Racap of Flow Control in the Data Link layer

- Recall the use of the Sliding Windows Flow Control technique:
  - This technique allows *multiple* **Frames** to be in transit in sucession,
  - This provides for more *efficient* ***Link Utilization.***
- Characteristics of the technique are:
  - Both stations use an extended buffer size to hold *multiple* frames.
  - The Sending/Receiving stations maintain a list of frames already sent/received.
- The transmission link is effectively treated as a *pipeline* that can be *filled* with many frames in transit.

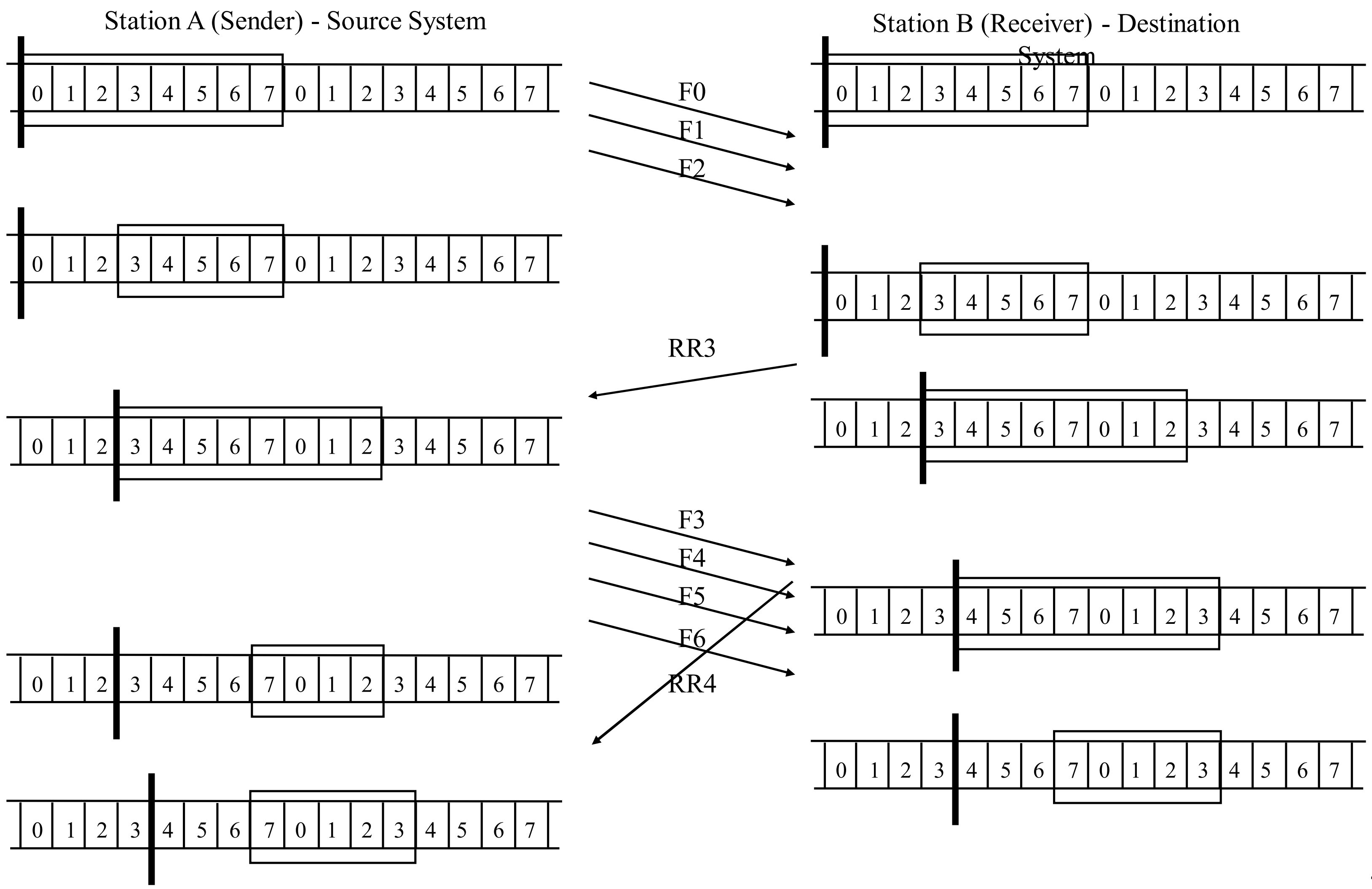
Example Sliding Windows Flow Control

- Consider two Stations, A and B exchanging data. Assume Station A is sending data to Station B:
  - Station A is the **Sender** and Station B is the **Receiver**.
- Before any data is exchanged, each station allocates buffer space for *W* frames (for example consider **W=8**)
  - This means that Station B can *accept* up to **W** (8) frames and, Station A can *send* up to **W** (8) frames, <u>without</u> individual frame acknowledgements (ACKs) being sent or received.

Example Sliding Windows Flow Control

- All frames contain a *sequence number:*
  - All frames from Station A to Station B contain a sequence number for the **current frame**,
  - All *acknowledgements* from Station B contain the sequence number of the **next frame** expected.
- Frames leaving Station A are stored in an **outgoing** buffer on Station A until an ACK is <u>**received**</u>.
- Frames arriving at Station B are stored in an **incoming** buffer on Station B until an ACK is <u>**sent**</u>.

Example Sliding Windows

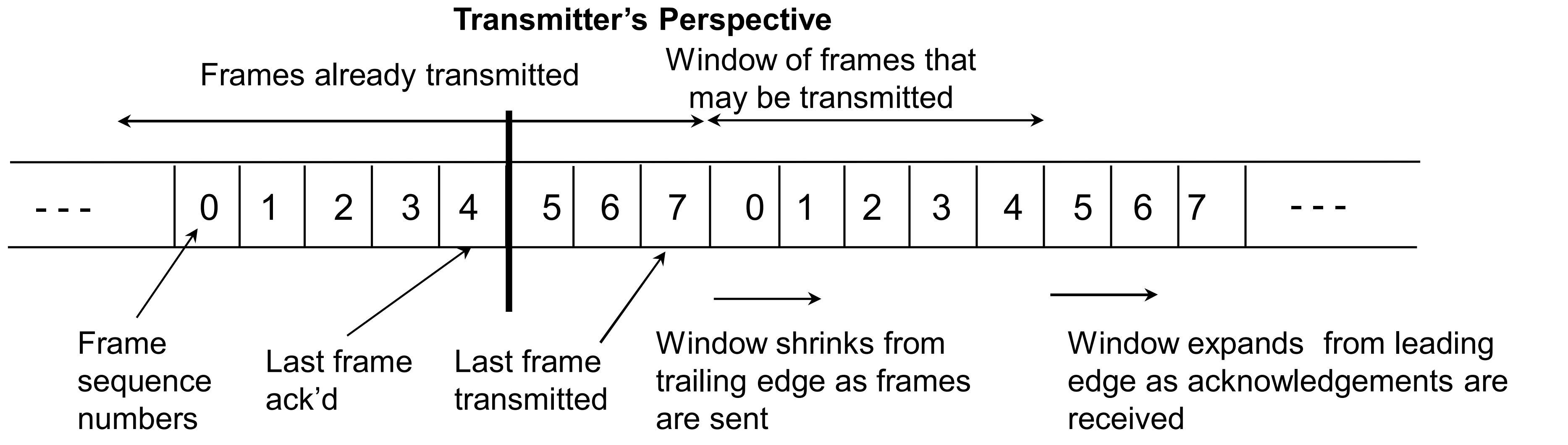


Station A (Sender) - Source System
Station B (Receiver) - Destination System
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
F0
F1
F2
RR3
F3
F4
F5
F6
RR4

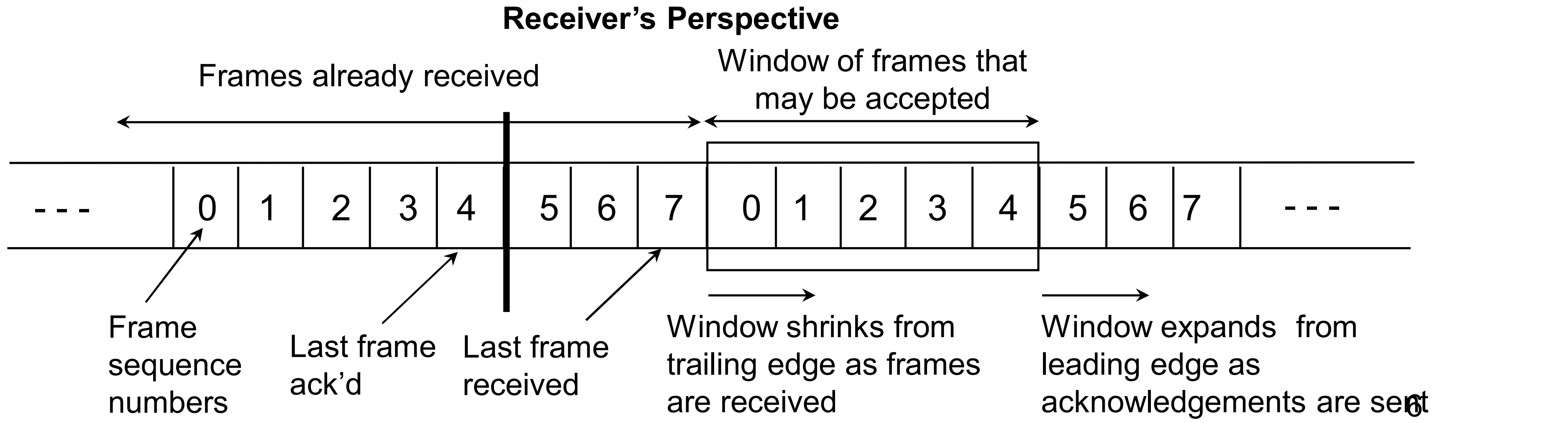
Sliding Windows Flow Control

- *Multiple* frames can be *acknowledged* using a single control message (*implicit ACK*):
  - e.g. Receipt of ACK for frame **2** (RR3) followed later by ACK for frame **5** (RR6) *implies* acknowledgement of frames **3** and **4.**
- Station A maintains a list of frame numbers it is allowed to *send* and, Station B maintains a list of frame numbers it is prepared to *receive*:
  - Each list cannot extend beyond the window size.
  - These lists can be considered as *windows.*

Sliding Windows Flow Control



Transmitter's Perspective
Frames already transmitted
Window of frames that may be transmitted
- - -
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
- - -
Frame sequence numbers
Last frame ack'd
Last frame transmitted
Window shrinks from trailing edge as frames are sent
Window expands from leading edge as acknowledgements are received



Receiver's Perspective
Frames already received
Window of frames that may be accepted
- - -
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
- - -
Frame sequence numbers
Last frame ack'd
Last frame received
Window shrinks from trailing edge as frames are received
Window expands from leading edge as acknowledgements are sent

Error Control

- Recall the purpose of Error Control:
  - Sender and Receiver stations co-ordinate activities to recover from **Lost** or **Damaged** Frames etc.
- Error Control involves enhancing Flow Control techniques with additional functionality such as:
    - **Transmission Timers**. Sender stations set a timer for each frame transmitted and takes action when a timer expires.
    - **Negative ACKs.** A Receiver station can reject an *out-of-sequence/damaged* frame with a REJ(5) or SREJ(4) message.
  - Example Error Control techniques include: **Go-Back-N** and **Selective Reject**:
    - Both techniques are based on the Sliding Windows Flow Control technique.

TCP Error/Flow Control

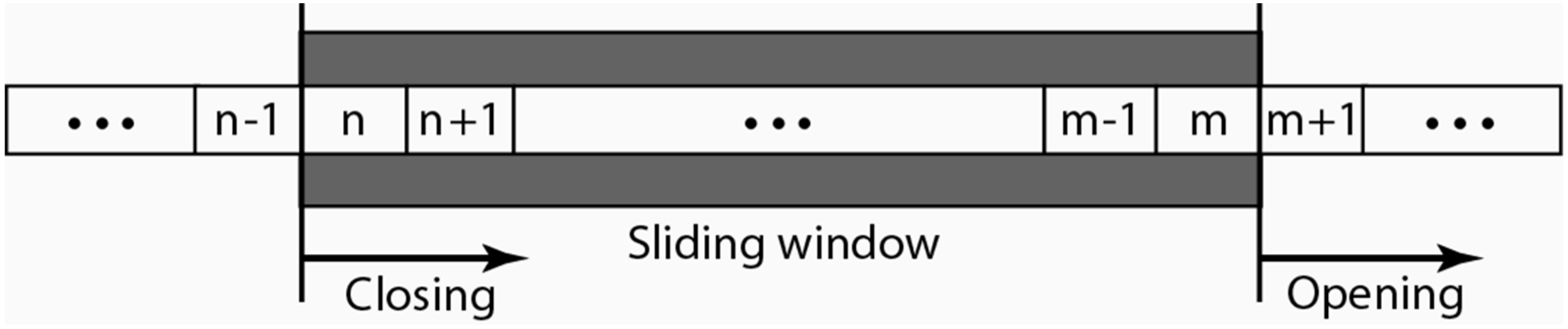
- *TCP* also uses an Error Control technique based on **Sliding Window Flow Control** technique.
- It is different to that used in the Data Link layer as follows:
  - Data is sent in **Segments** not **Frames.**
  - Sequence numbers relate to **Bytes** not Segments. Each **byte** in a segment is numbered:
    - Each Segment identifies the first byte in the data field.
    - ACKs contain the number of the next byte expected.
  - There are no **Negative** ACKs.

TCP Error/Flow Control

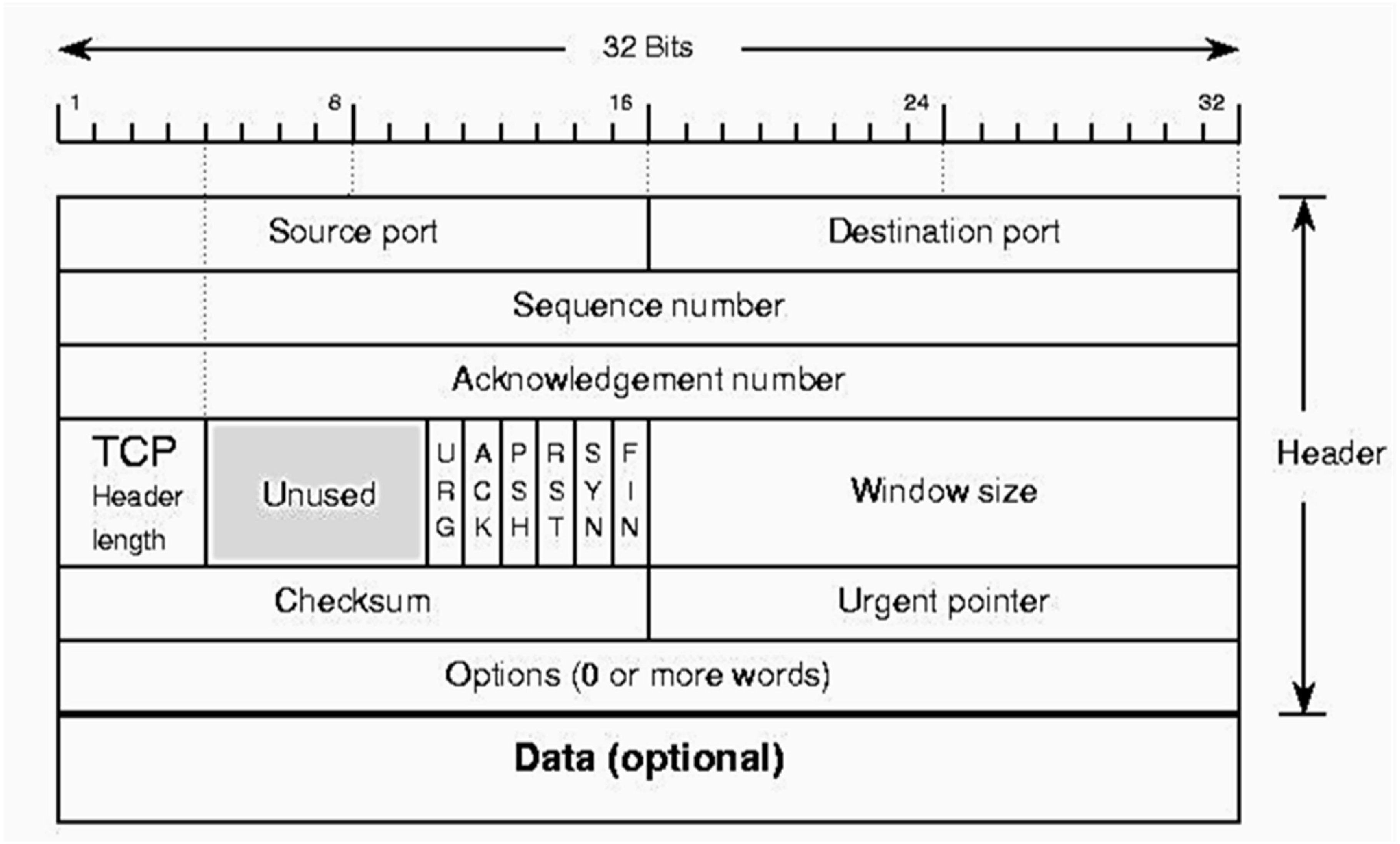
- Senders and Receivers maintain lists of bytes sent/received.
- Buffers used to hold incoming segments are measured in bytes (not segments).

TCP Sliding Windows

- Here it can seen that the available buffer space *decreases* as data is stored in the buffer and *increases* as data leaves the buffer:



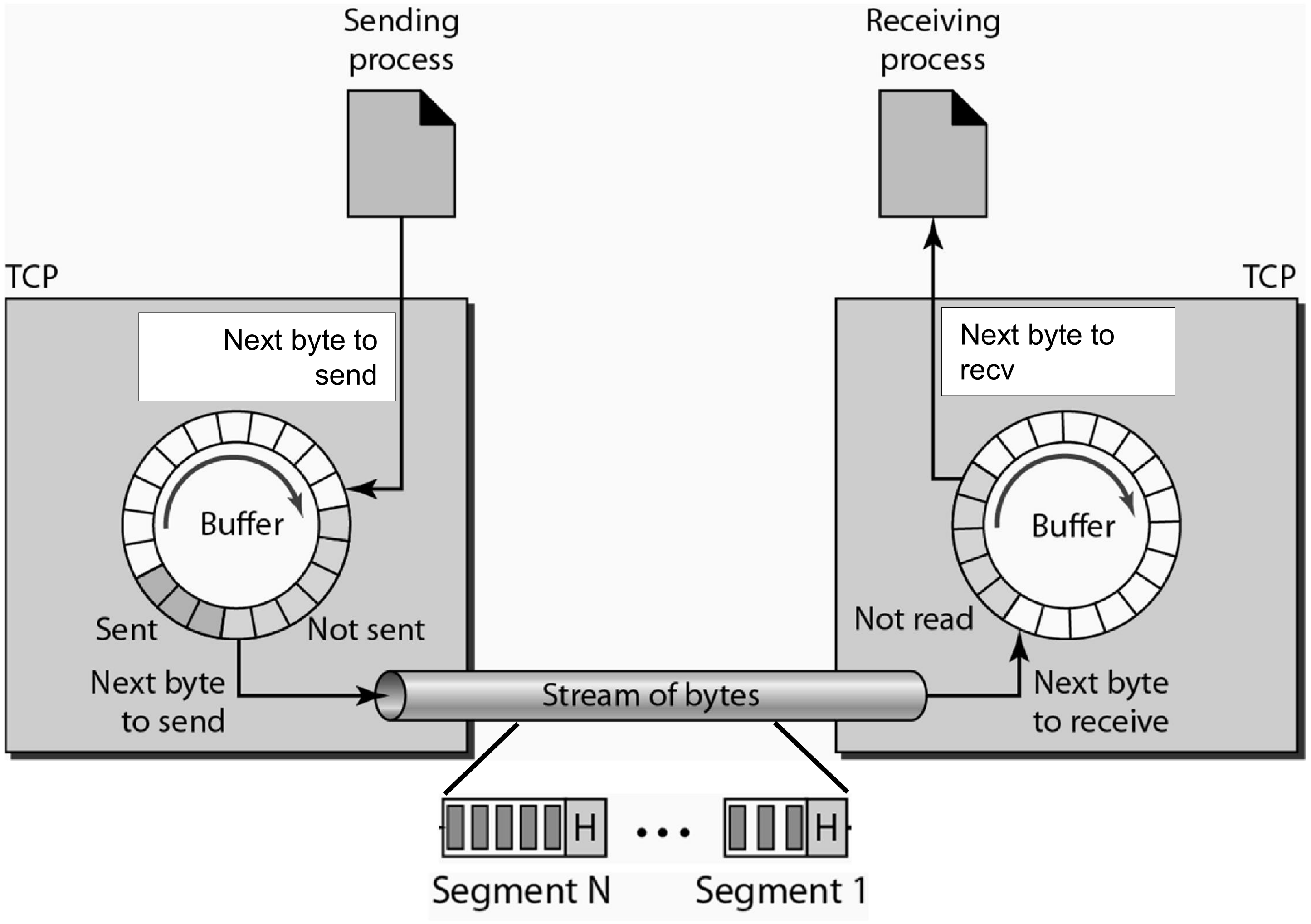
TCP Segment Format



TCP Flow Control – *Buffers* and *Windows*

- Recall that TCP creates two buffers per socket:
- These can be viewed with the **netstat** utility:
  - One for incoming data (known as **RECV-Q** in netstat)
  - One for outgoing data (known as **SEND-Q** in netstat)
- Incoming buffers can easily overflow.
- To prevent this, the receiving TCP entity uses a ***Window Mechanism.***

TCP Internal Data Buffers



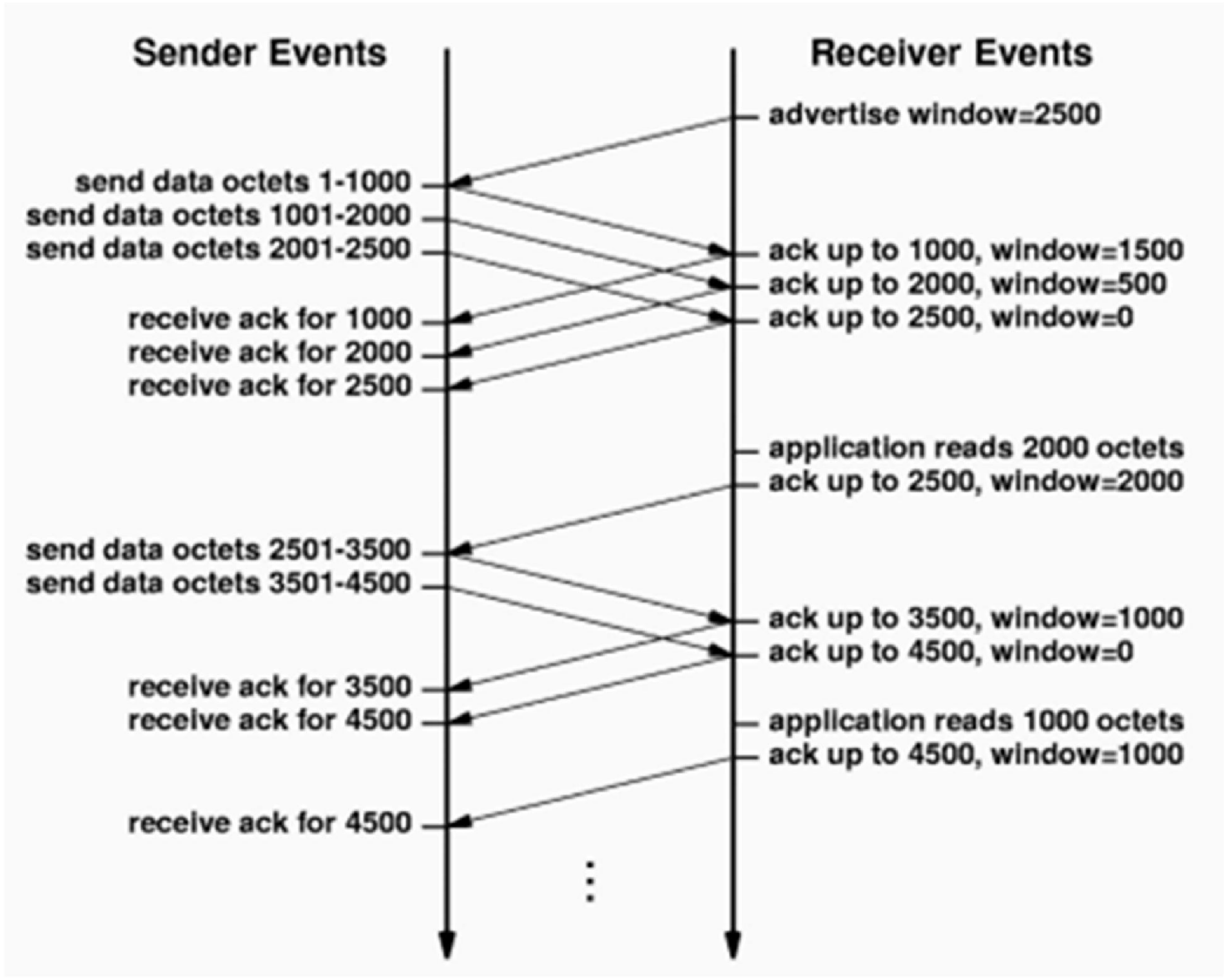
TCP Flow Control – *Buffers* and *Windows*

- Each end of the connection allocates a ***window*** (**RECVQ** buffer) to hold incoming data:
  - The size of the initial window, in bytes, is set using the *Window Size* field during *Phase 1* (*Connection Initialization*) when both sides exchange SYN messages.
  - This is known as a **Window Advertisement.**

TCP Flow Control – *Buffers* and *Windows*

- Thereafter, throughout *Phase 2* (*Data Exchange*), all ***ACKs*** messages include a ***Window Advertisement*:**
  - Again using the *Window Size* field.
  - The *window advertisement* can be positive or zero depending on the available space in **RECV-Q**.

Operation of *Window Advertisements*



Sender Events
Receiver Events
advertise window=2500
send data octets 1-1000
send data octets 1001-2000
send data octets 2001-2500
ack up to 1000, window=1500
ack up to 2000, window=500
ack up to 2500, window=0
receive ack for 1000
receive ack for 2000
receive ack for 2500
application reads 2000 octets
ack up to 2500, window=2000
send data octets 2501-3500
send data octets 3501-4500
ack up to 3500, window=1000
ack up to 4500, window=0
receive ack for 3500
receive ack for 4500
application reads 1000 octets
ack up to 4500, window=1000
receive ack for 4500

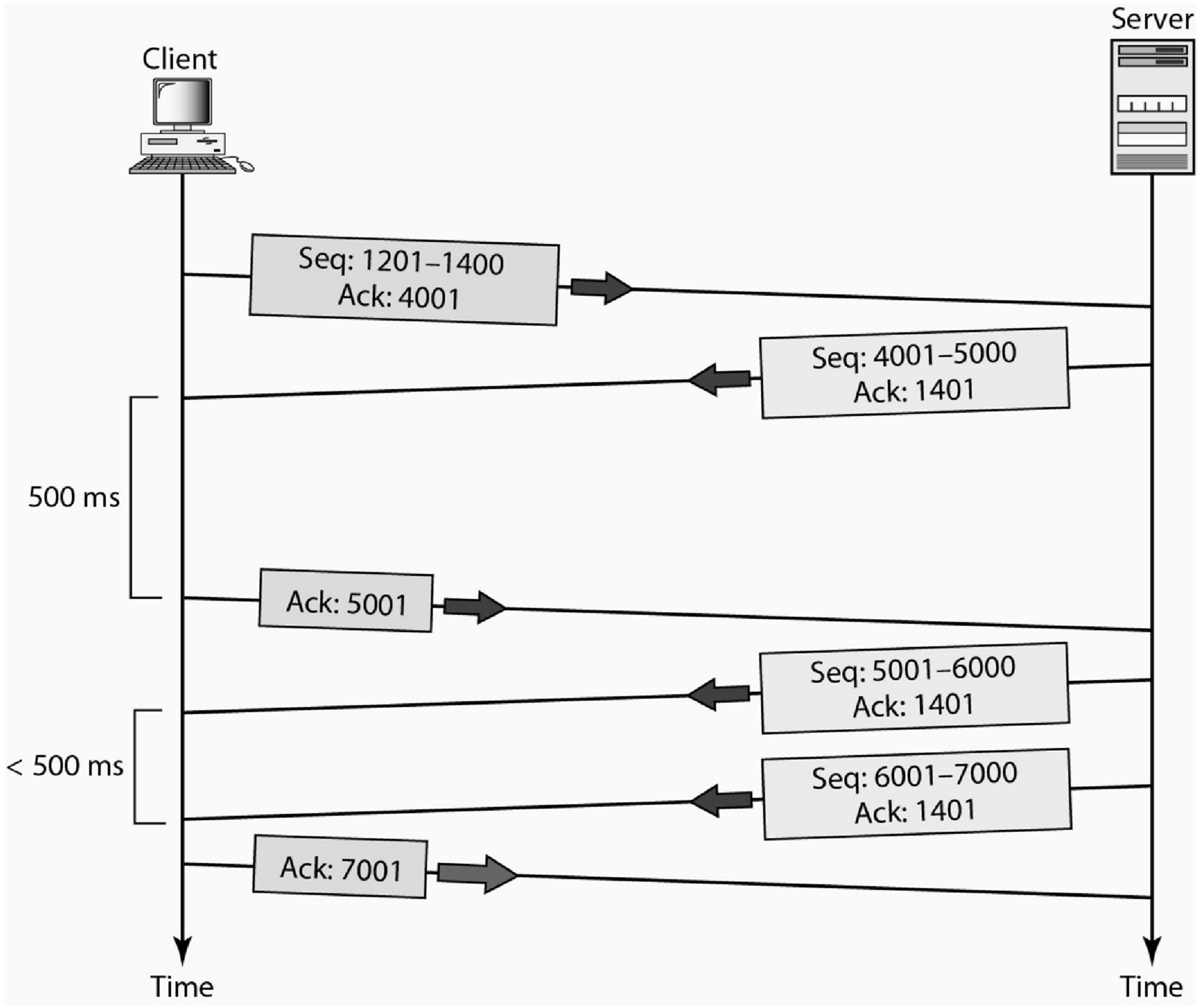
Achieving Reliability

- In addition to Flow Control, TCP must address the following reliability problems:
  - <u>Unreliable</u> delivery by the underlying communication system:
    - Segments can be ***lost***, ***duplicated***, ***delayed***, or delivered ***out-of-order*** by the underlying communication system (IP Layer).
  - Computer reboot:
    - During an active connection, either side may re-boot unexpectedly.
    - Segments arriving after re-boot need to be dealt with.

TCP Error Control

- This requires the use of some form of ***error control.***
- Interestingly to implement *flow* and *error control,* TCP is only equipped with two elements:
  - **Positive ACKs** and **Timers,**
  - Significantly TCP does <u>not</u> have a **Negative ACK.**
- The following slide shows the operation of TCP during <u>normal</u> *Data Exchange* without error.

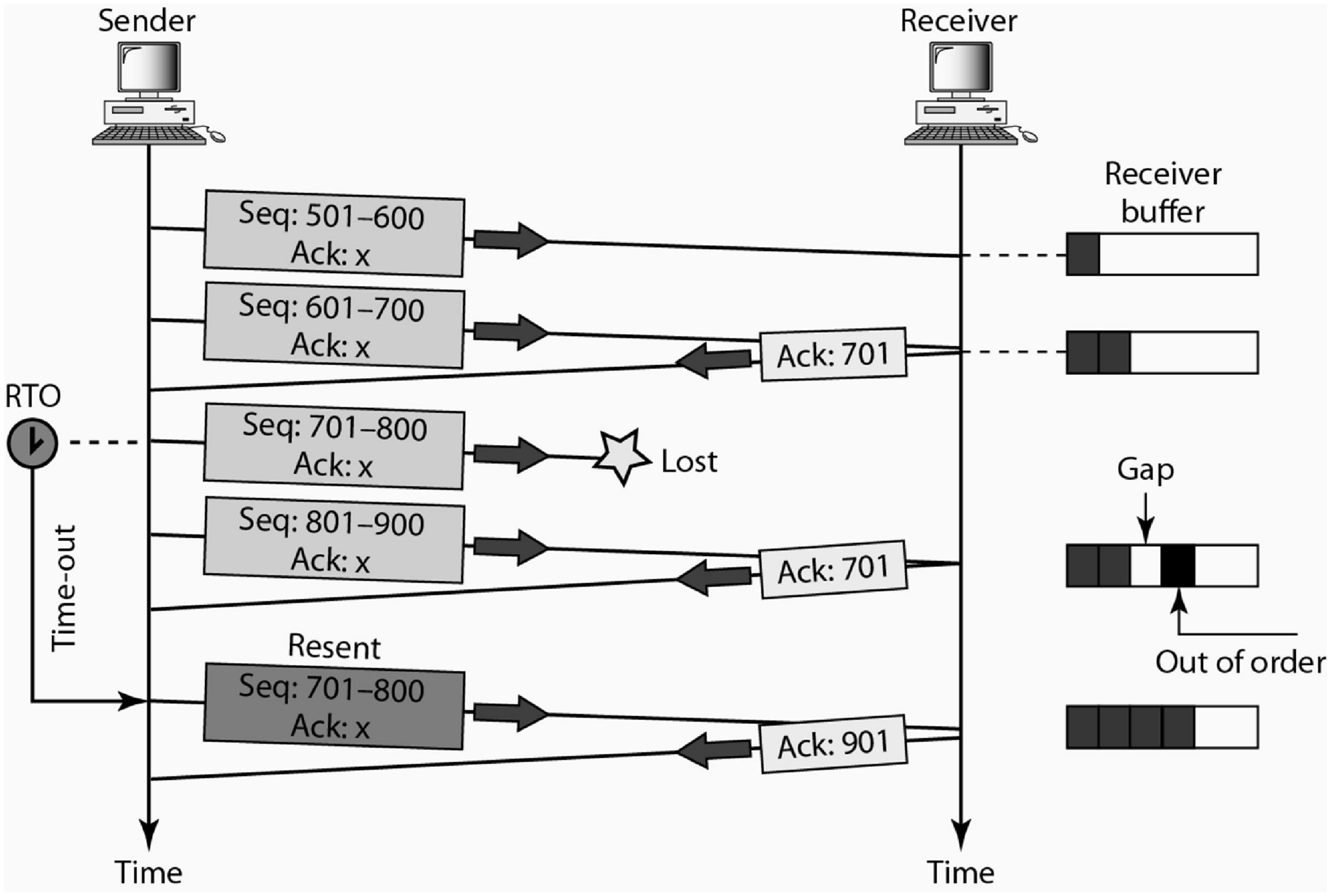
TCP Data Flow – Normal Operation



TCP Error Control - Segment Loss

- To deal with *lost* or *out-of-sequence* segments,TCP implements a ***Retransmission*** scheme:
  - This involves the retransmission of ACKs and/or lost segments*.*
- For the Sender, **timers** play a key role in error control:
  - Upon expiry of a timer (relating to an unacknowledged segment) the Sender simply re-transmits the segment,
  - A key question is “How long should TCP wait before retransmitting?”
- For the Receiver, sequence numbers play a key role in detecting **lost** or **out-of-sequence** segments:
  - A previous ACK is <u>re-transmitted</u> in response until the situation is rectified.

TCP Data Flow – Lost Segment Scenario



Factors affecting TCP's Retransmission scheme

- How long should a Sending TCP entity wait before re-transmitting a segment?
- The answer depends upon two factors:
  - The underlying network architecture, and,
  - Traffic levels across the network.
- TCP takes a measure of the delay between sending a segment and receiving an ACK.
  - This is known as Round-Trip Time (**RTT**).
- TCP uses the value for RTT to determine an appropriate value for the re-transmission timer (**RTO** - Retransmission Timeout).

Calculation of Retransmission Timeout (RTO)

- For each active connection, the RTT is continuously monitored:
  - It represents the *network latency*.
- TCP **adapts** its RTO timer to match the varying RTT values across the network:
  - It uses a <u>weighted</u> average of RTT <u>and</u> a variance factor,
  - It continuously re-calculates a value for RTO.

TCP's *Adaptive Retransmission* Scheme

- This is known as an *adaptive retransmission* scheme and is the key to TCP's success.
- This adaptability helps TCP to react <u>quickly</u> to changes in *traffic levels* and to <u>maximize</u> throughput on each connection.