Rechnernetze – Computer Networks

Lecture 4: Direct Link Networks
Error Correction: Automatic Repeat Request

Prof. Dr.-Ing. Markus Fidler

Institute of Communications Technology
Leibniz Universität Hannover

April 26, 2024

Repetition

Channel capacity

- AWGN channel: $C = B \log_2(1 + S/N)$

Forward error correction (FEC)

- triple modular redundancy
- linear block codes

Error detection: frame check sequence (FCS)

- Internet checksum
- cyclic redundancy check (CRC)

Today

- FEC can correct some but not all bit errors
- FCS can detect erroneous frames but cannot correct these
- Solutions
  - FEC at frame level: redundant repair frames
  - automatic repeat request (ARQ): retransmit erroneous frames

Outline

- Redundant repair frames
- Automatic repeat request
  - Stop-and-wait
  - Pipelining
  - Sliding window protocols
    - Go-back-n
    - Selective repeat
- Data link layer protocols
  - HDLC

Redundant repair frames

Institut für Kommunikations-Technik

1	2	3	4		original stream
1	2	3	4	1⊕2⊕3⊕4	redundancy
1	2		4	1⊕2⊕3⊕4	received stream
1	2	3	4		reconstructed stream

- exclusive or (XOR) operation: $a \oplus b = 1$ iff $a \neq b$
- thus $a \oplus a = 0$ and $a \oplus 0 = a$ for any $a = 0, 1$
- the XOR operation is commutative and associative
- hence $f_1 \oplus f_2 \oplus f_3 \oplus f_4 \oplus f_1 \oplus f_2 \oplus f_4 = f_3$
- example: redundancy recovers one lost frame out of five

Automatic repeat request (ARQ)

ARQ error control

- concerned with retransmission of lost or erroneous frames
- uses feedback from receiver to sender
- acknowledgements to confirm the correct receipt of frames
- timeouts to trigger retransmissions
- ideally, ARQ retransmits frames only if needed
  (as opposed to the constant overhead of FEC frames)

Frames may, however, also be lost despite error-free transmission

- if the receiver cannot process data at the speed of the sender
- flow control to adapt the sender to the receiver's ability
- uses feedback from receiver to sender

Often, the mechanisms for error and flow control are interlinked.

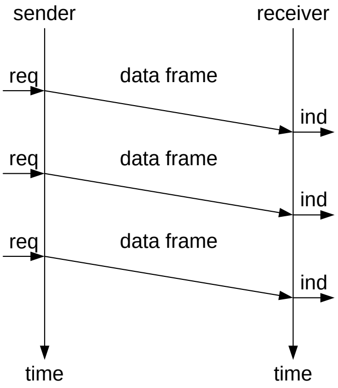
Protocol 1: Utopia

Assumptions

- error-free channel
- infinitely large buffer
- infinitely fast receiver

But

- finite receiver buffer
- finite processing speed
- ...



Assumptions

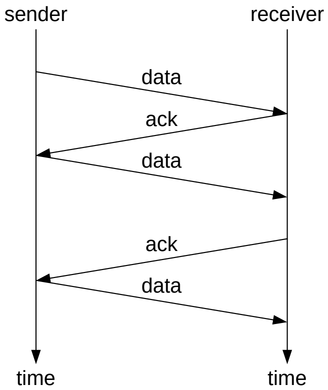
- error-free channel
- finite buffer
- slow receiver
- (half-) duplex

Solution

- feedback, here acknowledgements
- stop-and-wait flow control

But, additionally

- error-prone channel



Protocol 3: Stop-and-wait ARQ

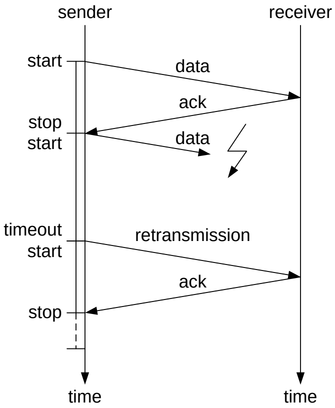
Assumptions: error-prone (half-) duplex channel

Solution

- positive acknowledgement with retransmit (PAR)
- also called automatic repeat request (ARQ)
- timeout triggers retransmissions
  - timeout too long ⇒ unnecessary waiting
  - timeout too short ⇒ needless retransmissions

But, additionally

- acks may be lost



Protocol 3: Stop-and-wait ARQ

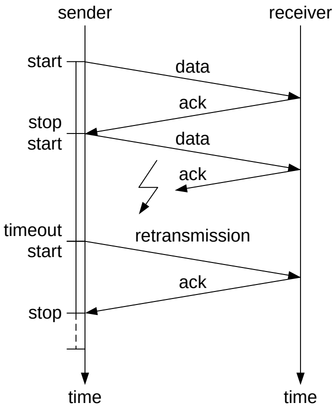
Assumptions: error-prone (half-) duplex channel

Problem

- loss of acknowledgements causes duplicated frames
- receiver accepts duplicates as if they were new data

Solution

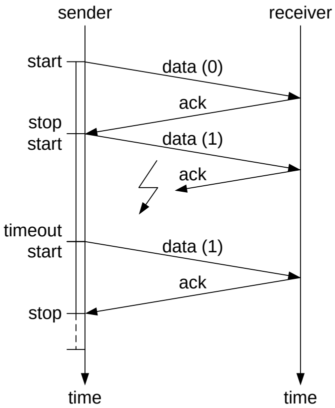
- need to mark retransmissions
- numbering of frames



Assumptions: error-prone (half-) duplex channel

Solution

- sequence numbers
- each frame is assigned a consecutive SeqNo
- retransmissions have to use the same SeqNo as the original transmission
- receiver discards duplicates
- range $[0, \ldots k-1]$ mod k where $k = 2^n$
- stop-and-wait [0,1] mod 2



Protocol 5: Stop-and-wait ARQ, SeqNo and ACK/NAK

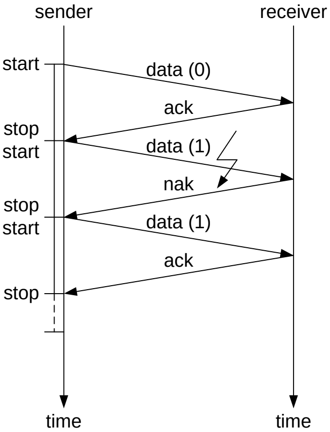
So far no differentiation between

- faulty frame
- missing frame

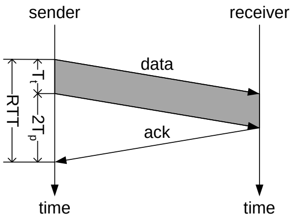
need to wait for timeout in any case.

Active error control

- positive acks
- negative naks



Performance of stop-and-wait ARQ



- transmission time T_t = frame length l / capacity C
- propagation delay T_p = distance / speed of light
- channel utilization $T_t/(T_t + 2T_p)$
- e.g. $T_p = 50$ ms trans Atlantic, $l = 1500$ Byte, $C = 2$ Mb/s gives $T_t = 6$ ms and a utilization of 5 %
- e.g. $T_p = 0.1$ ms, $l = 1500$ Byte, $C = 54$ Mb/s gives $T_t = 0.22$ ms, and a utilization of roughly 50 %

Effective data rate

$$\text{effective data rate } R = \frac{\text{number of bits accepted by receiver}}{\text{total time for delivery}}$$

- frame length l
- header/trailer length h
- payload $l - h$
- capacity C
- transmission time $T_t = l/C$
- propagation delay T_p

$$R = \frac{l - h}{\frac{l}{C} + 2T_p}$$

$$\left(\text{formally w. ergodicity: } R = \lim_{j \to \infty} \frac{j(l-h)}{\sum_{i=1}^{j} t_i} = \lim_{j \to \infty} \frac{l-h}{\frac{1}{j}\sum_{i=1}^{j} t_i} = \frac{l-h}{\mathsf{E}[t]}\right)$$

Effective data rate with frame errors

Consider transmission errors: Denote N the average number of transmission attempts until a frame is accepted (timeout $= T_t + 2T_p$).

$$R = \frac{l-h}{N\left(\frac{l}{C} + 2T_p\right)}$$

To derive N

- let P be the probability that the transmission of a frame is erroneous
- assume k frames shall be transmitted
- assume that errors in different frames are independent
- of the k frames transmitted $(1-P)k$ are on average received correctly, requiring Pk retransmissions

Effective data rate with frame errors

Consider transmission errors: Denote N the average number of transmission attempts until a frame is accepted (timeout $= T_t + 2T_p$).

$$R = \frac{l-h}{N\left(\frac{l}{C} + 2T_p\right)}$$

To derive N

- let P be the probability that the transmission of a frame is erroneous
- assume k frames shall be transmitted
- assume that errors in different frames are independent
- of the k frames transmitted $(1-P)k$ are on average received correctly, requiring Pk retransmissions
- of the Pk retransmissions $(1-P)Pk$ are on average received correctly, requiring another P^2k retransmissions
- continuing $(1+P+P^2+\dots)k$ transmissions are needed

Effective data rate with frame errors

The sum $k(1+P+P^2+\dots) = k\sum_{i=0}^{\infty} P^i$ is a geometric sum that can be easily solved (the trick is to solve $(1-P)\sum_{i=0}^{j} P^i$ first).

Effective data rate with frame errors

The sum $k(1 + P + P^2 + \dots) = k \sum_{i=0}^{\infty} P^i$ is a geometric sum that can be easily solved (the trick is to solve $(1 - P) \sum_{i=0}^{j} P^i$ first).

We can also find the solution from the following argument

- from n transmissions $(1 - P)n$ frames are received correctly
- hence, if we want k frames to be received correctly we equate

$$k = (1 - P)n$$

and solve for the number of transmissions required yielding

$$n = k/(1 - P)$$

Effective data rate with frame errors

The sum $k(1+P+P^2+\dots) = k\sum_{i=0}^{\infty} P^i$ is a geometric sum that can be easily solved (the trick is to solve $(1-P)\sum_{i=0}^{j} P^i$ first).

We can also find the solution from the following argument

- from n transmissions $(1-P)n$ frames are received correctly
- hence, if we want k frames to be received correctly we equate

  $$k = (1-P)n$$

  and solve for the number of transmissions required yielding

  $$n = k/(1-P)$$

Normalizing n by k the average number of transmissions required until one frame is received correctly is

$$N = \frac{n}{k} = \frac{1}{1-P}$$

Effective data rate with bit errors

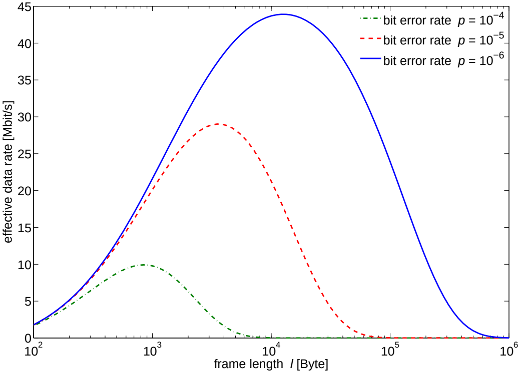
Computation of the frame error probability P

- assume bit errors occur with probability p
- hence a bit is transmitted correctly with probability $1 - p$
- we assume that bit errors are statistically independent
- the probability that an entire frame of l bits is transmitted correctly becomes $(1 - p)^l$
- the frame error probability follows as $P = 1 - (1 - p)^l$

Putting all pieces together the effective data rate becomes

$$R = \frac{(l - h)(1 - p)^l}{\frac{l}{C} + 2T_p}$$

Effective data rate with bit errors



$$R = \frac{(l-h)(1-p)^l}{\frac{l}{C} + 2T_p}$$

- $C = 54$ Mbit/s
- $T_p = 0.1$ ms
- $h = 52$ Byte

Pipelining

How to improve the performance of stop-and-wait ARQ?

- if the frame error rate is high
  $\Rightarrow$ reduce the frame size
- if the waiting time is large compared to the transmission time
  $\Rightarrow$ increase the frame size

Cannot do both at the same time!

Pipelining

How to improve the performance of stop-and-wait ARQ?

- if the frame error rate is high
  $\Rightarrow$ reduce the frame size
- if the waiting time is large compared to the transmission time
  $\Rightarrow$ increase the frame size

Cannot do both at the same time!

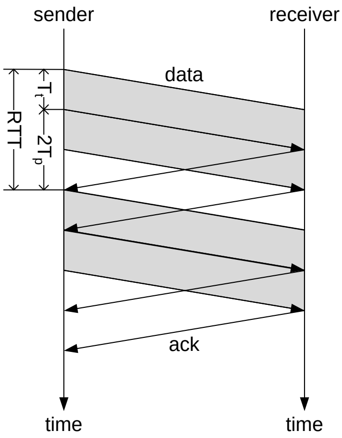
Solution: Pipelining

- several frames may be in flight simultaneously
- at any time at most n of the frames transmitted may be unacknowledged (can also be used for flow control)
- stop-and-wait is the special case $n = 1$
- ideally, there is a continuous flow of frames and acknowledgements such that no waiting occurs at all
- each frame can be small to account for high bit error rates

Pipelining

The sender can transmit n frames at line speed before waiting for acks. Consider the figure, which n achieves full utilization?

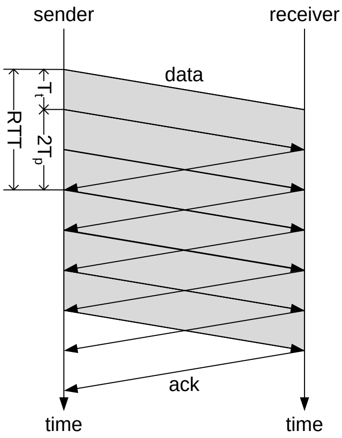
- $n = 2$: utilization 2/3 still limited by T_p



Pipelining

The sender can transmit n frames at line speed before waiting for acks. Consider the figure, which n achieves full utilization?

- $n = 2$: utilization 2/3
  still limited by T_p
- $n = 3$: utilization 1
  $n\, l/C$ matches RTT
- $n > 3$: utilization 1
  limited by the capacity C



Bandwidth delay product

A continuous flow of data (full utilization) is achieved if

$$n\, l \geq C \cdot RTT$$

The term $C \cdot RTT$ is called the bandwidth delay product.

- C and RTT are external parameters of the physical layer connection (from point of view of the data link protocol)
  - the RTT consists (mainly) of the transmission time T_t and two propagation delays T_p
  - T_p is distance divided by propagation speed
  - propagation occurs with speed of light v_l where depending on the medium $v_l \approx 200\,000 - 300\,000$ km/s
- n and l are internal parameters of the data link protocol that can be adapted to optimize the utilization

Number of bits in flight

The term $C \cdot T_p$ denotes the number of bits that are in flight simultaneously during a (serial) transmission at full utilization.

To see this:

- the duration of a bit sent at rate C is $T_b = 1/C$
- the bit propagates at speed of light v_l, i.e. it is spread over (occupies) a section of $s_b = v_l T_b = v_l/C$
  - on a 10 Mb/s Ethernet bits are 20 meters long

Number of bits in flight

The term $C \cdot T_p$ denotes the number of bits that are in flight simultaneously during a (serial) transmission at full utilization.

To see this:

- the duration of a bit sent at rate C is $T_b = 1/C$
- the bit propagates at speed of light v_l, i.e. it is spread over (occupies) a section of $s_b = v_l T_b = v_l/C$
  - on a 10 Mb/s Ethernet bits are 20 meters long
- it follows that a cable of length s_c has room for s_c/s_b bits
- the propagation delay is $T_p = s_c/v_l$ respectively $s_c = v_l T_p$
- there can be $C \cdot T_p$ bits in flight on the cable
  - on a 10 meter cable used at 100 Mb/s is room for 5 bits
  - on a 10 000 km trans Atlantic fibre used at 10 Gb/s is room for 500 000 000 bits = 62.5 MByte

Outline

Redundant repair frames

Automatic repeat request
- Stop-and-wait
- Pipelining
- Sliding window protocols
  - Go-back-n
  - Selective repeat

Data link layer protocols
- HDLC

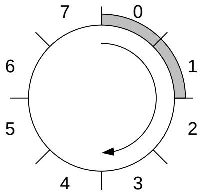
Frame numbering

Sender

- frames are numbered in ascending order
- sequence numbers SeqNo $\in [0, \ldots, k-1]$
- SeqNo is incremented modulo k

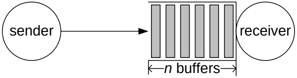
Receiver

- frames received in sequence are acknowledged and passed to layer 3
- acknowledgements contain an AckNo $\in [0, \ldots, k-1]$
- cumulative acknowledgements acknowledge all frames up to AckNo



Flow control with pipelining

With pipelining, how to ensure that the receiver is not flooded?



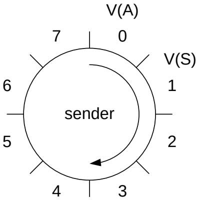
Solution: Sliding window protocol

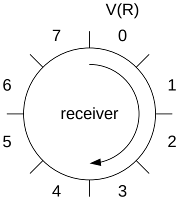
- receive window: number of receiver buffers n, advertised to sender
- send window: number of frames transmitted but not yet acknowledged

Condition: send window $\leq$ receive window

State variables

- sender: send window V(S)$-$V(A)
  - V(A): acknowledge state variable
    - smallest not acknowledged SeqNo
    - increment on receipt of an ack
  - V(S): send state variable
    - next SeqNo to be sent
    - increment when sending a frame
    - largest SeqNo allowed: V(A)$+n-1$
- receiver: receive window n buffers
  - V(R): receive state variable
    - next SeqNo to be received in sequence
    - increment on receipt of a frame
    - largest SeqNo allowed: V(R)$+n-1$

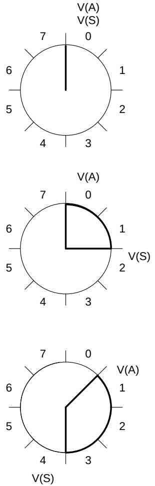




Send window examples

Window size $n = 3$

1. V(S) $=$ V(A)
   - no frames stored, i.e. all transmitted frames have been ack'ed
   - sender may send up to three new frames
2. V(S) $=$ V(A)$+$2
   - two frames stored, i.e. two transmitted frames have not been ack'ed
   - sender may send one new frame
3. V(S) $=$ V(A)$+$3
   - three frames stored
   - sender is not permitted to send further frames



Error control with pipelining

Expected order

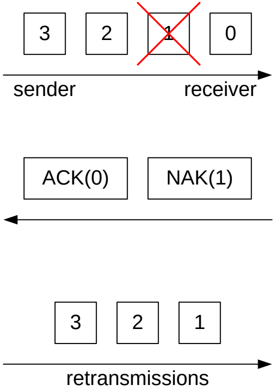
- window size $n = 1$ (stop-and-wait)
  - transmission/reception generally in sequence
- window size $n > 1$, two options
  - go-back-n
    - reception only in sequence
    - out-of sequence frames are discarded
  - selective repeat
    - no requirement for in-sequence reception
    - only limited by the receive window

Receiver

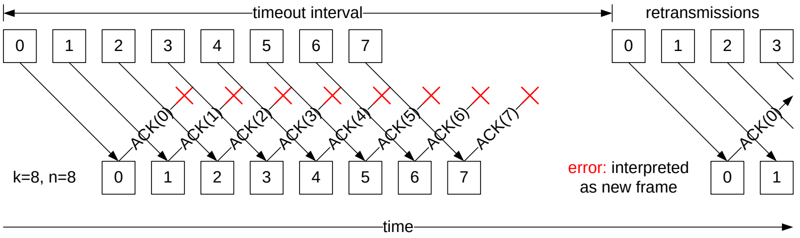
- accepts frames only in order of increasing SeqNo
- if a frame is erroneous or missing all subsequent frames are discarded

Sender

- transmits frames in sequence
- if a frame with a certain SeqNo is retransmitted all subsequent frames are retransmitted, too

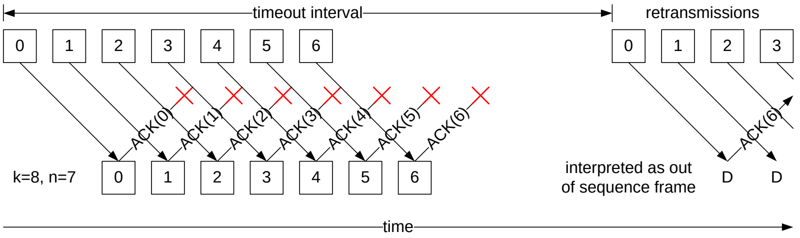


Maximum window size



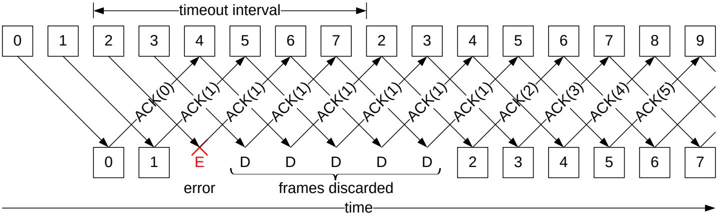
- number of distinct sequence numbers k
- sequence numbers $[0, \dots, k-1]$ modulo k
- window size n
- go-back-n: frames are only accepted in sequence

Maximum window size

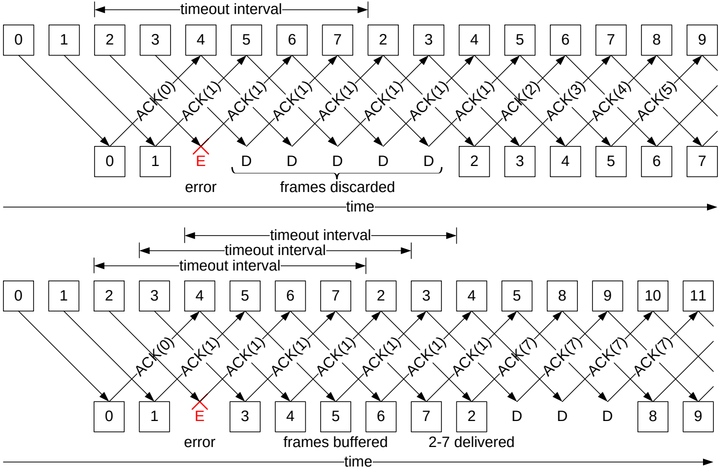


- number of distinct sequence numbers k
- sequence numbers $[0, \dots, k-1]$ modulo k
- window size n
- go-back-n: frames are only accepted in sequence
- need $n \leq k - 1$ to distinguish retransmissions

Optimization



Optimization



Selective repeat

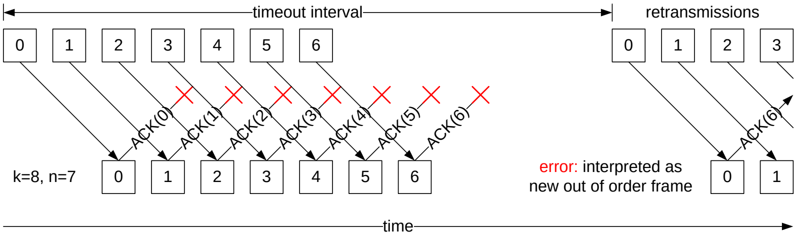
Receiver

- stores all correctly received frames
  - that fall into the receive window
  - regardless whether in sequence or not
- hands over data to the network layer only in sequence
- may involve extensive buffering of received frames while waiting for missing frames

Sender

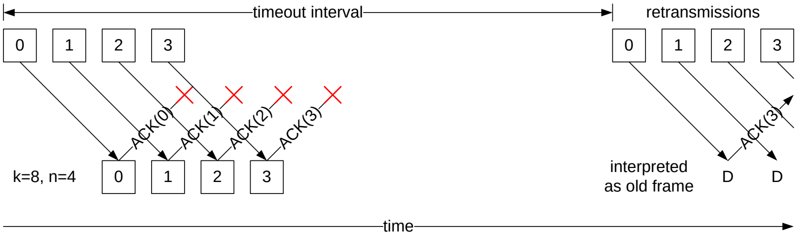
- seeks to retransmit only erroneous frames (as opposed to all frames starting at the erroneous one)
- notification of missing frames
  - timeout, one timer per frame
  - optional stalled positive acknowledgements
  - optional negative acknowledgements

Maximum window size



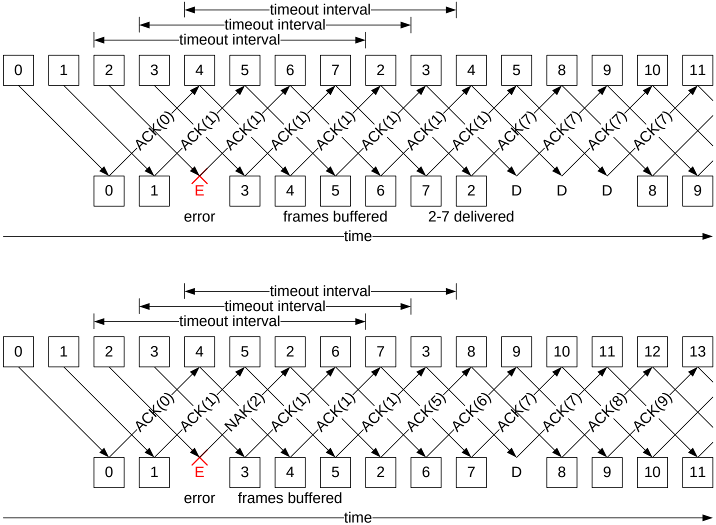
- number of distinct sequence numbers k
- sequence numbers $[0, \ldots, k-1]$ modulo k
- window size n
- out of sequence frames are buffered

Maximum window size



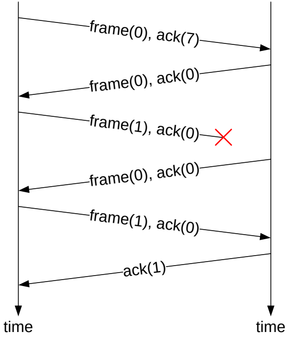
- number of distinct sequence numbers k
- sequence numbers $[0,\ldots,k-1]$ modulo k
- window size n
- out of sequence frames are buffered
- need $n \leq k/2$ to distinguish retransmissions

Selective repeat with negative acknowledgements



Duplex: piggyback acknowledgements

- adds acks to data frames
- frames have
  - SeqNo
  - AckNo
- need to determine when to send standalone acks
  - if a frame is received start a timer
  - if there is data for the reverse direction add the piggyback ack
  - otherwise send an ack once the timer expires
  - ack timer must be smaller than the retransmit timer

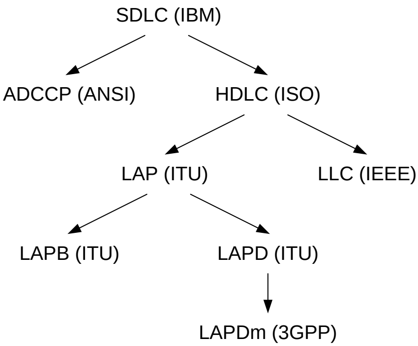


Redundant repair frames

Automatic repeat request
- Stop-and-wait
- Pipelining
- Sliding window protocols
  - Go-back-n
  - Selective repeat

Data link layer protocols
- HDLC

HDLC protocol family



Synchronous Data Link Control
Advanced Data Communication Control Procedure
High-Level Data Link Control
Link Access Procedure
Logical Link Control
Link Access Procedure Balanced
Link Access Procedure D signalling channel (ISDN)
Link Access Procedure D modified signalling channel (GSM)

"The nice thing about standards is that you have so many to choose from" [Tanenbaum]. The HDLC protocol family consists of

- bit oriented protocols
- with confusing little differences

Characteristics

HDLC protocol

- full duplex
- bit-oriented with bit stuffing (after five 1s a 0 is stuffed)

Frame format

- flag: 01111110
- address: used if not point-to-point
- control: sequence number, acknowledgement
- data: arbitrary number of payload bytes
- checksum: cyclic redundancy check

8 bit	8 bit	8 bit	m·8 bit	16 bit	8 bit
01111110	address	control	data	checksum	01111110

Types of frames

The control field specifies three different types of frames

- information frames

1 bit	3 bit	1 bit	3 bit
0	Seq	P/F	Next

- supervisory frames

2 bit		2 bit	1 bit	3 bit
1	0	Type	P/F	Next

- unnumbered frames

2 bit		2 bit	1 bit	3 bit
1	1	Type	P/F	Modifier

Information frames

1 bit	3 bit	1 bit	3 bit
0	Seq	P/F	Next

Sliding window with 3 bit sequence number
(extended version with 7 bit)

- Seq: sequence number of this frame
- Next: piggyback acknowledgement
  - specifies the next frame expected
  - last frame received in sequence plus one
- P/F: Poll/Final
  - master requests data transmission from a slave by setting P
  - the slave sends data with P and the final data frame with F

Supervisory frames

2 bit		2 bit	1 bit	3 bit
1	0	Type	P/F	Next

Type

- 00: RECEIVE READY
  - acknowledgement (not piggyback)
  - Next: next frame expected
- 01: REJECT
  - negative acknowledgement, go-back-n
  - Next: first frame to be retransmitted
- 10: RECEIVE NOT READY
  - stop command to make communication peer pause
  - start again after RECEIVE READY, REJECT, ...
- 11: SELECTIVE REJECT
  - negative acknowledgement, selective repeat
  - Next: frame to be retransmitted

Unnumbered frames

2 bit		2 bit	1 bit	3 bit
1	1	Type	P/F	Modifier

Type, modifier

- SNRM: Set Normal Response Mode
  - reports availability, initializes unbalanced master/slave mode
- SABM: Set Asynchronous Balanced Mode
  - reports availability, initializes balanced mode, i.e. equal communication peers
- DISC: Disconnect
  - reports unavailability
- UA: Unnumbered Acknowledgement
  - acknowledgement for unnumbered frames
  - only one unnumbered frame may be outstanding at any time
- FRMR: Frame Reject
  - report frames with incorrect semantics