Reliable Data Transfer: Sliding Window

In this lesson, we'll study sliding windows

WE'LL COVER THE FOLLOWING

Pipelining

Applications may generate data at a rate much higher than the network can transport it. Processor speed is generally much higher than the speed of writing out and reading data to/from the network (I/O).

Furthermore, reliable message communication is a multi-step process:

1. The processor emits the message to be sent, the network carries the message to the destination.
2. The receiver processor receives and emits an acknowledgment message.
3. The network carries the acknowledgment to the sender. So, instead of waiting for an acknowledgment of every packet before transmitting the next one, it's **more efficient** to **pipeline** the multi-step process. In other words, instead of waiting for the acknowledgment of a message before transmitting the next one, the sender keeps transmitting messages without waiting for an acknowledgment. This makes more efficient use of the processor's time.

While pipelining allows the **sender** to transmit segments at a **higher rate**, it may cause the **receiver** to become **overloaded** because the receiver may be running on a slower machine than the sender, or the receiving machine may be busy executing other processes.

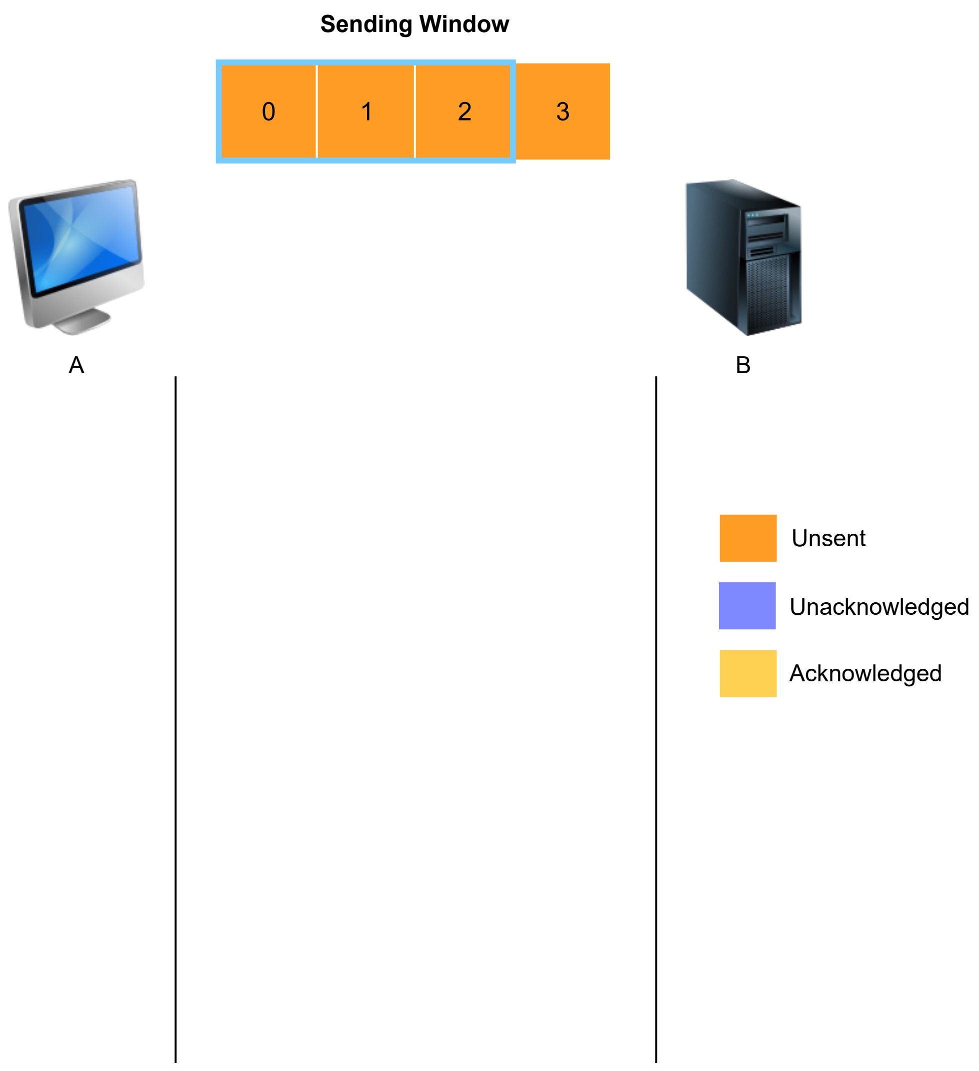
> **Note:** “**stop and wait**” is the term used in literature for when the sending entity waits for the acknowledgment of every transmitted message before sending the next one. That is also a means to recover from lost messages. It retransmits.

Sliding Window

The sliding window is the set of consecutive sequence numbers that the sender can use when transmitting segments without being forced to wait for an acknowledgment. At the beginning of a session, the sender and receiver agree on a sliding window size.

The figure below illustrates the operation of the sliding window. The sliding window shown contains three segments. The sender can thus transmit three segments before being forced to wait for an acknowledgment. The sliding window moves to the higher sequence numbers upon reception of acknowledgments. When the first acknowledgment (of segment 0) is received, it allows the sender to move its sliding window to the right, and sequence number 3 becomes available.

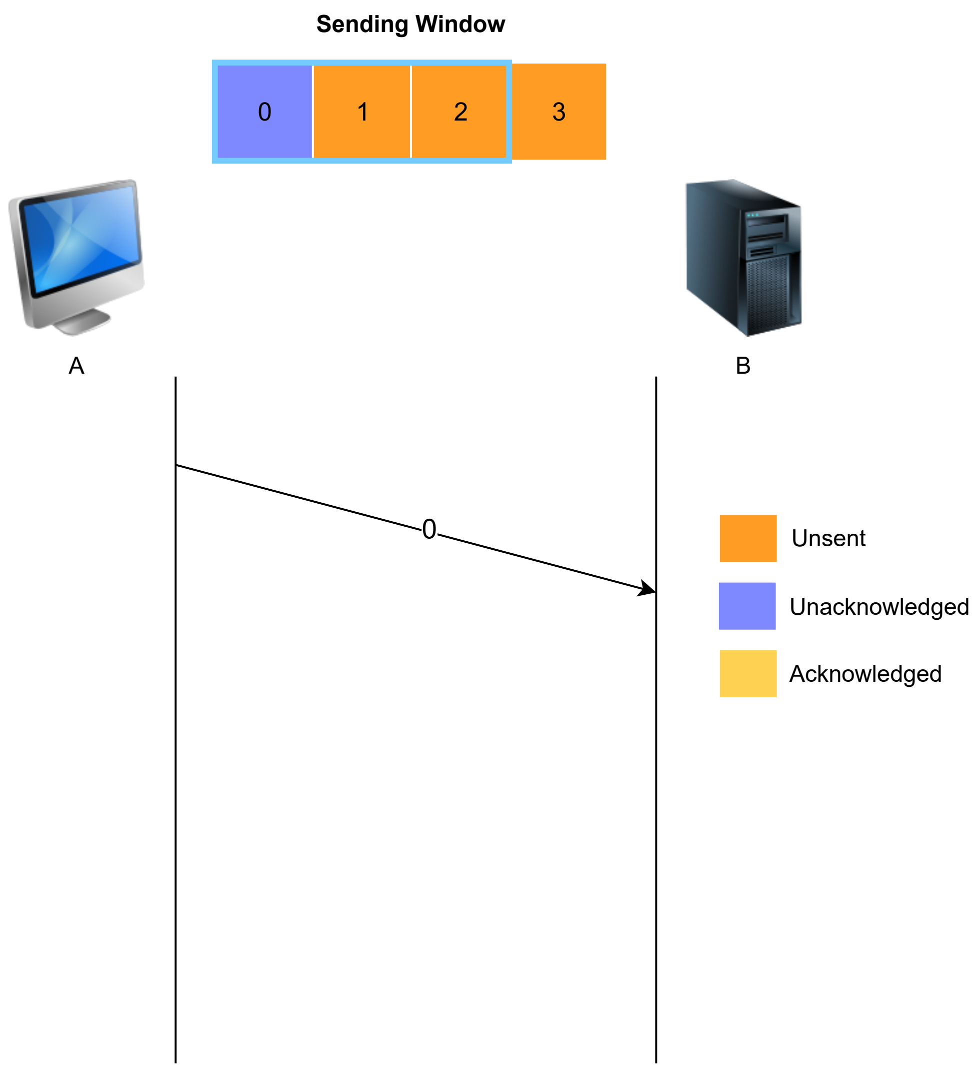
In practice, as the segment header encodes the sequence number in a n bits string, only the sequence numbers between 0 and 2^n – 1 can be used. There’s a problem here. What happens when an application has transmitted $2^n - 1$ messages and still has more data to send? Well, the same sequence number can be used for different segments and that the sliding window can wrap. This is illustrated in the figure below assuming that 2 bits are used to encode the sequence number in the segment header. Note that upon reception of the acknowledgment for the first segment, the sender slides its window and can use sequence number 0 again.



A sending sliding window of size 3 is initiated

The sliding window

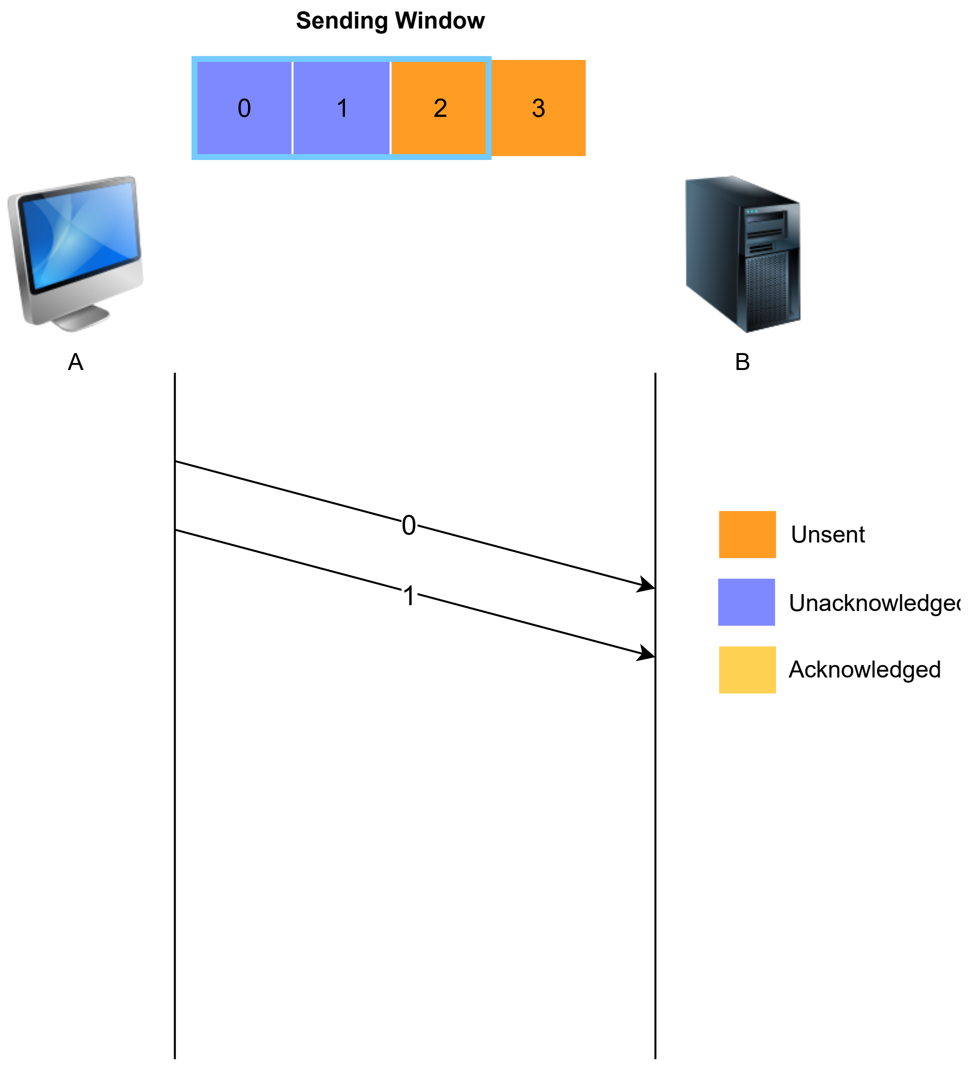
1 of 11



Segment with sequence number 0 is sent

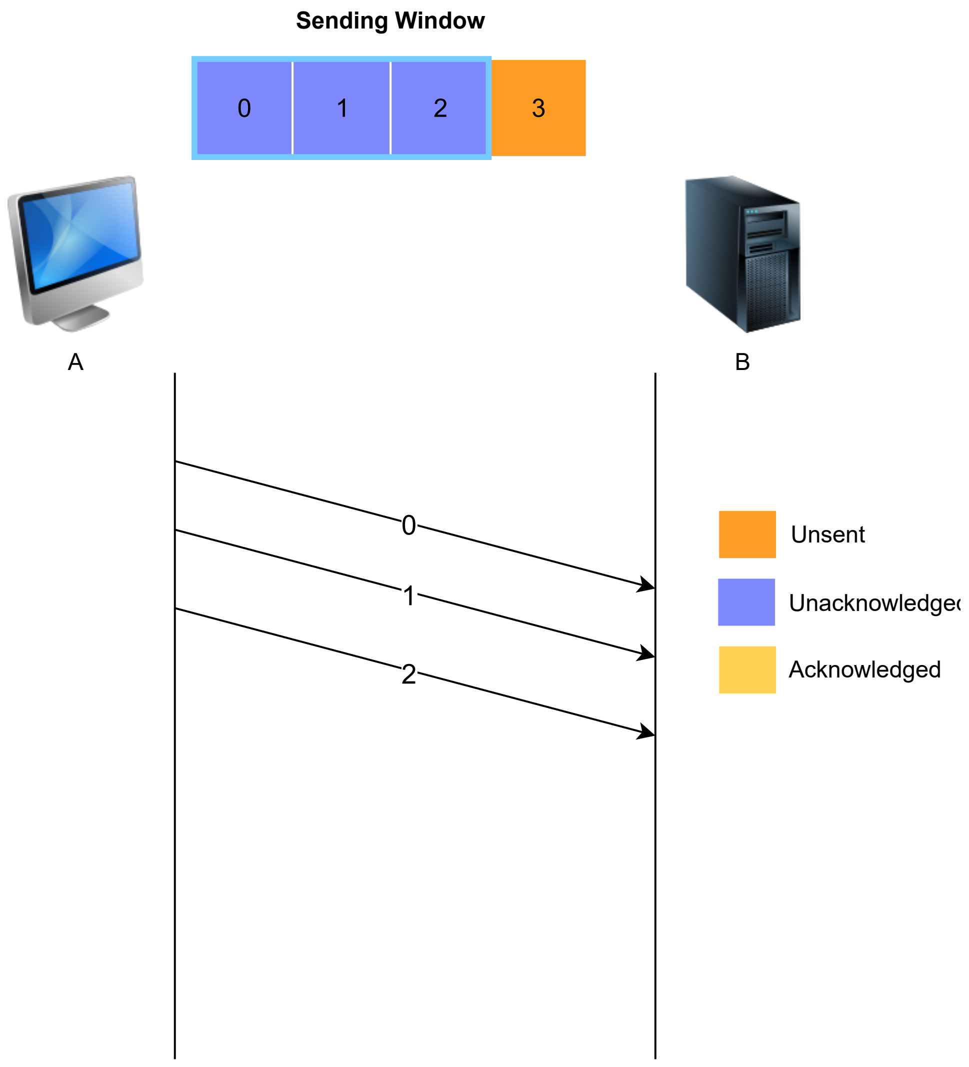
The sliding window

2 of 11



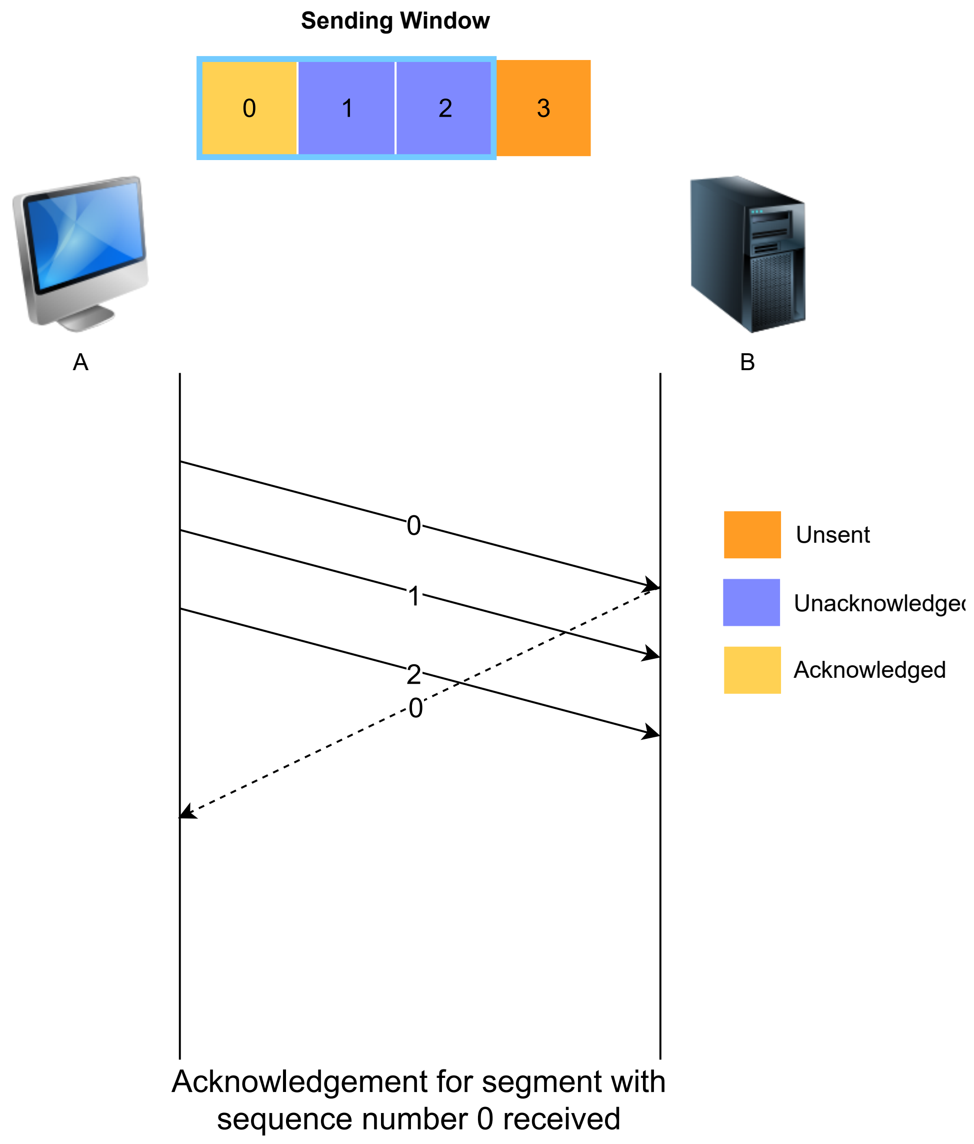
Segment with sequence number 1 is sent.
No acknowledgement is received yet

The sliding window

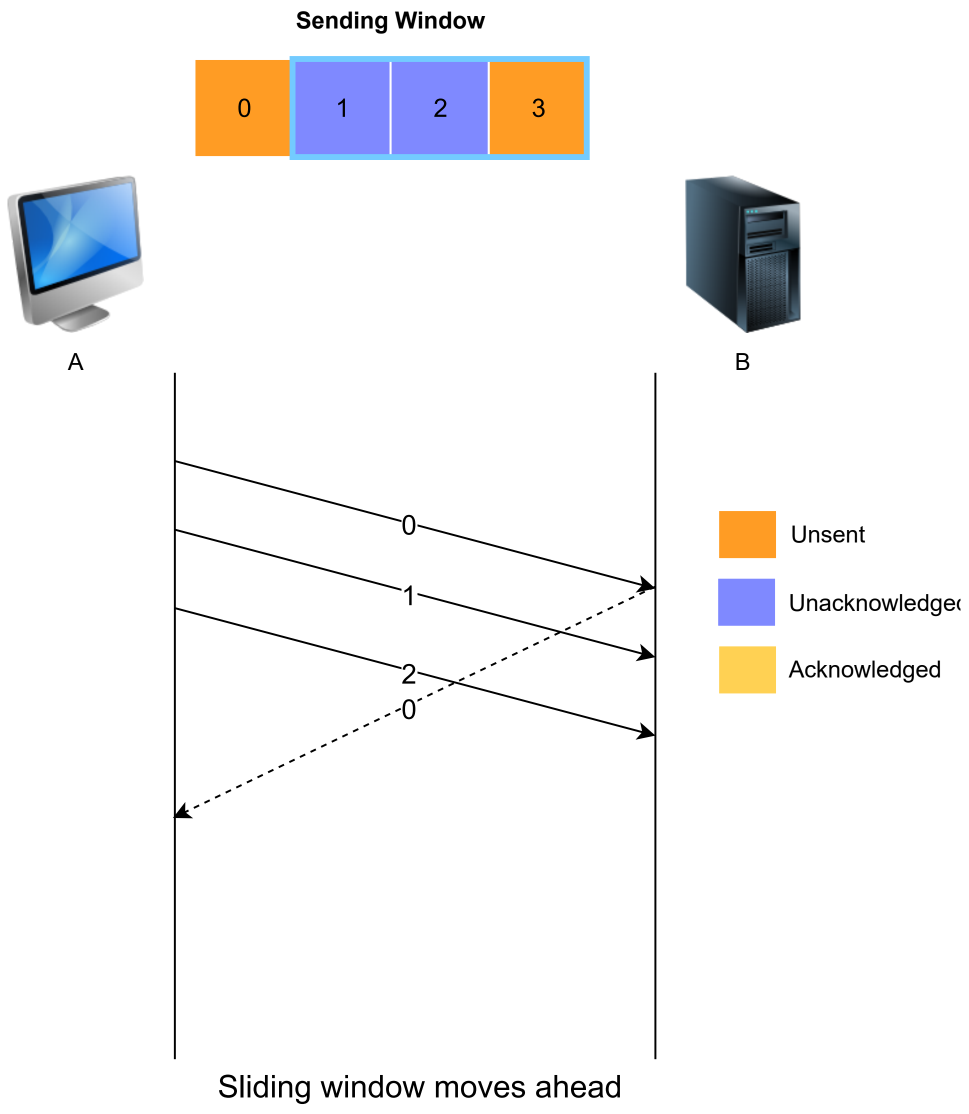


Segment with sequence number 2 is sent.
No acknowledgement is received yet.

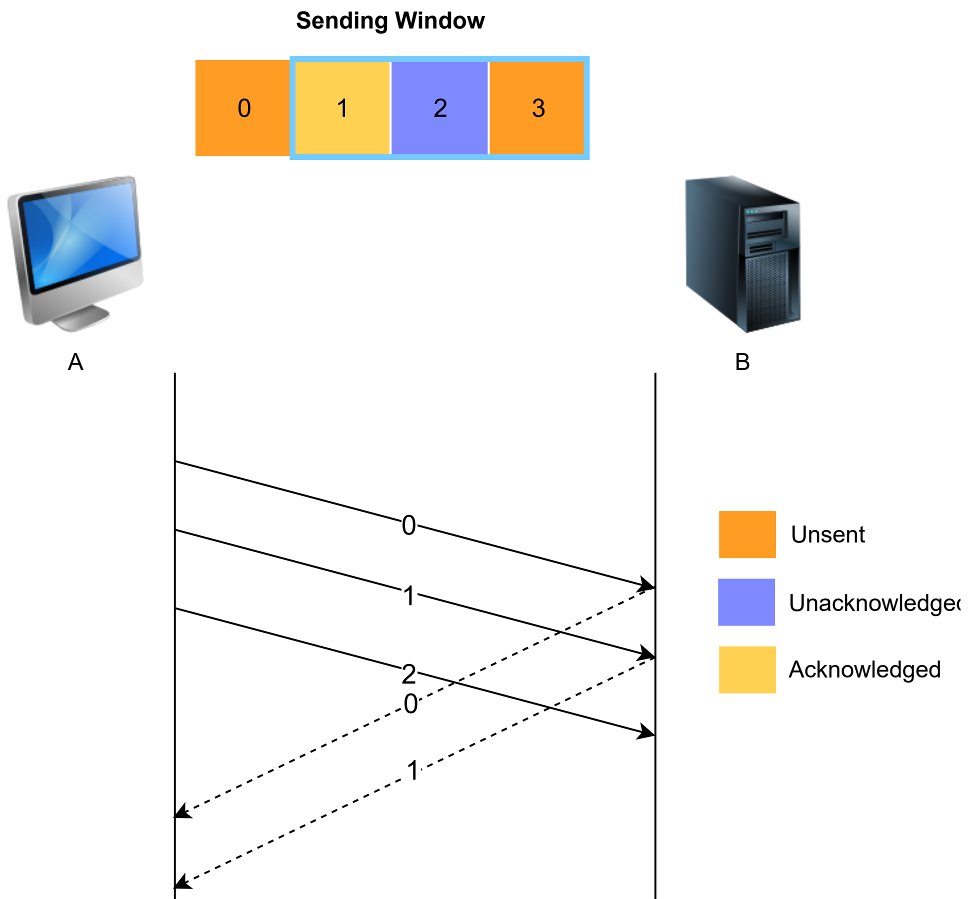
The sliding window



The sliding window

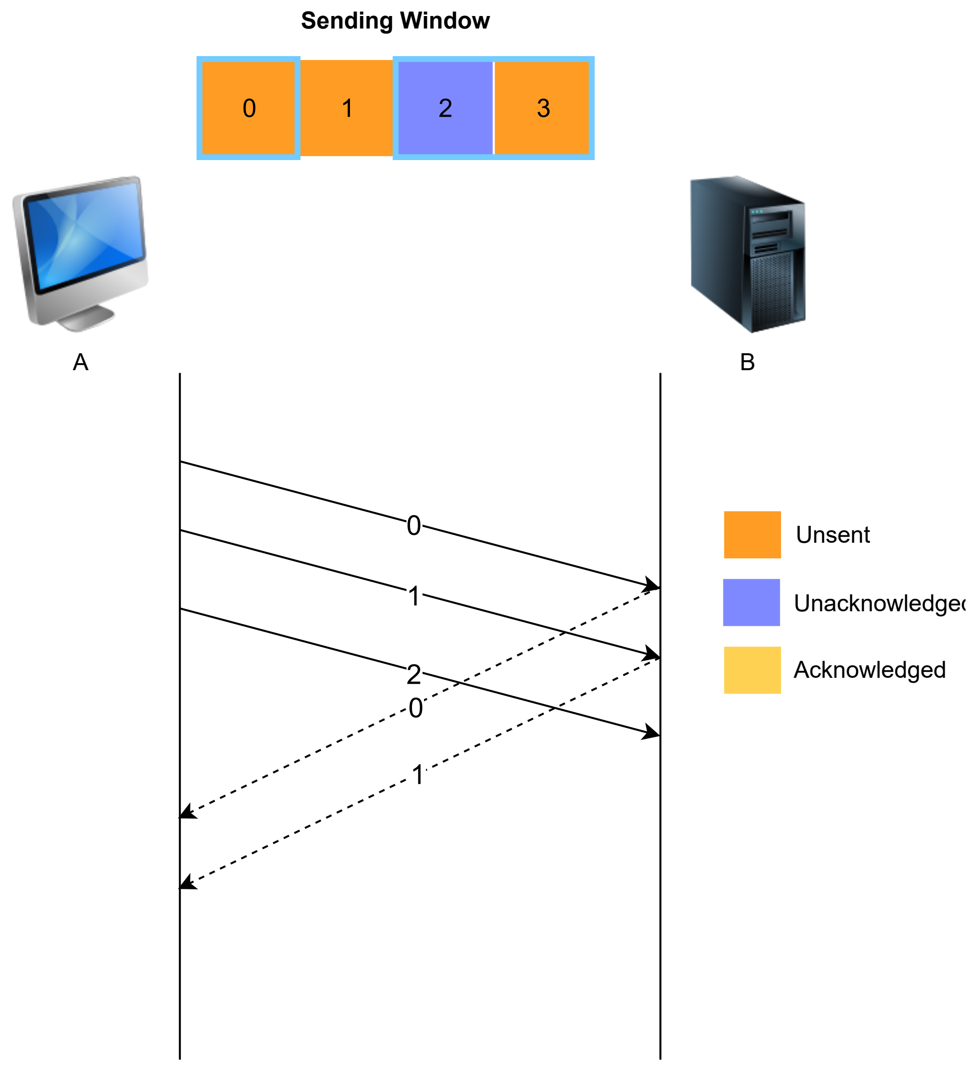


The sliding window



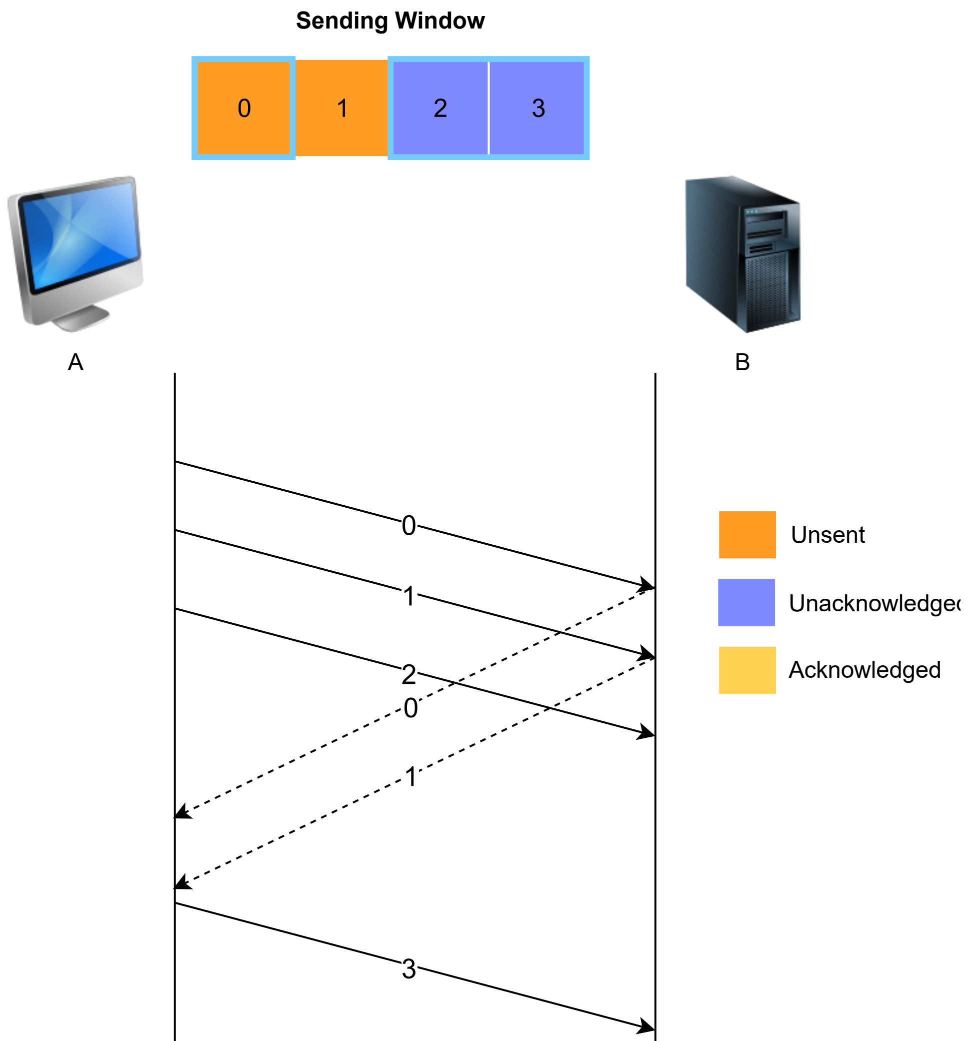
Acknowledgement of segment with sequence number 1 received

The sliding window



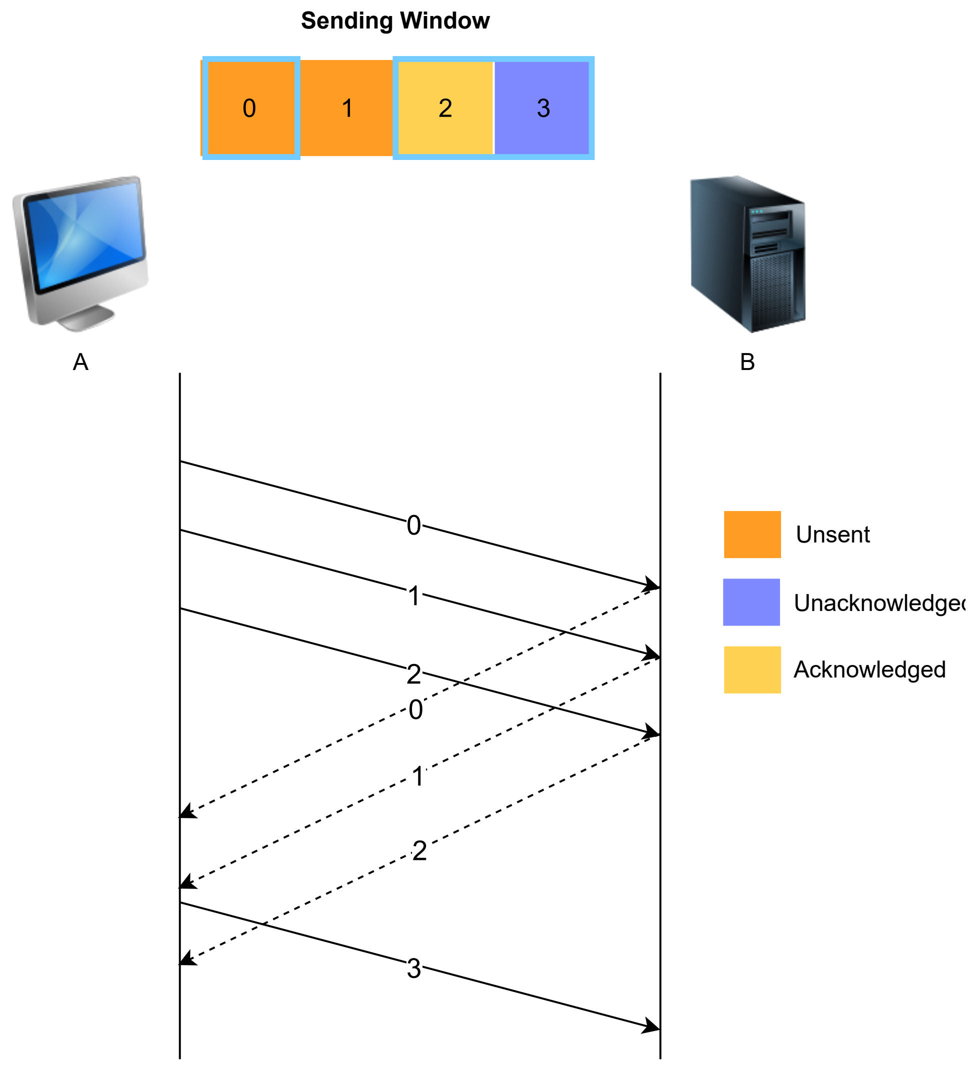
Sliding window moves ahead

The sliding window



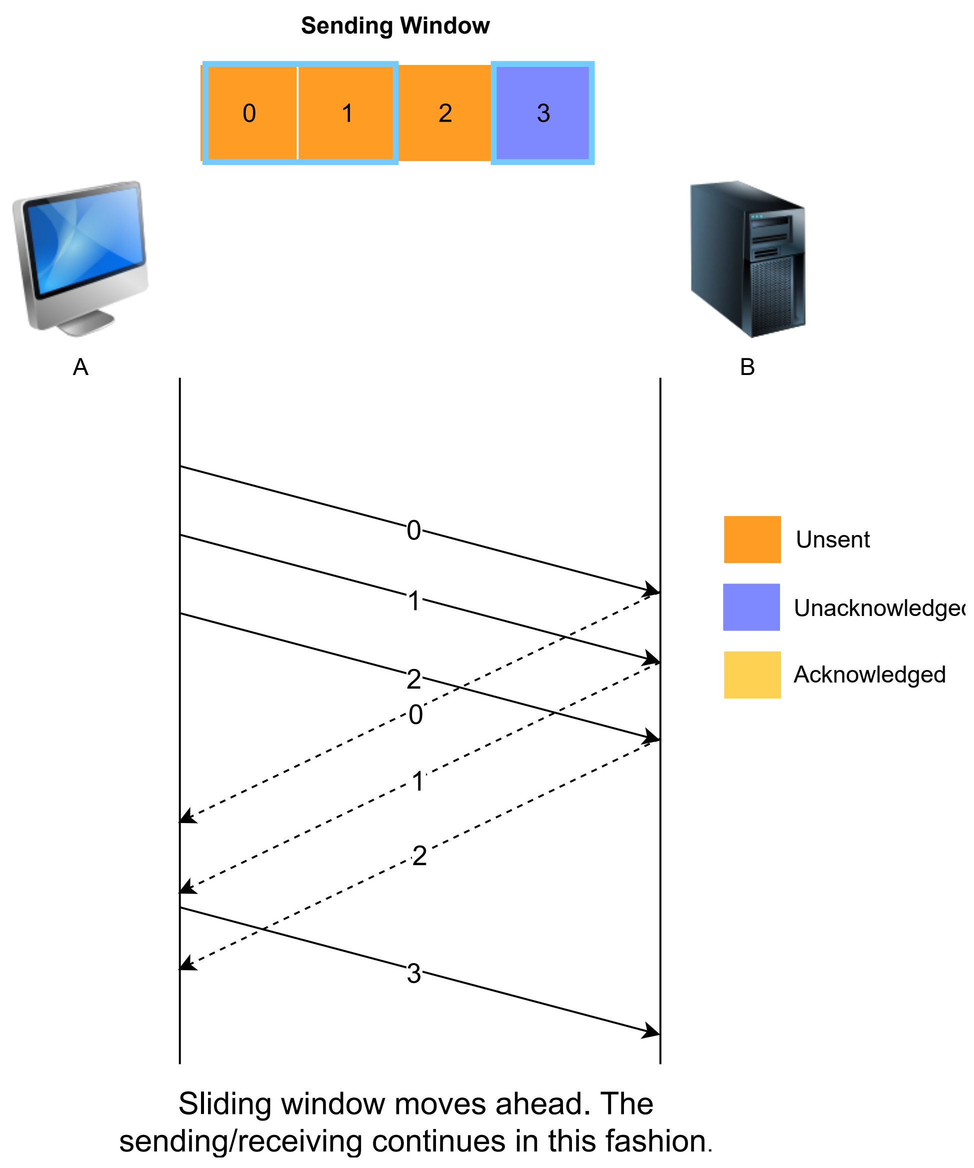
data for segment 3 has just arrived from the application (it wasn't available earlier) and data with sequence 3 is sent

The sliding window



Acknowledgement of segment with
sequence number 2 received

The sliding window


Sending Window
0
1
2
3
A
B
0
1
2
0
1
2
3
Unsent
Unacknowledged
Acknowledged
Sliding window moves ahead. The sending/receiving continues in this fashion.

The sliding window

11 of 11

Unfortunately, segment losses do not disappear because a transport protocol is using a sliding window. To recover from segment losses, a sliding window protocol must define:

- A heuristic to detect segment losses.
- A retransmission strategy to retransmit the lost segments.

Quick Quiz!

1

Pipelining and congestion control are competing objectives.

COMPLETED 0%

1 of 2

Let’s look at both in the next lesson.