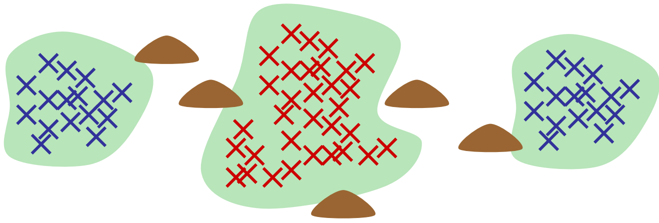
Data Communication Networks

Lecture 3

Saad Mneimneh
Computer Science
Hunter College of CUNY
New York

A flavor of distributed algorithms
the coordinated attack problem



- Three armies
  - two blue
  - one red
  - the red army separates the two blue armies
- Attacking
  - if blue armies attack simultaneously, they win
  - if they attack separately, the red army wins
- Communication
  - the only communication between the blue armies is to send a messenger through the red army
  - messenger can be captured $\Rightarrow$ message undelivered
- How to coordinate?

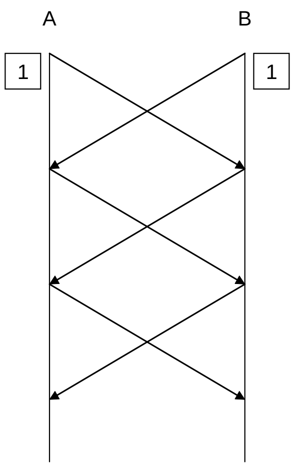
Fromal setting

- Denote two blue armies by A and B
- A and B both start with an individual decision, i.e. either 1 (let's attack) or 0 (let's not attack)
- A and B need to agree on either 0 or 1 (using some algorithm)
  - A and B can exchange messages
  - messages can be lost
  - the final agreement must be one of the original decisions (why?)
- Find an algorithm to reach agreement

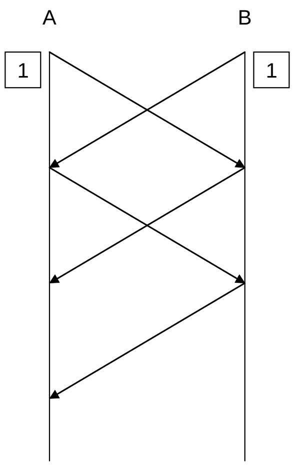
Fromal setting

- Denote two blue armies by A and B
- A and B both start with an individual decision, i.e. either 1 (let's attack) or 0 (let's not attack)
- A and B need to agree on either 0 or 1 (using some algorithm)
  - A and B can exchange messages
  - messages can be lost
  - the final agreement must be one of the original decisions (why?)
- Find an algorithm to reach agreement

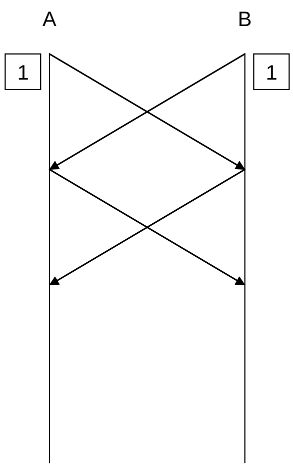
Impossibility result: There is no algorithm that correctly solves the problem if messages can be lost!



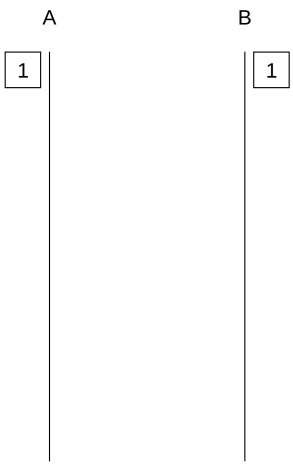
- Assume such an algorithm exists
- Both A and B start with 1
- They must both decide 1



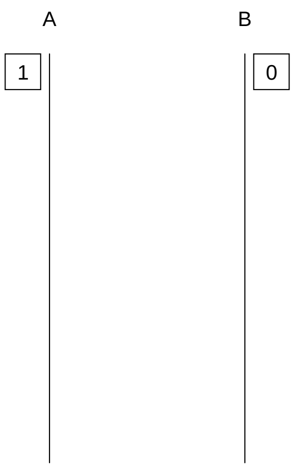
- Last message from A lost
- Execution looks the same to A
- A decides 1
- B must also decide 1



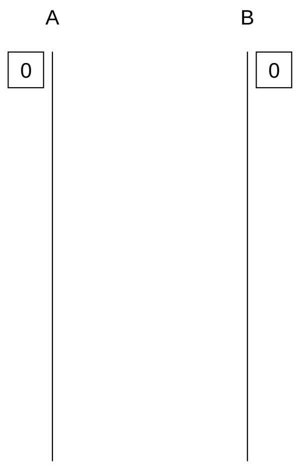
- Last message from B lost
- Execution looks the same to B
- B decides 1
- A must also decide 1



- Repeat the argument until all messages are lost
- Both A and B still decide 1



- Assume B's original decision is changed to 0
- Execution looks the same to A
- A decides 1
- B must also decide 1



- Assume A's original decision is changed to 0
- Execution looks the same to B
- B decides 1
- A must also decide 1
- Contradicts the requirement for final decision
- Therefore, such algorithm does not exist

So...

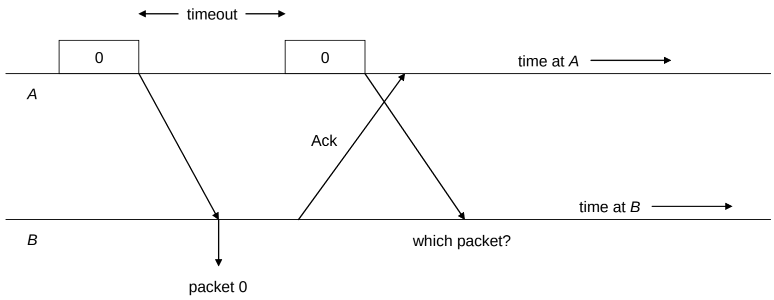
- How can we agree on anything in presence of message loss?
- The problem is in the setting itself
  - Purely theoretical result
  - For most problems of communication we only require that "eventually something good will happen"
  - A might be required to wait for a confirmation from B of this "eventuality"
- There is a probability > 0 that a message will be received
  - send multiple messengers (coordinated attack problem)
  - re-send a message (communication)

A sends a packet ... A does not know if B got the packet... A may resend... B sends an acknowledgement... B does not know that A got the ack... B may resend... A sends another packet upon receiving ack... A does not know that B got the packet... A may resend... B sends an acknowledgement... B does not know that A got the ack... B may resend...

At any point in time, there is no complete agreement... But there is eventual agreement with high probability.

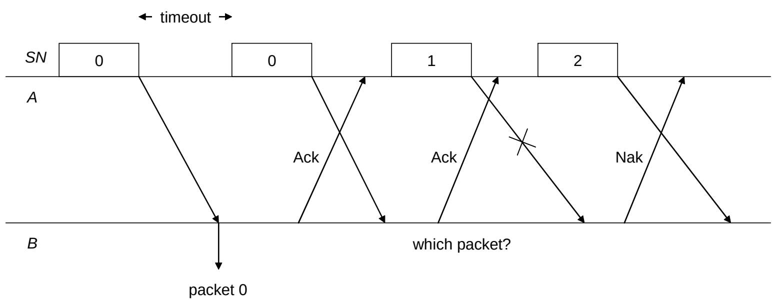
Stop and Wait

- Stop and Wait
  - A sends a packet to B
  - A waits for an acknowledgement from B
- Problem
  - either packet or ack may be lost (due to errors)
  - A might wait forever
  - use timeout



Stop and Wait (cont.)

- A can put a sequence number SN in the frame header
- B can use the sequence number to tell which is which
  - if B receives an error free packet, it sends an Ack
  - if B receives a packet with error, it sends a Nak (negative acknowledgement)



Problem with second Ack?

Stop and Wait (cont.)

- Ack and Nak must have sequence numbers too
- B sends a request number RN of the next expected packet
  - upon receipt of each packet
  - periodic intervals
  - arbitrary times
  - piggyback RN in frame header for packets going from B to A

	SN	*RN*		packet	CRC

Algorithm

```
A
SN ← 0
while (more packets)
        accpet packet from higher layer
        ack ← false
        while (!ack)
                send packet in frame with sequence number SN
                wait(timeout)
                if received frame from B with RN > SN
                   SN ← RN
                   ack ← true


B
RN ← 0
while (true)
        if frame with SN = RN received
           release packet to upper layer
           RN ← RN + 1
        with probability p > 0 send frame to A with RN
```

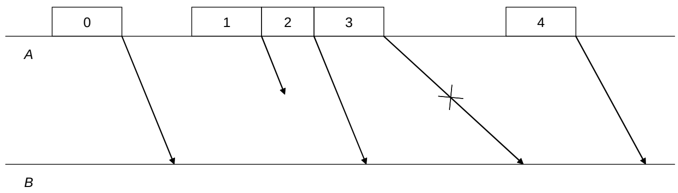
Unbounded sequence numbers

- Sequence numbers SN and RN are unbound
- How to fit in frame header?

Increment SN and $RN \bmod 2 \Rightarrow$ They alternate between 0 and 1

Would that work?

Need an extra condition: ordered delivery (why?)



Algorithm

$\underline{A}$
$SN \leftarrow 0$
while (more packets)
    accpet packet from higher layer
    $ack \leftarrow$ false
    while ($!ack$)
        send packet in frame with sequence number SN
        wait(timeout)
        if received frame from B with $RN \neq SN$
            $SN \leftarrow RN$
            $ack \leftarrow$ true

$\underline{B}$
$RN \leftarrow 0$
while (true)
    if frame with $SN = RN$ received
        release packet to upper layer
        $RN \leftarrow (RN + 1) \bmod 2$
    with probability $p > 0$ send frame to A with RN

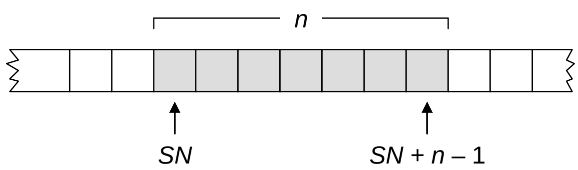
Throughput of Stop and Wait

- One packet is sent from A to B per RTT
  - B waits for packet
  - A waits for ack
- Example
  - link is 1.5 Mbps
  - RTT is 45 ms
  - frame size $= 1$ KB

  Therefore, we send 1000×8 bits every $0.045 + (1000 \times 8)/(1.5 \times 10^6)$ seconds, i.e. ≈ 160 Kbps
- We would like A to be able to send up to 10 frames before having to wait for acknowledgement
- ARQ Sliding Window ARQ

Sliding Window

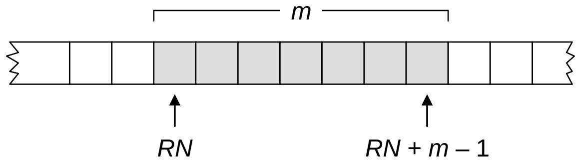
- In the previous scenario, we would like sender to be ready to transmit the 11^{th} frame at pretty much the same moment that the Ack for the first frame arrives
- The sender keeps a *window* of frames that if can send
- If the window size is n, the sender can transmit any frame with sequence number SN to $SN + n - 1$ before receiving $RN > SN$



- In Stop and Wait, the window size is 1, so the sender can send frames with sequence numbers in $[SN, SN + n + 1] = [SN, SN]$
- As before, if the sender receives a frame with request $RN > SN$, it sets SN to RN

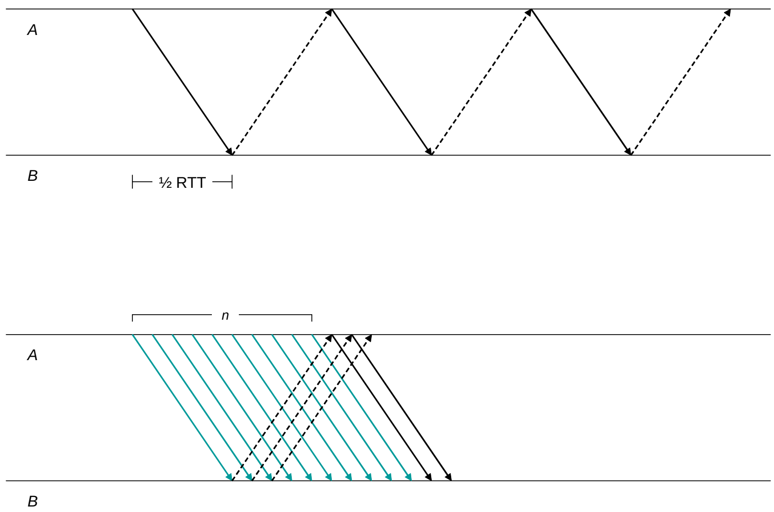
Sliding Window

- Similarly, the receiver keeps a window of frames that is willing to accept (but not necessarily deliver to the upper layer)
- If the window size is m, the receiver can accept any frame with sequence number RN to $RN + m - 1$ before receiving $SN = RN$



- In Stop and Wait, the window size is 1, so the receiver can accept frames with sequence numbers in $[RN, RN + m + 1] = [RN, RN]$
- Upon receiving a packet with $SN = RN$, the receiver sets RN to $RN + r + 1$, such that all packets with sequence numbers RN to $RN + r$ have been received
- Usually, $m \leq n$, e.g. $m = 1$ (Go Back n) or $m = n$

Stop and Wait vs. Sliding Window



Algorithm

<u>A</u>

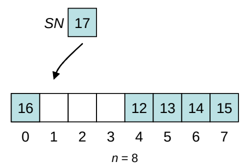
$SN \leftarrow 0$
while (more packets)
  accpet packets from higher layer
  $ack \leftarrow$ false
  while ($!ack$)
    send packets in frames with sequence numbers SN to $SN + n - 1$
    wait(timeout)
    if received frame from B with $RN > SN$
      $SN \leftarrow RN$
      $ack \leftarrow$ true

<u>B</u>

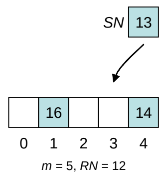
$RN \leftarrow 0$
while (true)
  if frame with $SN \in [RN, RN + m]$ received
    release packets RN to $RN + r$ to upper layer such that all r packets are received
    $RN \leftarrow RN + r + 1$
  with probability $p > 0$ send frame to A with RN

Buffers

- The sender needs to buffer at most n frames
  - if buffer is full, the sender does not accept more packets from upper layer
  - a frame with sequence number SN is stored in $buf[SN \bmod n]$



- Similarly, the receiver needs to buffer at most $m \leq n$ frames
  - if a frame is received with $SN \in [RN, RN + m - 1]$, it is accepted into the buffer
  - a frame with sequence number SN is stored in $buf[SN \bmod m]$



- Where does the receiver store frames with $SN = 12$ and $SN = 17$?

Algorithm

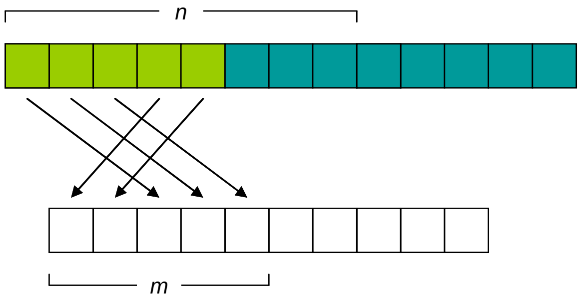
<u>A</u>
$SN \leftarrow 0$
...
if buf not full
  accept a packet and store the new frame in the buffer
...
if received a frame with $RN > SN$
  free $buf[SN \bmod n] \ldots buf[(RN-1) \bmod n]$
  $SN \leftarrow RN$

<u>B</u>
$RN \leftarrow 0$
...
if received a frame with $SN \in [RN, RN+m-1]$
  accept the frame and store it in $buf[SN \bmod m]$
  if $SN = RN$
    $RN \leftarrow RN + r + 1$ such that $buf[(SN+i) \bmod m] = SN + i$, $i = 0 \ldots r$
    free $buf[SN \bmod m] \ldots buf[(SN+r) \bmod m]$
with probability $p > 0$ send a frame to A with RN

Unbounded sequence numbers (again...)

- Sequence numbers SN and RN are unbound
- How to fit in frame header?
- For Stop and Wait, we used $SN \bmod p$ and $RN \bmod p$ with $p = 2$
- Would that work with Sliding Window?
  - The receiver needs to at least distinguish all sequence numbers in the sender's window
  - Therefore, we need to use $SN \bmod p$ and $RN \bmod p$ for some p (now are assume ordered delivery)
- Would $p = n$ work?
  - $p = n$ is enough to distinguish all sequence numbers in the sender's window
  - looking back at Stop and Wait ($n = 1$), we would argue for $p = 1$
  - The receiver needs to at least distinguish all sequence numbers in the sender's window plus a number that it has not yet seen
  - we need $p \geq n + 1$
  - that works for Go Back n ($m = 1$)
- In general, we need $p \geq n + m$

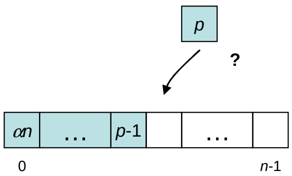
Why $p \geq m + n$?



- If Acks are lost, receiver will be seeing the light packets (and some dark ones)
- If Acks are not lost, receiver will be seeing the dark packets
- Therefore, all light and dark packets must be distinguished by the receiver
  - i am seeing these because Ack was lost?
  - or i am seeing these because Ack was not lost?
- Therefore, $p \geq m + n$

But...

- Although theoretically $p \geq m + n$ should be enough, with our particular implementation, it is not
- Consider the following situation



- When p is changes to 0, it will override frame n
- This cannot happen if p is a multiple of n
- If $m = n$ and $p = 2n$, we're fine
- What if $m < n$?
  - set p such that $p \geq m + n$ and p is multiple of both m and n
  - change implementation to use a circular queue, and keep a pointer to the head of the queue

Exercise

Think about how you would change the algorithm presented previously

Benefits of Sliding Window

- Reliably deliver frames across an unreliable link (can be also used to reliably deliver messages across an unreliable network)
- Preserve the order in which frames are transmitted
- flow control by changing window size and informing sender of how many frames it has room to receive (can also be generalized across network)