Data Link Layer, Part 2

CSC 343·643

Wake Forest University
Department of Computer Science

Fall 2013

Flow Control

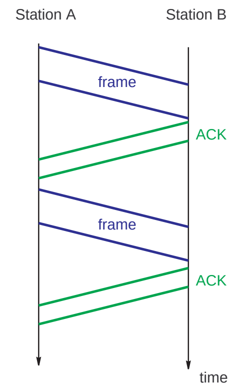
Another important issue (for the data link layer) is when the sender transmits faster than the receiver can process

- Flow control
  - Throttle the sender into transmitting no faster than the receiver can handle
  - Requires a *feedback* mechanism
- Rules typically are of the form - *Prohibit sending frames until permission given*
- There are two basic strategies for flow control
  1. Stop-and-wait
  2. Sliding window

Stop and Wait

- Rules (N.B. message broken into multiple frames)
  - Sender waits for an ACK after every frame sent
  - May send next frame after the ACK has been received
  - The data-ACK repeats until the end of the message
  - EOT sent to indicate end of message
- Acceptable for a few *large* frames

What happens if an ACK is lost?



Effect of Propagation Delay and Transmission Rate

Consider the maximum potential efficiency of a half-duplex point-to-point link using stop-and-wait

- Suppose a long message is broken into a long sequence of frames $f_1, f_2, ..., f_n$
- Using a polling procedure, the following events occur
  - Station S_1 polls station S_2
  - S_2 responds with f_1
  - S_1 sends an acknowledgement
  - S_2 sends f_2
  - S_1 sends ACK

    ⋮
  - S_2 sends f_n
  - S_1 sends ACK

- The total time to send the message is

$$T_D = T_I + n \cdot T_F$$

Where T_I is the time to initiate sequence

$$T_I = t_{prop} + t_{poll} + t_{proc}$$

and T_F = time to send one frame

$$T_F = t_{prop} + t_{frame} + t_{proc} + t_{prop} + t_{ack} + t_{proc}$$

To simplify, assume T_I is relatively small compared to the frame transmissions. Furthermore, assume the processing and ACK time is negligible. The formula is

$$T_D = n(2t_{prop} + t_{frame})$$

- Of the time spent, only $n \cdot t_{frame}$ is used for transmitting data. Therefore the link utilization is

$$u = \frac{n \cdot t_{frame}}{n(2t_{prop} + t_{frame})} = \frac{t_{frame}}{2t_{prop} + t_{frame}}$$

Defining $a = t_{prop}/t_{frame}$, the formula is

$$u = \frac{1}{1 + 2a}$$

This is the utilization of the link; however, why is the actual (what the user sees) utilization even lower?

Stop and Wait Utilization Examples

- WAN example
  - Two computers 1 km apart using ATM over a fiber optic link
  - ATM *frames* (cells) are 424 bits with a bit rate of 155.52 Mbps

$$t_{frame} = \frac{424}{155.52 \times 10^6} = 2.7 \times 10^{-6} seconds$$

$$t_{prop} = \frac{10^3}{2 \times 10^8} = 0.5 \times 10^{-5} seconds$$

$$a = \frac{0.5 \times 10^{-5}}{2.7 \times 10^{-6}} = 18.5$$

$$u = \frac{1}{1 + 2 \cdot 18.5} = 0.027$$

- LAN example
  - Two computers 0.1 km apart using 10 Mbps Ethernet
  - Frame size is 1000 bits and $v = 2 \times 10^8$

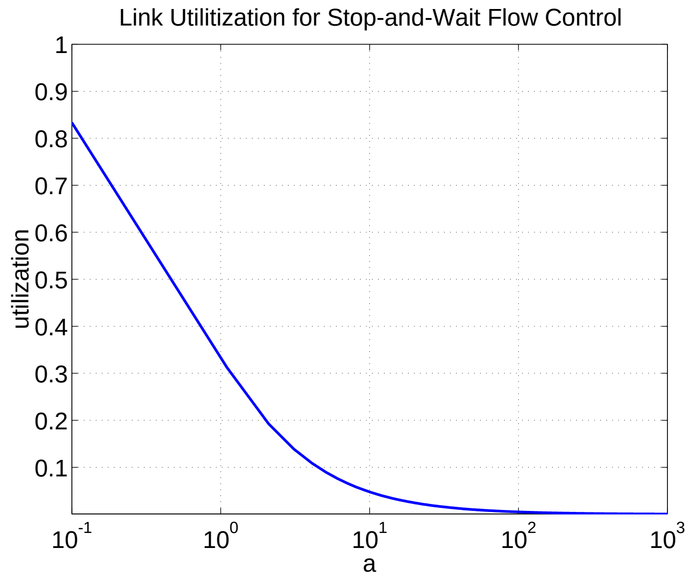
$$t_{frame} = \frac{1000}{10 \times 10^6} = 1 \times 10^{-4} seconds$$

$$t_{prop} = \frac{100}{2 \times 10^8} = 5 \times 10^{-7} seconds$$

$$a = \frac{5 \times 10^{-7}}{1 \times 10^{-4}} = 0.005$$

$$u = \frac{1}{1 + 2 \cdot 0.005} = 0.99$$

What can we conclude from the previous examples?



Sliding Window Flow Control

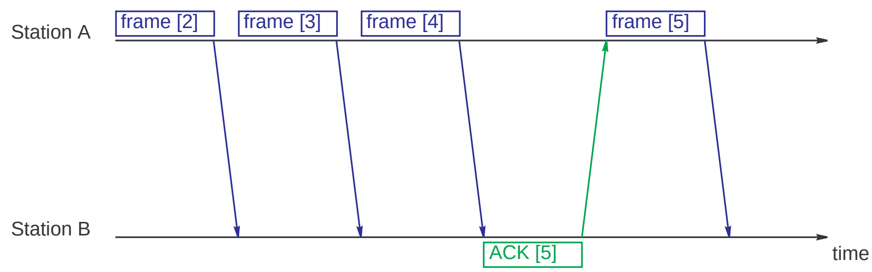
Overview

- Receiver allocates buffer space for n frames

  What is the buffer space for stop-and-wait?

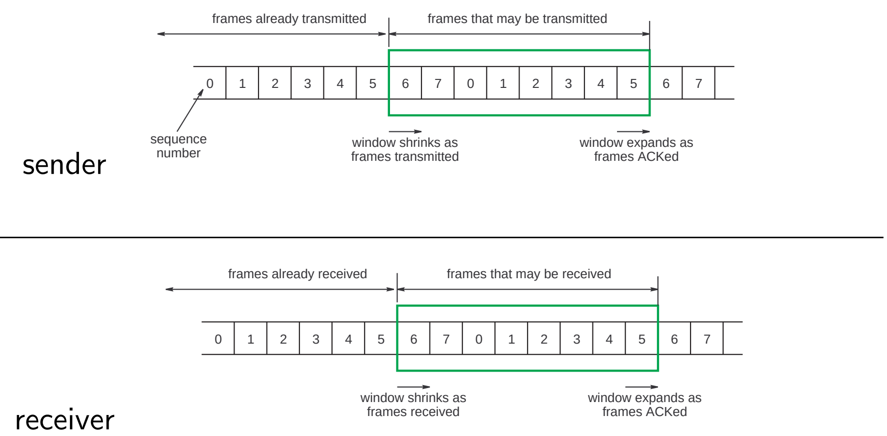
- Sender may transmit n frames without an ACK
- To keep track of which frames have be ACKed, each has a *sequence* number
- Receiver ACK's a frame by sending an ACK that includes the sequence number of the **next expected frame**
- Sender can send the next n frames **staring with** the last received sequence number

- A single ACK can acknowledge multiple frames
  - Receiver gets frames 2 and 3, but waits for frame 4 to ACK
  - Once frame 4 arrives, ACK[5] is sent to the receiver
  - Therefore frames 2, 3, and 4 are ACKed

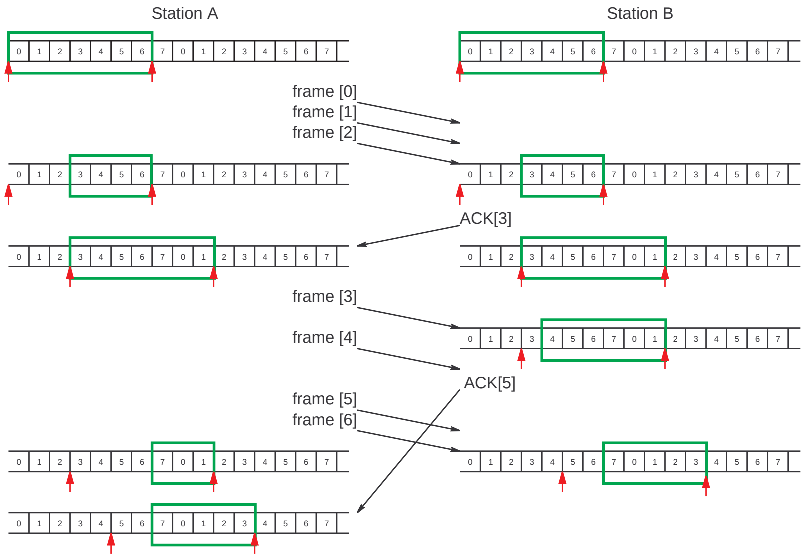


- Therefore, the sender can send multiple back-to-back frames, making better use of the channel

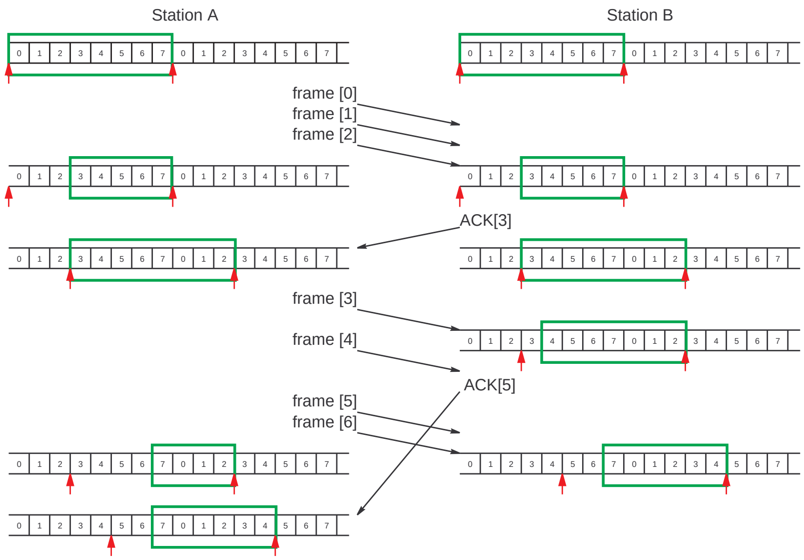
- Sequence number is a field in the frame $\Rightarrow$ finite size
- If k bits are reserved for the sequence number, then the values range from 0 to $2^k - 1$ (modulo counter)
- This can be depicted as a *sliding window*



- In the figures above, the window is represented using ______ bits



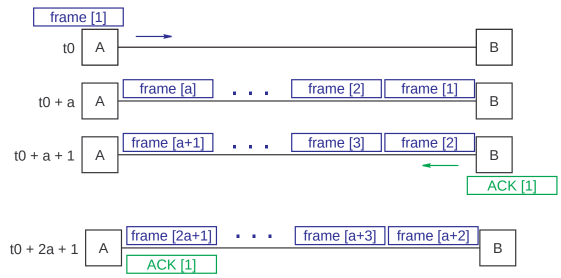
- N.B. can only send/receive n (window size) beyond last ACK[$\cdot$]
- In this example $n =$ ______ and $k =$ ______



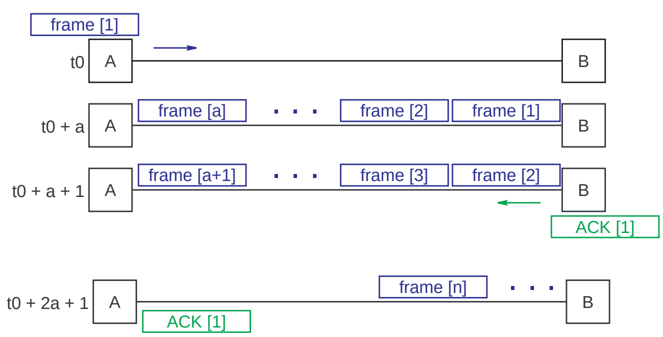
- In this example $n =$ ______ and $k =$ ______

Sliding Window Performance

- Assume a full-duplex point-to-point link using sliding window
- Efficiency depends on n (window size) and a ($\frac{t_{prop}}{t_{trans}}$)
- To simplify the analysis, normalize the frame transmission time; therefore, the propagation time is a
- There are two cases to consider
  1. ACK for the first frame reaches station A before the window is exhausted (*station A transmits without pausing*, $n > 2a + 1$)

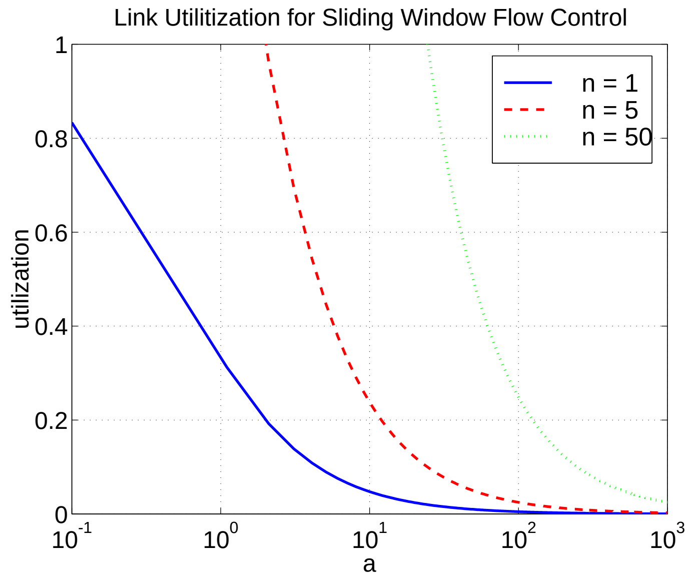


  2. Station A exhausts window before an ACK is received (*station A must pause*, $n < 2a + 1$)



- Using the two cases, the link utilization is

$$u = \begin{cases} 1 & n > 2a + 1 \\ \frac{n}{2a+1} & n < 2a + 1 \end{cases} \qquad (1)$$



Error Control

Mechanisms that detect (covered in previous slides) and correct (via retransmission) errors that may occur

- Two types of errors possible
  1. Lost frame - Frame (data or control) fails to arrive
  2. Damages frame - Frame recognized, but some bits have been changed

  What is the difference with error detection using parity?

- Error control mechanisms are referred to as **ARQ** (Automatic Repeat reQuest), there are three common versions
  1. Stop-and-wait ARQ
  2. Go-back-N ARQ
  3. Selective-reject ARQ

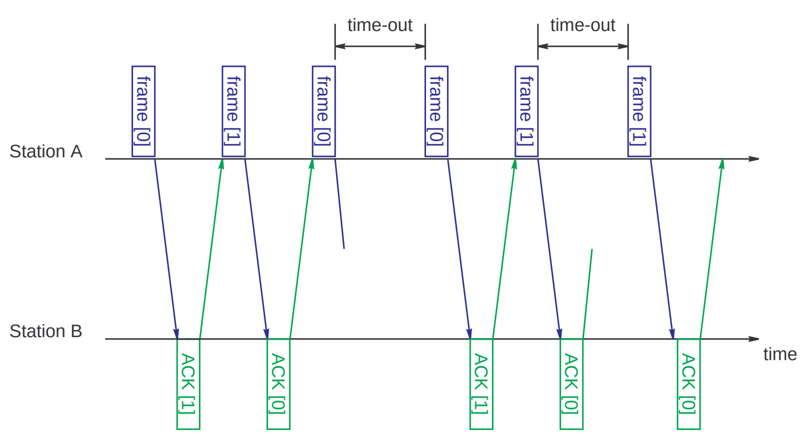
Stop-and-Wait ARQ

As previously described, sender transmits a frame then waits for ACK.

- Two types of errors may occur
  - Frame received in error (detected)
    - Receiver sends NACK to sender
    - Sender then retransmits frame
  - Frame (data) lost during transmission
    - Receiver neither ACKs nor NACKs
    - Sender *times-out* and retransmits

      What additional resources are required by the sender?
  - If the ACK is lost...
    - Frame received correctly, but ACK is lost
    - Sender times-out and retransmits
    - Receiver gets frame, it needs to identify that it is a copy

  - Avoiding the lost ACK problem
    - Label frames as 0 or 1 (alternating)
    - ACK[0] acknowledges frame[1], while ACK[1] acknowledges frame[0]



Stop-and-Wait ARQ Performance

- We have previously shown that error-free stop-and-wait achieves a maximum link utility of

$$u = \frac{1}{1 + 2a}$$

However, now we wish to account for the possibility of errors

- Utilization can be defined as

$$u = \frac{t_{frame}}{t_{total}}$$

where t_{frame} is the time to transmit a frame, and t_{total} is the total time the line is used to transmit the frame

- For error-free operation using stop-and-wait ARQ

$$u = \frac{t_{frame}}{t_{frame} + 2t_{prop}}$$

We will divide by t_{frame} and use $a = \frac{t_{prop}}{t_{frame}}$

- If errors occur then

$$u = \frac{t_{frame}}{n_r \cdot t_{total}}$$

where n_r is the number of transmissions required; therefore stop-and-wait ARQ utilization is

$$u = \frac{t_{frame}}{n_r \cdot (t_{frame} + 2t_{prop})}$$

Which can be rewritten as

$$u = \frac{1}{n_r \cdot (1 + 2a)}$$

- Must determine a value for n_r based on the probability that a frame is in error p
  - Assume ACKs and NACKs are never in error
  - The probability that it will take k attempts to transmit a frame successfully is

$$p^{k-1}(1-p)$$

    which is one success after $k-1$ failures

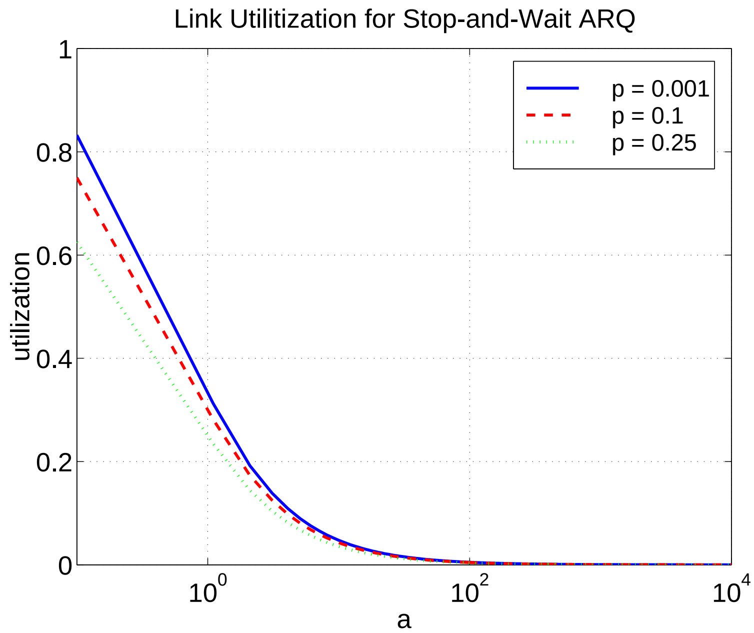
  - The expected number of transmissions is

$$n_r = E[\text{transmissions}] = \sum_{i=1}^{\infty} i \cdot p[i \text{ transmissions}]$$

$$= \sum_{i=1}^{\infty} i \cdot p^{i-1}(1-p) = \frac{1}{1-p}$$

- Therefore, the link utilization for stop-and-wait ARQ is

$$u = \frac{1-p}{1+2a}$$



Go-Back-N ARQ

A type of *continuous ARQ*

- Similar sliding window flow control, sender may transmit multiple frames based on window size
- ACK[i] still indicates expected frame
  - Every frame need not be explicitly ACKed

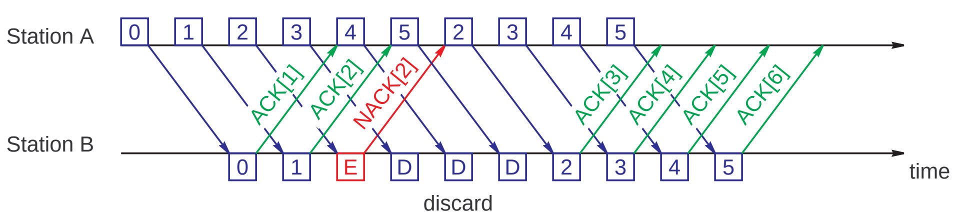
  Does sliding window flow control require each frame be explicitly ACKed?

- Modifications
  - Sender sets an *acknowledgement timer* for each frame
  - NACK[i] indicates
    * Frame i not received, resend along with any subsequent frames
    * Previous frames accepted
  - Receiver discards any frame out of order
  - Every lost/damaged frame must be NACKed
  - Maximum window size is $2^k - 1$
- There are two error cases
  1. Error/lost frame
  2. Error/lost ACK/NACK

Go-Back-N Error Cases

Frame Error

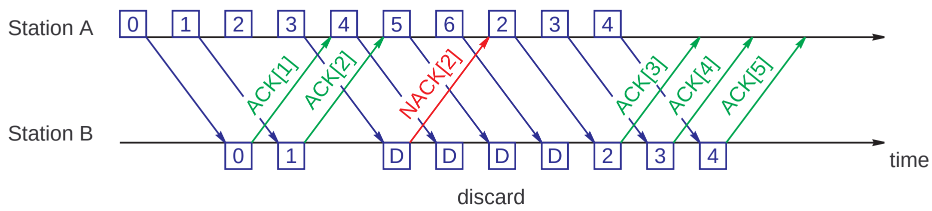
- Sender transmits FRAME[i]
- Receiver gets FRAME[i] then detects error
- Receiver transmits NACK[i], indicating FRAME[i] rejected
- Sender receives NACK[i] and must retransmit FRAME[i] and any subsequent transmitted frames



Lost frame

- FRAME[i] lost in transit (sender does not know)
- Sender sends FRAME[$i+1$]
- Receiver gets FRAME[$i+1$], which is *out-of-order*
- Receiver sends NACK[i] (first missing frame)
- Sender receives NACK[i] and must retransmit FRAME[i] and any subsequent transmitted frames

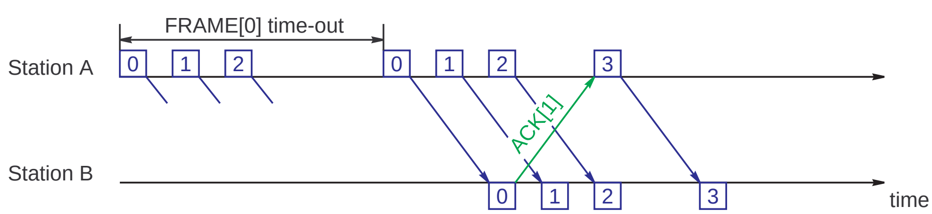
---



Sender Time-Out

- Sender transmits FRAME[i]

- FRAME[i] lost in transit
- Sender times-out (for ACK[i])
- Sender retransmits FRAME[i] and any subsequent frames



Damaged or Lost ACK

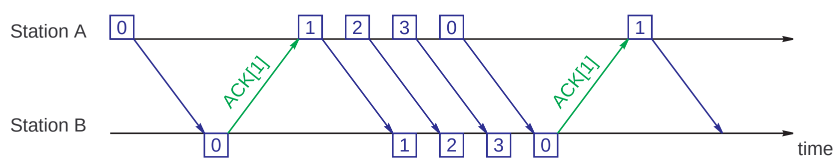
- Receiver get FRAME[i] and sends ACK[$i+1$]
- ACK[$i+1$] lost in transit
- Sender will eventually time-out or get subsequent ACK

Damaged or Lost NACK

- Receiver sends NACK[i], due to error
- NACK[i] lost in transit
- Sender will eventually time-out and retransmit

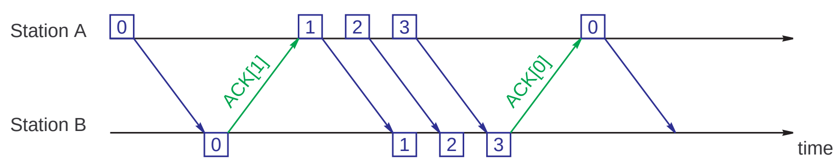
Go-Back-N Window Size

Although similar to sliding window flow control, go-back-n window sizes must be $\leq 2^k - 1$, where k is the sequence field length



$n = 4,\ k = 2$

(Sender does not know if the second ACK[1] is for the first or last FRAME[0])



$n = 3,\ k = 2$

Go-Back-N ARQ Performance

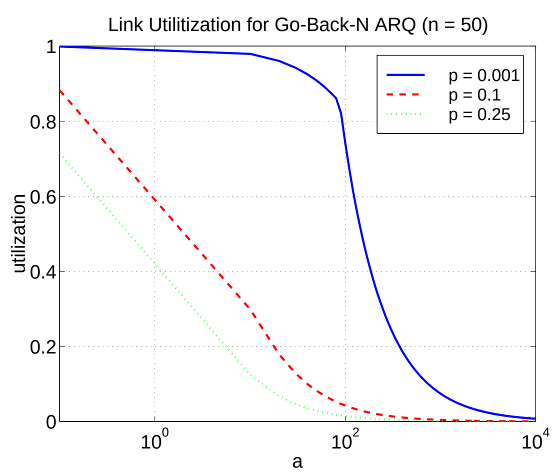
- Same assumptions as stop-and-wait ARQ analysis
- Approximating n is different, since each error may cause multiple frame retransmissions

$$n = E[\text{number of frames required to successfully transmit one frame}]$$
$$= \sum_{i=1}^{\infty} f(i) \cdot p^{i-1}(1-p)$$

where $f(i)$ is the total number of frames transmitted if the original frame must be transmitted i times

- Therefore, the link utilization for go-back-n ARQ is

$$u = \begin{cases} \frac{1-p}{1+2a \cdot p} & n > 2a+1 \\ \frac{n(1-p)}{(2a+1)(1-p+n \cdot p)} & n < 2a+1 \end{cases}$$

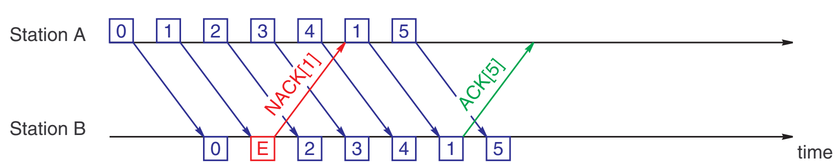


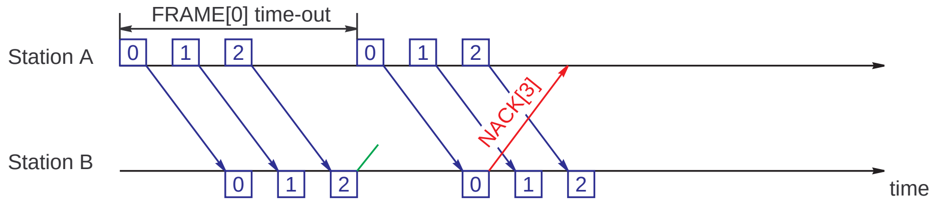
Selective Reject ARQ

Only the specific damaged or lost frame is retransmitted

- If FRAME[i] is lost or damaged, then a NACK[i] is returned
  - Only FRAME[i] is then retransmitted
- Receiving device must *place* frames in correct order
  - Since frames may arrive out of order
- Modifications
  - Receiver continues to accept frames after damaged frame
  - Smaller windows sizes are required, $n \leq 2^{k-1}$

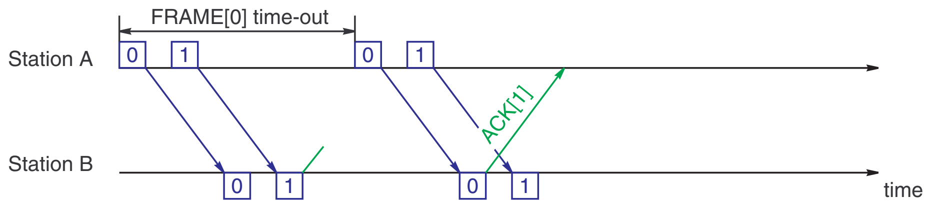
- ACK[i] and NACK[i] still imply the successful receipt of all previous frames
  - Frames after the error frame can not be ACKed until the damaged frame is received





$n = 3,\ k = 2$

(Receiver believes the second FRAME[0] is a new frame, not a repeat)



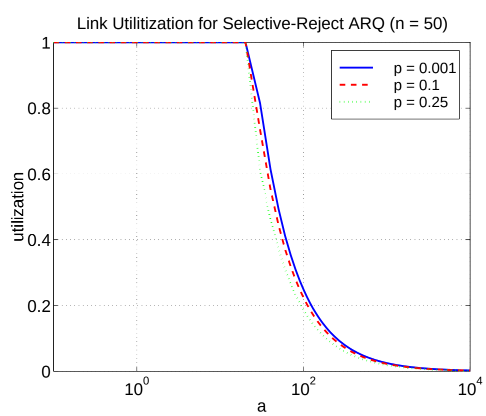
$n = 2,\ k = 2$

(Receiver knows second FRAME[0] is a repeat, since it expects either FRAME[2] or FRAME[3])

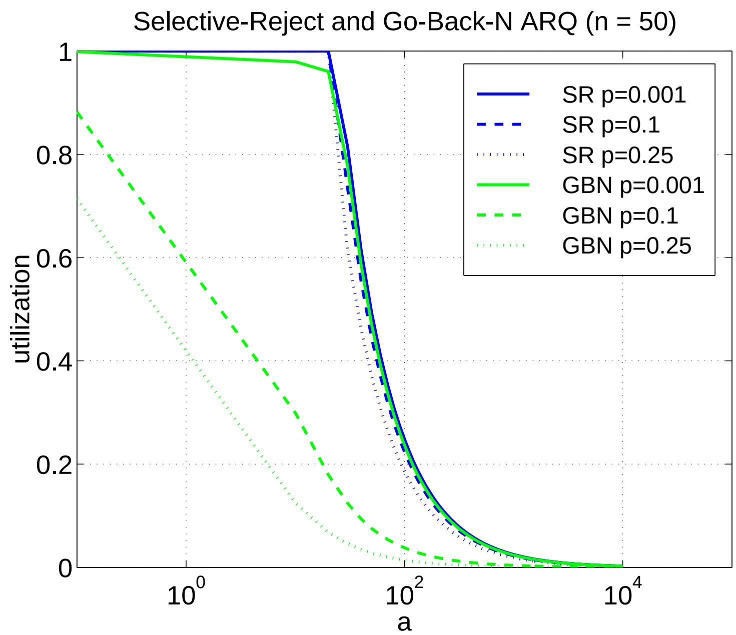
Selective Reject ARQ Performance

- Approximating n is the same as stop-and-wait

$$u = \begin{cases} 1 & n > 2a+1 \\ \frac{n(1-p)}{2a+1} & n < 2a+1 \end{cases}$$



ARQ Utilization Comparison



Go-Back-N and Selective Reject

- As seen in the previous figure, selective-reject yields a higher link utilization
- However, selective-reject is *expensive* to implement
  - Sender needs extra *state* information to retransmitt
  - Receiver needs complex storage and sorting
- Therefore, go-back-n is typically used for implementation