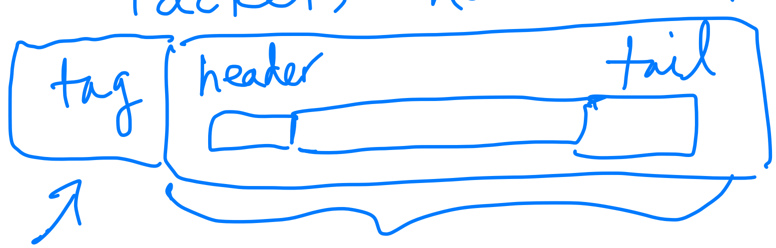
# AUTOMATIC REPEAT REQUEST (ARQ)

October 2, 2020

## AUTOMATIC REPEAT REQUESTs

- This technique is used to ensure the data is delivered accurately. automatically
- Assuming framing works, a strategy is required to handle frames with errors detected by the CRC (or whatever "error" detection technique is being used).
  - Such strategies are called **A**utomatic **R**epeat re**Q**uest (ARQ) strategies.
- Their purpose is to detect frames with errors at the receiving DLC and then to request the transmitting DLC to repeat the information.

Packets have been prepared by the DLC




5 101

3 011

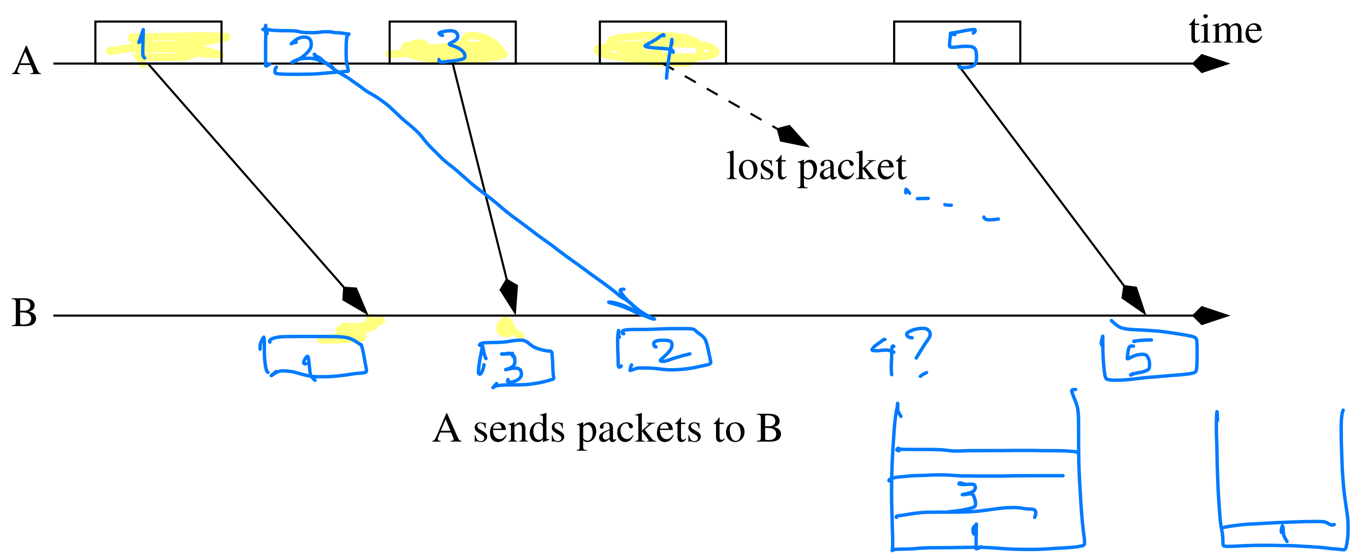
# CHARACTERISTICS OF STRATEGIES

- A ARQ strategy is characterized by its

  **correctness:** Is each packet released once and only once without errors from the DLC?

  **efficiency:** How much of the bit-transmitting capability is wasted by unnecessary waiting and by sending unnecessary retransmissions?

- Consider a flow from $A$ to $B$.




A sends packets to B

## ASSUMPTIONS IN ARQ STRATEGIES

1. Framing provides the begin and end of frame information for the receiving DLC.
2. A CRC (or other method) may be used for detecting errors.
3. All frames containing transmission errors are detected.
   - as a matter of fact actual probability for CRC error detection is $1 - 2^{-L}$, where $L$ is the length of the CRC.[a]
4. The bit pipe delivers frames in order (e.g., FIFO).
5. Each transmitted frame is delayed for an arbitrary and variable time before arriving at the receiving DLC.
6. The bit pipe may lose some frames.

$0, 1, \dots, N-1, \underbrace{0, 1, \dots, N-1}_{N=?}, 0, 1, \dots$

[a] We will not discuss this here in more detail.

## ASSUMPTIONS IN ARQ STRATEGIES

Peers

Send Receiver

- Assume two end systems connected by link (direct or otherwise).
  1. Source wishes to send messages
  2. Message broken into blocks which are sent individually.
  3. The reasons for this might be
     – limited buffer size
     – the longer the transmission the bigger the error probability $\frac{1}{100}$ Little's law
     – in LANs it is best not to have a single node occupy the medium for an extended period of time
- Question: What ARQ strategies can we use?
- Main Issue: the pipes may deliver in order but the queues may not be not be FIFO.

## ARQ STRATEGIES

- Several strategies are in use.
- Most important ones are
  - Stop-and-Wait
  - Sliding Window (this comes in two basic forms)
    1. Go-Back-$N$
    2. Selective Reject

Numerous Sliding Window

- Wireless
- Salletite
- Fiber Optic
- Wireline

## ERRORS IN STOP-AND-WAIT

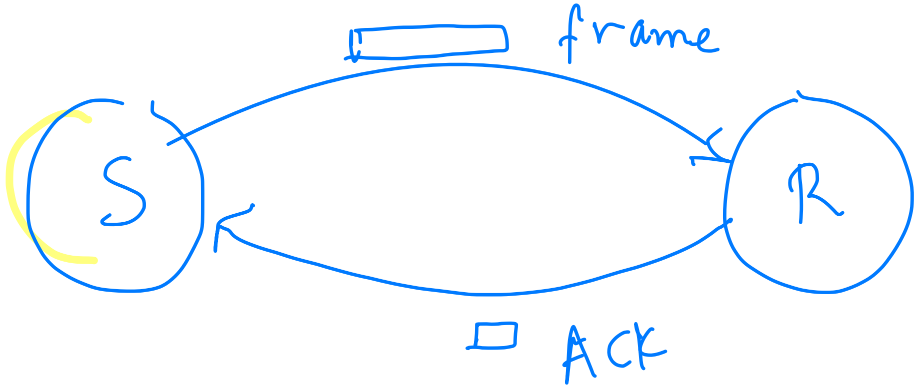
- There are two types of errors: Frame- and ACK-errors.

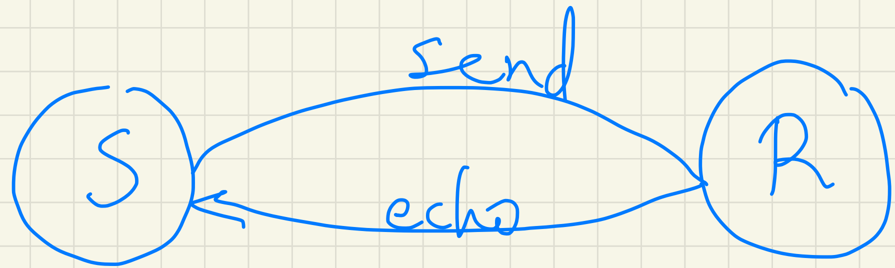
### 1. FRAME ERRORS

Frame that arrives in destination is damaged, i.e. one or more bits have been altered.

- if error is detected (e.g. via a frame check sequence based on CRC codes) the receiver discards frame.
- after frame is transmitted source waits for ACK within prespecified time frame (using a timer). If no ACK is received then the same frame is sent again.

How does S determine the waiting time?






- Can compute the RTT (Round Trip Time)
- ping
- The RTT depends on network circumstances

# ERRORS IN STOP-AND-WAIT

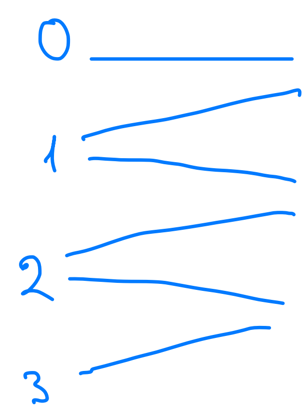
$0 + 1 = 1$

$1 + 1 = 0$

## 2. ACK ERRORS

Frame is received correctly but ACK is damaged.

Thus sender resends message and receiver accepts same message twice. You need to "add" a header

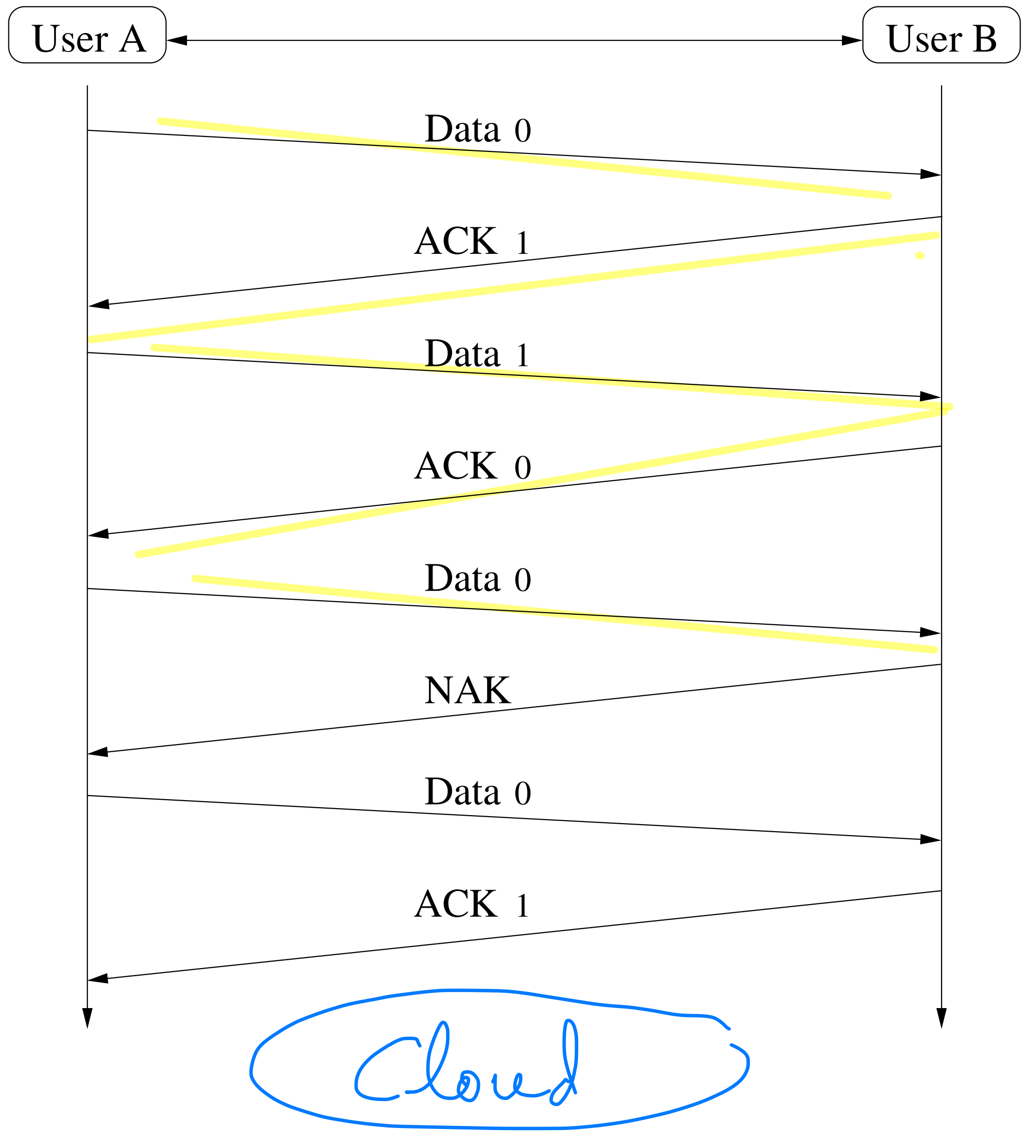



For this reason we use a labeling mechanism

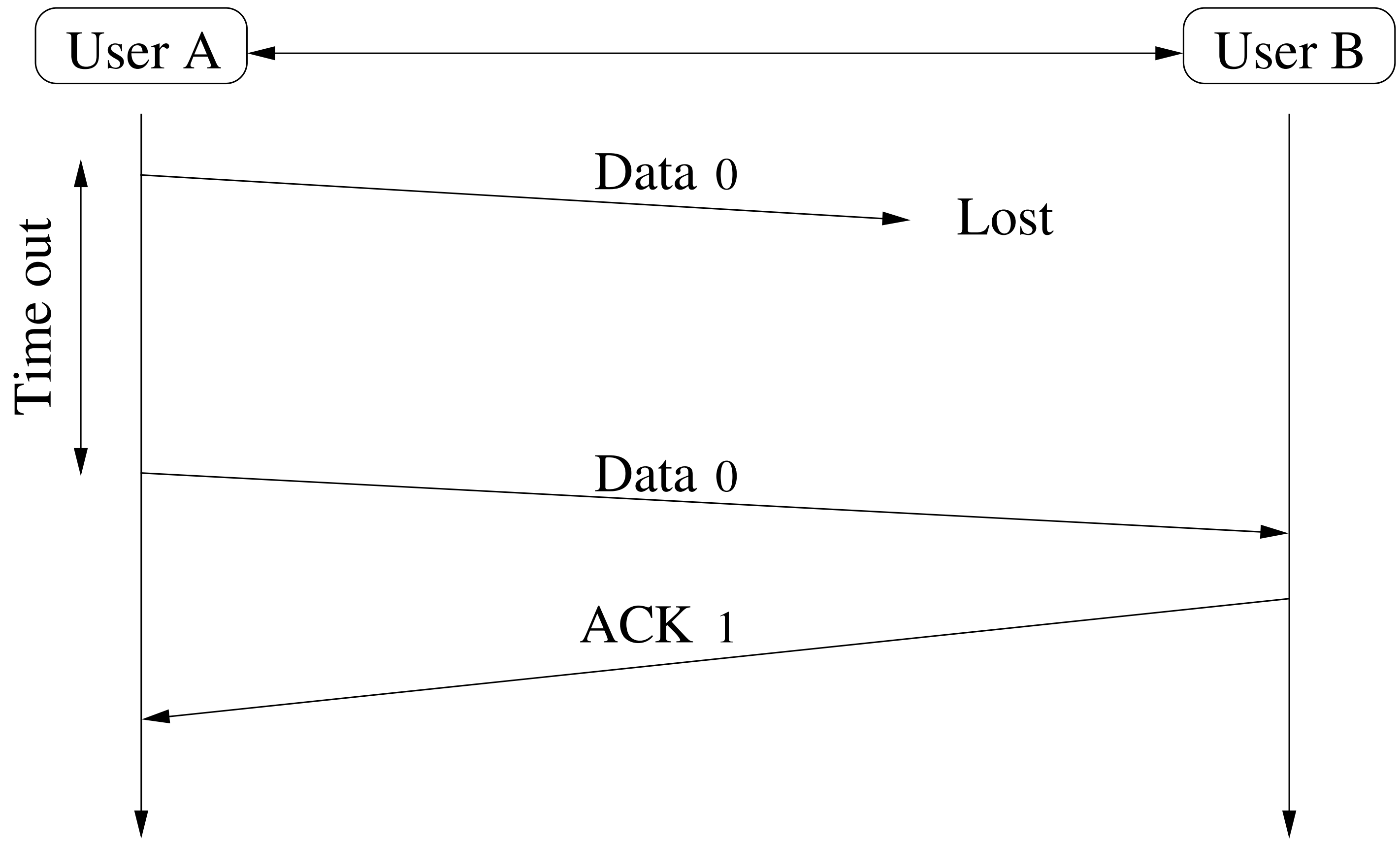
- Frames are labeled with a bit $b \in \{0, 1\}$.
- ACKs are also labeled with a bit $b \in \{0, 1\}$.
- ACK[$b$] acknowledges frame labeled $b + 1 \bmod 2$ and indicates that the receiver is ready for frame labeled $b$, where $b = 0, 1$.

It is sufficient to use as label a single bit $\in \{0,1\}$

# Stop-and-Wait ARQ

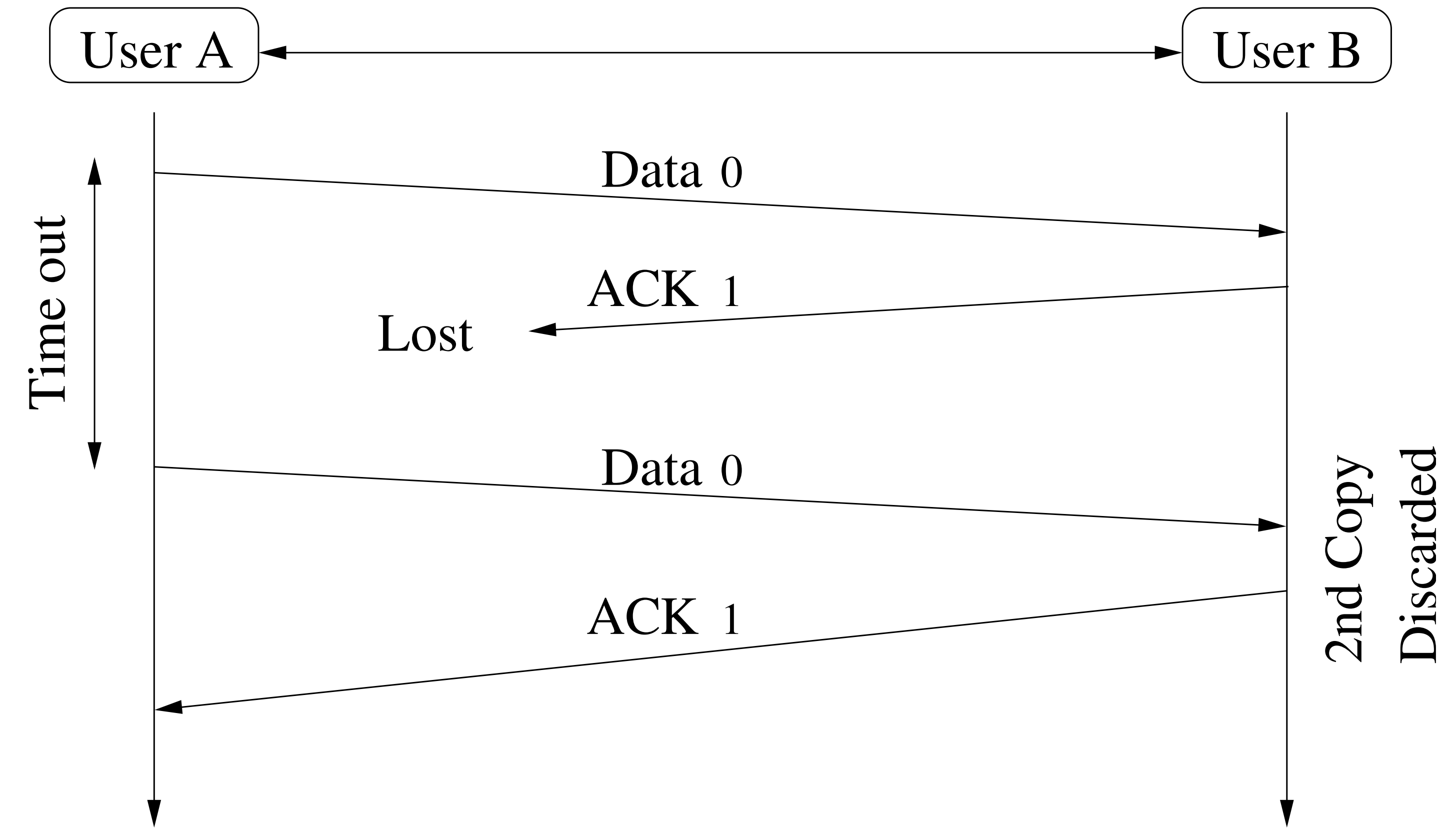


## Stop-and-Wait ARQ: Lost Frame

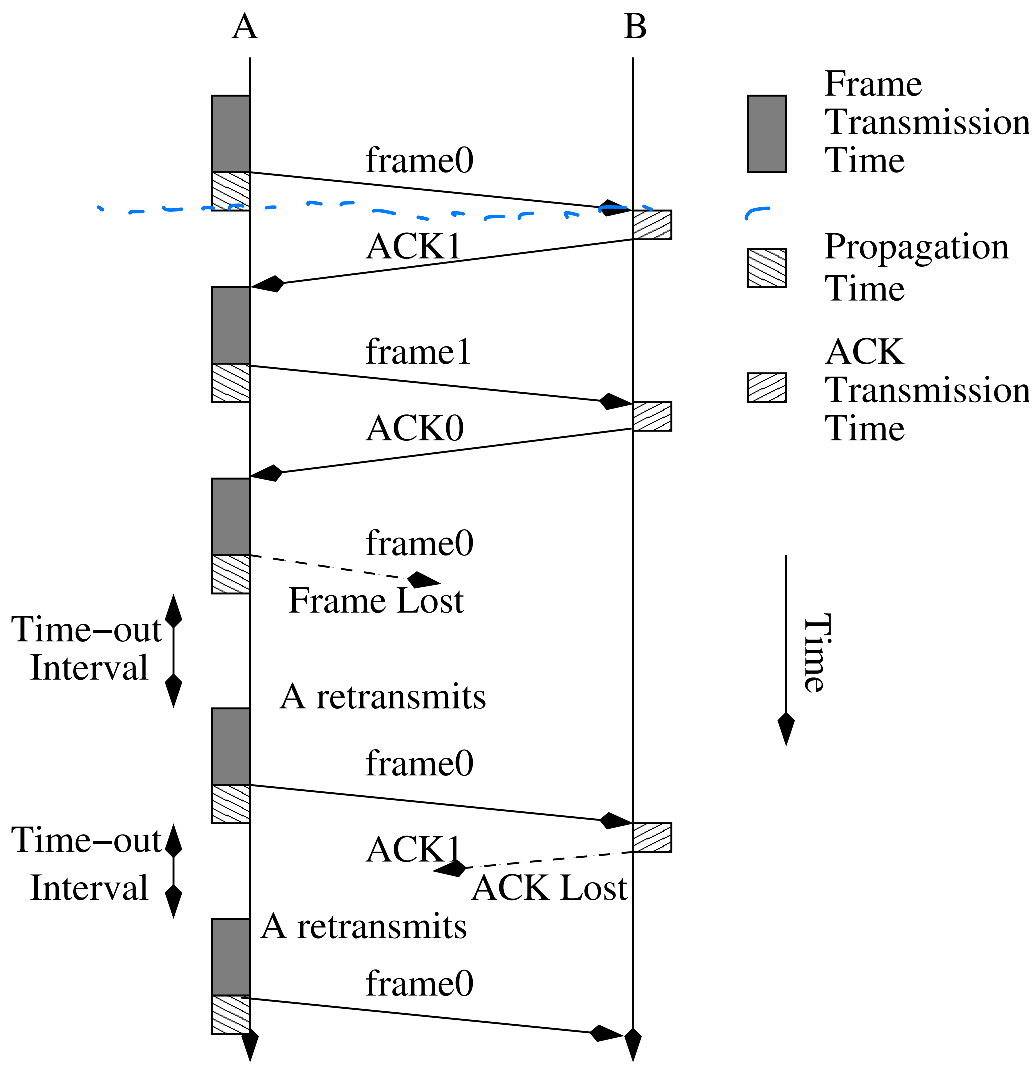



Time Out will be within set limits.

## Stop-and-Wait ARQ: Lost ACK



# Stop-and-Wait ARQ



## Correctness of Stop-and-Wait

- Can we show that a never ending stream of packets can be accepted from the higher layer at $A$ to the higher layer at $B$ in order and without repetitions and deletions?

You

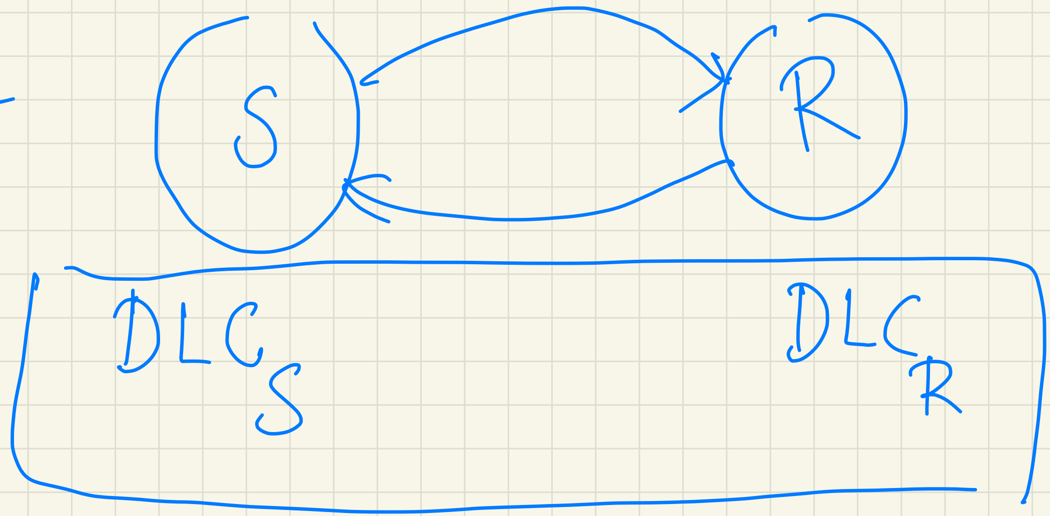
- Initial assumptions:

  1. all error frames detected by CRC,
  2. for some $p > 0$ all frames are received error free with probability at least $p$,
  3. link is initially empty,
  4. first packet from $A$ has label $b = 0$,
  5. $B$ is awaiting a packet with label $b = 0$.

Semantics: Program Verification

When you try to prove correctness

network layer




Assume that these protocols are correct

## Safety and Liveness of Stop-and-Wait

- Stop and Wait satisfies the following two important properties[a]
  - **Safety:** Stop Wait

    Algorithm never produces an incorrect result.

    In this case it means algorithm never releases a packet emanating from $A$ to the layer at $B$ out of the correct order.
  - **Liveness:**

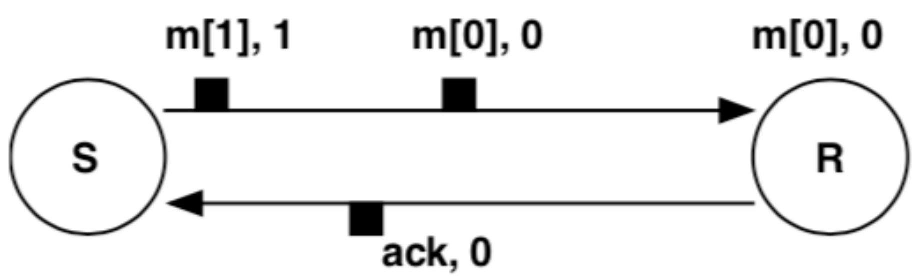
    Algorithm never enters a deadlock condition from which no further progress is possible.

    In this case it means algorithm continues to accept for ever new packets at $A$ and release them at $B$, and vice versa.

[a] We won't give a formal proof of these claims.

## Alternating Bit Protocol (ABP)

- Stop and Wait is also referred to as Alternating Bit Protocol.
- ABP can be thought of a special version of the sliding window protocol, with window size one. Works only when the channels are FIFO, which rules out message reordering. It is a suitable candidate for application in the data-link layer.
- With FIFO channels, the alternating bit protocol overcomes this problem by appending only a one bit sequence number to the body of the message.
- Global state of ABP



## ABP: Sender/Receiver

- Sender

```
define sent, b : 0 or 1; next : integer;
initially next = 0, sent = 1, b = 0, and both channels are empty;
do   sent≠b                  → send (m[next], b); next := next + 1;
                                sent := b
□   (ack, j) is received    → if j = b  →  b := 1-b
                              □ j ≠ b →   skip
                              fi
□    timeout (R,S)          → send (m[next-1], b)
```

- Receiver

```
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
```

## GO-BACK-$N$ ARQ

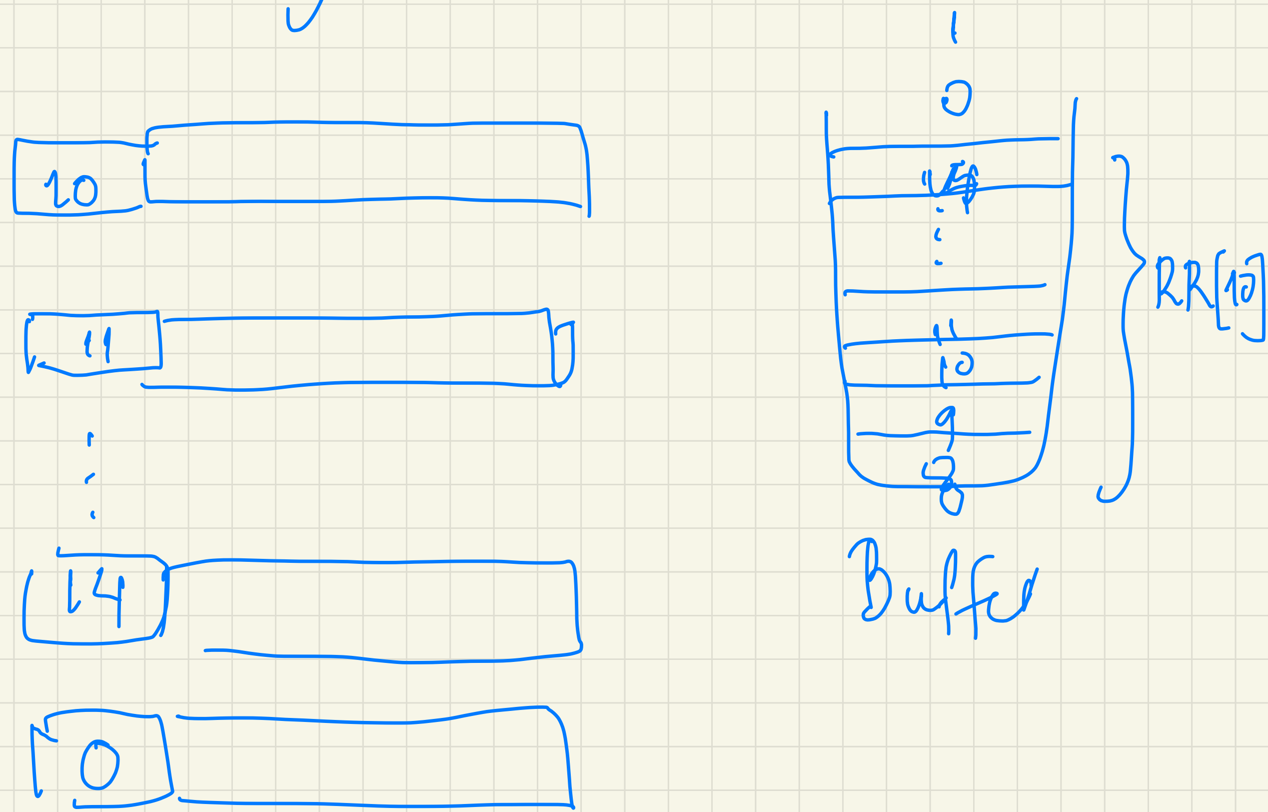
Go-Back-N is the most commonly used sliding window protocol.

- sender sends frames sequentially numbered modulo some max value
  - if receiver detects no error, an RR (Receive Ready) ACK is used for acknowledgement
  - if receiver detects error in a frame it sends REJ for this frame. This destination station will discard this and all future frames until the frame in error is correctly received.
- $N - 1$ frames may be sent before an ACK is received.
- Assuming stations $A$ (sender) $B$ (receiver) we have several cases to consider.

Unbounded sequence numbers is a major hurdle in implementing the sliding window protocols on **non-FIFO** channels.

# Sequentially numbered modulo N

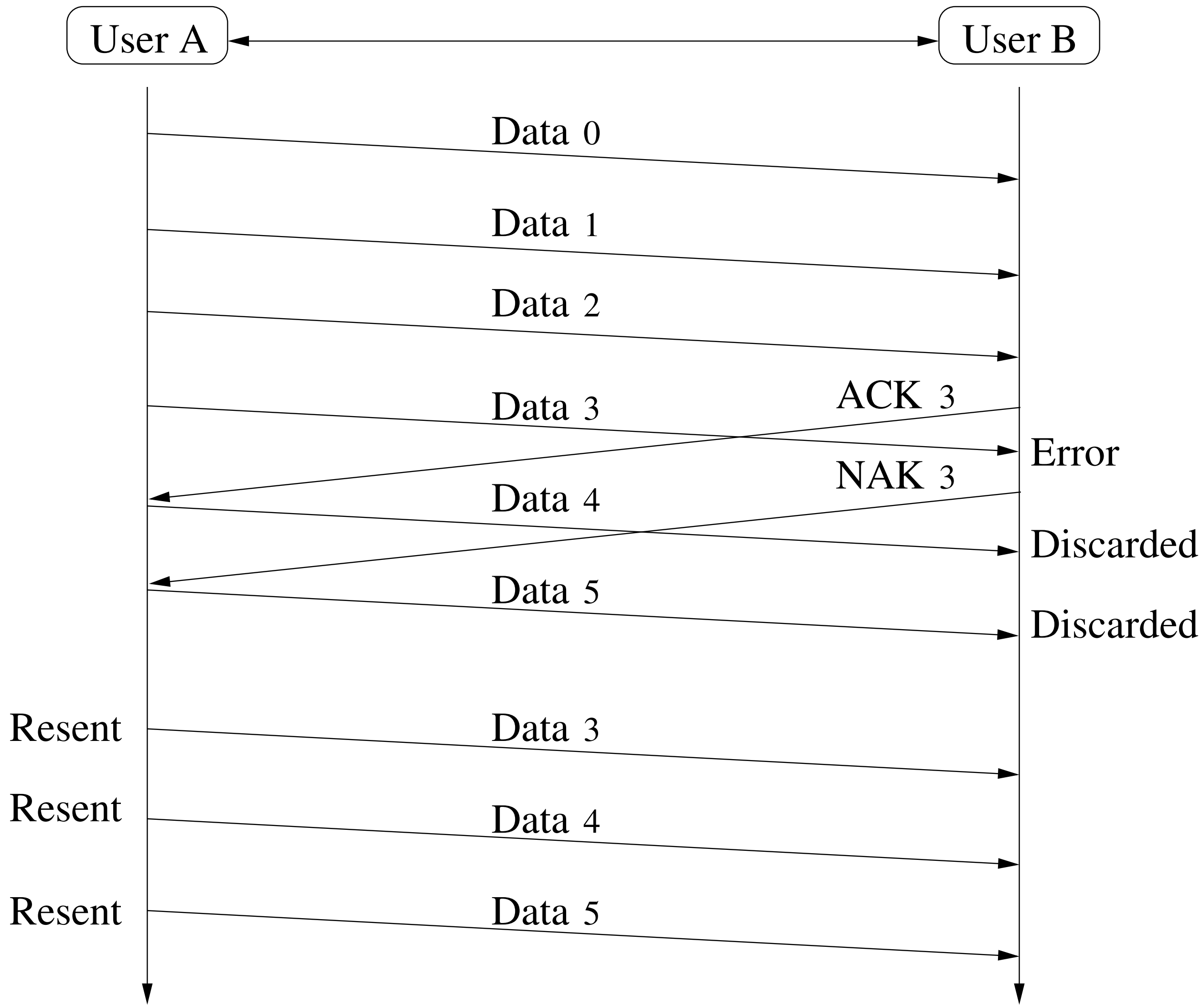
N = 15 :



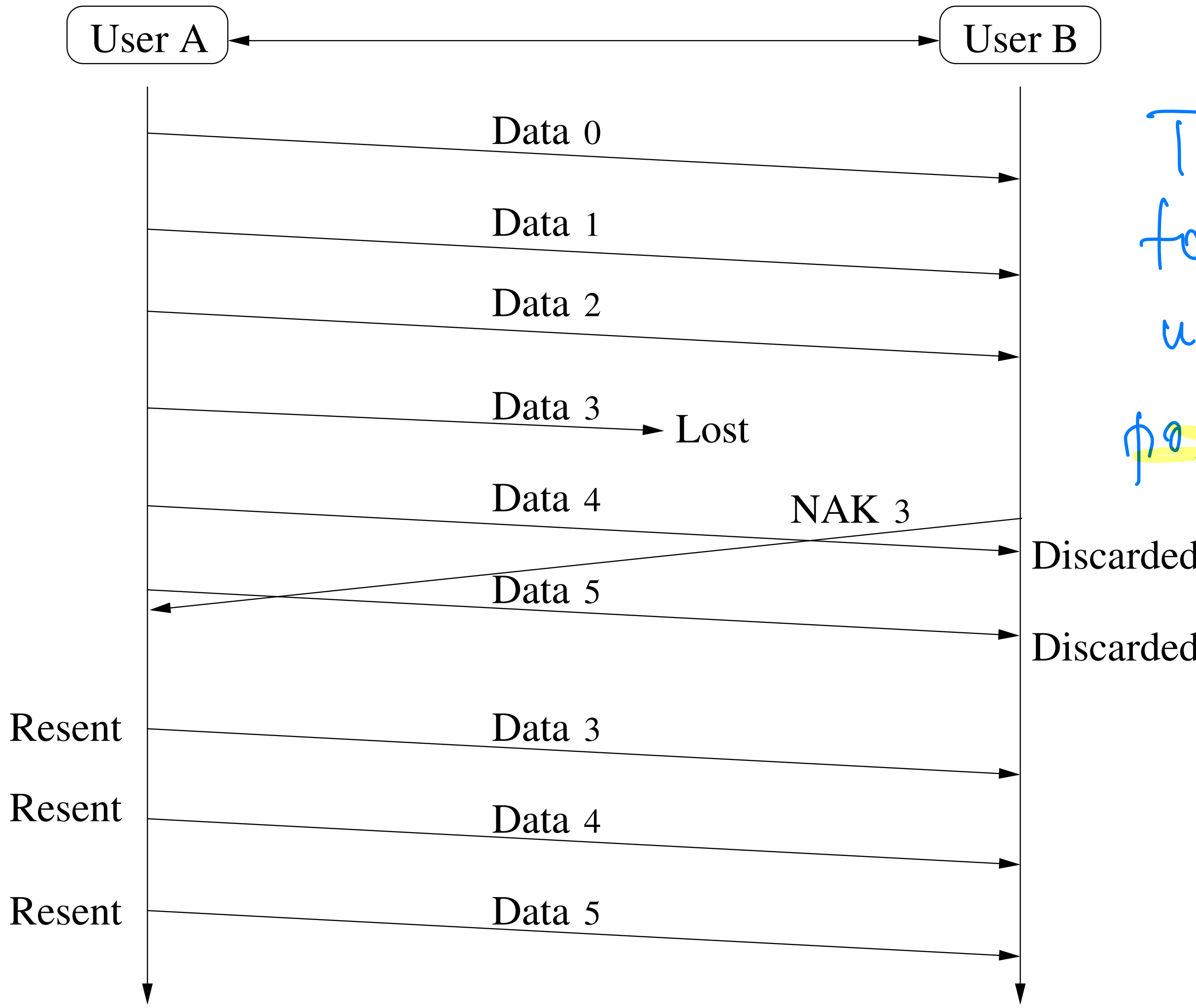
## GO-BACK-$N$ ARQ

- Sender keeps copies of all transmitted frames until they have been acknowledged. where in its buffer
- Receiver has the option to send either ACK or NAK if data frame is damaged. Because of the "continuity" of transmission both ACKs and NAKs are numbered. also places seq numbers on ACKs
- Sender is equipped with a timer in order to handle lost ACKs. Remember, $N-1$ frames may be sent before an ACK is received. If $N-1$ frames are awaiting acknowledgement the sender starts a timer and waits before sending any more!
- If allotted time runs out with no ACK received, the sender must assume frames were not received by receiver! RTT

# GO-BACK-$N$ ARQ: Damaged Frame

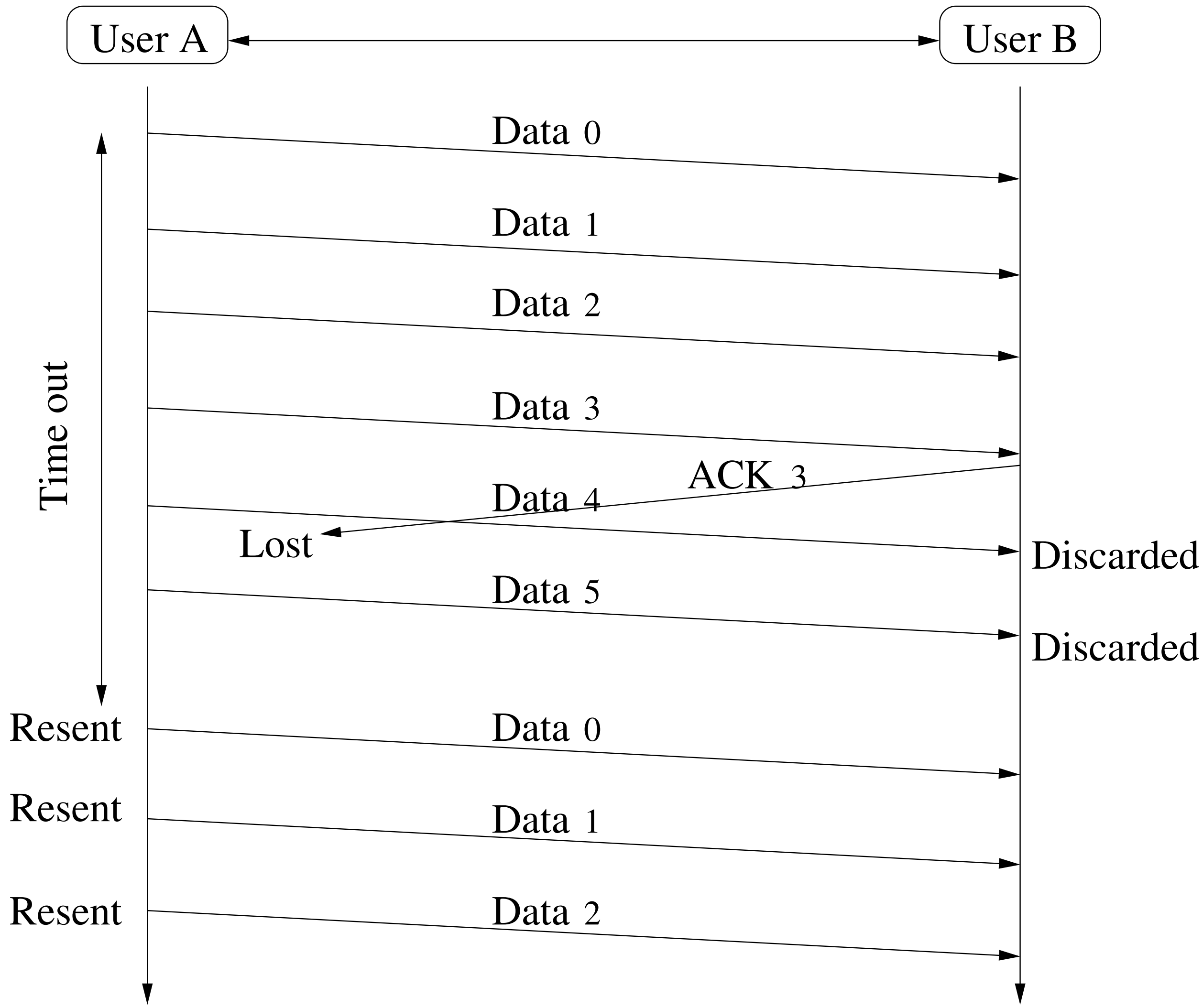


# GO-BACK-$N$ ARQ: Lost Frame




TCP likes to "say" it uses only positive ACKs.

## GO-BACK-$N$ ARQ: Lost ACK



## Damaged Frame: Interpretations 1.

$A \rightleftarrows B$

- **Case a.** $B$ detects an error and has previously received successfully $frame[i-1]$. $B$ sends $REJ[i]$. (Ack[i-1])

  When $A$ receives it must retransmit $frame[i]$ and all frames previously transmitted since original transmission of $frame[i]$.

- **Case b.** $frame[i]$ is lost in transit.

  cummulative

  $A$ subsequently sends $frame[i+1]$ which is received by $B$ out of order, and hence $B$ sends $REJ[i]$. $A$ must transmit $frame[i]$ and all subsequent frames.

- **Case c.** $frame[i]$ is lost in transit and $A$ does not soon send additional frames.

  $B$ receives nothing: sends neither $RR$ nor $REJ$. When $A$'s timer expires it transmits an $RR$ frame and a $P$-bit set to 1; $B$ interprets the $RR$ frame with a $P$-bit 1 as a command to be acknowledged by sending an $RR$ indicating the next frame that it expects. When $A$ receives $RR$ it retransmits $frame[i]$.

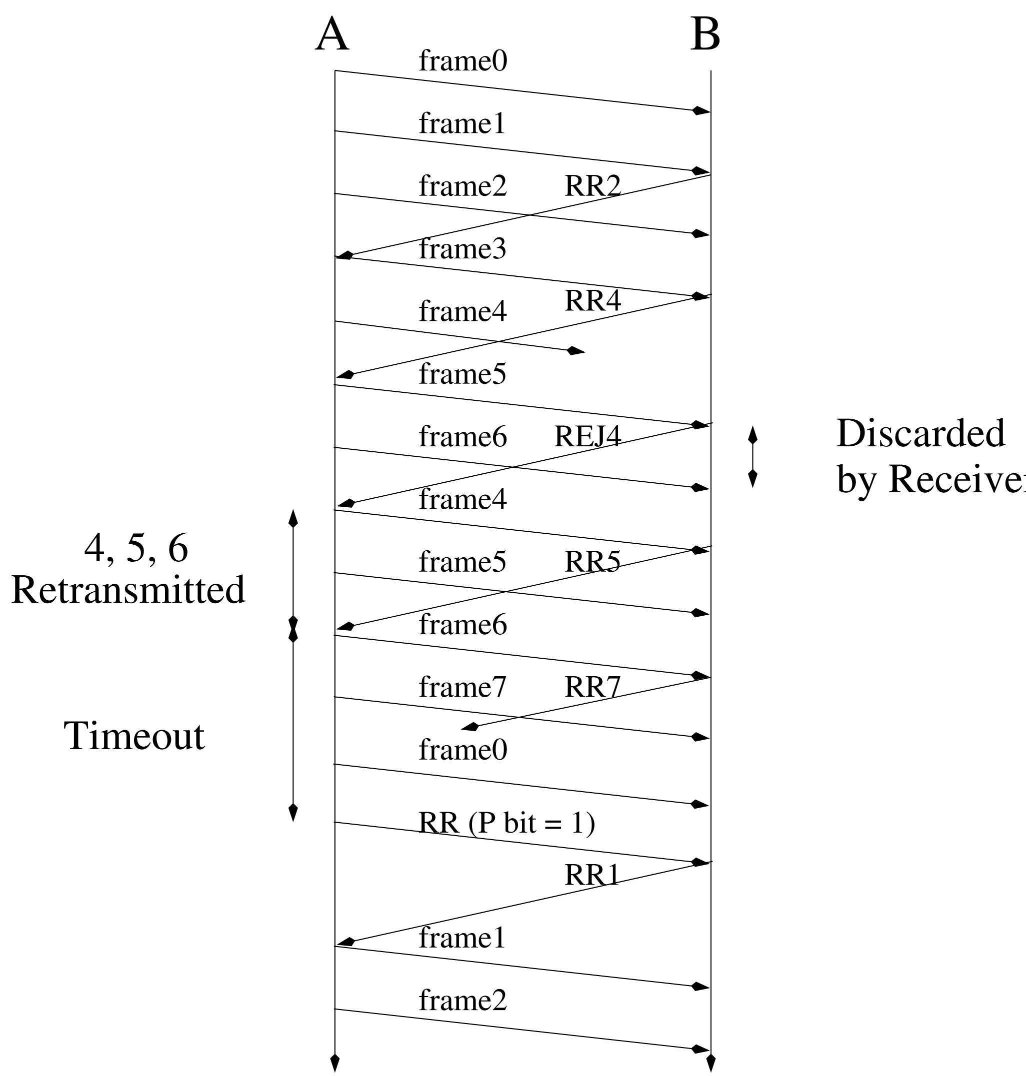
## **Damaged** $RR$ and REJ: Interpretations 2.

- **a.** $B$ receives $frame[i]$ and sends $RR[i+1]$ which is lost in transit. Because ACKs are cummulative (e.g. $RR[i]$ means all frames through $B$ are acknowledged) it may be that $A$ will receive a subsequent $RR$ to a subsequent frame and that it will arrive before the timer associated to $frame[i]$ expires.

- **b.** If $A$'s timer expires it retransmits an $RR$ command as in case 1.c. It sets another timer, called $P$-bit timer. If $B$ fails to respond to the $RR$-command or if its response is damaged then $A$'s $P$-bit timer will expire. At this point $A$ will try again by issuing a new $RR$-command and restarting the $P$-bit timer. This procedure is tried some maximum number of attempts. Failure of all these initiates a reset procedure.

Damaged **REJ** is similar to case 1c.

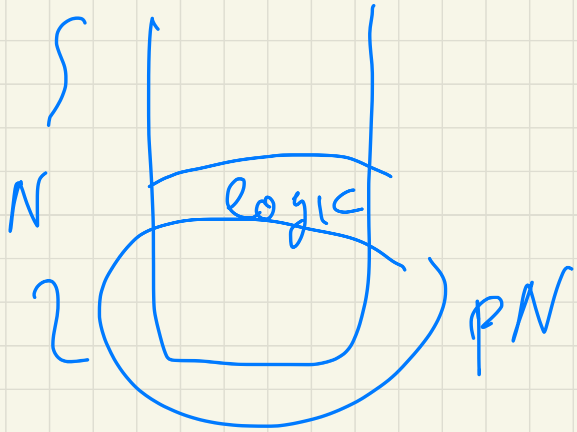
# Go Back N ARQ


A
B
frame0
frame1
frame2
RR2
frame3
frame4
RR4
frame5
frame6
REJ4
Discarded
by Receiver
frame4
4, 5, 6
Retransmitted
frame5
RR5
frame6
frame7
RR7
Timeout
frame0
RR (P bit = 1)
RR1
frame1
frame2

## SELECTIVE REJECT: SREJ

- Unlike Go-Back-$N$ ARQ, here retransmission is not cumulative.
- Only the specific lost frame is retransmitted (out of order)!
  - The receiving device must contain *sorting logic* to be able to reorder frames, and must be able to store frames received after a NAK.
  - The receiving device must contain a searching mechanism to be able to find only the requested frame.
- In general, it requires smaller window size than Go-Back-$N$.

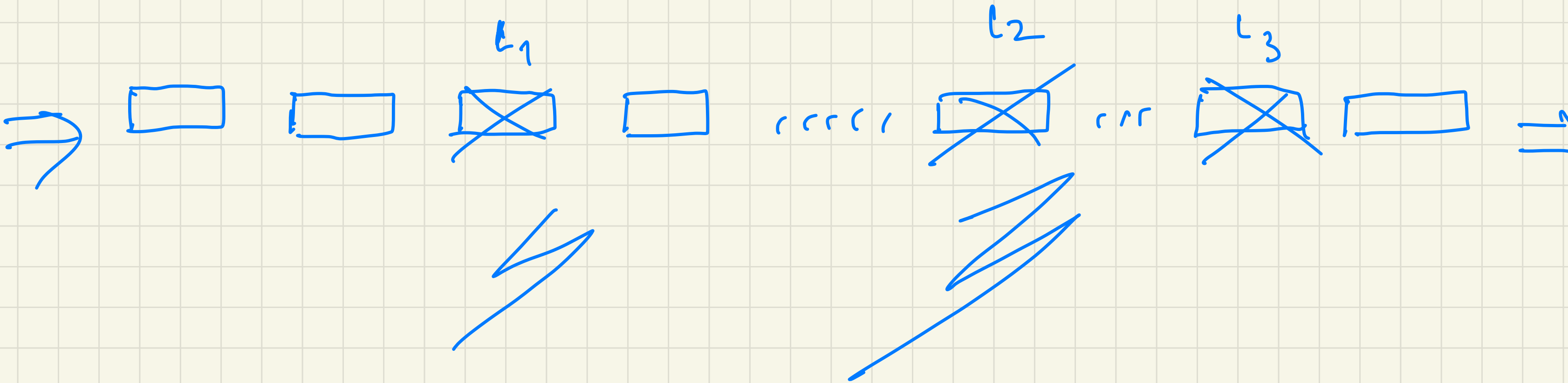
It works well in cases
where you have few errors




Errors w p    p

pN

# Problem in SREJ

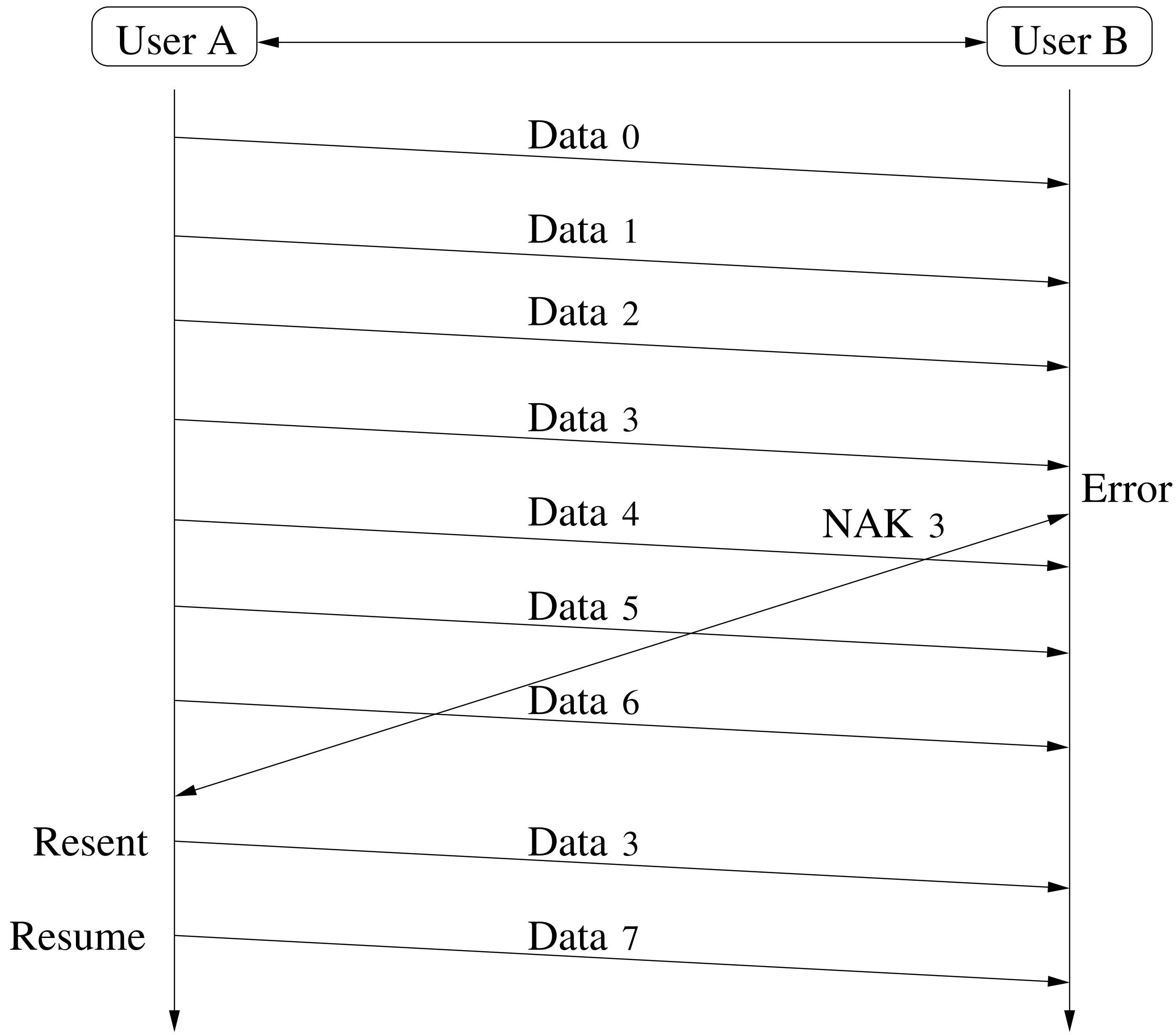



SREJ[$i_1$], SREJ[$i_2$], SREJ[$i_3$], ...

It creates overhead

"logic"

# SELECTIVE REJECT: SREJ



# Estimates on Overhead

## Go-Back-N

prob $p$

$\boxed{pN} = N_p$

In the worst-case you may have to sort them.

Sorting Alg: $N_p \log N_p$

Logic Alg. $L_p$

Transm. Cost $T_p$

$$\left( Cost_{sw} + N_p \log N + L_p + T_p \right) / Cost_{sw}$$