3. Layer 2 - The Data Link Layer

Read [Tanenbaum96] Ch. 3. You can skim Section 3.5.2.

This layer provides:

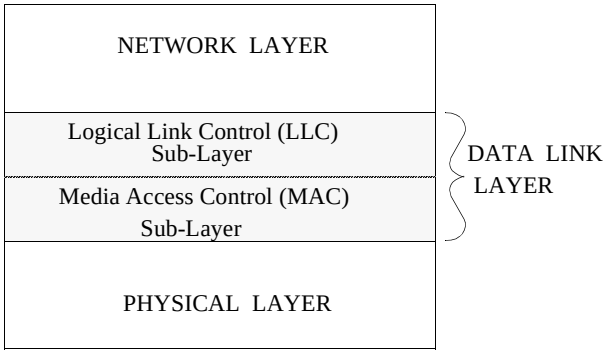
- transmission of basic data frames over,
- and control of,

a single hop link. This section only applies to non-multi-hop transmissions. i.e. point-to-point transmission, or semi-broadcast (but not store + forward) networks.

There are two sub-layers to this layer:

a) The top one does 'logical link control' and thus manages single-hop framing, errors, flow, and half-duplex turn control.

b) The bottom one manages media access control and is only present on broadcast (shared) media. It manages contention and multi-point turn control.



We will examine the Logical Link Control (top) sub-layer in this section of the course, so that we will quickly gain an understanding the exchange of basic frames of data. Much later in the course, after looking at this and basic store-and-forward networks, will we finally look at the complexities of networks based on shared media access.

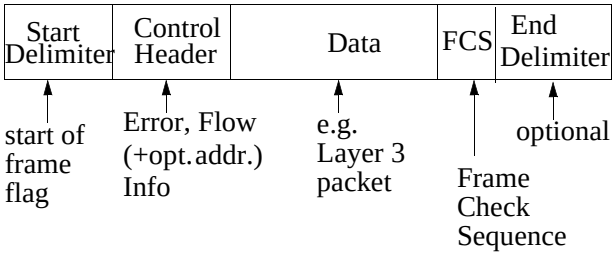
The basic LLC topics that need to be covered are:

1) framing
2) error detection and control (and it's subsidiary problem of single-hop re-sequencing)
3) flow control

TABLE OF CONTENTS - is located at end of the section.

3.1 Framing

A 'frame' is the basic unit of data at ISO Level 2.



Frames can be character-oriented, or bit-oriented.

3.1.1 Character-Oriented Framing

If a protocol will only ever carry traffic from one kind of computer to another computer of the same type (i.e. they use the same number of bits/character and the same character set), then character-oriented framing is sufficient.

In character-oriented frames, the frame length (in bits) is divisible by the character length (7 for ASCII, 8 for EBCDIC). i.e. the standard unit of transmission is a character.

The flag and some control characters are reserved special (non printing) characters ***in the character set***. (This make it impossible for an ASCII device to communicate with an EBCDIC device).

Since in asynchronous transmission a stream of characters is received, and in synchronous transmission a stream of bits is received, we must be able to determine where the beginning and end of a frame occurs. Frames must be ***delimited*** (i.e. have their starts and ends marked somehow).

The Binary Synchronous Communication (BSC) standard from IBM delimits frames by starting each frame with the two character sequence "syn syn" (syn = synchronous idle= 0010110 ASCII). When no data is being transmitted, the syn character is transmitted continuously.

The receiver, on power-up, can then search for and can determine character alignment within a frame in an already running stream of characters.

This is a prime requirement: that communicating devices be able to re-start a dialog after one or both lose synchronization for any of a number of reasons (e.g. power failure, or bit error that destroyed a "syn").

3.1.2 Character-Oriented Transparency

Note that if you were transmitting raw binary data (say floating point numbers), such data may contain what appears to be a "syn, syn" character pair. This would seriously confuse the framing mechanism just discussed.

We can protect the receiver from reacting to SYNs that are actually data, by inserting a "DLE" (Data Link Escape) character in front of any data SYNs. The receiver then strips any DLEs out of the received character stream and assumes the immediately following character is data, not a control byte(s).

Q: What if there is a "DLE" in the data?

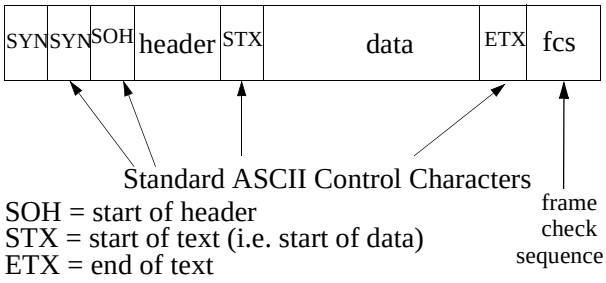
A: Send "DLE DLE" and have receiver discard 1st one and properly consider the second one data.

This is called ***character stuffing***, and has the side effect of changing the length of the frame irregularly.

(hand-drawn diagram next)

(Example of Char Stuffing diagram)

BSC Frame Format



Notes:

a) If noise causes a bit error, synchronization may be totally lost and the receiver will have to go back to hunting for "syn syn" flags. (This is **not** very easy if using synchronous transmission as a "syn syn" can occur in the middle of other characters.

e.g. "X" "X" "X"
1011000 1011000 1011000
SYN SYN

You can see how BSC, even through designed for synchronous communication, would actually work better for asynchronous transmission.)

b) If you are transmitting 7 bit ASCII using BSC, how would you transmit a 32 bit floating point number? Four 7 bit "characters" plus one extra 4 bit value? How would you transmit EBCDIC (which has more than 128 characters) over an ASCII BSC channel?

c) One way to address problems in b) above, or the ETX delimiting transparency problem in general, is to not delimit the end of the data field. Instead the header information contains a field which specifies the length of the text/data portion of the frame. This works well, unless a bit in the length field is received in error. Then, synchronization can be completely lost along with the location of the FCS. (We don't like putting FCS before the text. Why?). Note: DECnet uses length in header I think.

3.1.3 Bit-Oriented Framing

Bit-oriented protocols are based on the concept of needing to send a string of bits (not characters). Bit-oriented protocols:

- have the advantage of being independent of character codes (translation between character codes is now recognized as being a presentation layer function).
- are also independent of character size (5,6,7, or 8 bits)
- achieve transparency using ***bit stuffing***.

The delimiter flag is the sequence 01111110. This sequence is never allowed within frames, only before, between, or after the contents of frames. A frame is typically composed of 3 parts:

1) a header which is either a fixed, pre-planned length or which uses length extension bits.
2) a body which contains the data to be transferred.
3) and the frame check sequence, which normally occupies the 32 bits just prior to the ending delimiter flag.

3.1.4 Bit-Oriented Transparency

This is accomplished by stuffing a 0 bit after every occurrence of five 1's in a row **of data**. This prevents 6 ones in a row which would appear like the center part of a delimiter flag. The stuffing is done even if the next bit is a 0 anyway, as this is necessary to allow the receiver to correctly unstuff.

The receiver is constantly scanning the bit stream for 5 1's. If 5 1's are followed by a 0, the 0 is discarded. If the 5 1's are preceded by a 0 and followed by 10, it is regarded as a flag. If 5 1's are followed by 11, it is an error (as this should never happen).

The bit stuffing method is completely transparent. It doesn't matter if binary data such a 2's complement integers, or executable programs are transmitted; they will not cause a delimiter flag to be falsely detected. The data portion of a frame conveys only the desired info in whatever communication format (header, data, frame check sequence) is agreed on.

Notes:

a) Frames need only be separated by a single flag.

b) If a bit error occurs within a flag or causes a false flag, synchronization will be lost, but bit-stuffing allows reliable re-synchronization ***at the very next flag***.

c) In b) above, as in character-oriented protocols, loss of sync could result in a frame being split by a false flag, or 2 frames merging into one by loss of a flag. This will be detected by the frame check sequence and corrected (as discussed later) by the layer 2 logical link protocol.

d) Bit stuffing works fine over asynchronous (character-oriented) links if considered a bit stream. The only problem is frames must somehow be padded out to an integral # of bytes. A methodology must be adopted to remove the padded bits.

3.1.5 Code Violation Framing

There is one other way of frame delimitation. A few systems use physical layer code violations to delimit a frame. e.g.

- A Manchester symbol with no transition at the symbol center could be transmitted and have the meaning of a delimiter.
- Similarly, certain 4B/5B codes are unused as data symbols can are used as delimiter control symbols.

In summary, framing is accomplished by one of 3 possible methods:

1) Delimiter character or bit strings (with stuffing for transparency).
2) Using a character count in each header.
3) Physical layer code violations.

3.2 The Data Link Control Concept

Data sent to a destination needs to be "acknowledged" by some mechanism which assures the sender that the data frame:

1) Got to its destination,

2) Got there correctly, and

3) Got written to disk (or otherwise removed from the layer 2 task's RAM) so that there is room to receive the next frame.

The first two of these items constitute the concept of error control, while the third is flow control.

These 3 elements of data link control are handled jointly by 2 mechanisms:

a) By numbering each frame with an ID called a "sequence number" the receiver can then send return packets suggesting whether that specific packet was received, received correctly, and whether the receiver is ready to receive another.
There are two types of return messages:
- an acknowledgment ("ACK"), or
- a negative acknowledge ("NACK") which suggests the frame was not received correctly.

b) A time-out scheme, wherein if the sender gets no response within a certain period of time, it will presume the transmitted frame was lost in its entirety (due to severe channel interference) and automatically try to send it again. If after re-trying several times unsuccessfully, the sender will give up and report a "link failure" to its user!

3.2.1 Automatic Repeat Request (ARQ)

Automatic Repeat Request (ARQ) is the generic name given to any protocol system which will automatically trigger a re-transmission of damaged or completely lost frames, so that the user at the destination will not detect that the channel sometimes induces errors in the data.

The ARQ technique of the sender waiting for an ACK (acknowledge) or NACK (negative acknowledge) requires that there exist *a reverse direction channel* upon which the acknowledgment message can be transmitted (note: pagers do not have a reverse channel; they handle errors in another way).

ARQ is used to control errors. We will later see that a consequence of error control is that sometimes frames will appear to arrive out of order. Thus ARQ must be also have a frame re-sequencing capability.

The major strategies that can be taken to implement ARQ are:

a) Stop + Wait ARQ

b) Go-Back-n ARQ

c) Sliding window ARQ

d) Selective repeat ARQ

3.3 Basic Stop + Wait ARQ

Transmit algorithm:

1) Set retransmission count to zero.
2) Send frame.
3) Start timer. Wait for acknowledgment (ACK) from the receiver that the frame has been received correctly, and has been stored so that the receiver is able to accept the next one.
4) If timer runs out with no ACK having been received (a 'time-out'), then re-transmit the frame, and increment the retransmission count. Goto step 2 above.
5) If retransmission count > 10, give up and report a "link failure" to user.

Receive Algorithm:

1) Wait for two flags separated by some data info.
2) If frame has bit errors, discard it.
3) Handle or store this (valid) frame. Then send an "ack" message back to the sender.

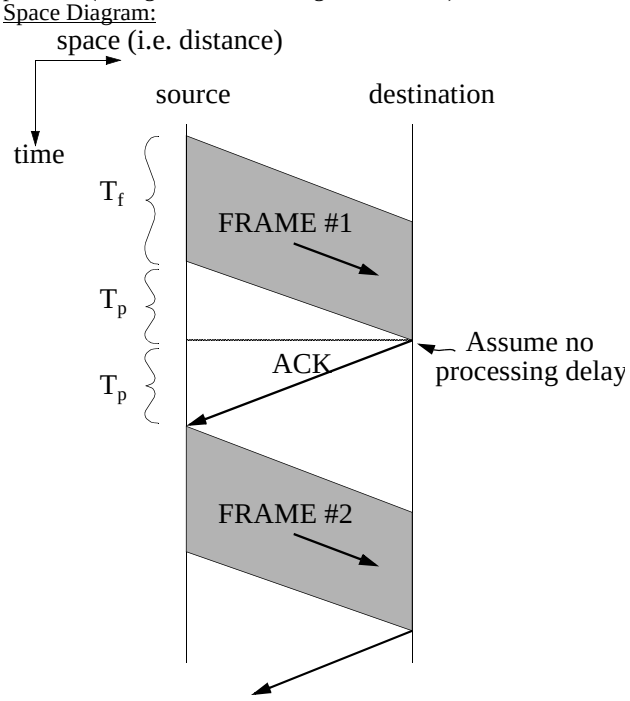
3.3.1 Link Utilization for Stop-and-Wait

Now we will make our first foray into the study of data communication performance.

If we have a 9600 bps link and a large file to send, we would like to be able to make the most efficient use of the available capacity (i.e. transmission time).

Unfortunately, for flow and error control reasons, we need to send data in packets. And if using stop-and-wait, we cannot achieve a full utilization of U=1.0

We will analyze the net throughput of the stop-and-wait protocol (to begin with, assuming no bit errors) with a Time Space Diagram:



As with all analyses in this course, we analytically evaluate the throughput utilization (i.e. protocol efficiency) by determining the fraction of time a transmitter is usefully transmitting (cf. not 'waiting').

Let:

T_f = Frame Transmission Time.

= time for source to "spit" a frame of bits out.

This is another rate problem. It is different in that the rate is ***not*** a spacial speed, but a ***bit transmission rate***. And the quantity is not a distance, but the number of bits in a frame needing to be transmitted.

We know that time = quantity/rate. Therefore, to calculate T_f we will need to define:

L = length of frame measured in bits.

R = (gross) bit transmission rate in bits per second.

Therefore:

$T_f = L/R$

Also, let:

T_p = Propagation Time for the data link.

= time for an edge of a bit (e.g. first or last)

to propagate from source to destination.

This duration can be determined from the simple rate problem discussed in the Physical Layer.

Define:

D = the distance between source and destination.

V = velocity (i.e. speed) of propagation of the signal energy down the physical channel.

Therefore:

$T_p = D / V$

Note: If D is in meters, then V must be in m/s.

The above diagram illustrates the cyclic nature of transmitting a large file using the stop and wait protocol with small frames. The fraction of time usefully transmitting is:

Utilization U = time to transmit a frame / total time before can start next frame

U = T_f / (duration of stop-and-wait cycle)

With the simplifying assumption that the length of an acknowledgment is infinitely short, we can see that:

$U = T_f / (T_f + 2T_p)$

$= 1 / (1 + 2T_p/T_f)$

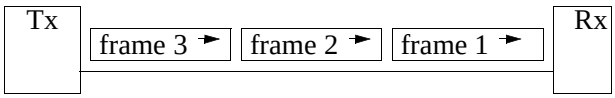
Now define the link timing ratio to be:

$$\mathbf{a = T_p / T_f.}$$

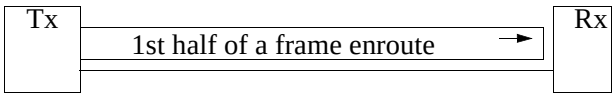
Then: $$\mathbf{U_{s+w} = 1 / (1 + 2a)}$$ } Memorize!

The link timing ratio "a" is a measure of whether the first bit of a frame arrives at the destination before the last bit of the frame leaves the source, or not. (c.f. train analogy.)

e.g. a=3



e.g. a=0.5



When the energy of a frame is enroute to its destination, it is spread out in distance along the route. If its length is <D, then a>1. In fact, a = the number of frames that will fit along the route.

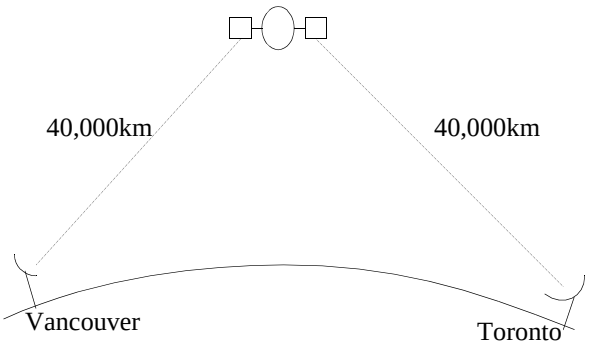
Example Calculations

1) Consider using a 9600 bps modem between your residence and SFU. Assume the distance is 10 km., the speed of propagation in the wire is 2/3rds of the speed of light, and you are transferring 500 byte frames.
   Note: You must memorize the speed of light ($3x10^8$ m/s) and the fact that in metal wires, the speed is approximately 2/3rds of that.
   $T_p = D/V = \frac{10 \text{ km x } 1000 \text{ m/km}}{.66 \text{ x } 3x10^8 \text{ m/s}} = .00005 \text{ s.} = 50 \mu s.$
   $T_f = L/R = \frac{500 \text{ bytes x } 8 \text{ bits/byte}}{9600 \text{ bps}} = \frac{4000 \text{ bits}}{9600 \text{ bps}} = .417 s.$

Therefore:
$a = T_p/T_f = \frac{5x10^{-5} \text{ s}}{4.17 \text{ x } 10^{-1} \text{ s}} = 1.2 \text{ x } 10^{-4} << 1$
Now we can calculate:
$U_{s+w} = 1/(1+2a) = 1/(1+.00024) = .9997 = 99.97\%$

Thus your throughput would be .9997 x 9600 bps = 9597 bps. Your throughput is making almost complete use of your modem capacity, as there is only .03% of the time wasted 'waiting'.

2) 56 kbps satellite channel between Vancouver and Toronto. Assume a 500 byte frame size.



$T_p = D/V = \frac{2 \text{ x } 40{,}000\text{km x } 1000 \text{ m/km}}{3 \text{ x } 10^8 \text{ m/s}}$

$= \frac{8 \text{ x } 10^7 \text{ m}}{3 \text{ x } 10^8 \text{ m/s}} = .27 \text{ seconds}$

$T_f = L/R = \frac{500 \text{ bytes x } 8 \text{ bits/byte}}{56 \text{ kbps}}$

$= (4000 \text{ bits}) / (5.6 \text{ x } 10^4 \text{ bps}) = .071 \text{ seconds}$

Therefore:

$a = T_p/T_f = .27/.017 = 3.8$ (which is >>1!)

So:

$U_{s+w} = 1/(1+2a) = 1/(1+2x3.8) = .12 = 12\%$ efficiency

This is terrible! How would you like to rent a 56 kbps satellite channel for several thousand dollars per month, yet only be able to get 12% x 56 = 6.5 kbps throughput?

Obviously, on a<<1 links, stop-and-wait has good utilization efficiency because the transmitter spends most of its time transmitting. T_p (the time you have to wait for the last bit of a frame to get completely to its destination, plus the time for the ACK to return) is small compared to the frame time, so having the source wait for ACKs and NACK before proceeding with the next frame takes little extra time relative T_f.

But for a>1, we must look for better ways to do flow (and error) control. With stop-and-wait we cannot transmit 3 frames and have them all packed enroute at once. We only transmit the second one once the first has arrived and the ACK has returned to the originating station. Later we will look at improved protocols that are more suitable than stop-and-wait for systems where a>>1.

3.3.2 Stop-and-Wait with Added NACKs

In the receive algorithm for the simple stop-and-wait protocol listed several pages back, if a damaged frame is received, the receiver does nothing but discard the frame (it does not send a NACK). The time-out duration is often quite long to take into account a possibly slow ACK response of a busy, multi-tasking computer at the destination (c.f. loose vs. tight timeout). This is inefficient for a sender on a noisy channel (where frames are frequently received in error), because of time wasted waiting for acknowledgments that will never come.

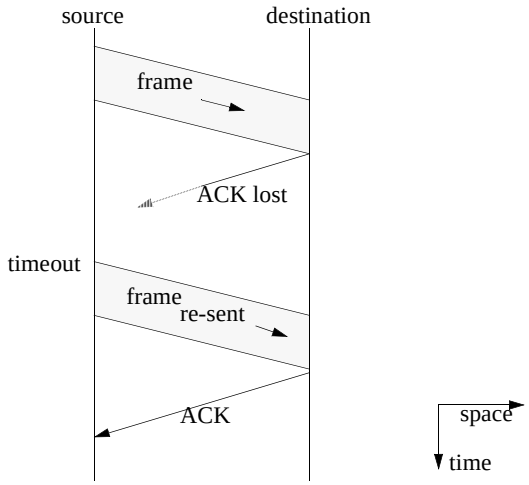
Stop-and-wait therefore almost always includes a negative acknowledgment (NACK) capability. If the receiver *realizes* it has received a damaged frame, it can promptly transmit a NACK message back to the original sender. This makes the delay before the originator re-sends a data frame as small as possible.

Note: In this case, a send timer is still needed in case either:

i) The original frame was totally lost.
ii) The returning ack was lost or garbled.
iii) A returning nack was lost or garbled.

3.3.3 Handling Lost ACKs

The preceding stop + wait protocol has a number of fatal flaws! First, consider the following interaction:



Due to a lost or garbled ACK, the source thinks the destination never got the data, and thus re-sends it. But the destination thinks it got and acknowledged the first transmission, so it accepts the next frame as a valid next data frame. Unfortunately, the destination user gets two copies of the message (which could debit your bank account twice!).

Conclusion: The second frame must be detected as a duplicate and discarded by the destination's Layer 2 algorithm. The only way to do this is to label frames with a ***sequence number***. A frame number is put in the data link header, where the receiver

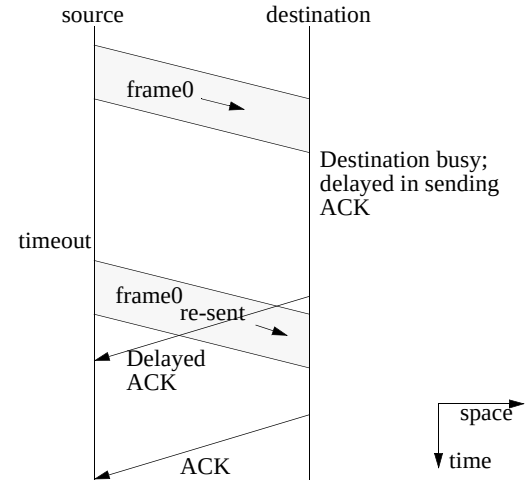
can check to see if it just, because of some kind of communications error, received another frame with the same number as the last.

Note that because of the 'one at a time' nature of stop-and-wait, modulo-2 numbering (i.e. alternating 0 and 1 frame labels) is adequate.

This handles any occurrence of whole frames lost or damaged (I consider an ack to be a return frame).

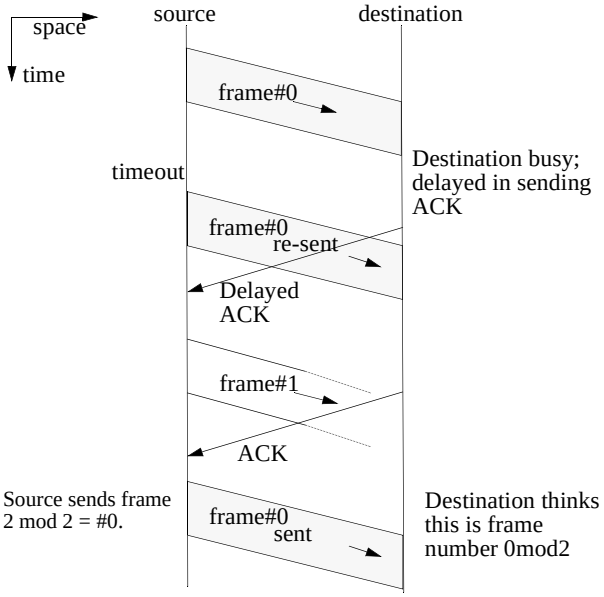
3.3.4 Handling Duplicate ACKs

What if the receiver is a multi-tasking computer and is delayed momentarily by another task before sending an ACK. Assume a full duplex channel.



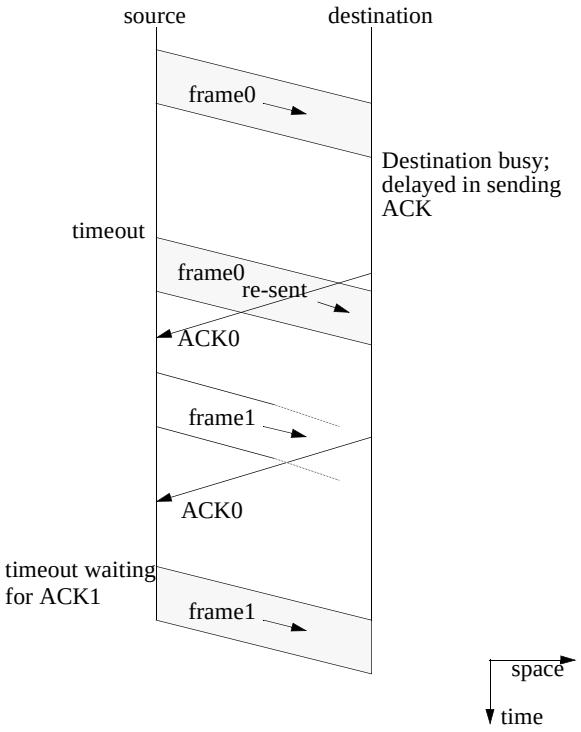
This situation is handled OK, *if the source is programmed to ignore/throw away the second ACK*.

But, consider the following:



The source will (wrongly) assume the first frame0 got lost and re-send frame0. The Delayed ACK will be taken as the ACK to the re-sent frame0. The source will then proceed to send frame1. Since the second ACK arrives after frame 1 is sent, the source will (wrongly) figure that the second ACK is confirmation that frame1 was received. But in fact frame 1 was lost for some reason (say radio interference). The source will go on to send frame 2, not realizing frame 1 (crediting your bank account with $1M) was lost! Neither station detects error!

Conclusion: We must number ACK's (modulo-2 numbering is adequate for stop-and-wait). Then the system would work as follows:



Note: The destination ***must*** send the second ACK0 (regarding frame0), as it *could be* that the reason the source is re-sending frame0 is that the destination's first ACK0 never got through. If the destination didn't send the second ACK0, and the situation was not as shown, but that alternatively the destination's first ACK0 never got thru, then the receiver could, in that alternate situation, be stuck forever waiting for an ACK0. The destination must be programmed to send the second ACK0 as it doesn't know which of the alternatives has happened, and must protect the protocol against either alternative.

3.3.5 ACK Numbering Conventions

There are two different conventions to number ACKs currently in use in the datacom world. Neither has any technical advantage over the other!

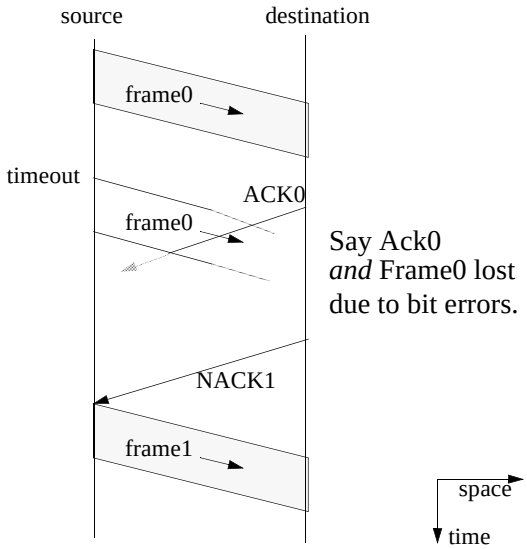
a) '**Ack Number Received**' Convention:
As shown in the last section, an ACK of frame #n is labelled with the number n. This seems sensible, but does not agree with the way numbered NACKs work (which we will study in a few minutes).

b) '**ACK Next Expected**' Convention:
Many destination systems, in response to frame #n, send an ack numbered n+1. This suggests to the data frame sender that the next frame that should be sent is #n+1. This also seems an equally sensible convention, though it often takes students a bit of getting used to. It also agrees with the way NACKs are numbered: when a garbled frame is received, we don't know what frame # it was, so the destination station numbers the NACK with the frame # it was expecting).

I may specify *either ACK numbering convention* to be used on an exam. In addition, in any assignment or exam you do, you should in the absence of a specification, state the numbering convention you use.

3.3.6 NACK Numbering

Unless the header of a frame (where the sequence number is located) is separately FCSed (there are rare protocols like this. e.g. DDCMP), it is unwise to accept the sequence number of a damaged frame as "likely valid", and send a NACK numbered with that possibly damaged #. It is better to send some useful information that the destination does know for sure: the next frame number it is expecting.



Above is an example of where a NACK numbered with the 'next frame expected' is useful. The NACK1 basically acts as an ACK0 stating that the destination has 0 and is now ready to accept DATA 1 next. This is good info, but required a particular double packet loss situation to be useful. It saves having to send frame 0 a third time.

Note: Not only do NACKs ***not*** have to be numbered, but they are ***not*** even essential. Stop-and-wait will work without NACKs of any sort, just a little slower if there are errors and the

source has to wait the full duration for a timeout. So even unnumbered NACKs are not necessary. But they help performance. And numbered NACKs slightly help the source guess better what the current situation it.

You see now that intricate protocols are required. You learned in your operating system course that one of the biggest problems of asynchronous cooperating processes is rigorous validation of timing variances, deadlock prevention, and critical section protection. Distributed systems and communication systems are by their very nature always asynchronous! This is one reason why distributed systems products such as network management are 3 times harder to develop than an equivalent, non-distributed system!

3.3.7 State Machine Implementation

Because of the intricacy of the behavior needed by ***each*** end of a communication system, communication node behavior should be specified and programmed in terms of finite state machines. Finite state machines are a terrific behavioral specification representation that allows the behavior under each and every circumstance (i.e. state) to be documented in either a diagram or a table. The diagram is more visual for design reviews, but the table format is particularly good as it checks for completeness (i.e. there will be blank cells in your table if you have not defined how every possible event in every possible state should be handled!).

State machines should later be implemented in programs as a loop containing doubly-nested CASE/switch statements. e.g.


```
TYPE State enum{idle, just_sent_frame0, just_sent_frame1};
TYPE Event enum{ack0, ack1, nack0, nack1};
VAR state: State;
VAR received: Event;

state := idle;
LOOP
    CASE state OF
    just_sent_frame1:
        CASE received OF
         ack1:send frame0;
              state := just_sent_frame0;
        |ack0:discard this ack;
              state := just_sent_frame1
        |nack1: send_frame1;
              state := just_sent_frame1
        |nack0:send frame0;
              state:= just_sent_frame0
        END (*inner case1)
    |just_sent_frame0:
        CASE received OF:
        ack1:  ......
        ......
        END (*inner case0)
    |.......
    ........
    END (*outer case)
END (*loop*)
```

Isn't this a beautiful, systematically-organized programming style!

Question: Could you re-write this code so the CASE state OF was nested inside of the CASE received OF?

3.4 Flow Control

Flow control is the ability of a receiving station to apply back pressure along the link back toward the sender, in order to indicate that the receiving station is not ready to receive another frame (or has recently been receiving frames in too rapid a sequence to handle/store/process).

We will examine 5 major protocol mechanisms which can be used to implement the back pressure concept. They are:

i) Stop and wait (withholding acks)

ii) Stop and wait (repeated acks)

iii) Stop and wait (using RNR messages)

iv) Sliding window

v) Credit allocation

3.4.1 "Withholding Ack" Flow Control

As previously described, acks should not be returned to the sender unless the previous frame has been received, ***and has been stored so there is Layer 2 receiver RAM space*** to handle reception of the ***next*** frame. Even if a message has been received correctly, the receiver needn't immediately send an ack. It may, for any reason, withhold for a while from sending the ack. If the receiver does this after receiving each frame, it can slow the sender down to an acceptable rate of frames per second.

Disadvantages of withholding ACKS:

- If receiver takes too long to acknowledge, the sender will time out and retry sending the same frame. This is useless, as the receiver already has a valid copy. Re-sends only serve to clog up the net with useless extra traffic.

- If the receiver delays for too long, the sender will retry several times, then give up and inform it's user the link is broken!

3.4.2 Repeated ACK Flow Control

To prevent the sender from timing out or worse "giving up", the receiver can occasionally re-send an ACK appropriate to the previous frame. This would neither confirm or deny the proper reception of the present frame, but would at least re-assure the sender process that the receiver process is still powered-up and alive. The sender's transmit finite state machine would have to be modified so that upon receiving an ACK(previous #), it would either:

a) Reset the "retransmission" counter. This would prevent the sender from declaring the link broken after say 10 tries. But the repeated ACKs reassuring the sender of the receiver's health would only add even more unnecessarily traffic to the re-sends of the frame due to withholding.

b) Or, reset the timer. This would prevent the frame re-sends, but would require repeated acks more frequently (every [time_out] seconds instead of every [time_out] x [max_retries] seconds).

3.4.3 RNR Flow Control

In this scheme, instead of replying with an ACK or NACK, there is a third option. The receive could send a special message called a Receiver Not Ready (RNR). In HDLC/LAPB protocols there is a special supervisory message format called RNR (which actually means "acknowledge frame # but don't send more"). Later, when the receiver is ready to accept more, it follows with a regular ack (RR).

In character-oriented ASCII protocols, instead of a special packet, certain characters may be recognized as RNRs.

ASCII	Keyboard	HEX	Message
DC3	CNTL-s	13_{16}	XOFF
DC1	CNTL-q	11_{16}	XON

I think the X is an abbreviation for "transmission" (sometimes we abbreviate transmitter by "TX", and receiver by "RX"). Note: If using XON/XOFF, don't forget to stuff DLEs in front of each if transmitting binary data!

Disadvantage of RNR Flow Control:

If the XON or RR gets corrupted, the TX will never resume. SOLN: RX should time out if TX never resumes. Problem: What if TX has no more data to send (i.e. no data to resume for)?

3.4.4 Sliding Window Flow Control

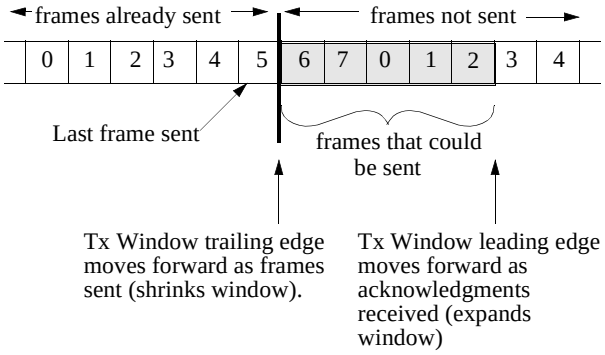
Improved performance is possible if the receiver is known to have more than one Layer 2 frame buffer. A buffer is just a record in RAM where a frame can be stored (not to be confused with a graphics frame buffer). With extra buffers in the destination, the source can safely send a second frame without having to wait for an ACK to the first. If a destination is known to have N buffers, the source is allowed to send N packets without waiting for an ACK to the first. When it gets an ACK to the first, it may then send the (N+1)'st frame of the message.

To understand this scheme, you must first understand modulo frame numbering. To transmit a frame # in a packet header we allocate a fixed number of bits, say 7. Then we can use those bits in the frame format to label the frame (or ACK) as a number between 0 and $2^7 - 1 = 127$.

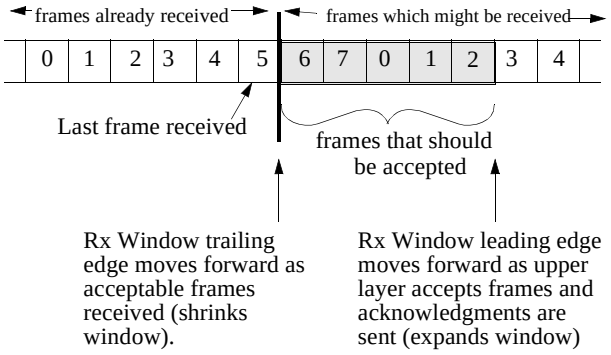
This is fine, except after sending 128 packets, we have no choice but to "wrap around" and start with zero again. We must be able to handle sequence number 'wrap-around' We might avoid this if the field size was much greater than 7, but communication systems run for years and would eventually wrap anyway. We must handle this. For ease of illustration, we will use a field width of 3, frames numbered 0..7 (i.e. modulo-8)

The transmitter keeps track (modulo-M) of two variables: TX_window_back and TX_window_front. It is only allowed to send the few frames numbered between the back (or trailing edge) of the window and the front of the window.

Transmitter's perspective of *its* window.



Receiver's perspective of *its* window.



Each time a frame is transmitted, the source station's layer 2 program updates the TX_window_back variable as follows:

TX_window_back: = (TX_window_back +1) Mod 8. The source can keep transmitting until the TX_window_back catches up to TX_window_front. At that point, the transmitter must stop and wait for some acknowledgments, because:

1) It does not know that the destination Layer 2 task has enough buffer space in RAM to accept any more frames.
2) And, the source is having to keep a copy of every frame it sent until each frame is acknowledged (lest one be lost and need to be retransmitted). This fills up the source's Layer 2 frame buffers in the source's RAM.

TX_window_front, RX_window_back, and RX_window_front are also modified as shown by the events on the diagram (using modulo 8 arithmetic).

Question: Can you write the assignment statement for each?

Note that some students find it confusing why there has to be both a TX_window_back and an RX_window_back. The reason is that the source and destination have a different view of what the situation is. The send and receive back window edges will not coincide if:

a) A frame or ack is in transit.

b) A packet has not been stored due to a bit error.

c) The receiver is withholding acks.

Optionally see Figure 5.7 of [Stallings94]

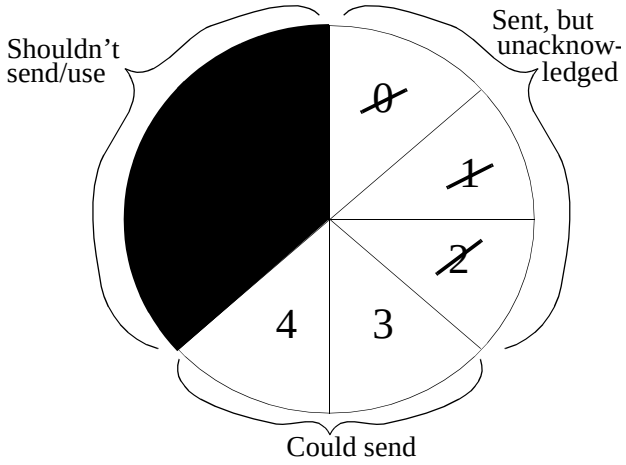
Note that there is a nice alternative to Stallings sliding window diagram that lucidity identifies what sequences numbers:

i) Have been sent but must be stored as they are as yet unacknowledged.

ii) Which have not been sent but could because we know the receiver has layer 2 RAM space for them.

iii) Which cannot be sent as the receiver is not ready to receive them.

Consider Source TX Window for M=8 and N=5:



Note the shaded portion means something different that in Stallings drawings. In Stallings, shading meant you could send those frames. I have found that students prefer to have the shaded/blacked region mean you can't use those sequence numbers at the moment.

You will have to draw these diagrams on assignments and exams.

Note that when the source sends the next frame (i.e. #3), it would noted with a slash.

Note that when this source receives the next acknowledgment (e.g. ack to # 0), it discards buffer 0's contents (i.e. can overwrite it with anything else it wants) and blackens sector 0. Also since N=5, it can unblacken sector (0 + 5) mod 8 = 5.

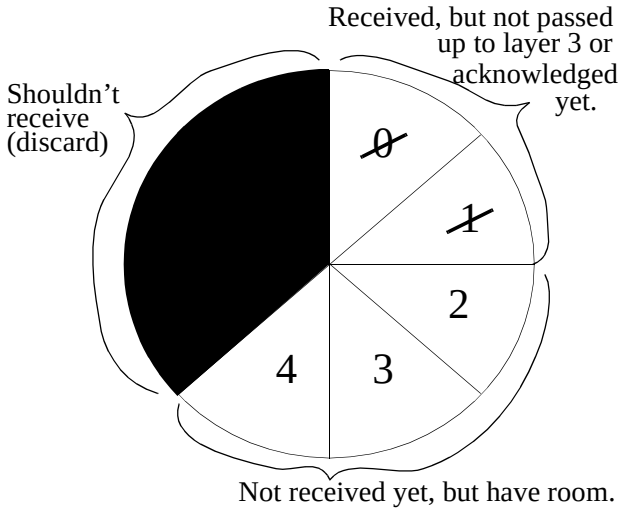
This diagramming technique really nicely illustrates:

1) The sequence # modulo recycling, and

2) The 3 different portions of the sequence space.

Note that you could alternately augment Stallings sliding window diagrams with a "transmitted but not acknowledged" notation.

The RX uses the same notation but the regions have slightly different meanings.



In the receiver:

- When a frame is received, the appropriate numbered sector (e.g. 2) is slashed.
- When a frame (e.g.#0) is passed to Layer 3 and no longer needs be retained in one of the five Layer 2 buffers, the 0 sector is blackened, and sector (0+5) mod 8 is unblackened to permit reception of a frame #5

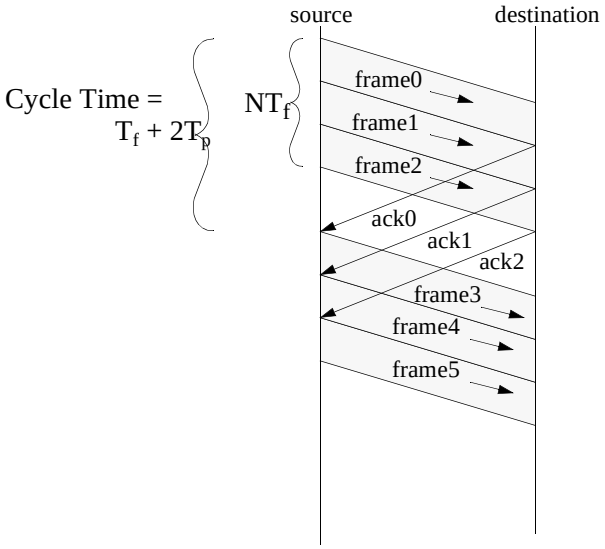
Note that there are only 5 frame buffers, not 8! Typically when frame #5 arrived it would overwrite the old buffer no longer needed for frame #0!

You could program this protocol with 8 buffers, but at any moment only 5 would be needed if N=5. This is wasteful as the frame sizes for a very high speed fibre optic link could each be as big as 100,000 bytes long! On the other hand, it is not trivial to program modulo-8 sequencing with only 5 buffers. You need to use the buffers in a kind of round-robin way for the 8 sequence numbers.

Also note that you don't have to keep track of 3 variables for each window (e.g. front of not received, front of received, front of shaded). This is because if you know M, N, and two of these three window boundaries, you can determine the third.

(hand drawn disk diagram)

Let's determine the sliding window throughput utilization for N=3:



This pattern would repeat over and over again:

transmit N=3 frames then wait for ACK to first.

Utilization = utilization fraction for 1 cycle

$$= \frac{NTf}{Tf+2Tp} = \frac{N}{1+2Tp/Tf}$$

Therefore:

$$U_{slide} = \begin{cases} \frac{N}{1+2a} & \text{if } N < 1+2a \\ 1 & \text{if } N >= 1+2a \end{cases}$$

Note there are two different answers depending on the relative size of N to 'a'. If N >= 1+2a, then the source can keep transmitting continuously, as it has enough buffer space to keep everything sent until ACK0 returns. Recall it must keep every frame sent until each is acknowledged, lest one not be acknowledged and have to be re-sent.

On the other hand, if '2a' is very large compared to N, the transmitter will have to stop and wait for acknowledgments occasionally, as these will free up transmit buffers so that future frames sent can be stored in case they are lost.

Note that the simple stop-and-wait protocol is just a sliding window scheme with

N = 1

M = modulo sequence size = 2

3.4.5 Credit Allocation Flow Control

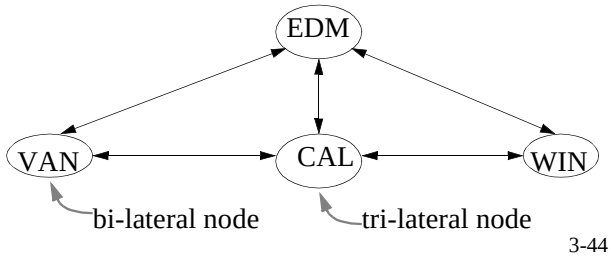
This last scheme is one of the best, but does use up a few extra bits in each header. I am not aware of any Layer 2 protocols that use this scheme, but the layer 4 protocol TCP uses a variant of it. I consider it if I were designing a new protocol.

Credit allocation de-couples the acknowledgment mechanism from the flow control mechanism. It works by having the acknowledgment message contain two fields: an indication of the last packet received correctly, and (depending on how many of them have been passed up to the destination's layer 3) how much buffer space is left in the destination to receive further frames.

De-coupling the acknowledgment mechanism from the flow control mechanism is very useful for three reasons.

1) It's simpler to deal with the 2 issues separately.

2) The transmitter also must have N frame buffers. It must keep copies of all frames sent but not yet acknowledged! But the sooner it gets acknowledgments, the sooner it can free up each buffer (which may come from a shared pool of buffers being used by all links into and out of a multi-lateral network node. Multi-lateral nodes have a Layer 2 task which controls more than one link).



3) If one link is running low on buffer space, the layer can dynamically reduce the maximum number of buffers, N, that are available to a not-heavily-used outgoing link, and lend to the needy link. i.e. You can reduce N for some of the links and lend some to others, dynamically, without causing any protocol anomalies.

In the following example, I will use:

a) ACK next expected (i.e. ACK i+1) convention.

b) ACK2, credit 3 to mean I'm ACKing #1 and expecting 2,3 and 4 (i.e. 3 more).

c) M=8 shown as an 8 sector disk with frames not allowed to be sent blocked out. This is a great illustration of the modulo wrap around and could be used/asked for on a test.

d) N=4

e) Cumulative ACKs (which I don't want to talk about yet).

CREDIT ALLOCATION EXAMPLE (M=8, N=4)

(diagram)

Note:

See the "repeated" ACK4 at the bottom of the previous page. This is one example of the much better 'fidelity of control' offered by the credit allocation scheme. In particular, as a destination you can ACK a frame(s) without the resultant worry of immediately getting more frames fired at you by the source. This has the advantage of allowing the transmitter to discard it's copy of a frame sooner (rather than have to wait until destination's upper layer has accepted a packet). Thus the transmitter releases its buffer as soon as it knows the destination has received that frame correctly (not as soon as it knows destination has room for more)!

Warning:

If you send a CREDIT=0,

then later send a CREDIT=N,

and the latter message is lost, => DEADLOCK!

Solution: 1) Window credit timer, or

2) Source ACKs credit changes.

Credit allocation still uses the concept of a window of frame numbers that are acceptable to send/receive, but it is more definitive in the way the destination tells the source how well things are going. Because credit allocation still uses the window concept, it therefore has the same utilization efficiency formula as sliding window flow control!

3.5 Error Detection & Correction

By sending additional, ***somewhat redundant*** information with a frame, the receiver can with 100% certainty detect whether a "few" bits have been distorted (complemented) by the transmitter/channel/receiver. This is done by detecting inconsistencies of the redundancies in the received frames, and using "automatic repeat request" (ARQ) to request the message be re-transmitted.

If there are *more than a "few" errors*, this also can usually, but not with 100% certainty, be detected.

"Well designed" systems will,

- by appending only r extra bits,

- fail to detect only the tiny fraction $(1/2)^r$ of the frames with errors.

For r = 32, that means the technique misses only 1: 4 billion frames with errors, no matter how "few" or many bits of a frame are received in error!

Also note that this fraction is of the number of frames in error. If only 1% of the frames arrive with any errors within them, the fraction of all frames passed up to layer 3 with errors is 1/100th of 1:4 billion!

In addition, this works no matter how many bits the frame contains.

Nonetheless, if a nuclear weapons system is listening for a 10 ms. long radio packet 24 hours a day, 365 days a year, you may have a problem in less than a year if all it hears is static (noise).

A similar, but less critical problem, is laser bar code readers in super markets reading all the miscellaneous writing on packages, or the pattern on your shirt!

3.5.1 FEC vs. ARQ

Forward Error Correction (FEC) is a technique that adds even more redundant information to a data stream, such that if a "few" bits of a message are received in error, this can be detected and corrected (without an ARQ re-send of the data).

WHEN TO USE F.E.C.

- When you can't be delayed for a re-transmission. e.g. real-time video or audio. Or very long range space communication to Saturn (Tp=10 minutes).
- When there is no return transmission system, and you expect errors, you must use F.E.C. (as ARQ requires a return path for acks/nacks).
  * All types of broadcast-only systems:
  - pocket pagers.
  - future newspapers being broadcast by satellite to all homes.
  * Storage systems where errors are expected (storage systems of course have no 'return path'!).
  - Ram memory in space where radiation is high and can complement bits.
  - Disk memory sectors where you can use cheaper media if a few bits can be corrected (magnetic, or fingerprints on CDs).
- When you have lots of bandwidth to transmit the many extra bits (e.g. space communication.)
- When there is little received signal strength. You can use coding to get coding gain (measured in dB) to compensate for a weak signal-to-noise ratio (SNR).
  e.g. very long range space communication.
- On a "really bad" channel, ARQ schemes may detect an error in most frames, dropping the utilization (i.e. throughout) tremendously. In this situation, a little F.E.C. plus ARQ may be best.

Unfortunately, FEC requires significantly more redundancy to detect and correct up to C errors in a frame, than to just detect up to C errors. This is because you must additionally determine how many errors there were (not just that there was at least 1), and which bits in the frame they were, so you can correct (i.e. complement) them.

In addition, this determination can be in error too, if more than C errors have occurred. And unfortunately you usually can't even conclude whether more than C occurred and fall back to ARQ! Forward error correction coding assumes C or less have occurred. If between C+1 and ~2c errors have occurred, you will correct (i.e. complement) the wrong ones, and if >2C occur, who knows? ***In both cases a wrong message will be accepted "as correct" by the receiver!***

Note: Error detection will usually detect more than a few errors. F.E.C. will usually not correct more than a few errors, and in addition usually doesn't even realize when more than a few errors have occurred.

WHEN TO USE ARQ:

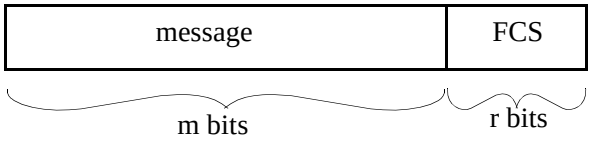
- FEC is computationally burdensome, so when data rates are very high, F.E.C. requires expensive, custom high speed hardware.
- When data rate is limited by the channel bandwidth. ARQ uses fewer check bits to detect the same # of errors in a frame.
- When the transmission rate needn't be constant (i.e. the delay caused by a NACK and re-transmission is not serious). One of the nice things about ARQ is that it uses the extra capacity on a link for retransmissions only when needed (during noisy channel times), and the rest of the time uses it to increase throughput. F.E.C. on the other hand, is burdened with the "extra" redundancy all the time (i.e. even when the channel is quiet and few errors are occurring).

Note: Layer 2 error handling is not employed as part of a multi-hop network if the hops are relatively error free (requiring only layer 4 error handling).

Note: Some systems only require detection without retransmission. e.g. Digitally coded voice at <4800 bps just extends a syllable a little extra long when no new accurate vocal information is available due to a frame error.

3.5.2 Intro to Error Detection

Consider a message frame M which is m bits long.



We append a frame check sequence which is r bits long (called that as r is often the remainder from a division). Note that r is usually (but not necessarily) <m.

The resulting n= m+r bits is called a code word, which is transmitted over the channel.

The "rate" of the code is m/(m+r). A rate 2/3rds code then, has only 2/3rds of its content as actual message. The other 1/3 is redundant data used to decide if there is an inconsistency upon reception, thus indicating an error during transmission.

3.5.3 Vertical Parity

Vertical parity is a simple scheme to get some error detection capability by adding only one extra bit to a message. i.e. r=1

Parity is one way to choose the value of the extra bit. In fact there are two alternate algorithms for choosing the extra bit, one called even parity and the other called odd parity.

Many IBM computers use 9 bits to store each byte in RAM. Each 8 bit byte has a parity bit appended to it. If a memory chip fails, or even becomes loose in your computer, the computer will detect it and shut down, indicating to its user there was a parity error that should be looked into.

I'm not sure whether IBM uses even or odd parity, but either are equally effective. Here is how even parity works:

a) Add up the number of bits in the byte that are true (i.e. 1).
b) If the result is even, add a 0 as the ninth bit. If not, add a 1 as the ninth bit.
c) The effect of this algorithm is that the number of bits which are true in the resulting 9 bit string is even!
d) If when reading or receiving such a string you find the number of bits which are true is odd, you know there has been an error!

Odd parity works only slightly differently, as you might expect.

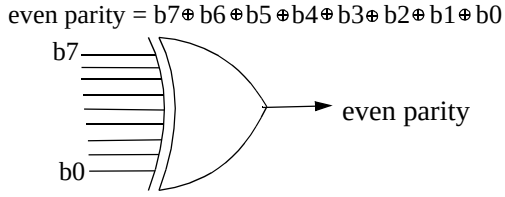
Parity will detect all 1, 3, 5, 7, or 9 bits in error in the codeword, but not even #'s of errors. In other words, if the channel complemented 3 of the bits of the 9 bit string, the destination's parity detection circuitry will detect this. On the other hand, if the channel complemented 4 of the bits, this would not be detected by the receiver, and the destination user would be fed corrupt data!

Vertical parity will reduce the byte error rate (by requiring retransmission) by a factor 2-3, and the block (frame) error rate by a factor of ~10 (depends on frame size).

Definition: Block error rate is fraction of frames with >= 1 bit error.

Simple parity is used for several reasons:

- often we cannot afford more than 1 extra bit per frame, especially if m is small.
- in some situations, we just want to eventually find out that our systems are failing.
- it can be implemented ***very simply and rapidly*** with nothing other than the exclusive-or function or hardware gates.



Notes:

1) The above circuit is for parallel calculation of even parity. In serial transmission, where one bit at a time is sent, the 8 data bits are sent first and the bits 'added'. Then the least significant bit of the sum is appended to the transmission as even parity. This way the parity is being computed 'on the fly' during the transmission. Similarly at the receiver, the 9 bits are added one at a time as they arrive. Immediately upon the 9th bit arriving you know whether there is an error or not as the least significant bit of the sum should be 0!
2) odd parity = NOT(even parity)

3.5.4 Hamming Distance Theory

Consider m=3 and r=1 (e.g. 3 bits plus odd parity)

message	codeword
000	0001
001	0010
010	0100
011	0111
100	1000
101	1011
110	1101
111	1110

All other 4 bit values are not legal codewords. If a destination received an illegal codeword, it would know that a transmission error had occurred.

Definition #1: The "Hamming distance between two particular legal codewords" is the # of bits that need to be complemented to change one to the other.

e.g.

0001
↓↓
0010

Note that two bits differ, therefore the Hamming distance between these two particular codewords is 2.

You will note that ***every*** codeword in the above codeword set (called a 'code') is ***at least*** a (Hamming) distance of 2 from ***every other legal codeword***. Some codewords pairs are farther apart than that. But every codeword above has a **nearest** other codeword of distance 2 away (in the Hamming distance sense). This is true of a 'perfect code'.

Definition #2: Perfect codes are a **set of codewords**, each of which's ***nearest*** other codeword is the same distance. Non-Perfect Codes have codewords which differ in distance to the ***nearest*** (in the Hamming distance sense) other codeword in the set.

Pay attention to what I said just there! This is not to say the codewords are all the same distance apart!

If a given code has some codewords whose ***nearest*** other codeword is 3 bits and others whose ***nearest*** other codeword is 4 bits, it is not a perfect code.

We don't have to use perfect codes, but perfect codes **make the most use of the space** between codewords in guaranteeing how many bit errors a code can detect.

Definition: The "Hamming distance of a code word set" (not just between two codewords) is the smallest Hamming distance of any codeword pair in that codeword set.

Let d be the Hamming distance of a code (i.e. code word set). Then this code will be guaranteed to either:

a) detect $e_{max} = d-1$ bit errors in any codeword.

b) Or, correct $c_{max} = ((d-1)\ DIV\ 2)$ bit errors in any codeword

(but not both).

Note the DIV operator is integer division that rounds the result down.

Therefore 3 bits plus parity,

which has d=2, has:

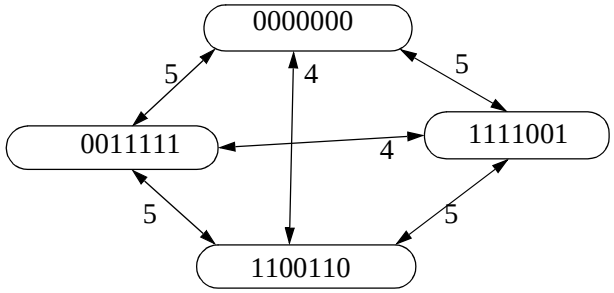
- e_{max}=1, and
- c_{max}=0.

Let's look at a different example where we have a bigger and more interesting d (so c_{max} is not zero):

message	codeword
00	0000000
01	0011111
10	1100110
11	1111001

Note that it is convenient that the first part of a codeword be the raw data message, it needn't necessarily be so for a coding system to work (just use a table look-up instead).

We will examine the above code with a ***Hamming Distance Connected Graph***:



- Each codeword is a node, and there is a line from each codeword to ***every other*** codeword.
- The Hamming distance between the individual codewords is determined and shown as a label on the lines connecting the nodes.

For the above code, we can see the distances range from 4 to 5. Each node's "nearest" other codeword is 4 away. It is thus a Perfect Code.

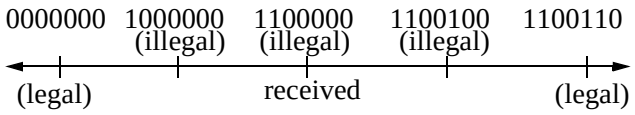
(If one node were to have no connections of 4 (all 5 or more), then it would be a non-perfect code).

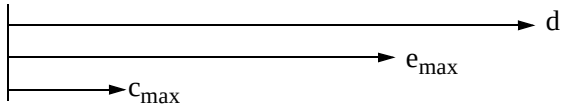
If 0000001 arrives, we could be "pretty sure" 0000000 was sent (more sure than 1111001 sent since 1111001 is farther away and would have had to experience the improbable bad luck of 4 bits complemented by noise on channel!). **This is an example of error correction**. Notice though that this conclusion:

- is based on the probability that a 1 bit error is more likely that a 4 bit error.
- is thus not 100% sure. We must conclude that error correction is not a surety, unless we know that <= c_{max} errors occurred. And how could we ever know that!

If 1100000 arrives, we know a bit error has occurred because it is illegal. It is likely (but not guaranteed) to be a two bit error from either of the nearest legal codewords 0000000 or 1100110. Both are distance two away. 1111001 is 3 away (possible), while 0011111 is 7 away (unlikely was the codeword sent). Error correction is based on this simple principle: "Assume the nearest legal codeword was sent". Let's look at

this in more detail. We will use a non-euclidean "Hamming axis".





Here we can see than 1000000 and 1100100, if received, can be "rounded" to the nearest legal codeword to correct them. But we don't know which way to round 1100000. On a CD with finger prints, take a guess. You have a 50% chance of being right.

(c.f. CMPT290 - If the data is Gray-coded, you might not notice an error as the resultant distortion would be quite small).

Note that we can detect 3 bit errors (or ones that look like them), and with a back channel, request retransmission. But if 1100100 is being corrected, we are really assuming it is not a 3 bit error from 0000000 but a 1 bit error from 1100110. Therefore, if 3 bit errors do occur when we are using correction, we <u>won't</u> detect them, <u>nor</u> will we correct them properly!

When using c=1, our e is reduced to e_{max} -c = 3-1 =2!

SUMMARY:
- $e_{max} = d-1$
- c_{max} = [(d-1)/2] rounded down
- choose c <= c_{max}
- then e = e_{max} - c is left over.

When to give up some e in order to use some (forward error correction) c:

a) Very noisy channel where even the ACKs and retransmissions often don't get there.

b) Space communications where re-transmission could take up to 20 minutes (which may be unacceptable for remote driving on Mars).

c) Military radio where the <u>request</u> for re-transmission may reveal your position to the enemy.

3.5.5 Longitudinal Parity

This section removed from course.

3.5.6 Check Sums

Another frame check sequence is the check sum. This is simply the sum of all bytes into an accumulator register. The register, or a lesser significant portion of the register, is then used as the frame check sequence. If the portion of the register is larger than a byte, a check sum provides more reliable error detection than a similar, related technique called longitudinal parity (which we won't discuss. A check sum's only disadvantage is the requirement for adder hardware (which is more complicated and slower that the Exclusive-OR hardware required for longitudinal parity).

3.5.7 Cyclic Redundancy Codes (CRC)

Also called Polynomial Codes. The theory of CRCs is quite complicated, and can be studied in either 4th year Math or Engineering courses that are offered at SFU. More importantly for you it that you understand the specification and implementation mechanism, as you may have to write software or design high speed hardware to do this in industry.

Consider message M composed of m bits. m may be quite large.

Let G be a special generator pattern (or polynomial) with r+1 bits (i.e. polynomial of order r)

e.g. G=10110111=$2^7+ +2^5+2^4+ +2^2+2^1+2^0$

(Often the literature uses X instead of 2 in the polynomial! This may be because the theory is based on the manipulation of polynomial functions.)

How do CRCs work? It is based on a fast binary variant of the following: If you divide 13, say, by 4, the result is 3 with remainder 1. You can then conclude that 13 - 1 = 12 is exactly divisible by 4!

CRC Procedure: Consider M to be an m bit long number.

1) Divide ($M \times 2^r$) by G using modulo-2 arithmetic.
   - Note 1: $Mx2^r$ is just M shifted left by r zeroes appended on right.
   - Note 2: Modulo-2 arithmetic is binary arithmetic with no carries to or from binary columns.
2) Subtract, using modulo-2 arithmetic, the remainder (which is called the frame check sequence and will be r bits long) from $Mx2^r$.
3) Transmit the resulting frame (which is exactly divisible by G).
4) On reception, divide the received frame by G. If there is a non-zero remainder, an error in transmission has occurred. If the remainder is zero, it is highly unlikely an error has occurred.

Example:

Given M= 1010001101 (m=10 bits)

G= 110101 (6 bits long, so r=5)


```
                 11010 10110   <- Quotient
        ____________________
110101 )1010001101 00000      <- Mx2^5
G <-    110101
         111011
         110101
          011101
          000000
           111010
           110101
            011111
            000000
             11111 0
             11010 1
              0101 10
              0000 00
               101 100
               110 101
                11 0010
                11 0101
                 0 01110
                 0 00000
                   01110   <- Remainder
                           (r bits long)
```

Using modulo-2 subtraction

```
            1010001101 00000
          -           01110
Transmit:   1010001101 01110
```


But since this looks like M concatenated with r remainder bits, why do the subtraction. Just concatenate the remainder on the end of the message!

On reception 101000110101110 should be exactly divisible (modulo-2) by G. Try it!

Selection of G is very important. Some standard Gs with known good frame check characteristics are:

CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	$x^{16}+x^{12}+x^5+1$
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

No matter how large m is, 'good' CRCs will detect:

- all odd numbers of bit errors (no matter how widely dispersed).
- all burst errors of length r,
- bursts of length r+1 with probability $1-(.5)^{r-1}$
- all longer bursts with probability $1-(.5)^r$

e.g. CRC - CCITT has been proven to detect all single, double, and odd # of bit errors in a frame, and:

- All bursts <= 16 bits long
- 99.997% of 17 bits long bursts (e.g. 4, 6, 8, ..16 errors within 17 bit string)
- 99.9985% of >17 bit long bursts.

Definition: Burst Length is first bit in frame which is in error until last bit in error (even if some in the middle of the substring aren't in error). This comes from the mathematical theory of CRC codes.

To elaborate more on CRC's error detection capabilities, CRC-CCITT for example can detect:

a) 100% of all 2 bit error patterns, no matter how far apart in the frame

b) 100% of all error patterns with an odd number of errors (e.g. 3, 5, 21, 999), no matter how widely scattered in the frame they are.

c) 100% of all error patterns confined to a 16 (or less) bit sub-string of the frame.

d) 99.997% of any error patterns confined to exactly a 17 bit sub-string of a frame (e.g. 4, 6, 8,...., or 16 bits wrong within a 17 bit long sub-string of the frame).

e) 99.9985% of any patterns not described above. (i.e. patterns of even numbers of errors (>=4) spanning a >17 bit substring of the frame).

Advantage of CRCs:

- Fast algorithm and hardware.
- Good for bursty channels.
- Number of bits of FCS does not grow for larger frames. It is fixed, so as frame grows longer, the % overhead taken up by the FCS gets smaller.

Disadvantage of CRCs:

- Not as good for 4, 6, ... bit random errors spanning a >r bit long sub-string.

There are other (non-CRC) schemes which also use, say, a 16 bit FCS, but which would 100% reliably detect 4 random errors (i.e. 4 bit errors widely spread within the frame), but would not with 100% reliability detect 5 to 16 errors occurring in a confined burst! You can't win both ways (unless you use a longer FCS).

3.5.8 Hardware Implementation of CRCs

Division looks and sounds slow, but modulo - 2 division has no carries or borrows. It can be simply implemented with only a high speed r-bit-wide shift register and a few Exclusive-OR gates! Optionally see [Stallings94] Figure 4-6.

Intentionally left blank for shift register diagram.

1) Clear shift register.

2) Feed M concatenated with r zeroes in from right.

3) When last bit is in, (parallel) read out the contents of the shift register and use it as the remainder.

On reception, clear register, feed in frame, check final contents = 00000 (i.e. divides evenly).

3.5.9 Residual Frame Error Rate

Definition: The Residual Frame Error Rate (RFER) after ARQ is the fraction of frames sent up to layer 3 that are likely to be in error.

This can be calculated as:

RFER = (fraction, P_f, of frames corrupted by channel)

x (fraction of corrupt frames which are not detected as being corrupt).

e.g. (1/1000 wrong) x (1/65536 undetected) = (1/65.536M)

3.5.10 Frame Error Rate

In this section, we will see how to calculate the frame error rate given the bit error rate, assuming random errors. Please note that frame error rate is sometimes called the block error rate, as it is the fraction of fixed size blocks of bits within a message that have one or more bit errors in them, and thus need to be re-transmitted. Unfortunately, this is sometimes called P_b which wrongly might lead you to believe it is the bit error rate. I will avoid the subscript 'b' for this reason.

Given the probability, P_e, of a ***bit*** error, we need to determine the probability P_f of a frame error. With P_f we can then determine the performance (or utilization) of a particular error protocol.

Given a frame of length L bits, the probability of one or more bit errors in a frame is ***not*** $P_f = P_e \times L$

Instead, we must calculate the probability $(1-P_f)$ that a frame is good. Now $1-P_e$ is the probability that the first bit is good and (i.e. times) the second is good and the third is good .... So:

$1-P_f = (1-Pe) \times (1-Pe) \times ..... \times (1-Pe)$ {L times}

$= (1-Pe)^L$

Thus:

$P_f = 1-(1-Pe)^L$

You may be tested on this derivation.

3.6 Error Protocols and Performance

We will be looking at how protocols handle errors. In particular, we will look at stop and wait, go-back-N, and selective repeat error control protocols. In addition we will try and estimate the performance under varying conditions of frame error rate and link parameter 'a'.

3.6.1 Stop-and-Wait Utilization with Errors

In addition to a flow control mechanism, stop-and-wait is used as a simple error control mechanism. For links were a<<1, there is no need to use a more complicated protocol.

In stop-and-wait, if either a NACK is received or a timeout occurs, the data source assumes there was a garbled or lost message (or garbled or lost ACK or NACK). It re-sends the current packet. Since we have seen how this works already, we will not discuss the mechanism again.

How do errors affect the throughput utilization, U, previously derived for stop-and-wait flow control? First we make some simplifying assumptions:

- The receiver is never busy and can thus send immediate NACKs.
- Either the timeout interval is exactly 2Tp, or frames are never lost (only garbled), so the source knows as soon as possible of an error.
- ACKs (and NACKs if used) are infinitely short and have a transmit time of 0.
- ACKs and NACKs are very unlikely to have a frame error as they are extremely short.

Note: During the assignments I may remove some of these assumptions and let you analyze a more general case.

Recall for flow control: $U_{s+w} = \frac{1}{(1+2a)}$

If due to errors, some packets have to be retransmitted (and in unlucky cases retransmitted several times), we will have:

$$U_{s+w} = \frac{1}{\bar{N}_{tx}} \frac{1}{(1+2a)}$$

Where $\bar{N}_{tx}$ is the average number of transmissions needed to get a packet thru to the destination (averaged over all source packets needing to be transferred). e.g. if 1.5 tries are required, on average, then,

$$U_{s+w} = \frac{1}{1.5} \frac{1}{(1+2a)}$$

We now attempt to derive $\bar{N}_{tx}$ given the frame error rate expected. The probability of occurrence that it will take specifically i attempts (i.e. i-1 failures followed by success on the i^{th}) is:

$$P_f^{i-1}(1-P_f)$$

The average i is just the sum of all possible (i's times their probability of occurrence).

$$\bar{N}_{tx} = \sum_{i=1}^{\infty} i(P_f)^{i-1}(1-P_f) = \frac{1}{1-P_f}$$

e.g. If P_f=.1, $\bar{N}_{tx}$=1/.9 = 1.1111

So one in every 9 will need to be retransmitted (not 1 in 10)! In 10 packets, 1 will be a retransmission of one of the other 9. In other words 9 in 10 get through.

So, the throughput of stop-and-wait is reduced by $1/\bar{N}_{tx}$. Therefore:

$$U_{s+w} = \frac{1}{\bar{N}_{tx}} \left(\frac{1}{1+2a} \right) = \frac{1-P_f}{1+2a}$$

3.6.2 Go-Back-N ARQ

Warning: Be wary of Figure 3-15 in [Tanenbaum96] as he shows ACK leaving before the packet they acknowledge has fully arrived, etc.

Go-back-N was the first error control protocol that allowed "continuous ARQ". Continuous ARQ is just the term used for any ARQ technique that allows continuous transmission (without stops to wait).

In the early satellite systems, it was quickly realized that $U_{stop+wait} << 1$ if a>>1. But early CPUs and their memories were primitive, small, and expensive. Especially in systems which broadcast to many ground stations, it was important to keep the ground station cost low, so Go-Back-N was developed because it only required one buffer at the destination.

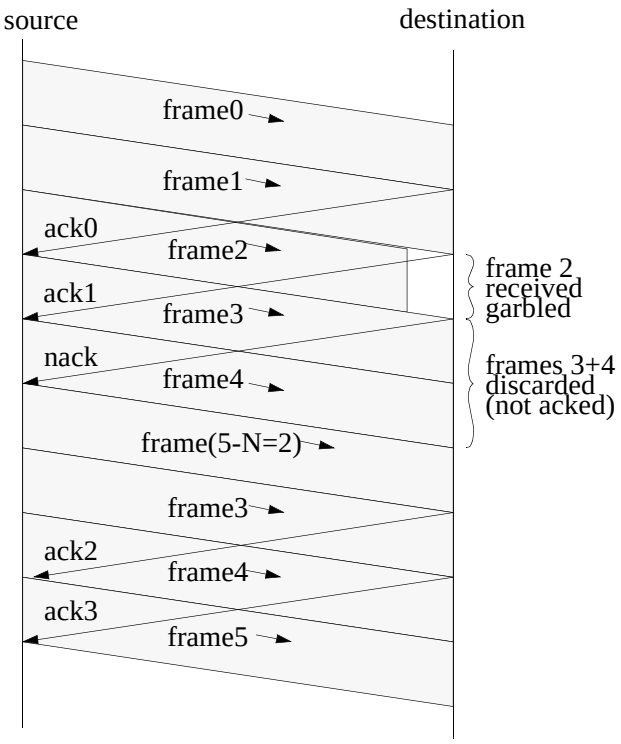
It has these advantages:

- Buffer memory in the receiver for only 1 frame is required in the destination. This is important as satellite transmission rates and frame sizes can be quite large.
- CPU intelligence needed was minimal (GBN doesn't have to re-sequence retransmitted frames catching up to their adjacently numbered colleagues).
- No waiting required for ACKs.

Go-Back-N is related to sliding window protocols, but uses unequal transmit and receive window sizes: N and 1 respectively!

When the destination receives a garbled frame, or misses one, it sends a NACK. When the source transmitter receives a NACK or (if the frame or NACK was lost) times out, it backs up N from the frame number it was about to transmit, and starts re-sending as shown below.

Assume N=3 (>=1+2a)



When the source receives a NACK or (if the frame or NACK was lost) times out, the source backs up N *from the sequence number of the next frame* ***it was about to transmit***, and instead starts again from that earlier numbered data frame. This allows the receiver to throw away any subsequent frames it receives up until it receives the re-sent frame. This means the destination only needs 1 frame buffer, and no re-sequencing logic.

NOTES ON GO-BACK-N:

a) Because the source has N buffers, it still has copies of the last N frames it transmitted. It is thus able to, temporarily not accept anymore from the network layer above, and go back and re-send its stored frames.

b) Disadvantage of go-back-n is that at minimum 2a frames *properly received* are wastefully thrown away. This is the inefficiency you have to pay for simplicity. And it is particularly inefficient if errors are frequent and the destination is throwing away 2a = N-1 (e.g. 19) frames regularly.

c) What should N be? Well, to even get continuous transmission:

$$N_{min} >= 1+2a$$

d) If the destination's responses are delayed, say Tf, by either other processing or needing to be piggy-backed on a similar size, data-carrying return frame, then:

$$N_{min} >= 2+2a$$

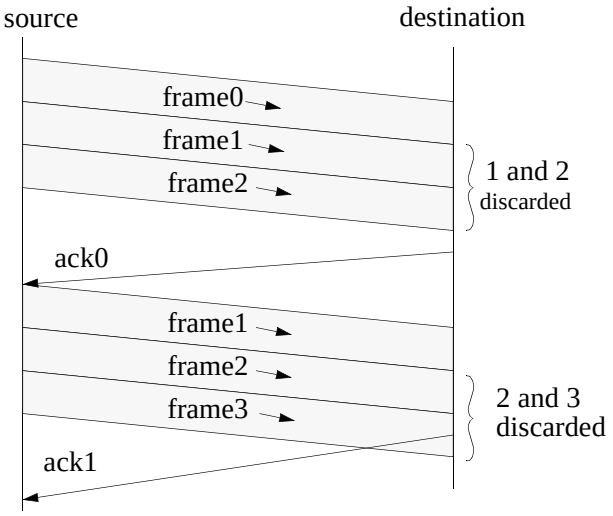
e) For best performance (least delays and frame throw aways in the case of an error), the time out period should be 'tight' (i.e. not much longer than):

$T_{timeout} > 2T_p$ - if not piggy-backing

$T_{timeout} > 2T_p+T_f$ - if piggy-backing

f) If N is too small, or there is a lost frame or ACK/NACK, or using "withholding ack" flow control, the transmitter will stop and wait until timeout.

g) As a result, Go-Back-N is a terribly wasteful form of flow control. See the example below which shows the channel being wastefully tied up by Go-Back-N 'flow control':



3.6.3 Go-Back-N Utilization

If no errors, no processing delays, and

N = 1+2a: $U_{gbn}=1$

N > 1+2a: $U_{gbn}=1$ [but some buffers never used]

N < 1+2a: $U_{gbn} = N/(1+2a)$ [has to wait sometimes]

But if there are receiver processing delays, or if $T_{timeout} >> 2T_p+T_f$, U will be even smaller that this last expression.

If there are errors, no processing delays, infinitely short ack/nacks, and ack/nacks are never lost, we can expect:

$U_{gbn} = 1/(\overline{N}_{txgbn})$ - if N>= 1+2a

$U_{gbn} = (1/\overline{N}_{txgbn})(N/1+2a)$ - if N<1+2a

But now $\overline{N}_{txgbn}$ must not be interpreted as the average # of transmit tries of an individual packet, since a frame error normally causes N extra transmissions, not just one!

Let's call this $\overline{N}_{txgbn}$, the average number of "resulting" transmissions per original source data frame.

Now, a particular frame may not get through on it's 2nd try either, or its 3rd,....., or its 5th!

Let $N_{txgbn}(i)$ be the number of resulting transmissions in failing to transmit a particular frame i-1 times and succeeding on the i^{th}. To calculate this we need to find the average number of resulting transmissions

$N_{txgbn}(i) = N(i-1) +1$

$= Ni + (1-N)$

To average this over all i, we will also need the probability of this occurring:

$Prob(i) = P^{i-1}(1-P)$

where P is the P_f we discussed previously.

Putting these together we get:

$$\overline{N_{txgbn}} = \sum_{i=1}^{\infty} N_{txgbn}(i) \times Prob(i)$$

$$= \sum_{i=1}^{\infty} [Ni + (1-N)]P^{i-1}(1-P)$$

$$= N(1-P)\sum_{i=1}^{\infty} iP^{i-1} + (1-N)(1-P)\sum_{i=1}^{\infty} P^{i-1}$$

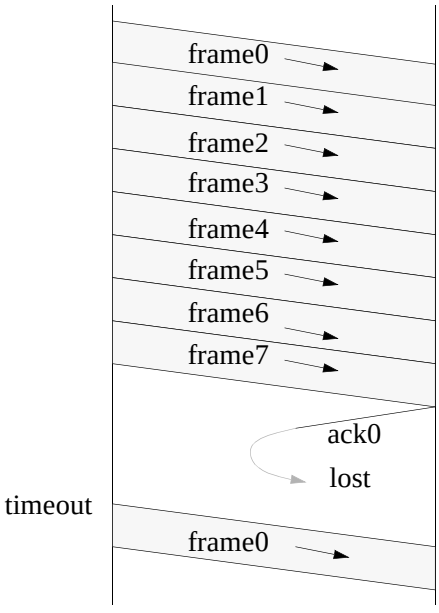
$$= N(1-P)\frac{1}{(1-P)^2} + (1-N)(1-P)\frac{1}{1-P}$$

$$= \frac{N}{1-P} + 1 - N \quad = \quad \frac{1-P+NP}{1-P}$$

Thus:

$$U_{gbn} = \begin{cases} \frac{1-P}{1-P+NP} & \text{for } N > 1+2a \\ \frac{1-P}{1-P+(1+2a)P} = \frac{1-P}{1+2aP} & \text{for } N = 1+2a \\ \frac{1-P}{1-P+NP}\left(\frac{N}{1+2a}\right) & for (N < 1+2a) \end{cases}$$

The middle case is best as it is most efficient, and uses least buffers, while still allowing "continuous ARQ".



The problem is there is an overlap/intersection between the set of unacknowledged sequence numbers from the source point of view, and the set of acceptable next sequence numbers from the destination's point of view! This must never be allowed to happen, and won't in Go-Back-N if M is larger than N.

3.6.4 GBN Modulo Sequence Space Requirement

The modulo sequence size, M, must be > N.

i.e. M>= N+1

N<= M-1

The reason for this is the same reason we needed M=2 for Stop-and-Wait (N=1). If we do not do this, then as shown below, if a burst of data is sent and after some processing delay the first ACK is lost, the source will re-transmit the previous frame 0. But the destination will incorrectly think it is the next frame 0 (modulo-8)!

e.g. Assume (wrongly) that M=N=8

EXERCISE 1:

If destination processing delays are quite variable (requiring $T_{timeout} >> 2T_p$), how could numbered NACKs be used to improve things slightly? Explain when this would help, and by how much? Answer: Go Back only as far as indicated by Nack.)

EXERCISE 2:

What variable(s) must a Go-Back-N destination maintain? Give a code fragment which shows how it/they are updated.

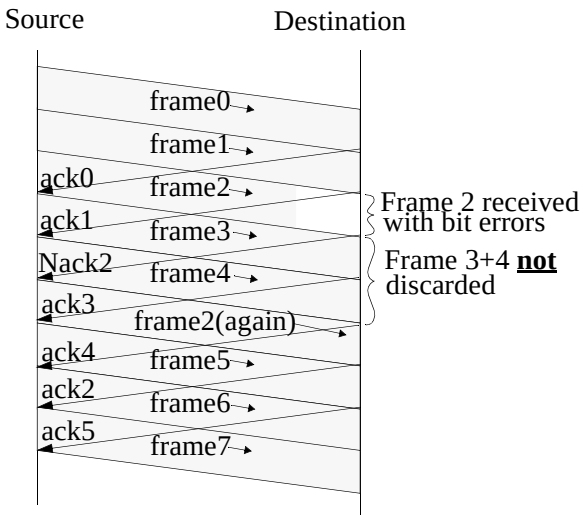
3.6.5 Selective Repeat ARQ

Selective repeat ARQ is similar to Go-Back-N except the receiver is expected to:

1) Also have N buffers, and
2) Have the ability to re-sequence valid frames received out of order (due to errors and subsequent re-sends).

This ensures the destination data link layer can fulfill its job of passing received frames up to its network layer in correct order. Go-Back-N did this in a dumb, brute force manner. Selective repeat does this intelligently.

The selective repeat transmitter does this by only re-transmitting a frame for which it has received a specific NACK or has timed out on. It does ***not*** Go-Back-N and re-send all N frames in it's buffers (as the destination probably received most of them correctly already).



Notes:

a) Frame #2 was simply re-sent between frames #4 and #5. Absolutely nothing was wasted! Compared to Go-Back-N, we are 2 frames further ahead by the end of the diagram. More properly, we are 2a frames ahead (as the diagram uses slopes for a=1).

b) Selective Repeat is a perfect protocol to handle error prone, large 'a' channels. But it is considerably more complex, having to store frames 3 and 4 until 2 re-arrives, then sending the re-arrived 2 followed by 3 and 4 up to Layer 3 (this is called ***re-sequencing***).

b) There are some imposter protocols (namely HDLC) in widespread use which have a restricted form of selective repeat functionality. They use "**cumulative acks**" (where an "ACK i" reply is interpreted as an acknowledgment to frame i and all previous frames too). If this were the case above, ACK3 and ACK4 have been withheld lest they imply 2 was originally received properly. An ACK4 rather than ACK2 would then later convey an acknowledgment to 2, 3, and 4. Only if the protocol adopted the convention of ***selective ACKs***, should an ACK have been sent immediately on reception of frames 3 and 4. Actually the big problem with HDLC over and above the ack withholding is that only one *outstanding* NACK is allowed! This is poor, as the only time selective repeat is much better than Go-Back-N is if errors are frequent, and thus the probability of having two frames in the window needing re-transmitting is also frequent.

3.6.6 U of Full Selective Repeat

For selective repeat, $\overline{N}_{tx}$ is the same as for stop and wait: $\overline{N}_{tx} = 1/(1-P)$. But selective repeat performance is not penalized by 'a' at all (except you need more buffer memory). This leads us to conclude:

$$Usr = \begin{cases} 1-P & for (N \geq 1+2a) \\ \frac{1}{\overline{N}_{tx}}\left(\frac{N}{1+2a}\right) = \frac{(1-P)N}{1+2a} & for (N < 1+2a) \end{cases}$$

U is a function of both 'a' and P. Let's compare the utilization efficiency of selective repeat, go-back-n, and stop-and-wait as a function of P for a particular 'a' where Selective Repeat is useful (e.g. a=3).

Then we will look at U for these three error control protocols as a function of 'a' for a particular P where selective repeat really shines.

Graph of U vs. P

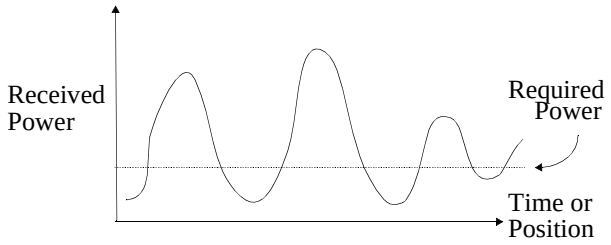
graph of U vs. a

- Notice that for small P, Go-Back-N is almost as good as Selective Repeat! The complications and extra memory needed for Selective Repeat are ***not*** needed on an FDDI fibre optic link where the bit error rate is 10^{-12} and even with million bit frames, P would be of the order of 10^{-6}! On the other hand, for channels with ***both*** large 'a' ***and*** large P, selective repeat is the error protocol of choice.
- Notice Stop-and-Wait is terrible for large 'a'. But for small 'a', simple Stop-and-Wait is almost as good as the most powerful error control protocols, even for large P. In fact, Stop-and-Wait actually does better that Go-Back-N if 'a' is small and N unnecessarily large.
- Go-Back-i is a protocol that uses numbered NACKs to go back only as far as is necessary. It is equivalent to Go-Back-N with a tight time out and as small an N as reasonable (rather than the fixed N=7 shown in the diagram).
- Selective Repeat, Go-Back-i, and Stop-and-Wait should be the same for a=0.

Notes on Selective Repeat:

1) In selective repeat, because the transmitter has to retain for the duration of 2 round trips a frame which was NACKed (i.e. for 1 + 4a frame times), one might think you need 1 + 4a transmit buffers. This is true if you don't want to do complicated buffer management. But if you can manage the pool of Tx buffers with some tricky programming, you only need $N_{txwind} > 1 + 2a$ buffers (as in Go-Back-N). This works even if adjacent frames are also in error, or if a frame re-transmission also fails, because you still only need 1+2a buffers because you don't have any more newly transmitted data to retain.
2) Hint to fading channel protocol designers: When driving through the city, VHF radio receivers are passing through alternating areas of constructive and destructive

interference caused by reflections from buildings and mountains, and the received signal strength regularly dips below adequate levels:



If your frames are not short enough to get through during a peak, none will get through, i.e. $P \simeq 1$, $U \simeq 0$. It is important then, not to use frames which are too long in this environment, lest none get through!

3.6.7 SR Modulo Sequence Space Requirements

Let's look at how large M needs to be for Selective Repeat. Consider the following (malfunctioning) example:

Malfunctioning Example diagram

We notice that if N_{rxwind} were 3 rather than 5, the transmitter's set of unacknowledged frames would not overlap/intersect ever with the receiver's set of acceptable "next" frame numbers. From this we can conclude:

$N_{rxwind} <= M - N_{txwind}$

or **$M >= N_{rxwind} + N_{txwind}$**

Since we usually prefer $N_{rxwind}=N_{txwind}=N$,

$M >= 2N$

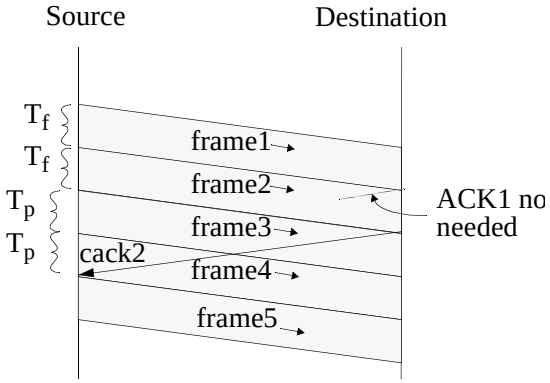
or $N <= M/2$

SELECTIVE REPEAT SUMMARY:

- is an excellent protocol when P > .03 ***and*** a>1.
- Unfortunately, not widely used by data processing professionals who grew up using Go-Back-N with cumulative acknowledgments and only one outstanding NACK allowed.
- But radio data communication engineers, when dealing with urban or short wave radio where P >.03, use it.
- With very high P, Selective Repeat is improved by using a bit field ACK (e.g 00111000 means acknowledgment of frames 2, 3,+ 4) so that the source is totally kept up to date on status of reception at destination.

3.6.8 Cumulative ACKs

If N > 2+2a then ACKs need not necessarily be returned for every frame. Say a=1 and n=4. One could receive frames 1 and 2 (without sending an ACK #1), and then send an CACK2 which is agreed to be interpreted as acknowledging all frames numbered 2 or less (modulo back to the edge of the window). As shown below, frame #5 can then be transmitted without a pause.



Advantages of Cumulative ACKs:

1) Saves on need for return direction acks. e.g. less processing, less network traffic.

2) Offers protection against a lost ack 1, had one been sent.

Disadvantages of Cumulative ACKs:

1) Number of buffers must be greater than 1+2a.

2) If an error occurs, Go-Back-N requires more re-sends since N larger.

3.6.9 Cumulative Nacks

Most techniques that use Cumulative ACKs (e.g. HDLC, LAP-B, and most implementations of Go-Back-N), also use Cumulative NACKs.

A Cumulative NACK#i is an **ack**nowledgment to all frames prior to i and a negative acknowledgment to frame i!

It has the same advantages as a cumulative ack.

Note: In protocols like HDLC which also have (a weak form of) selective repeat, two different forms of nack messages must exist:

1) REJ- a cumulative NACK (reject)

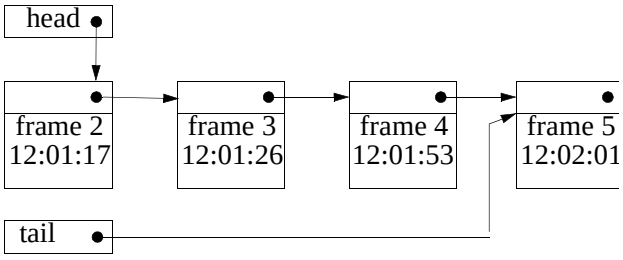
2) SREJ - a selective (or specific) NACK

3.6.10 Implementing Timeouts

Some books suggest that a separate timeout timer is needed in the sender for each departed frame. This is ***not*** true.

Instead, all you need an ordered queue of times you should time out at, and only 1 timer. Each element in the queue contains the frame number and expected ACK time for a frame. The one

timer waits for the time specified by the first element in the queue.



When sending a new frame add it to the tail.

When you get an ack, remove the appropriate queue element

When setting the time out timer, set it to the time of the first element in the queue.

3.7 Flow and Error Control Summary

Q: Do you understand how the 4 different schemes of error control work on a system with no need of flow control (i.e. very fast receiver)? e.g.

Stop-and-wait: - good for half-duplex and small 'a', any P_f.

Go-Back-N - only need 1 rcv. buffer (good for large 'a' AND small P_f.)

HDLC Sel. Rep. (don't study this one) - some advantages of selective repeat but hindered by cumulative acks, so little advantage and only 1 rejected packet allowed at a time.

Sel. Repeat - good on large 'a' noisy (large P_f) channels but complex to implement and requires >1 receive buffer.

Q: Do you understand how the 6 different methods of flow control work on an error free channel?

Withholding ACKs - disadvantages? (stop and wait)

Repeated ACKs - disadvantages?

RNR/XOFF - disadvantages?

Go-Back-N - terrible method of flow control as tries to transmit continuously!

- M>N needed.

Sel. Rep./Sliding Window - $M >= (N_{txwind} + N_{rxwind})$

Credit Allocation - an improved sliding window.

3.8 Optimum Frame Length

For a given probability of bit error, E (sometimes called P_e), a long frame length L will cause frequent frame errors $\mathbf{P_f = 1-(1-E)^L}$, and therefore low utilization efficiency, U. Thus we should choose L small.

But the header of each frame is a fixed necessary size. i.e L = h header bits + d data bits. Note: h is deemed to include all administrative overhead bits. For HDLC:

h = flag + address + control + (CRC -16)

= 8+8+8+16 = 40 overhead bits

So if we use too small an L, there will be little room left in the frame for d and the effective U (after subtracting the administrative overhead), will be small. Thus we should choose L large.

We would guess then, that there is a maximum of U somewhere between the two extremes of independent variables. Sounds like a job for calculus!

For stop and wait, U_{s+w} is a function of P_f and 'a'. But 'a' is a function of the data rate R, of L=h+d, and of T_p. And P_f is a function of P_e and h+d.

i.e. $U(R, h, d, T_p, P_e)$

In fact, U is compose of 3 multiplicative factors:

1) losses due to header overhead.
2) losses due to errors requiring re-transmissions.
3) losses due to stopping and waiting.

We can determine the d which results in maximum U by taking the partial derivative of U with respect to d, setting the derivative to zero, and solving for d. See [Tanenbaum88 - an old edition] Section 4.5.1. It's very messy to differentiate such

a function (i.e. the derivative of the first factor times the second and third, plus the derivative of the second times ...).

To make the math simpler, Tanenbaum's analysis makes several simplifying assumptions:

1) $P_e << 1$, so $\ln(1-P_e) \sim -P_e$, and another simplification shown in Tanenbaum can be made since P_e is usually small.

2) The time it takes to send an ACK is $T_h = h/R$ (it is quite normal that an ACK is the same size as an empty data frame).

The resulting optimum data length is: $d_{opt} = \sqrt{(h + RT_{min})/P_e}$

where T_{min} is the minimum time out period:

$T_{min} = T_p+T_{ack}+T_p = 2T_p+T_h$

Substituting this and re-arranging, we obtain an important result:

$d_{opt} = \sqrt{(2h + 2T_pR)/P_e}$

This expression is of the expected form:

Increasing P_e -----> causes decreased d_{opt}.

Increasing h ------> causes increased d_{opt}.

Increasing T_p -----> causes increased d_{opt} (to reduce the number of "waits").

Increasing R ------> causes increased d_{opt} so useful % of cycle time spent transmitting is not reduced by shorter T_f.

So, if $T_p = 0$ (i.e. a=0),

$$d_{opt} = \sqrt{(2h)/P_e}$$

This formula is also applicable to the Selective Repeat protocol even if $T_p>0$, as long as the selective repeat is operating with lots of buffers so that propagation delays are irrelevant.

Note also that if $T_p=0$, d_{opt} is also independent of R!

Therefore, given a selective repeat protocol using an HDLC-like frame format (where h = 40 bits), for:

$P_e=.01$, then d_{opt} = 89 bits = 11 bytes

$P_e=.0001$, then d_{opt} = 894 bits = 111 bytes

$P_e=10^{-6}$, then d_{opt} = 8944 bits = 1118 bytes.

Note that for the above 3 cases you also have to add 5 bytes to get Lopt.

d_{opt} graph here

3.9 The HDLC/LAP-B Protocol Standard

SDLC
- Synchronous Data Link Control
- Developed by IBM for SNA, their Systems Network Architecture.

ADCCP and HDLC
- Advanced Data Communication Control Procedure (ANSI)
- High-Level Data Link Control (ISO)
- are essentially the same: a modification of SDLC.

LAP-B
- Link Access Procedure - Balanced
- is a subset of HDLC adopted by CCITT.

Notes:
- IEEE-802.2 LLC standard is based on one of these (but I'm not sure which).
- Selective Repeat not available in SDLC or LAP-B.
- All the above protocols use the "next frame # expected" form of ACK numbering convention.

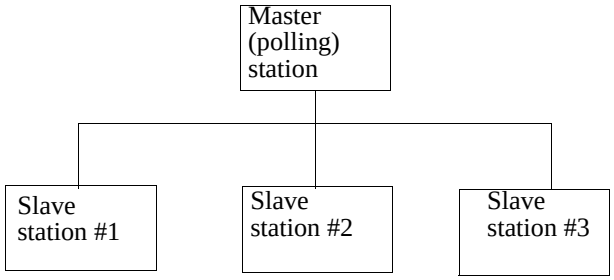
We will study only HDLC as being representative of all the others.

3.9.1 HDLC Frame Format

See Figures 3-24 and 3-25 of [Tanenbaum96] for HDLC frame format and options.

Intentionally left blank for HDLC frame structure diagram.

HDLC has the following features:
- Is bit-oriented to achieve character length and code independence. Uses flag "01111110" and bit stuffing to achieve transparency.
- Full-duplex full sliding window protocol (some advantages of selective repeat) with optional ACK piggy-backing if return frame available (else use supervisory frame to ack/ nack).
- Address field in header available for use on multi-drop (point-to-multipoint) links. Address field is 8 bits long and indicates the address of the slave station being sent to, or received from. Address 11111111 is a broadcast to all stations.



This allows a primitive, polling-based network to be operated on a shared media (i.e. some Media Access Control features are provided for). We will study polling in the MAC sub-layer lectures. Note that the slaves are sometimes called 'secondary' stations.

The multi-use control header contains:
- Frame type (starts with)

0 - Info
10 - Supervisory
11 - Un-sequence numbered (utility)

- 'Send Sequence Number, N(S), and Receive Sequence Number, N(R)
- Poll/final bit:
  - In a frame from master-to-secondary, p/f bit true indicates master asking secondary if it has data to send. (i.e polling)
  - In a frame from secondary-to-master, p/f bit true indicates this is the final frame of a multi-frame data message (i.e. the master need not poll this station for more right now).
- An extended format can be negotiated at connection set-up which can do Modulo-128 rather than Modulo-8 sequence numbering, thus allowing for satellite channel delays. The control portion of the header for information and for supervisory frame types is then 16 rather than 8 bits long.

Two bits determine the function of a supervisory frame:

S = 00 - Receiver Ready (RR) - An cumulative ACK (not piggy backed)

S = 01 - Reject (Rej) - A cumulative NACK (go-back-n or sliding window).

S = 10 - Receiver Not Ready (RNR)

S = 11 - Selective Reject (SREJ) (a **specific** NACK used for selective repeat. i.e. **not** a cumulative NACK)

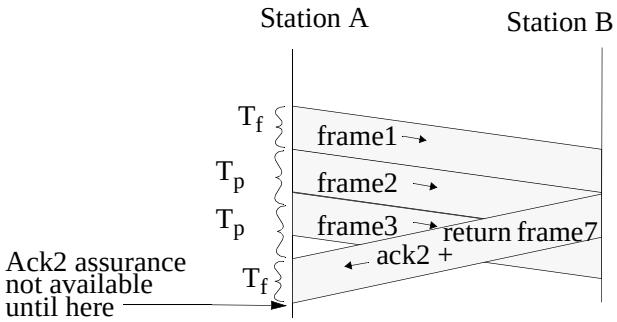
Note1: RNR tell Tx to stop until it subsequently receives an RR.

Note2: HDLC only allows one outstanding selective rejection. Lap-B may (I'm not sure) not allow SREJ at all. This and cumulative acks is why HDLC and Lap-B are not full proper selective repeat protocols.

3.9.2 Piggy-backed ACKs

Often there is considerable return data traffic on a duplex link and it is wasteful to frequently send special RR (i.e. ACK) frames which contain a considerable % overhead in control and flag bits. Many protocols use a loose timeout to wait for the next ***return*** data frame, and accept the penalty of a slightly larger N. The application in the receiver often has reply data to the frame, and the ACK can be piggy-backed on the reply. If using modulo-8 this *only costs 3 bits*!

But now an ACK is not fully received for an extra frame transmission time! Even though the ACK (i.e. N(R)) may be located at the front of the frame, you can't trust its value until you have received and confirmed the FCS is correct.



The return packet contains an N(S)=7 and N(R)=2 (using the next number expected convention), and the data for return frame 7.

Please note that in full duplex communication, there is data flowing the other way too (not just acks and nacks); that is why I have not used the labels source and destination above. Most importantly, the opposite direction data traffic is sequence numbered by a completely independent numbering sequence. This is necessary as there may be twice as many packets going one way as the other.

Note: Need N > 2+2a buffers for piggy-back continuous ARQ, and therefore:

$$\text{maximum } U_{piggy} = \begin{cases} 1 \\ N/(2+2a) \text{ for piggybacking} \end{cases}$$

(not including any extra wait time).

Note: HDLC/Lap-B always fills an info or supervisory frame sequence number fields with something. So the data source gets a lot of repeated acks (most of which it ignores).

3.9.3 HDLC Dialog Notation

Let us set up notation for illustrating HDLC packet exchange between a master, and several secondaries on a multi-drop, full-duplex link. We will annotate the messages on HDLC time-space diagrams with the following notation:

ADDR, Ftype (send#) P/F (rec#)

The P/F bit is P or $\overline{P}$ when going ***to*** a secondary, and F or $\overline{F}$ when coming ***from*** a secondary.

Here are various examples:

e.g. B, I (2) $\overline{P}$ (6')

- is an Information frame #2 for secondary B, P/F bit = 0, and it acknowledges frame 5. It would normally be followed by the info bits, FCS, and flag.

e.g. C, REJ $\overline{P}$(2)

- is a supervisory frame of type = REJ addressed to C NACKing #2 which was garbled or absent (#1 was received). Poll bit =0. Note that (send #) is absent in supervisory frames.

e.g. B, SABME, P

- is an unnumbered frame of type = SABME addressed to B. The message is also polling for a reply. Note that none of the bits are used for sequence numbers (thus the term "unnumbered").

3.9.4 HDLC Possible Frame Types

Information Frame Types (only one type):

I - Information frame including seq. numbers.

Supervisory Frame Types:

RR - Receiver ready (Ack) or (restart after RNR)

REJ - Reject (Nack)

RNR - Receiver not ready

SREJ - Selective reject

Unnumbered Frame Types:

SNRM - Set Normal Response Mode (i.e. master + slaves)

SNRME - Set NRM Extended (modulo 128)

SABM - Set Asynchronous Balanced Mode for peer to peer links (but must be point to point as only one address field)

SABME - Set ABM extended

SARM - Set Asynchronous Response Mode

SARME - Set ARM extended

RESET - Reset sequence numbers without resetting mode.

UA - Unnumbered Ack (for set/reset commands)

RIM - Request Initialization Mode (only in ARM)

SIM - Set initialization mode

RD - Request disconnect (only in ARM, or in response to poll)

DISC - Disconnect command

DM - In Disconnect Mode (ack to DISC)

Notes:

- The first 6 set command above reset the sequence number counters to 0 in preparation for a new connection mode.
- Sets and resets can only be issued by a master.
- UA, RIM, RD, DM can only by issued by a secondary (or combined master secondary) station.
- Info and supervisory frames can be issued by either.

I will now show the HDLC dialog notation and frame types in action:

HDLC exchange Diagram page 1 of 2

HDLC exchange Diagram page 2 of 2

Note 1: This was a full-duplex connection, data was going both ways simultaneously, with two completely separate systems of sequence numbers!

Note 2: On a multi-drop link, the master could connect to 127 other stations by doing a connection set-up to each, then polling one at a time. The master would be working then with 254 sets of sequence numbers (127 different send and 127 receive).

Here is another example, with window disks included:

???Diagram with disk page 1 of 2

???Diagram with disks page 2 of 2

3.9.5 HDLC ARQ

(Do not study this section; only know that HDLC ARQ is bad as it has no selective Ack frame type).

HDLC has a very weak form of selective repeat. Because it only has the cumulative form of acknowledgments, doing selective repeat is very awkward and susceptible to certain pathological problems. Because of the following weaknesses, it is rarely turned on/enabled:

- It cannot have the full advantage of selective repeat because of the cumulative ack restriction.
- Even if well programmed, the pathological problems can cause occasional failures.
- Because of the intricacies of implementing it right, it is often not implemented correctly. In fact, two former CMPT 371 texts [Tanenbaum 88], and Figure 5.55 of [Stallings 94] both make serious errors when presenting the material.
- Selective repeat is only an advantage (cf. Go-Back-N) if both a>>1 and the error rate is large. In most applications of HDLC in business, the error rate is low. Since it is rarely used on noisy radio channels, this feature is enabling rarely.

HDLC only has 3 types of ack/nacks:

1) cumulative ACK (i.e. RR)
2) cumulative NACK (i.e. REJ)
3) selective NACK (i.e. SREJ)

The root of the problem is that HDLC uses cumulative ACKs. HDLC should not cumulatively ACK frames received correctly following one received in error, lest the transmitter interpret them as an ACK to the frame that was in error! It instead should have a special 4th type of frame: 'selective ACK'. Since

it doesn't have such a frame type, in order to have any advantage over Go-Back-N, it must send the cumulative ACK and hope that the source temporarily 'interprets' cumulative ACKS as 'selective' because it follows a selective NACK. This idea has some serious errors in it if the original selective Nack was lost! In addition, only one packet in error can be outstanding (selectively NACKed) at a time. But on a noisy channel where you would want selective repeat, often two packets within the window could be in error.

The result is that 'HDLC selective repeat' is terribly designed and HDLC should thus not be used on noisy channels, nor enabled on non-noisy channels.

3.10 Other LLC Protocol Standards

3.10.1 LAP-M

This is a variation of LAP-B designed for use over modems. Originally, modems did not have any built-in protocol or error detection and correction. They relied on the end computer applications to detect this, yet some end applications had no error detection. In the late '80s, students often dialed in to SFU with old 1200 and 2400 bps modems. If the telephone line was noisy, students received garbled messages from SFU, and sent garbled commands to SFU's computers. It was very frustrating.

Later model 2400 bps modems and all faster ones have built-in error detection and correction. LAP-M during link set-up allows modems negotiate the setup of either the MNP5 or V.42 error correction standard (and possibly compression).

3.10.2 SLIP and PPP

SLIP and PPP are used mainly by dial-up Internet sites. Serial Line Internet Protocol (SLIP) is an early point-to-point protocol used between Unix systems on dial up modems. It has no error correction, and its main purpose is just to extend an internet address out to a non-permanent site (like your home).

SLIP is falling into disfavor now that Point-to-Point Protocol (PPP) is becoming widely deployed. PPP's frame format is loosely based on HDLC's, but it uses character stuffing rather than bit stuffing. In fact, PPP is just a framing mechanism for a variety of network layer packets (e.g. IP, AppleTalk, XNS etc.). You will see as we progress in the course that at each layer, the header should contain a field indicating the nature of the data payload. At layer 2, this field indicates whether the payload should be passed to the IP process in layer 3 above, the AppleTalk process at layer 3, or the XNS process at layer 3. It

is possible the destination computer might have several layer 3 processes.

You can optionally read a bit more about SLIP and PPP in Section 3.6.2 of [Tanenbaum96].

3.10.3 LAP-D

LAP-D is a variant of HDLC use for the D channel of an N-ISDN link. Optional Reading: Section 16.3 and 16.5 of [Stallings94].

The D channel in N-ISDN can handle 8 connected devices (slaves) in your home. It thus has some medium access control (MAC) aspects to it. These 8 channels are (asynchronously/ statistically) time division multiplexed together, so each can communicate with the telephone company (the telephone company's computer in your local exchange is the master station; remember HDLC from which LAP-D is derived, can handle a simple master/slave form of MAC). The resulting 16 kbps D channel is synchronously time division multiplexed with the two 64 kbps B channels. The D channel therefore takes up 16/(16+2x64) = 1/9th of the bits being exchanged in each direction on the 144 kbps N-ISDN link to the telephone company. Because the D channel is synchronously allocated 1/ 9th of the data slots, it fills them with flags if not busy.

The following few pages describe the LAP-D frame format, and some of its fields. It is optional, but interesting reading. Note that the payload designator is called the SAPI. Also note that when an incoming phone call is announced over the D channel, it is not necessarily addressed to a particular digital phone (TEI). The message is addressed to TEI=127, which is a broadcast designator. All phones in the house ring, and then the one that is picked up by the user replies with its specific TEI.

Note that LAP-D is a layer 2 protocol. As such it makes sure the calls to a specific piece of hardware are handled. If that

piece of hardware (e.g. your PC) wants to set up multiple data connections (using multiple Layer 3 or 4 sockets), that is handled by Layer 3 and does not show up in LAP-D.

LAP-D always uses Asynchronous Balanced Mode with Extended numbering (modulo-128) so it works on long distance telephone channels (i.e. a local loop via satellite if your home is in the backwoods). The sequence numbering is on a per address basis. Each logical connection has its own set of sliding windows.

Lap-D frame Octets and other pages go here

3.10.4 LLC Layer for ATM

The LLC layer for B-ISDN is broken down into a number of complicated sub-sub-layers. We will look only at the frame format at this time

The whole concept of Asynchronous Transfer Mode (ATM), is that it is a digital technology that is not based on synchronous time division multiplexing. All previous digital technology that the telephone company has used, especially internally between exchanges, is based on digitizing the continuous analog signals from voice phones. Since this data is being feed through the network at a known rate (64 kbps), it made sense to pre-allocate capacity for each call such that one byte of data for each call was sent each 1/8000th of a second. Any user data (non-voice) traffic that the telephone company carried was usually dumped into their digitized voice system. If the user data connection had nothing to send, useless fill flags were sent. ATM is an attempt for the telephone company to get away from their synchronous-only infrastructure.

ATM is designed so that:

- It can carry a mixture of kinds of traffic. e.g. bursty user data, as well as multimedia (video and voice) and telephone voice which are synchronous.
- It can carry and easily route very high speed data (51.84 - 622.08 Mbps).
- It takes advantage of high quality fiber optic links that rarely have bit errors.

One of the problems with very high speed data is that too much more of it comes into a node (and must be buffered/stored) before you can figure out that it is correct (CRC) and figure out where to route it next. So, ATM packets are:

a) very short and all exactly the same length: 53 bytes
b) only have a CRC on the header portion of the frame.

These packets are so small they are called 'cells', and in fact, ATM is sometimes called cell relay (to differentiate it from frame relay which allows variable length frames).

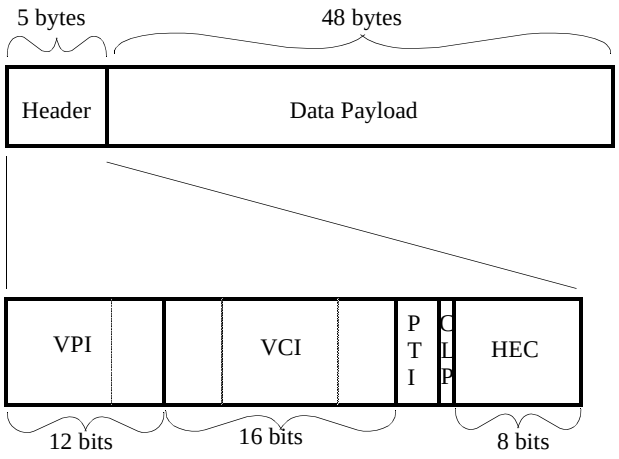
Because they are fixed length, they are more easily handled by fast hardware and software.

Because they are short, buffering one as it is being received only takes up 53 bytes.

Because the CRC is only for the 5 byte header, in fact, they can start to be forwarded even before the cell is completely received. The CRC is called the Header Error Check (HEC) field.

The payload data is not checked for bit errors. That is so rare on good ATM links that it is left to layer 3 or 4. It is ok if the data at layer 2 arrives wrong, as long as it doesn't arrive at the wrong destination (which is why the header is checked).

ATM FRAME FORMAT (53 bytes)



The length of the payload was chosen to be 48 bytes as a compromise between what the telephone companies wanted (32 bytes) and what the data users wanted (64 bytes so the % overhead represented by the header was less). The compromise means the overhead is 5/53 = 9%.

The last part of the header is the header error check field. It is last in the header so it can be calculated while the first part of the header has already started being transmitted. It will correct all 1 bit errors and 90% of multi-bit errors (most fiber optic errors are 1 bit errors, not bursts).

The Virtual Path Identifier (VPI) and Virtual Circuit Identifier (VCI) together form the connection identifier. Since ATM is

connection-oriented, the source and destination addresses have already been specified at call set up time. After that, all that is needed in each cell is a connection ID, which all nodes on the route have been informed of. On receiving a cell, a router looks the connection ID up in a table to determine which outgoing link the cell should be forwarded out on. The breakdown of the connection ID into a path identifier and circuit identifier is done because often a number of circuits must take the same path.

The 1 bit flag called Cell Loss Priority (CLP) indicates which frames are best to discard in the event of congestion. Note that one brief syllable of voice would hardly be missed by a telephone listener.

The Payload Type Identifier indicates whether what kind of user data is in the cell and how congested the route the cell travelled was, or call setup/teardown management information. See Figure 5-63 of [Tanenbaum96].

Note: the first 4 bits of the VPI on cells between a user and the telephone company is reserved for a General Flow Control field. This seems to be a bug in the standard as no one has decided on what the specific values of this field should be. These 4 bits are used to make the VPI a full 12 bits between telephone company exchanges.

ATM is often carried on other MAC layer fiber protocols such as SONET (Synchronous Optical Network) or FDDI, in which case frame delimitation is provided by the MAC sub-layer. If raw ATM is used, another way to find frame boundaries is needed. Flags with stuffing or code violations are not used. The technique is interesting. The incoming data (possibly at 622 Mbps) is fed into a 5 byte long shift register (not 1 byte long). Each bit of the shift register is output to frame check sequence hardware (a big 40 bit input, one bit output combinational logic circuit). This hardware instantly calculates whether the 40 bits in the shift register represent a valid header, and if so, outputs a boolean true. If not, the register is shifted

one bit, and this is tried again. Most of the time (approximately 255/256th of the time) when the output is true, the output of true indicates frame alignment. Approximately 1 time out of 256, random data in a payload will trigger a true. With a state machine that waits to see true several times in a row 53 bytes apart, you can determine where the beginning and end of frames are. Neat eh?

3.10.5 Frame Relay

Frame relay is a hybrid protocol. It's purpose fits partly in Layer 2 and partly in Layer 3. Often, both layer 2 and 3 have error and flow control mechanisms. This seems somewhat redundant, and also tends to restrict data rates to about 56 kbps because of all the extra processing involved at each node.

In addition, often a business wanting to connect to a public data network service like X.25 Datapac does not need any Layer 3 routing between the business and the X.25 node as this is a direct wire connection. The idea of frame relay is to remove one layer of flow control, one or both layers of error control, and all routing, to improve maximum packet processing speeds available from the node computers.

But, businesses often need multiplexing out of their computer (via the dedicated link to the Datapac node), to several other stations simultaneously. Frame relay therefore provides bare bones virtual circuit identifiers, frame delimitation, and throw away error detection. Frame relay is a reduced complexity hybrid of layer 2 and 3 services that allows speeds to 1.5 Mbps.

Frame relay is discussed briefly in Section 1.6.3 of [Tanenbaum96] and more completely (optional reading) in Section 17.2 of [Stallings94]. You can also check the description of B.C. Tel's Hyperstream frame relay service at http://www.bctel.com/office/index.html.

3.11 References

[Stallings94] "Data and Computer Communications" 4th ed., by William Stallings, Macmillan, 1994.

[Tanenbaum96] "Computer Networks" 3rd ed., by Andrew S. Tanenbaum, Prentice-Hall, 1996.