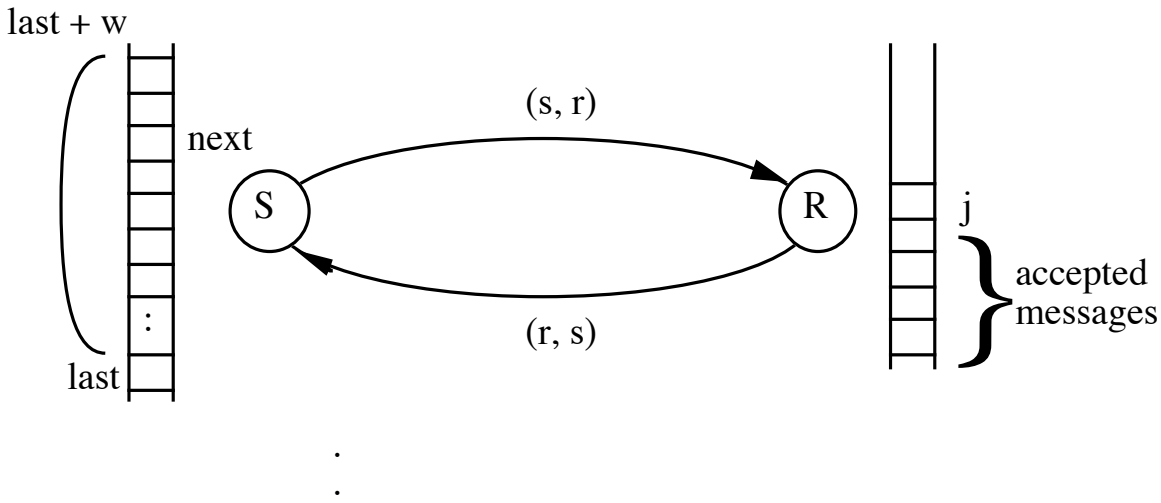
The sliding window protocol

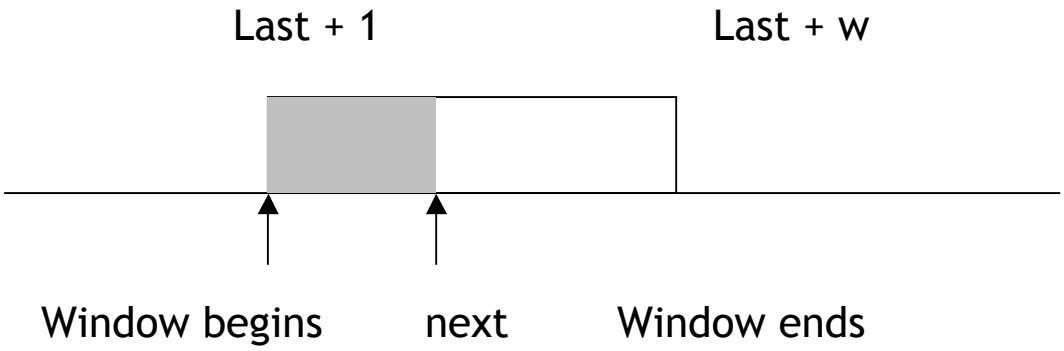


Creates a ***reliable FIFO channel*** on top of an unreliable channel that can lose and reorder messages. The requirements are:

No loss

No duplication

No reordering



1. Sender can send up to **w** messages without receiving acks. If no ack is received, then the entire window of messages is retransmitted

2. Receiver accepts a message if it is anticipated. Otherwise, it sends back an ack for the *last message* that it received.

{Sender's program}

{m[k] = **k**th message to be transmitted}

do last+1 ≤ next ≤ last + w □ send (m[next], next);
next := next + 1

(ack, j) is received □ **if** j > last □ last := j
j ≤ last □ skip
fi

timeout (R,S) □ next := last + 1 {Retransmission begins}

od

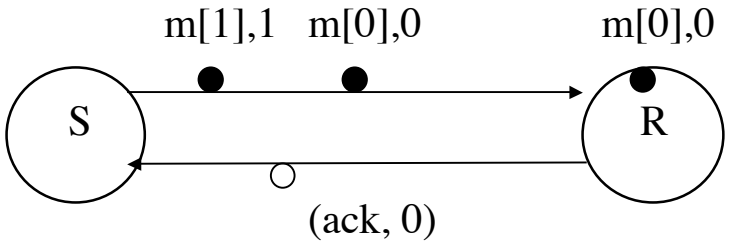
```
{Receiver's program}

define  j  :    integer (initially 0);

do   (m[next], next) is received     □
          if     j = next     □        accept  the message;
                                       send (ack, j); j:= j+1
                 j ≠ next     □        send (ack, j-1)
          fi;
od
```

Question 1. Why does it work?

Question 2. Can we solve it using bounded sequence numbers?

The Alternating Bit Protocol

It is a special version of the window protocol that works only on FIFO channels. The window size w = 1.



{Sender's program}

initially next = 0, sent = 1, b = 0;

{Both channels are empty};

do sent ≠ b □ send (m[next], b);

next := next +1;

sent := b

(ack, j) received □ **if** j = b □ b := 1-b

j ≠ b □ skip

fi

timeout (r,s) □ send (m[next-1], b)

od

{Receiver's program}

define j : **0** or **1**;

initially j = 0;

do (m[next], b) is received □

if j = b □ accept the message;

send (ack, j);

j:= 1 – j

j ≠ b □ send (ack, 1-j)

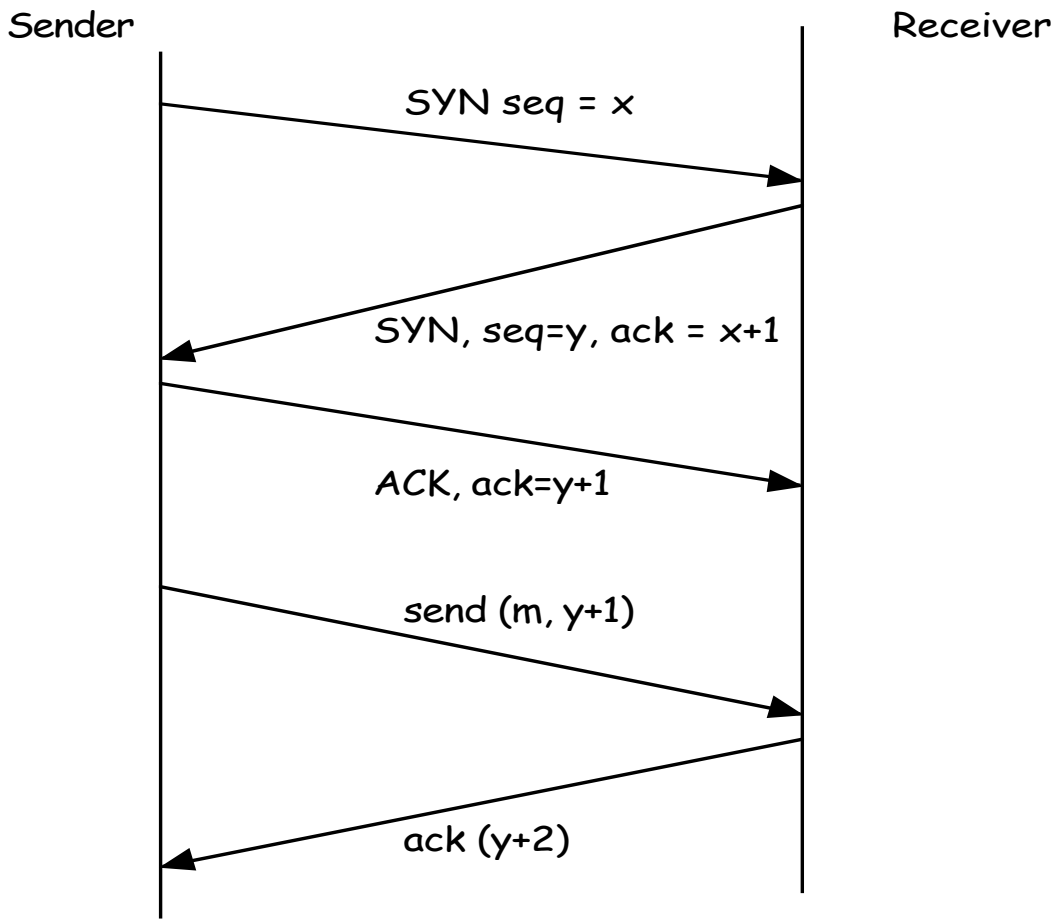
fi

od

How TCP works

TCP is a polished version of the sliding window protocol. *What about generating unique sequence numbers?*

Randomly chosen 32/64-bit pattern is most likely unique. Also, all sequence numbers older than 2□are discarded, where □ is the round-trip delay.

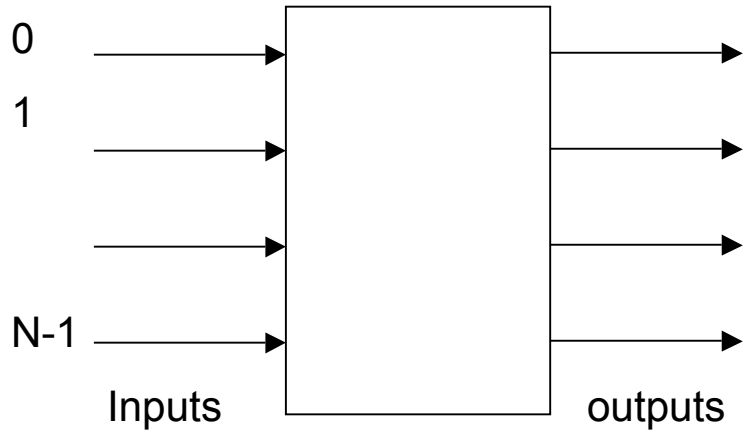


3-way handshake

Consensus

The consensus problem (crudely speaking, reaching agreement) is the mother of all (well, most) problems in distributed computing. Examples are

Leader election
Mutual exclusion
Decision to commit or abort in transactions
Clock synchronization



The problem becomes tricky if some of them are faulty. We are interested in this non-trivial version.

Requirements

Termination. Every non-faulty process must eventually decide.

Agreement. The final decision of every non-faulty process must be identical.

Validity. If every non-faulty process starts with the same initial value **v**, then their final decision values must be **v**.

Review the significance of these requirements.

An impossibility result

FLP Theorem (*due to Fischer, Lynch, Patterson*):
It is impossible to design a consensus protocol that will tolerate the crash failure of even a single process

(to be continued)