ECCS-3631
Networks and Data Communications

Module 5-2: User Datagram Protocol
Principles of Reliable Data Transfer
Pipeline Protocols

Dr. Ajmal Khan

UDP: User Datagram Protocol

No Frills, Bare-bones transport protocol

UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer. The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.

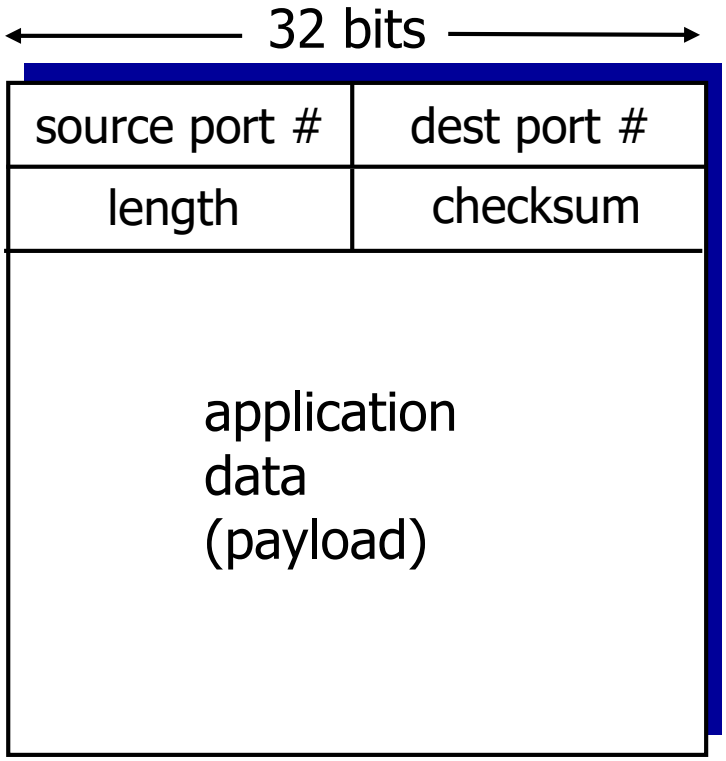
There is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be ***connectionless***.

UDP: User Datagram Protocol

- UDP is used in:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: segment header

UDP header has only four fields, each consisting of two bytes.
The length field specifies the number of bytes in the UDP segment (header plus data).
The checksum is used by the receiving host to check whether errors have been introduced into the segment.



UDP segment format

why is there a UDP?

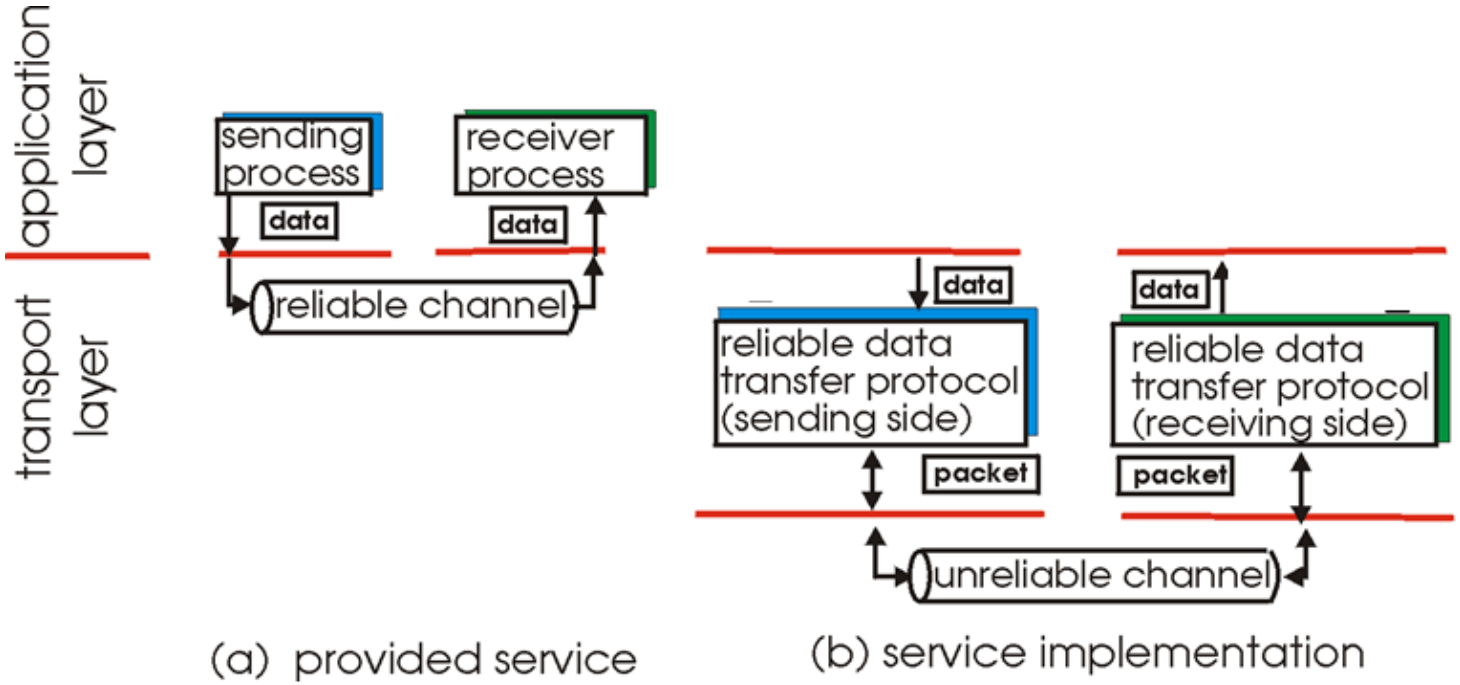
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

List of Applications that use UDP or TCP

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Principles of Reliable Data Transfer

The following figure illustrates the framework reliable data transfer (rdt). The service abstraction provided to the upper-layer entities is that of a reliable channel through which data can be transferred. With a reliable channel, no transferred data bits are corrupted (flipped from 0 to 1, or vice versa) or lost, and all are delivered in the order in which they were sent. This is precisely the service model offered by TCP to the Internet applications that invoke it.



We now study a series of protocols, each one becoming more complex, arriving at a flawless, reliable data transfer protocol (rdt).

Reliable Data Transfer over a Perfectly Reliable Channel (rdt 1.0)

- Underlying channel perfectly reliable
  - no bit errors
  - no loss of packets

All packet flow is from the sender to receiver; with a perfectly reliable channel there is no need for the receiver side to provide any feedback to the sender since nothing can go wrong! Note that the receiver is able to receive data as fast as the sender happens to send data. Thus, there is no need for the receiver to ask the sender to slow down!

Reliable Data Transfer over a Channel with Bit Errors (rdt 2.0)

A more realistic model of the *underlying channel is one in which bits in a packet may be corrupted.* Such bit errors typically occur in the physical components of a network as a packet is transmitted, propagates, or is buffered. It is assumed that all transmitted packets are received (although their bits may be corrupted) in the order in which they were sent.

Fundamentally, three additional protocol capabilities are required:

Error detection: A mechanism is needed to allow the receiver to detect when bit errors have occurred.

Receiver feedback: The positive (ACK) and negative (NAK) acknowledgment replies rdt 2.0 protocol back from the receiver to the sender. In principle, these packets need only be one bit long; for example, a 0 value could indicate a NAK and a value of 1 could indicate an ACK.

Retransmission: A packet that is received in error at the receiver will be retransmitted by the sender.

The sender will not send a new piece of data until it is sure that the receiver has correctly received the current packet. Because of this behavior, protocols such as rdt 2.0 are known as stop-and-wait protocols.

Reliable Data Transfer over a Channel with Bit Errors (rdt 2.0 and rdt 2.1)

Flaws in rdt 2.0: The possibility that the ACK or NAK packet could be corrupted!

When sender receives a garbled ACK or NAK packet, the sender simply resends the current data packet. This approach introduces duplicate packets into the sender-to-receiver channel. The fundamental difficulty with duplicate packets is that the receiver doesn't know whether the ACK or NAK it last sent was received correctly at the sender.

Solution (rdt 2.1): A simple solution is to add a new field to the data packet and have the sender number its data packets by putting a sequence number into this field. The receiver then need only check this sequence number to determine whether or not the received packet is a retransmission.

Since it is assumed the channel does not lose packets, ACK and NACK packets do not themselves need to indicate the sequence number. A 1-bit sequence number will suffice to determine whether or not the received packet is a retransmission.

Reliable Data Transfer over a Channel with Bit Errors (rdt 2.1)

Protocol rdt 2.1 uses both positive and negative acknowledgments from the receiver to the sender. When an out-of-order packet is received, the receiver sends a positive acknowledgment for the packet it has received. When a corrupted packet is received, the receiver sends a negative acknowledgment.

Flaws in rdt 2.1: Acknowledgement for every single packet. Too many acknowledgements.

Solution (rdt 2.2): Only send an ACK for last correctly packet received. Receiver must explicitly include sequence number of packet being ACKed.

Reliable Data Transfer over a Channel with Bit Errors (rdt 2.2)

Instead of sending a NAK, send an ACK for the last correctly received packet. A sender that receives two ACKs for the same packet (that is, receives duplicate ACKs) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice. So NAK-free reliable data transfer protocol for a channel with bit errors is rdt 2.2.

Reliable Data Transfer over a Lossy Channel with Bit Errors (rdt 3.0)

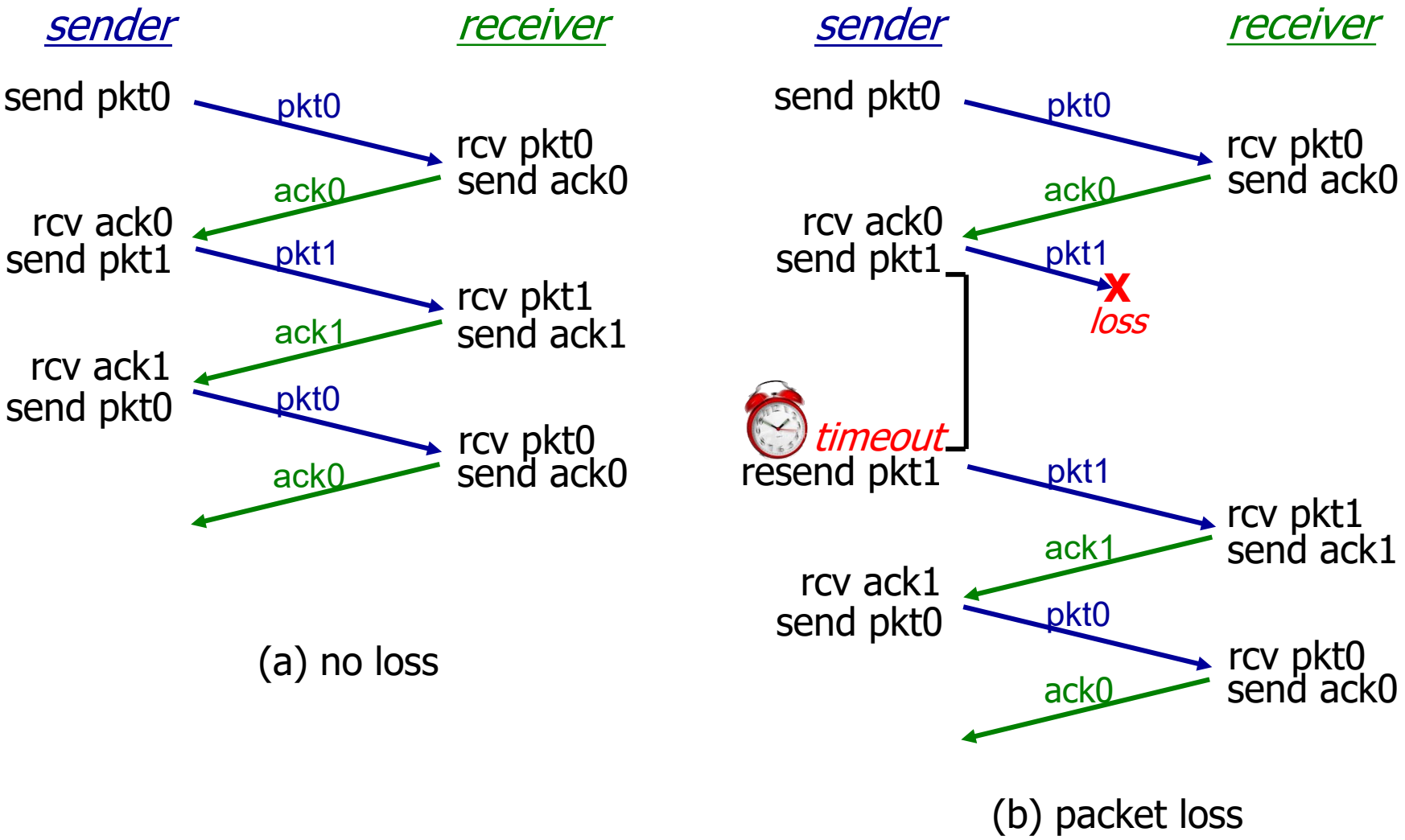
Suppose now that in addition to corrupting bits, *the underlying channel can lose packets* as well, a common event in today's computer networks.

Two additional concerns must now be addressed by the protocol: how to detect packet loss and what to do when packet loss occurs. The use of checksumming, sequence numbers, ACK packets, and retransmissions — the techniques already developed in rdt 2.2 — will allow to answer the latter concern.

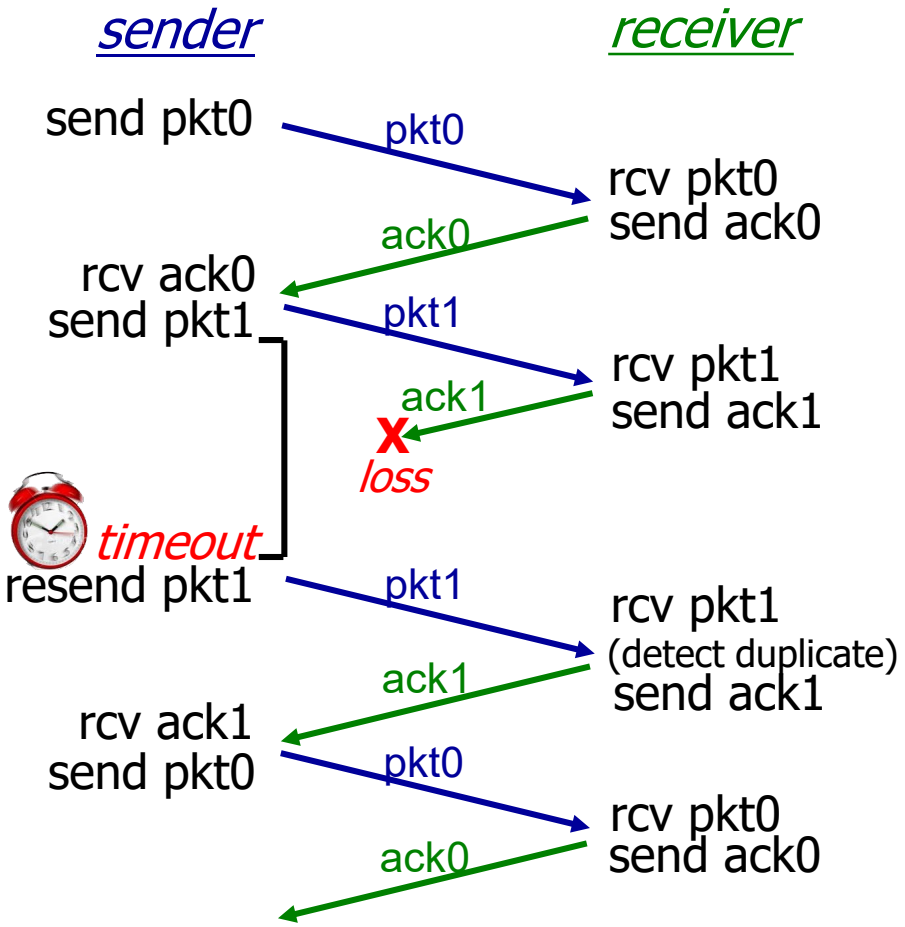
Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait, but how long must the sender wait to be certain that something has been lost? *The sender must clearly wait at least as long as a round-trip delay between the sender and receiver* (which may include buffering at intermediate routers) plus whatever amount of time is needed to process a packet at the receiver.

If an ACK is not received within this time, the packet is retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost. This introduces the possibility of duplicate data packets in the sender-to-receiver channel.

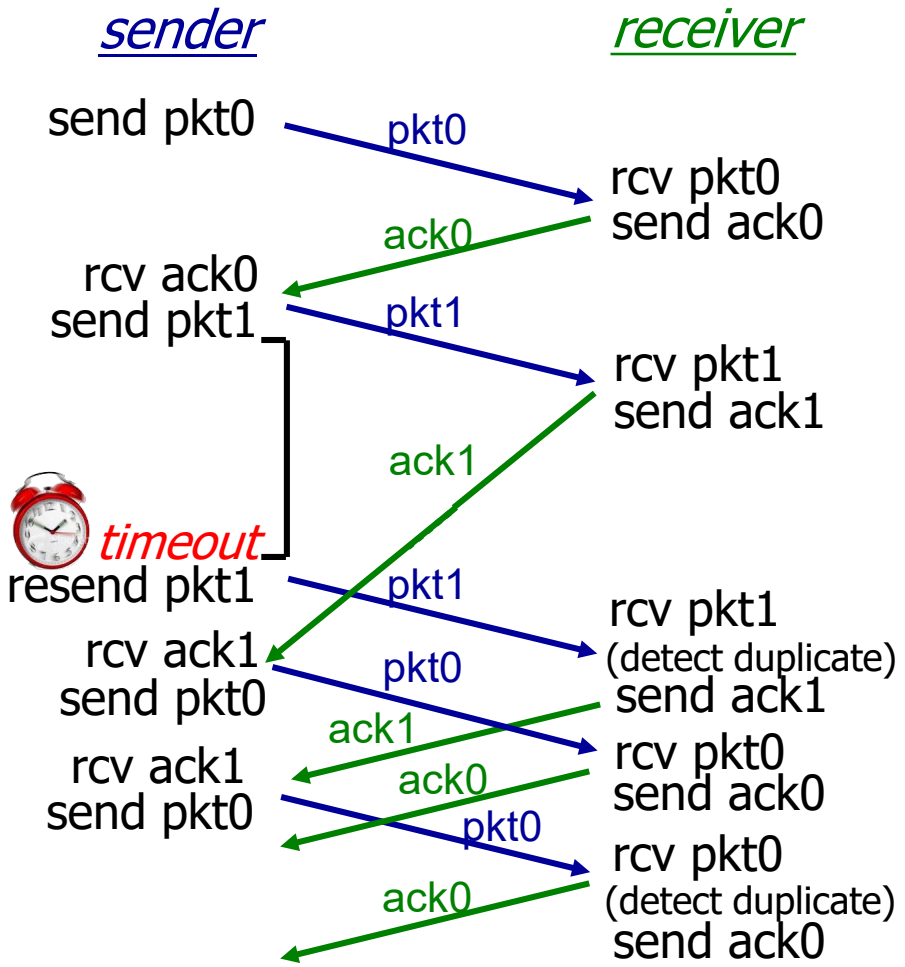
rdt 3.0 in action



rdt 3.0 in action

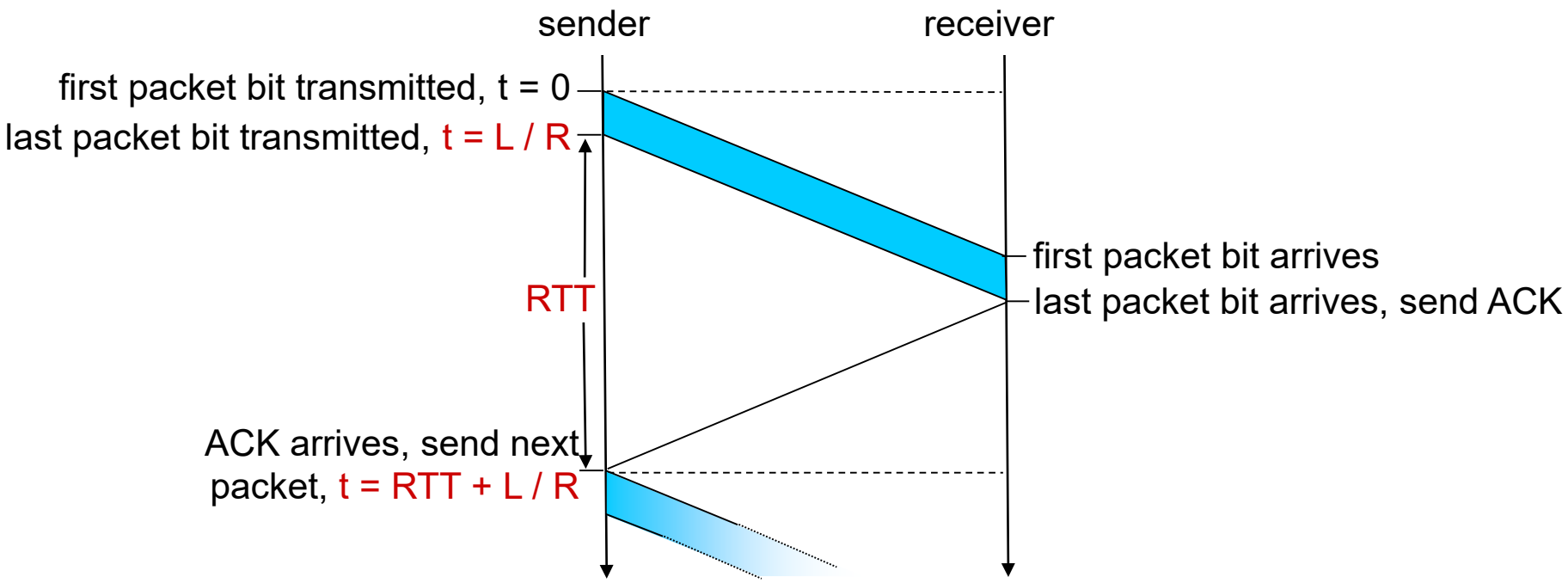


(c) ACK loss



(d) premature timeout/ delayed ACK

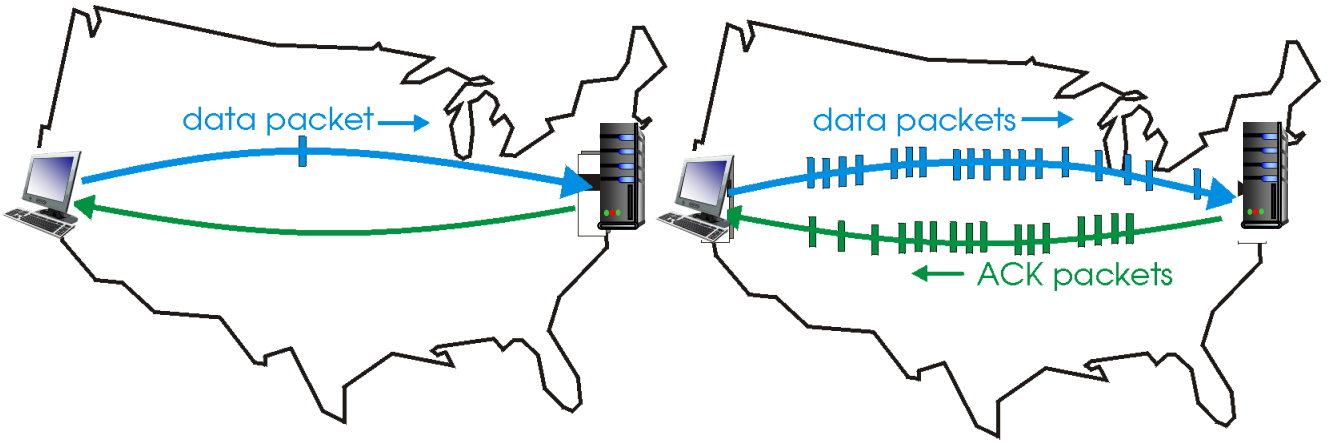
rdt 3.0: stop-and-wait operation



Pipelined Protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets

- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelined Protocol

The sender is allowed to send multiple packets without waiting for acknowledgments. Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as pipelining.

Pipelining has the following consequences for reliable data transfer protocols:

- The range of sequence numbers must be increased, since each in-transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.
- The sender and receiver sides of the protocols may have to buffer more than one packet. Minimally, the sender will have to buffer packets that have been transmitted but not yet acknowledged. Buffering of correctly received packets may also be needed at the receiver.
- The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets.
- Two basic approaches toward pipelined error recovery can be identified: Go-Back-N and selective repeat.

Pipelined Protocol – Seq # and Ack

The sequence number is the byte number of the first byte of data in the TCP packet sent (also called a TCP segment).

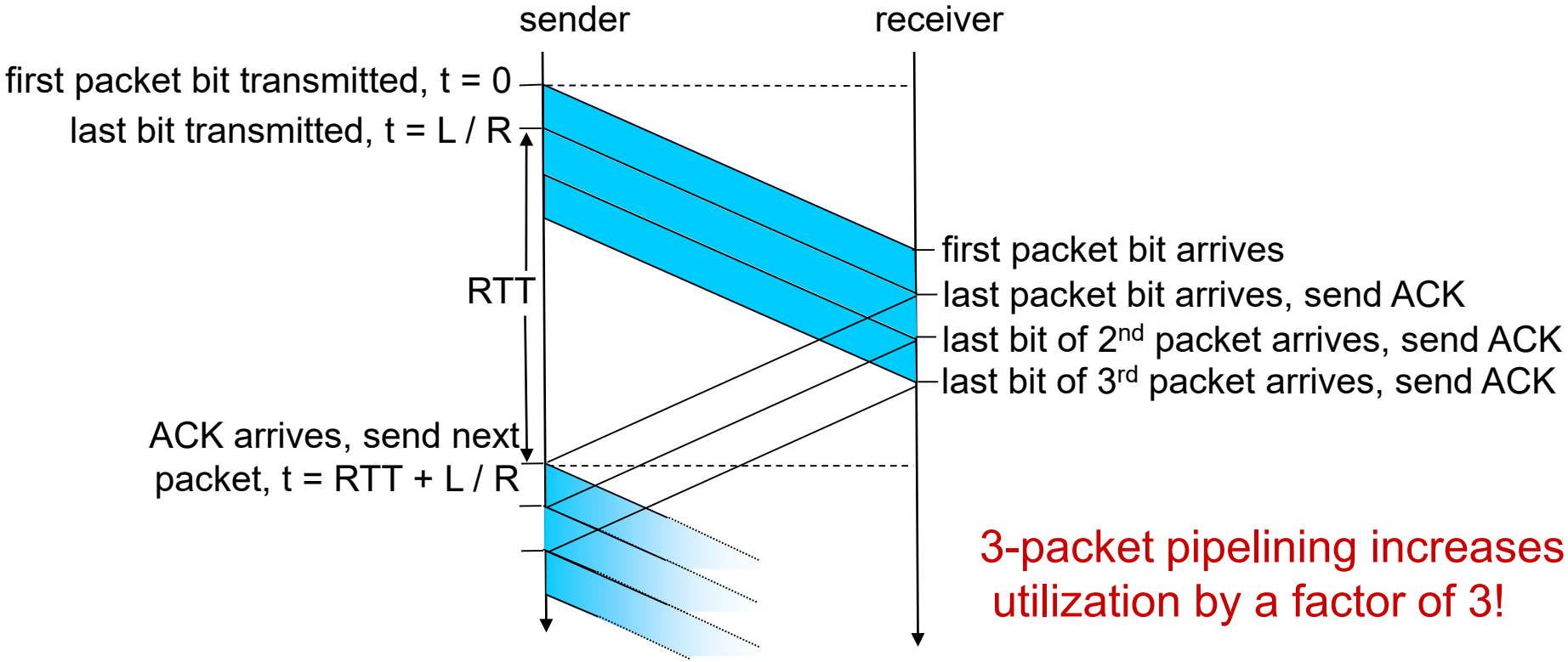
The acknowledgment number is the sequence number of the next byte the receiver expects to receive.

The receiver acknowledging sequence number ***x*** acknowledges receipt of all data bytes less than (but not including) byte number ***x***.

The sequence number is always valid. The acknowledgment number is only valid when the ACK flag is one.

The only time the ACK flag is not set, that is, the only time there is not a valid acknowledgment number in the TCP header, is during the first packet of connection set-up.

Pipelining: increased utilization



3-packet pipelining increases utilization by a factor of 3!

Pipelined Protocols: Overview

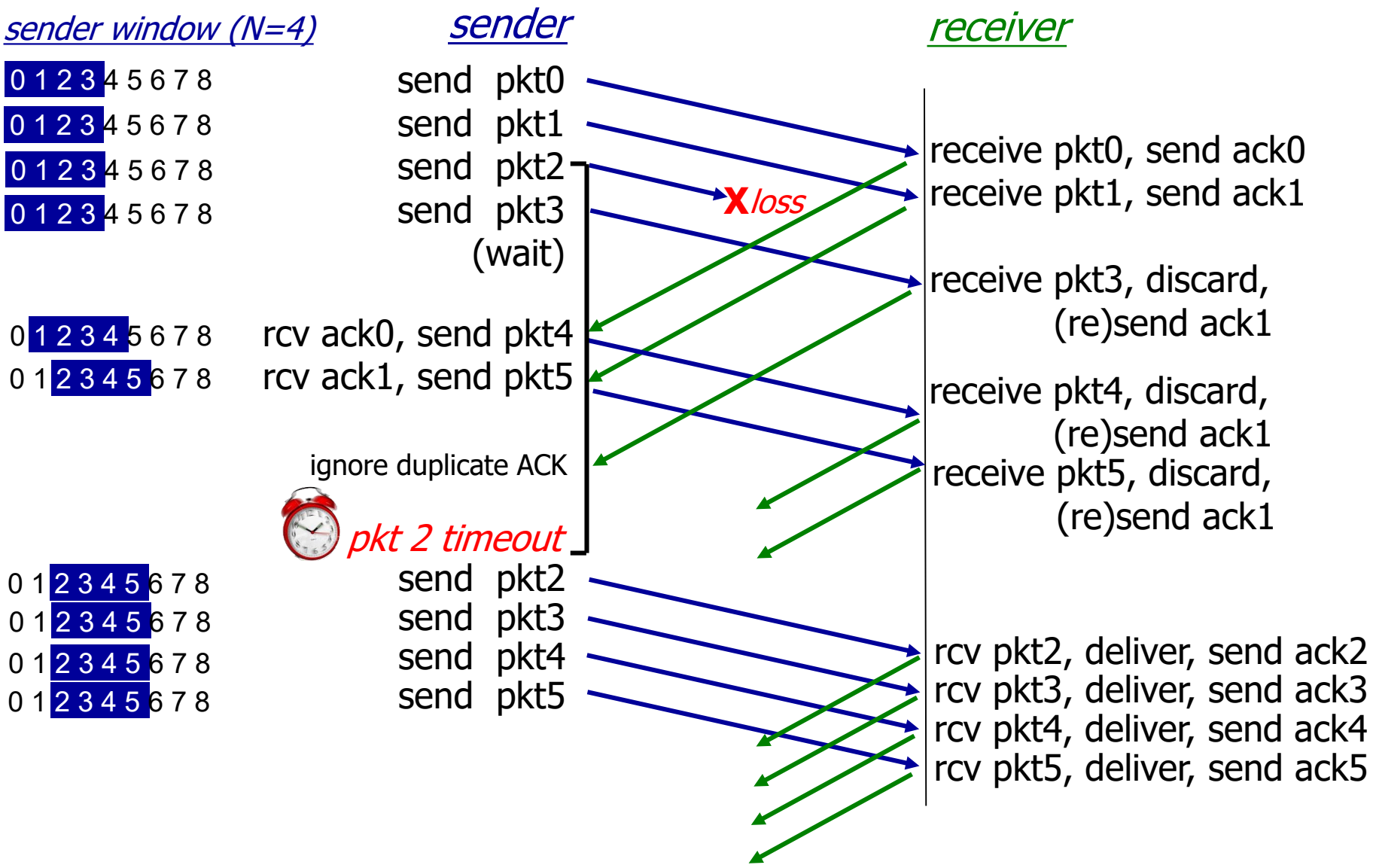
Go-back-N:

- Sender is allowed to transmit multiple packets without waiting for an acknowledgement.
- Receiver only sends cumulative acknowledgement, indicating that all packets with a sequence number up to and including n have been correctly received.
- Sender has timer for oldest unacked packet
- When timer expires, retransmit all unacked packets

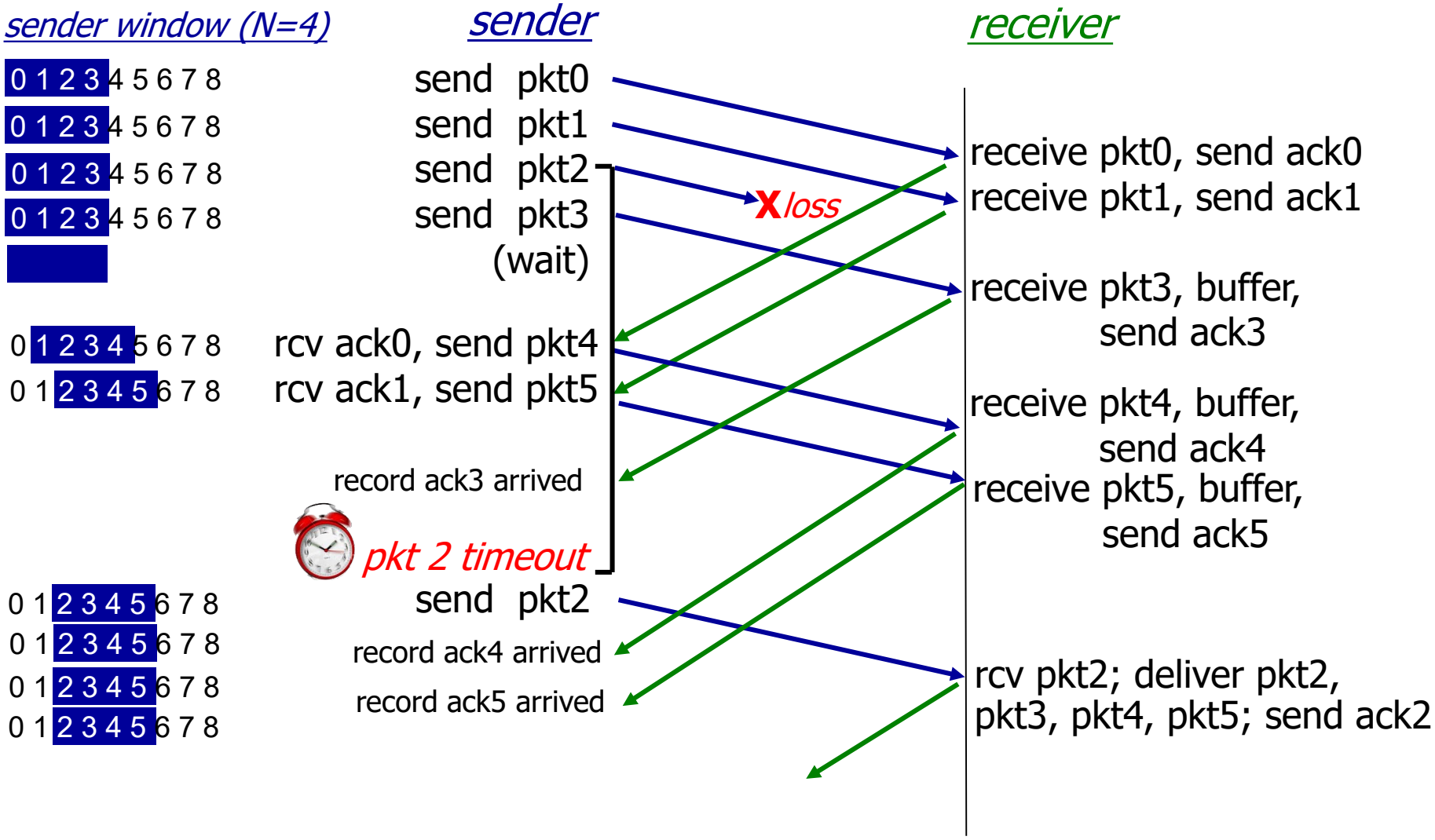
Selective Repeat:

- Sender is allowed to transmit up to N unacked packets in pipeline
- Receiver sends individual ack for each packet.
- Sender maintains timer for each unacked packet
- When timer expires, retransmit only that unacked packet

GBN in action



Selective Repeat in Action



Practice Problem 1

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 100. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 70 and 50 bytes of data, respectively. In the first segment, the sequence number is 101, the source port number is 225, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

i. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

ii. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

iii. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

iv. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.

Practice Problem 1

(i) In the second segment sent from Host A to B: the sequence number is 171, the source port number is 225 and destination port number is 80.

(ii) If the first segment arrives before the second segment, in the acknowledgement of the first arriving segment, the acknowledgement number is 171,
the source port number is 80,
and the destination port number is 225.

(iii) If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, the acknowledgement number is 101, indicating that it is still waiting for bytes 101 and onwards.

Practice Problem 1