OVERVIEW

RFCS: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
 - one sender, one receiver
- **reliable, in-order *byte stream*:**
 - no “message boundaries”
- **pipelined:**
 - TCP congestion and flow control set window size
- ***send & receive buffers***
- **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- **connection-oriented:**
 - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- **flow controlled:**
 - sender will not overwhelm receiver



TCP SEGMENT STRUCTURE



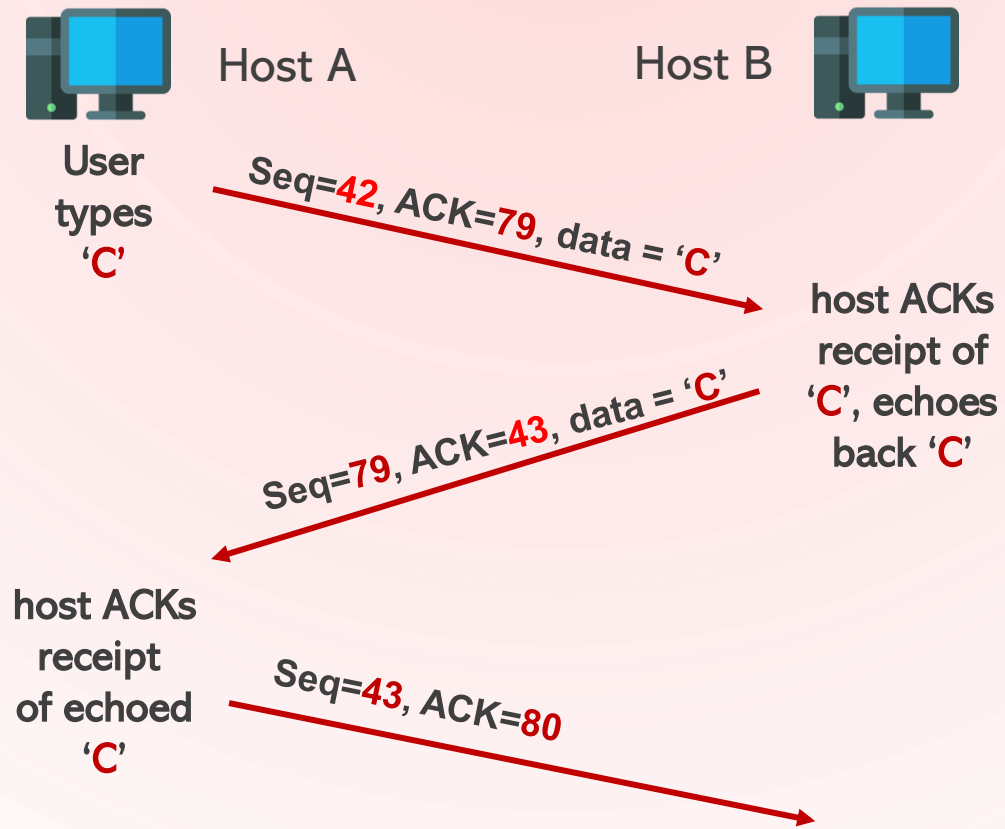
TCP

SEQUENCE NUMBER & ACKs

- ال **Sequence number** يكون رقم يحدد علي حسب حجم ال **Packet** وعدد ال **Segments** اللي **Packet** هتقسم ليها
- ال **Sequence Number** لاول **Segment** بتكون بتساوي 0
- يعني مثلا لو هبعث **Packet** حجمها **100 byte** و هقسمها ل **20 Segment** يبقى ال **Sequence Number** هيزيد بمقدار **100/20** اللي هي بتساوي 5
- يعني في المثال ده ارقام ال **Sequence** هتبقى , اول **Segment** هتكون ب 0 , ثاني **Segment** هتكون ب 5 , وتالت **Segment** هتكون ب 10 , وهكذا
- ال **Sequence Number** برضو حل لمشكلة ال **Duplication** اللي بتحصل في ال **RDT 2.0**

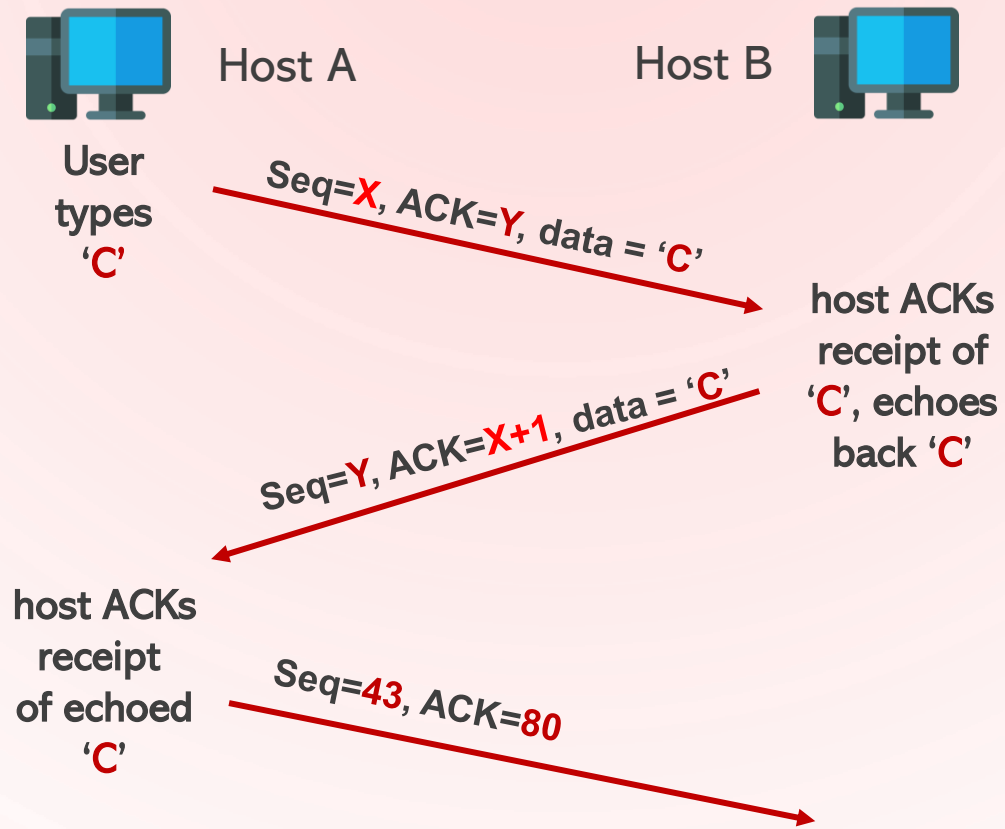
TCP

SEQUENCE NUMBER & ACKs



TCP

SEQUENCE NUMBER & ACKs



ANOTHER VIEW

TCP RELIABLE DATA TRANSFER

- **TCP** يستخدم ال **RDT 3.0** علشان يقدر يحقق **Reliable Data Transfer**
- بنستخدم برضو ال **Pipelined Segments** علشان مشكلة ال **Performance** اللي كانت موجودة في ال **RDT 3.0**
- ال **Sender** بيبيع ال **Packet** تاني لل **Receiver** في حالة انه حصل **Timeout** ومستقبلش **ACKs** لل **Packet** , او في حالة انه استقبل **Duplicated ACKs** لنفس ال **Packet**
- لما ال **Sender** يستقبل **Duplicated ACKs** يبقى ده معناته انه ال **Receiver** استلم ال **Segments Out-of-Order** او ال **Segment** حصلها **Lost**

TCP SENDER EVENTS:

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`

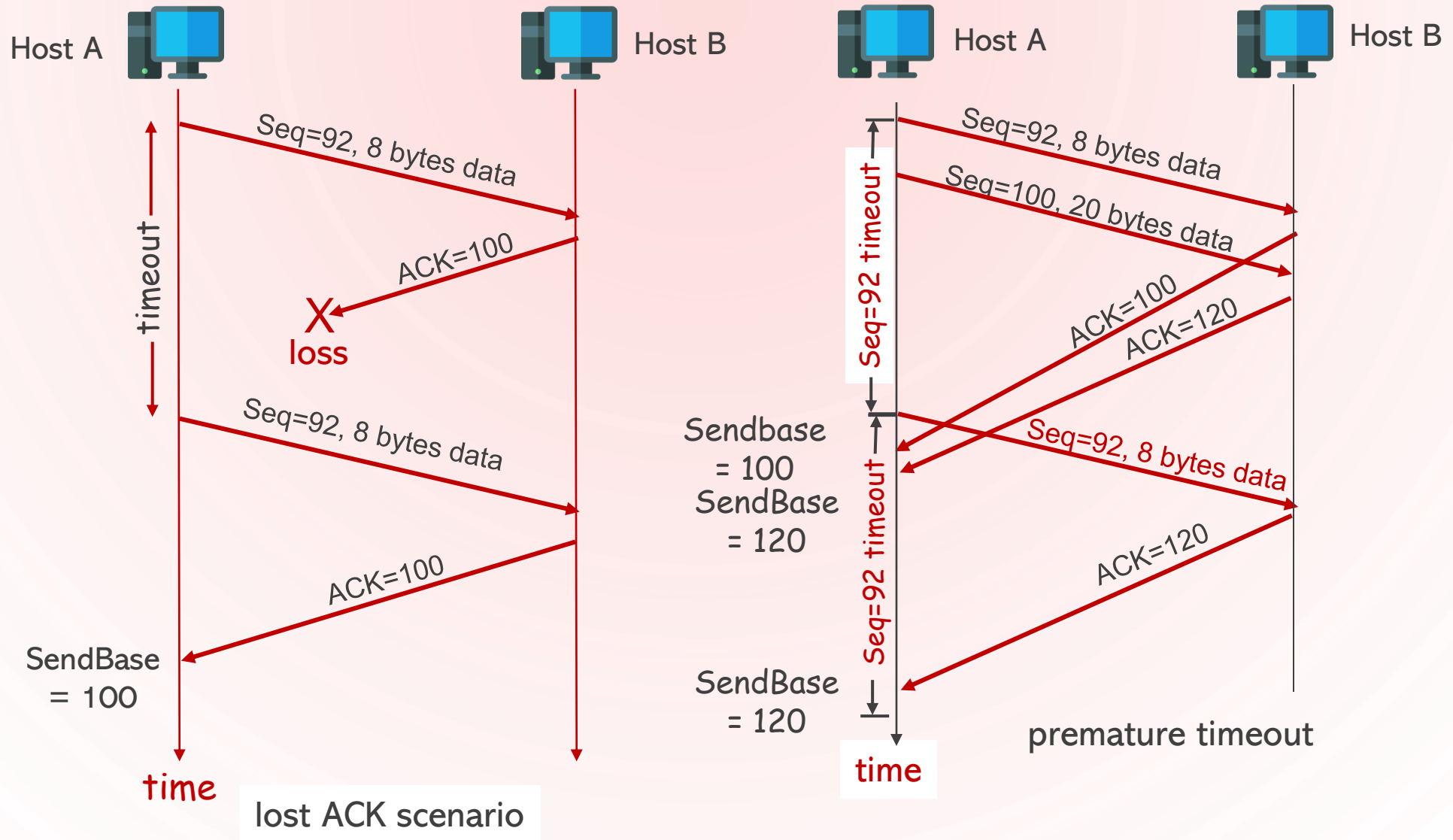
timeout:

- retransmit segment that caused timeout
- restart timer

Ack rcvd:

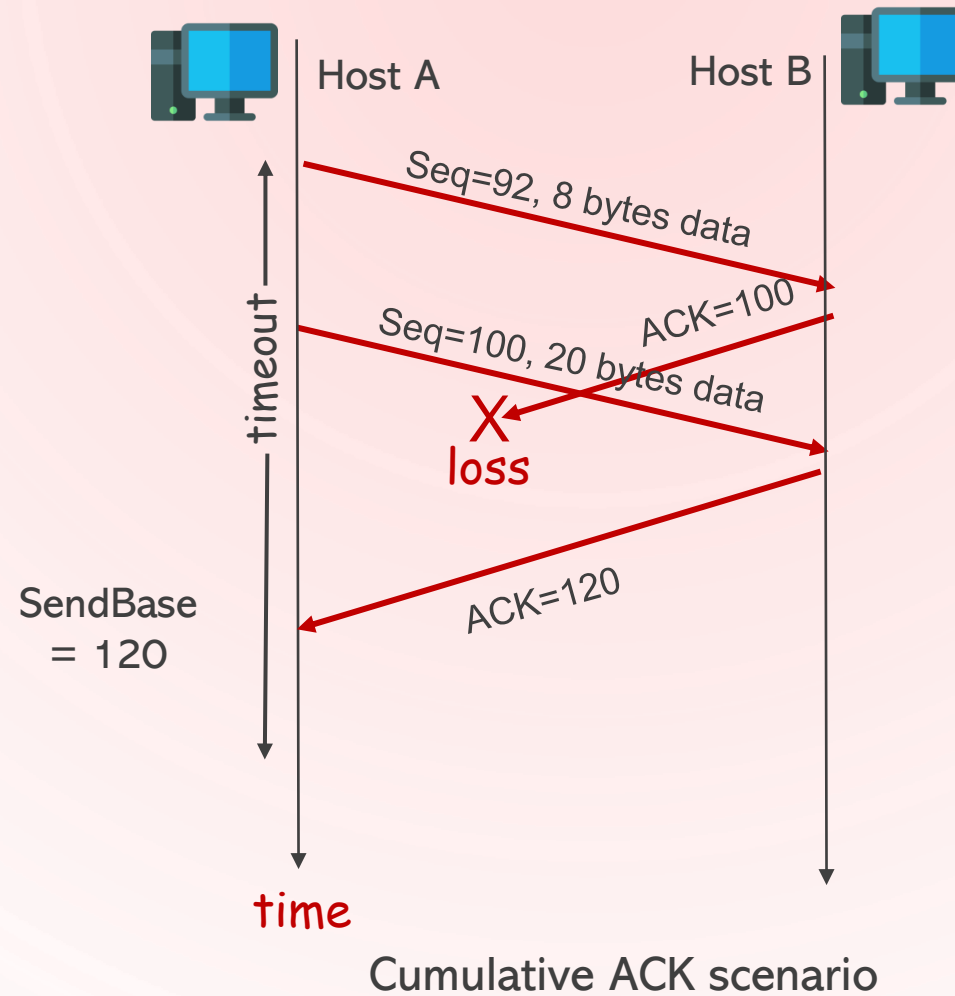
- If acknowledges previously unacked segments
 - update what is known to be acked
 - start timer if there are outstanding segments

TCP: RETRANSMISSION SCENARIOS



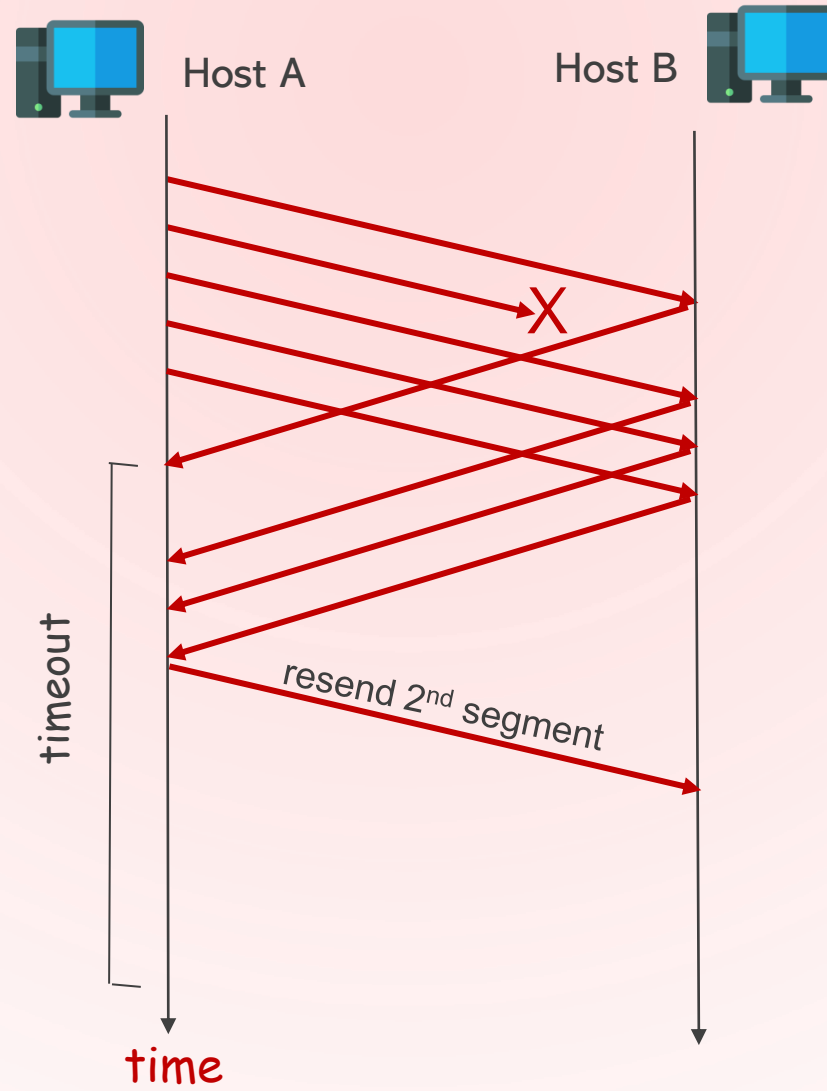
TCP RETRANSMISSION SCENARIOS

(MORE)



FAST RETRANSMIT

- 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.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
 - fast retransmit: resend segment before timer expires



A decorative graphic on the left side of the slide, consisting of a network of red lines and small circles, resembling a circuit board or a neural network structure.

TCP

FLOW CONTROL & CONGESTION CONTROL

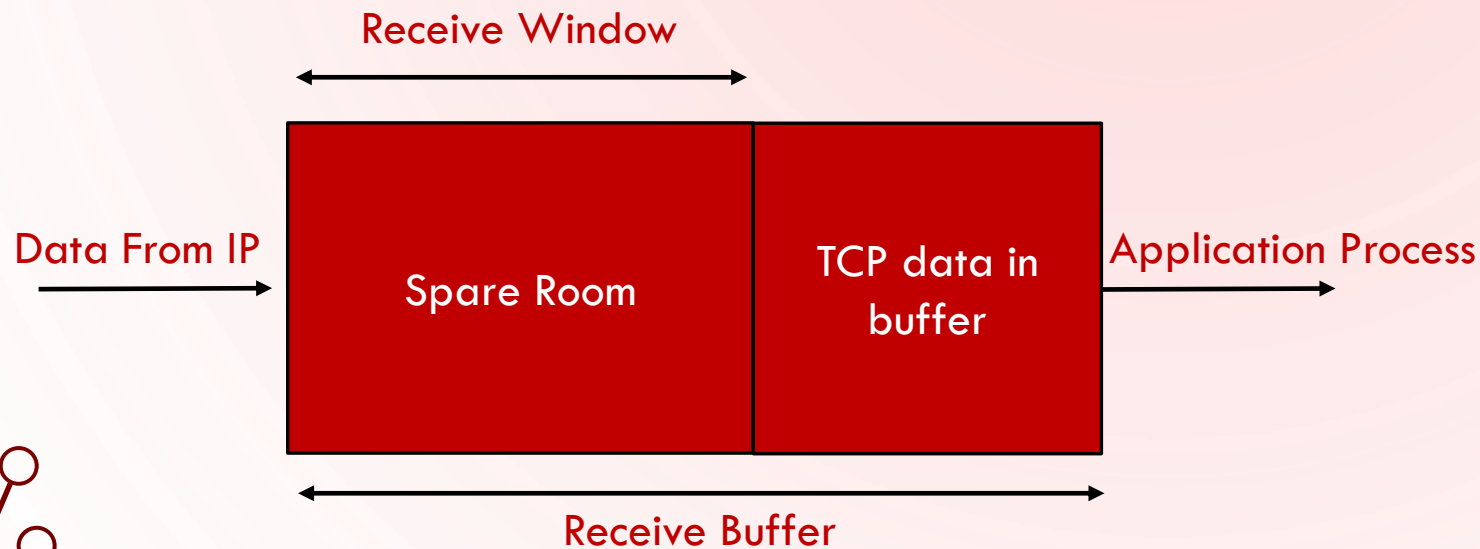
TCP

FLOW CONTROL:SENDER WON'T OVERFLOW RECEIVER'S BUFFER BY TRANSMITTING TOO MUCH , TOO FAST

- ال **Receiver** يكون عنده **buffer** ليه **مساحة** او **حجم** معين لو اتملي هحصل ال **Packet loss** او **Congestion**
- ال **TCP** بيقدم ال **Flow Control** وده عن طريق انه بيحسب هيبعت قد ايه من ال **Packets** كل مرة علي حسب مساحة ال **Buffer** اللي عند ال **Receiver**
- لما ال **Sender** بيبعت **Segment** لل **Receiver** , ال **Receiver** بيبعت لل **Sender** ال **ACKs** او ال **Feedback** وبيكون معاه ال **Receiver Window**
- ال **Receiver Window** هو بيكون مقدار ال **Data** اللي يقدر ال **Receiver** يستقبلها
- ال **Sender** لما يستلم ال **ACKs** بيعرف كمية ال **Data** اللي يقدر ال **Receiver** يستقبلها في المرة الجاية اللي هيبعت فيها **Packets** , وعلي اساسها هيبعت ال **Packets** بكميات معينة بحيث ميحصلش **Overload** لل **BufferSize** ويحصل **Packet Loss**

TCP

FLOW CONTROL:SENDER WON'T OVERFLOW RECEIVER'S BUFFER BY TRANSMITTING TOO MUCH , TOO FAST



- spare room in buffer
= $RcvWindow$
= $RcvBuffer - [LastByteRcvd - LastByteRead]$

- speed-matching service:
matching the send rate to
the receiving app's drain
rate

- app process may be
slow at reading from
buffer



TCP

CONNECTION MANAGEMENT – 3 WAY HANDSHAKE

الجزئية ده صعب شرحها كتابتا فا ده فيديو كويس بيشرحها

https://youtu.be/xMtP5ZB3wSk?si=_krxSN-0kaze1mls



PRINCIPLES OF CONGESTION CONTROL

END TO END & NETWORK - ASSISTED CONGESTION CONTROL

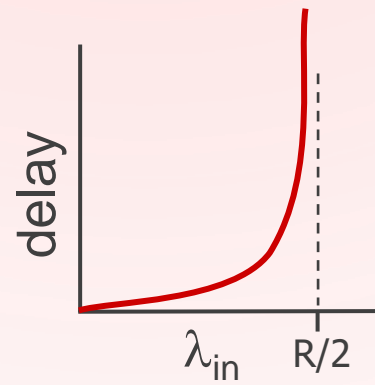
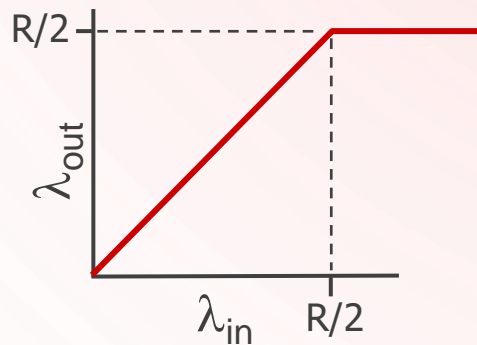
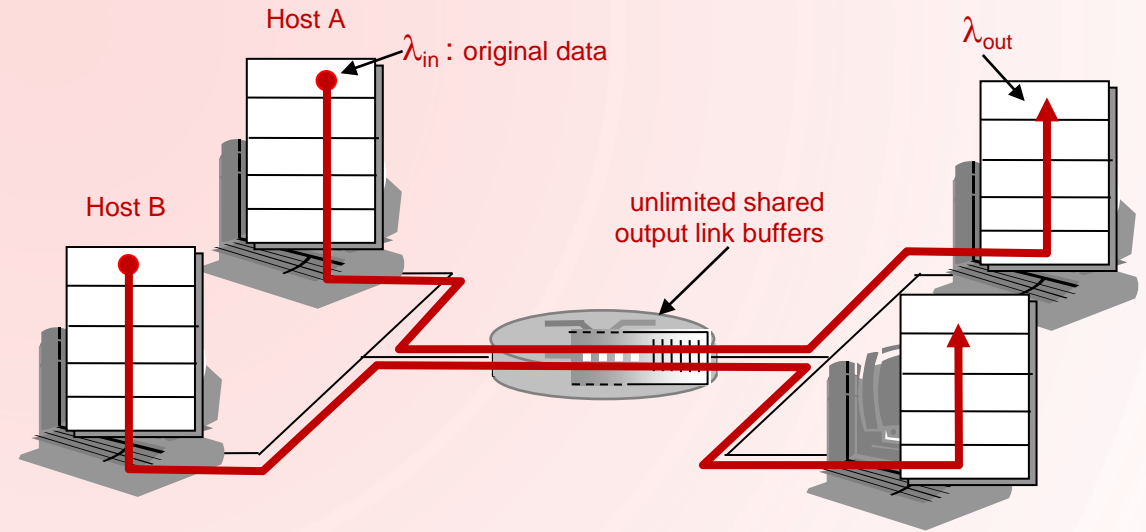
CONGESTION

TOO MANY SOURCES SENDING TOO MUCH DATA TOO FAST FOR NETWORK TO HANDLE

- مشكلة ال **Congestion** تحصل لما اكثر من مصدر او **Sender** يبيعت لنفس ال **Receiver** , فا ال **Buffer Size** الخاص ب **Router** بيتملي ويحصل **Congestion** او تزامم لل **Data** وده بيؤدي لل **Packet loss**
- ال **Congestion Control** وال **Flow Control** مشكلتين مختلفين
- ال **Packet loss** يحصل بسبب انه حصل **Overflow** لل **Buffer** الخاص بال **Routers**
- ال **Congestion Problem** ممكن تسبب **Delay** وده بسبب انه ال **Packet** لما توصل لل **Router** وال **Buffer** بيكون **Full** , بتستني في **queues** لحد ما يكون في مكان في ال **Buffer**

CAUSES/COSTS OF CONGESTION: SCENARIO 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



- large delays when congested
- maximum achievable throughput
- maximum per-connection throughput: $R/2$
- large delays as arrival rate, λ_{in} , approaches capacity

END TO END VS NETWORK-ASSISTED

CONGESTION CONTROL

END TO END CONGESTION CONTROL	NETWORK-ASSISTED CONGESTION CONTROL
no explicit feedback from network	routers provide feedback to end systems
congestion inferred from end-system observed loss, delay	single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
approach taken by TCP	explicit rate sender should send at This approach aims to provide more precise and timely information about network conditions to the end systems, enabling them to adjust their behavior more effectively to prevent congestion.



TCP

CONGESTION CONTROL

[HTTPS://YOUTU.BE/GBQPLCZ1MGC?SI=JFWKLKGCWI3PZUVG](https://youtu.be/GBQPLCZ1MGC?si=JFWKLKGCWI3PZUVG)

(FROM 14:50) , FROM SLIDE 63 IN THE CHAPTER 3 POWERPOINT