Error Detection

- Errors exist in <u>all</u> *transmission systems:*
  - Consequently, it is necessary to include some form of *Error detection*.
  - Error detection capabilities allow the Receiver station to detect if any of the bits contained in a received frame have been modified.
  - If they have, then something needs to be done about it. Typically this would involve a re-transmission of the errored frame.

Error Detection

- A general principle of error detection is:
  - All *frames* include additional bits which constitute an error-detecting *code.*
  - The code is calculated using an algorithm that includes <u>other</u> bits from within the frame.
  - The Receiver performs the <u>same</u> calculation.
  - The presence of an error will cause a mismatch (although not always i.e. some errors go undetected)

Error Detection

- There are three common techniques to consider:
  - *Parity checking.*
  - *Checksums.*
  - *Cyclic Redundancy Check.*
- *Parity Checking:*
  - A bit is added to the end of <u>every</u> character.
  - This bit is called a *parity bit.*
  - The value of the bit is selected to make the total number of <u>one</u> bits in the character even (*even parity*) or odd (*odd parity*).
  - The Receiver counts the number of bits received to verify the validity of the parity bit.

Parity Checking

- Parity checking can detect *single-bit* errors, or, an *odd* number of bit errors i.e. where 1, 3, 5…..etc. bits are in error.
- It cannot detect *double-bit* errors or, where an *even* number of bits are in error e.g. 2, 4, 6 …etc.
- See example in class.

Checksums

- This mechanism operates at the *frame* level (or as shall be shown later at the *packet* level).
- The Transmitter treats the *entire* frame as a sequence of binary integers and computes their numeric sum:
  - 16 or 32-bit checksums can be used.
- Once again not all errors can be detected
  - See example on next slide

Checksums - Example

H	e	l	l	o		w	o	r	l	d	.
48	65	6C	6C	6F	20	77	6F	72	6C	64	2E

4865	6C6C	6F20	776F	726C	642E

- In this example a 16-bit checksum is being determined hence the result must be 16 bits long.
- The data is separated into 16-bit quantities and added together to give a result of **271FA.**
- Note the **2** on the LHS takes the result over 16-bits hence it is added back onto the **A** to give a 16-bit result (**71FC).**

Cyclic Redundancy Check (CRC)

- This technique is more involved:
  - Here the *Transmitter* adds a *bit sequence* to each frame i.e. not just a *single* bit nor an *integer* value as before
  - This *bit sequence* is known as a *Frame Check Sequence* (FCS)
  - The FCS is derived from the bits associated with the *message* to be transmitted using a *predetermined* number (**the P-value**)
  - The *Receiver* divides each incoming frame by the same *predetermined* number (divisor) and checks the remainder
  - If remainder is zero it assumes no errors have occurred

CRC – The Proof (Not Examinable)

T = (k + n) – bit frame to be transmitted, with $n < k$

M = k – bit message, the first k bits of T

F = n – bit FCS, the last n bits of T

P = pattern of n + 1 bits; this is the predetermined divisor

We would like T/P to have no remainder.

Now,

$$T = 2^n M + F \text{ and,}$$

$$T/P = (2^n M + F)/P$$

$$= 2^n M/P + F/P$$

CRC – The Proof (Not Examinable)

Mathematically the term $2^nM/P$ can be rewritten as:

$$2^nM/P = Q + R/P \text{ – equation A}$$

where, Q = quotient and R = remainder (e.g. $5/3 = 1 + 2/3$)

From before: $T/P = 2^nM/P + F/P$

Substituting equation A gives: $T/P = Q + R/P + F/P$

Q R

If the remainder R from equation A is assumed to be the FCS then:

$$T/P = Q + R/P + R/P$$

Recall adding a number to itself in modulo 2 arithmetic results in zero, therefore

$$T/P = Q$$

No remainder implies T is exactly divisible by P.

Therefore the FCS is the remainder from the calculation $2^nM/P$.

This is guaranteed to result in a zero remainder when the transmitted frame is divided by P

CRC – In Conclusion

- What is required is an error detection code (called the **FCS**) such that received frame **T** is exactly divisible (no remainder) by the **P-value** in the absence of any errors i.e. T/P has no remainder.
  - Where **T** (the transmitted/received) frame is the concatenation of the Message <u>and</u> the n-bit **FCS** i.e. T = M + FCS
  - The **P-value** is an (n+1 bit) divisor known to both the Transmitter and the Receiver and is always one bit longer than the **FCS**

- From the above Proof such an FCS can be found from the <u>remainder</u> of the calculation $\mathbf{2^n M/P}$.
  - All calculations are performed using Modulo 2 arithmetic
  - When this FCS is used T/P is guaranteed to result in a zero remainder when there are no errors

CRC – Step Approach

- To determine the FCS the Transmitter:
  - Uses the remainder from the following calculation: (2^n**M) / P**.
- The FCS is included in the frame with the message M:
  - i.e. FCS is concatenated onto M to give the new transmitted frame (T).
- The Receiver divides T by the P-value:
  - If no remainder results it concludes that there are no errors.

CRC – Example (not Examinable)

- For message: M = 1010001101 (10 bits)
  and a P-value: P = 110101 (6 bits)

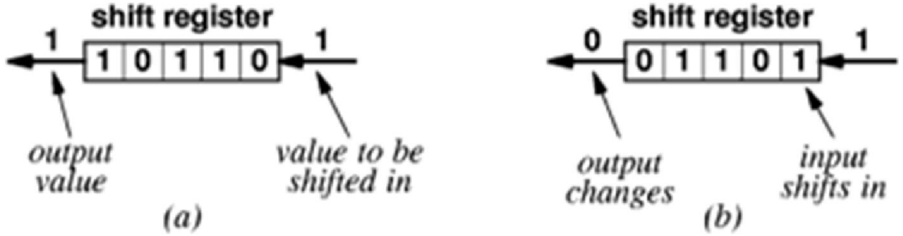
- To obtain the FCS:
  - Multiply M by 2^5 and divide by P
  - The FCS is the remainder i.e. 01110

- The transmitted 'word' is derived by:
  - Appending the FCS to M
  - This is the same as multiplying M by 2^5 and adding the FCS
  - For this example T = 1010001101<u>01110</u>
- The Receiver divides *T* by *P* and should get **<u>zero</u>** remainder if <u>no</u> errors have occurred

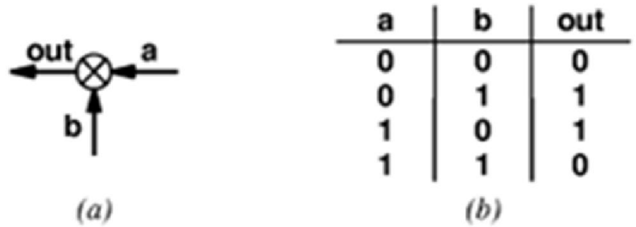
CRC Hardware (Not Examinable)

- The hardware used to compute a CRC consists of *shift registers* and *exclusive OR gates*
- The bits associated with the original message are shifted into the device
- After <u>all</u> bits have been inputted the shift registers contain the 16-bit FCS

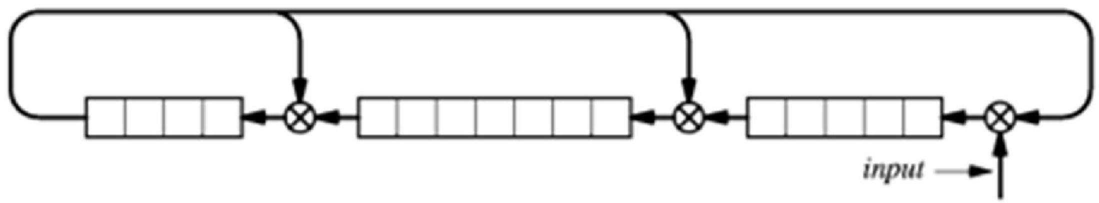
CRC Hardware Components (Not Examinable)



Shift Register



Exclusive OR Gate



CRC Calculator

Error Control

- Having examined the operation of three error detection techniques it is useful to explore how they are used.
- The Detection <u>and</u> Recovery from errors is known as *Error Control.*
- To discuss Error Control the following assumptions are made:
  - Data is sent as a sequence of ***frames***.
  - Each time a frame is transmitted a **Timer** is set to some value.
  - Frames must arrive in the order in which they were sent otherwise an error is assumed.
  - Each frame suffers an arbitrary, variable delay called *propagation delay*.

Error Control

- Two types of error can occur:
  - Lost frame/Lost ACK. Either a **Data** frame or an **ACK** is lost in transit. Recall the effects of *Impulse Noise*.
  - Damaged frame. Here a frame arrives but it fails the Error Detection check.

Error Control

- Error Control provides a number of techniques to address each of these types of error.
- These techniques can be considered a *tool-chest*:
  - Each technique can be considered a tool which can be applied when something goes wrong (or, indeed when everything is working fine).

Error Control - Elements

- Error Control provides the following elements:
  - ***Error detection*** - discussed previously e.g. CRC
  - ***Positive ACK****.* Here the Receiver sends positive ACKs to error-free frames received (implicitly or explicitly).
  - ***Retransmission*** (after *time-out*). Here the Transmitter may retransmit a frame that has not been acknowledged after a time-out period.
  - ***Negative Acknowledgement.*** Here the Receiver can send a negative ACK for damaged or out-of-sequence frames that arrive. The Transmitter may simply retransmit these frames

Error Control

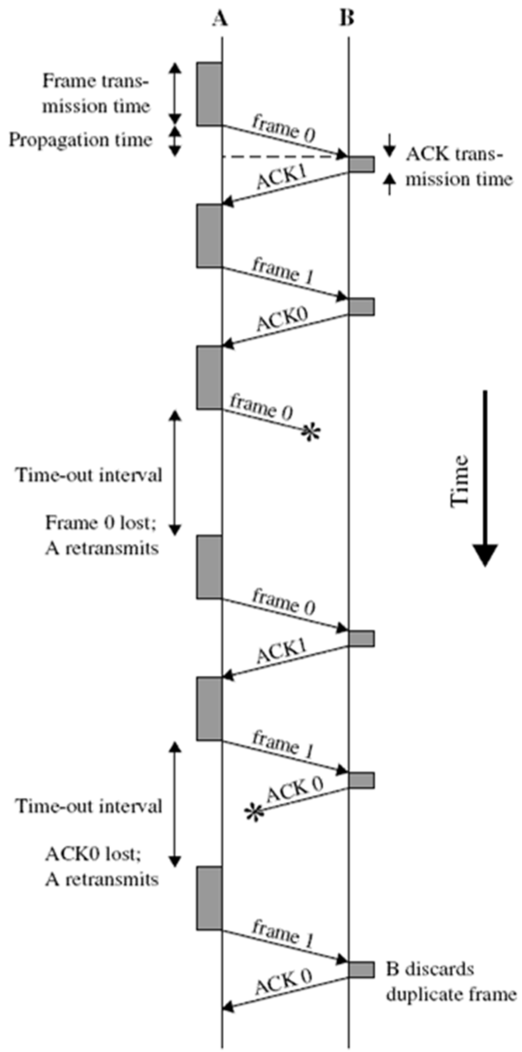
- The most common *error control* techniques use some or all of these elements.
- Collectively these *error control* techniques are called *Automatic Repeat Request* (ARQ)
- Three ARQ techniques are in common use:
  - Stop-and-wait ARQ.
  - Go-back-N ARQ.
  - Selective-reject ARQ.

Stop-and-Wait ARQ

- This technique is an enhancement to the *Stop-and-Wait* Flow Control technique:
  - Recall that the Tx transmits a frame and awaits an ACK before continuing.
  - If the frame contains an error, the Receiver simply discards it and does not return an ACK.
  - If the frame is not acknowledged after a *time-out* period the Transmitter simply retransmits the frame.

Stop-and-Wait ARQ

- Problem:
  - If ACK is sent but is damaged in transit, the Transmitter will resend the frame after time-out.
  - Receiver will now get <u>two</u> copies of the frame.
- Solution:
  - Frames are alternatively numbered 1 or 0 and ACKs are correspondingly numbered 0 or 1.



Go-back-N ARQ

- This technique is an enhancement to the Sliding Window Flow Control technique:
  - Recall that the Tx can send a number of frames (up to some window size) *without* having to receive some form of ACK from the Rx.
- In relation to the types of error (or non-error) conditions that can occur:
  - Successful Transmission,
  - Lost Data or Lost ACK frame,
  - Damaged Data frame.

Go-back-N ARQ

- If **NO** errors occur i.e. frames arrive in sequence and undamaged:
  - The Receiver returns a positive ACK e.g. RR4 or RNR6, with the number of the next frame expected.
  - This ACK can be sent as an explicit message in its own frame, or, it can be *piggybacked* onto an outgoing Data Frame (implicitly).
  - The Transmitter continues to send Data frames up to the agreed *window* size.

Go-back-N ARQ

- If **<u>Errors</u>** occur there are two responses depending on the type of error:
  - *Lost Frames/Damaged Frames. (Lost Frames* are detected as <u>out-of-sequence</u> frames).
  - *Lost ACKs.*
- *For Lost Frames* and *Damaged Frames:*
  - These are treated the same.
  - The Receiver sends a Reject (REJ) message with the number of the next frame expected e.g. REJ 4.

Go-back-N ARQ

- The Sending station, upon receipt of this REJ message, retransmits <u>all</u> frames starting at the number included in the message:
  - This is regardless of whether subsequent frames have arrived intact – they are simply discarded.

Go-back-N ARQ

- *For Lost ACKs*:
  - Here the Transmitter station has not received any messages (positive or negative ACKs) from the Receiver station.
- Recall the use of a Timer for each frame that is transmitted:
  - This timer has a part to play here.

Go-back-N ARQ

- Depending on whether the timer for a frame expires, the Transmitter can respond as follows:
  - If a timer has not expired it can continue to send more frames up to the *window* size. This is not considered an error condition.
  - If a timer does expire………..

Go-back-N ARQ

- The Transmitter immediately stops transmitting Data frames:
  - It then sends a ‘special’ RR message with the *poll* bit set (explained in class):
  - This is considered a *command* frame.
  - The Receiver must respond with a RR message of its own identifying the number of the next frame expected.

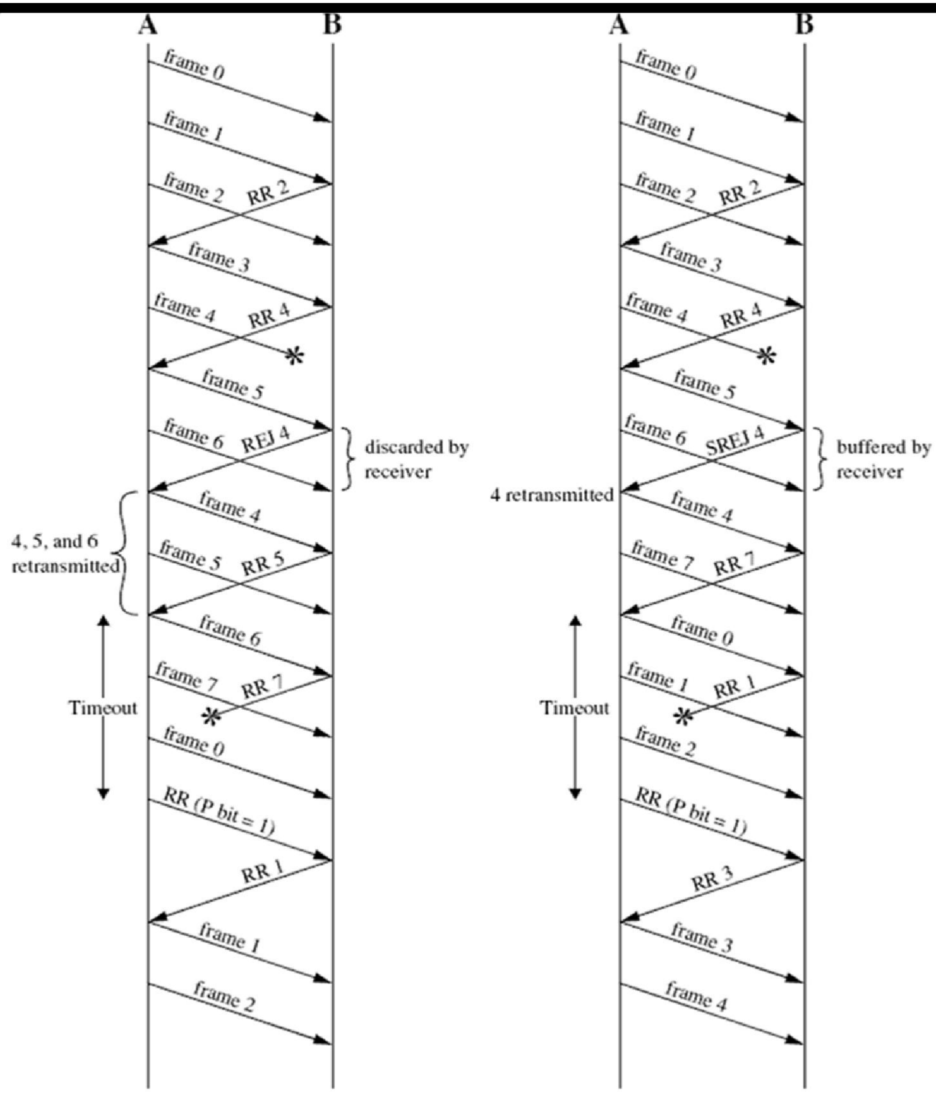
Selective-Reject ARQ

- This is very similar to the Go-Back-N technique with the exception that the only frames retransmitted are:
  - Transmitted frames for which a timer has expired i.e. a *Lost ACK* or,
  - Transmitted frames that were *Lost* or *Damaged*. The Receiver uses a slightly different Negative ACK – it's called a *Selective Reject* (SREJ) message.
  - The number of the frame to be re-transmitted is also included e.g. SREJ 5.
- This is considered more efficient than Go-Back-N as it minimises the amount of re-transmitted frames.

Selective-Reject ARQ

- However, this technique is also more complex than Go-Back-N as the Receiver station has to:
  - Store frames received <u>after</u> a Damaged frame has been rejected.
  - It has to re-sequence retransmitted frames as they arrive *out-of-sequence*.
  - This requires more complex logic on both the Transmitter and Receiver stations.
- Hence *Go-back-N* ARQ is often preferred over *selective-reject* ARQ.

Go-back-N and Selective-Reject ARQ



(a) Go-back-N ARQ (b) Selective-reject ARQ