# Computer Networks
# Chapter 2 - Getting Connected

Prof. Marino Miculan

# Problems

- In Chapter 1 we saw networks consists of links interconnecting nodes. How to connect two nodes together?
- We also introduced the concept of “cloud” abstractions to represent a network without revealing its internal complexities. How to connect a host to a cloud?

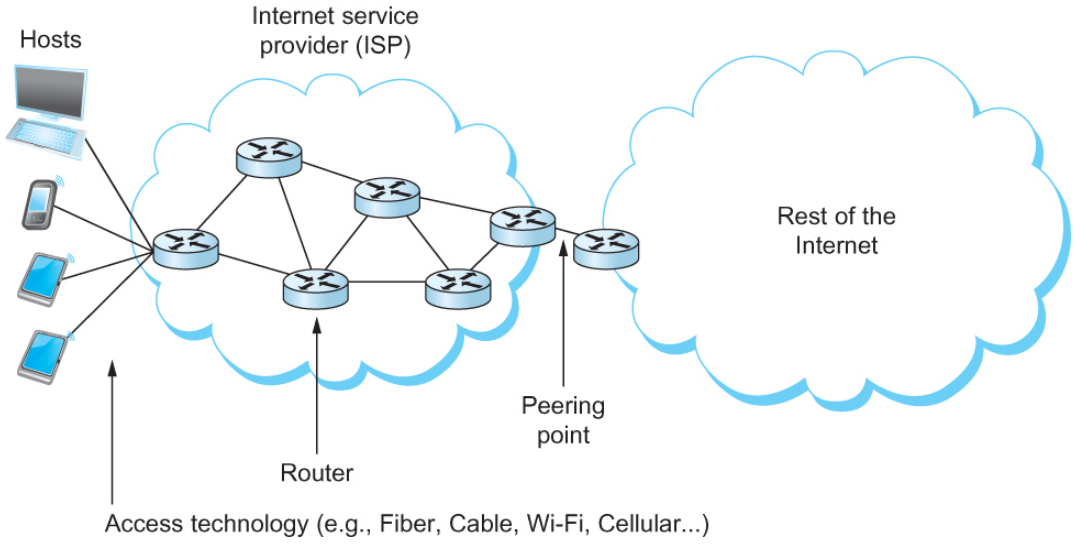
# Chapter Outline

- Perspectives on Connecting nodes
- Encoding
- Framing
- Error Detection
- Reliable Transmission
- Ethernet and Multiple Access Networks
- Wireless Networks

# Chapter Goal

- Exploring different communication medium over which we can send data
- Understanding the issue of encoding bits onto transmission medium so that they can be understood by the receiving end
- Discussing the matter of delineating the sequence of bits transmitted over the link into complete messages that can be delivered to the end node
- Discussing different technique to detect transmission errors and take the appropriate action
- Discussing the issue of making the links reliable in spite of transmission problems
- Introducing Media Access Control Problem
- Introducing Carrier Sense Multiple Access (CSMA) networks
- Introducing Wireless Networks with different available technologies and protocol

# Perspectives on Connecting




An end-user's view of the Internet

# Links

- All practical links rely on some sort of electromagnetic radiation propagating through a medium or, in some cases, through free space
- One way to characterize links, then, is by the medium they use
  - Typically **copper wire** in some form (as in Digital Subscriber Line (DSL) and coaxial cable), → RAME
  - **Optical fiber** (as in both commercial fiber-to-the-home services and many long-distance links in the Internet's backbone), or
  - **Air/free space** (for wireless links) → ACCESSO SATELLITARE

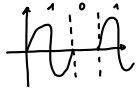
MEZZI per i collegamenti

le informazioni possono essere trasmesse via cavo variando alcune proprietà fisiche, es. TENSIONE o CORRENTE

# Links

Un mezzo ha un range di frequenze che lo possono attraversare, es. la luce non attraversa il muro

- Another important link characteristic is the **frequency**
  - Measured in hertz, with which the electromagnetic waves oscillate
- Distance between the adjacent pair of maxima or minima of a wave measured in meters is called wavelength
- Speed of light divided by frequency gives the wavelength.
  - Ex. (phone communication): Frequency on a copper cable range from 300Hz to 3300Hz; Wavelength for 300Hz wave through copper is:
    speed of light on copper / frequency = about $2 \times 10^8 / 300 = 667 \times 10^3$ meters.
  - (in fact, the speed depends of the kind of cable)
- Placing binary data on a signal is called **encoding**. → codifica dei bit, trasmettersi con delle onde



- Modulation involves modifying the signals in terms of their frequency, amplitude, and phase (B unchanged!)

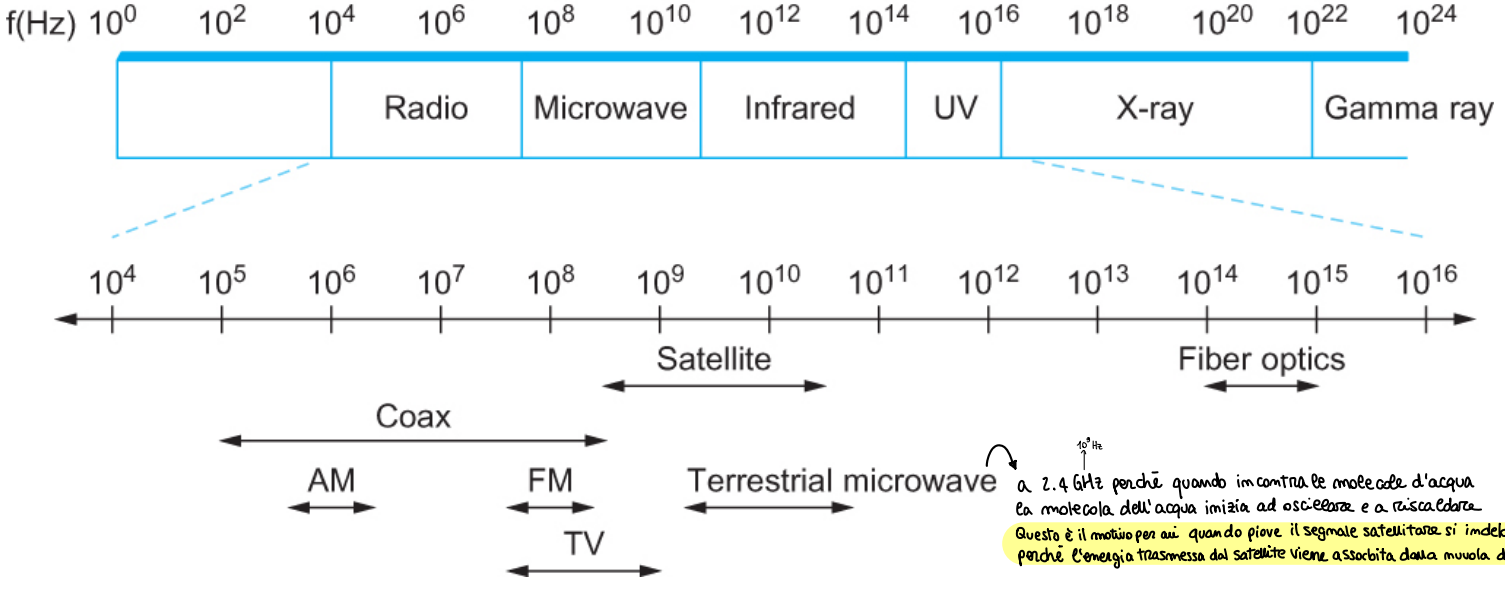
↳ MODULARE, cambiare il segnale

- Ex.: shifting a 0-3000 Hz voice signal into phone link frequency range 300-3300 Hz.

# Links

più aumento la frequenza più è debole verso gli ostacoli

- Electromagnetic spectrum




$10^9$ Hz

a 2.4 GHz perché quando incontra le molecole d'acqua la molecola dell'acqua inizia ad oscillare e a riscaldarsi

Questo è il motivo per cui quando piove il segnale satellitare si indebolisce perché l'energia trasmessa dal satellite viene assorbita dalla nuvola d'acqua

# Links

| Service | Bandwidth (typical) |
|---|---|
| Dial-up | 28–56 kbps |
| ISDN | 64–128 kbps |
| DSL | 128 kbps–100 Mbps |
| CATV (cable TV) | 1–40 Mbps |
| FTTH (fibre to the home) | 50 Mbps–1 Gbps |

Common services available to connect your home

# Encoding - Signals and Symbols → come codificare i bit sul canale

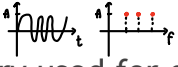
- But how much data can we encode and transmit on a channel?
- There are two aspects to consider:
  - How many bits we can stuff in a symbol
  - How many symbols we can transmit in a second

→ è uno stato in cui si trova il canale e che cambia spesso
Se uno stato è fisso non trasmetto niente

- A symbol is any state in which the link can be set by transmitter, and observed by receiver

→ un possibile stato in cui è il canale

- Symbols are what are actually transmitted on the channel. Examples:
  - Voltage levels (like in circuits) → es il filo può essere a 0V o 5V
  - Amplitude of a carrier frequency (AM) 

(la frequenza è la stessa, vario l'ampiezza)
  - Frequency shift (FM) 

(la ampiezza è la stessa, vario frequenza)
  - Phase shifts (PSK. Very used for digital encoding)
  - On/off light
  - Letters on the paper. Each letter + space is the alphabet
  - Language phonemes in the voice
  - Smoke puffs, etc.

=> La trasmissione di segnali analogici è ottenuta variando la TENSIONE nel tempo, per rappresentare un flusso di informazioni

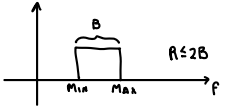
# Encoding - Symbols and Nyquist-Shannon theorem

- A symbol is any state in which the link can be set by transmitter, and observed by receiver
- The faster we can change the state of the line, the faster we can send symbols, the more data we can transmit over the channel
- The number of symbols per second is called **baud rate**, usually denoted with R, and measured in **baud = symbol/s**
  - ↳ #max simboli in un secondo
- **Nyquist-Shannon theorem**: on a channel of bandwidth B we can transmit symbols at rate no more than 2B:

  **R = 2B**

  ↳ max numero di simboli al secondo

  ↳ larghezza banda o ampiezza dello spettro




- (actually this is a corollary of the real N-S theorem, which says that with a sampling frequency R we can correctly reconstruct a signal of frequency B up to R/2)

Una linea a n baud trasmette n simboli al secondo e un BAUD può contenere (essere riconosciuto) più bit es. un simbolo è 6 bit

⇒ bitrate = 14,4 Kb/s

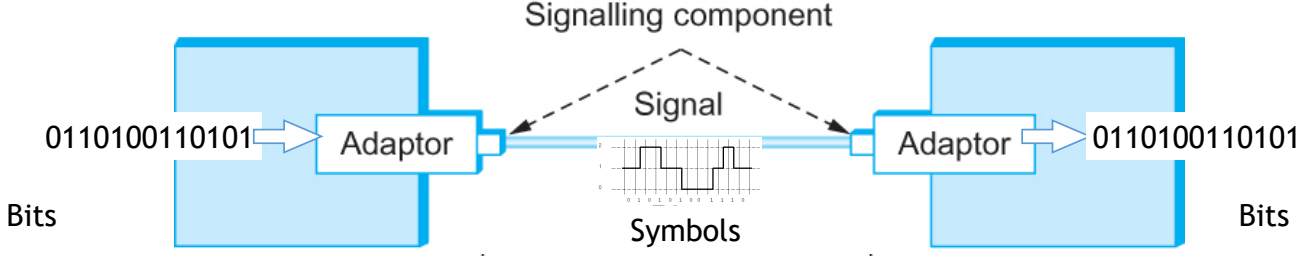
baud rate = $\frac{14,4 \cdot 10^3}{6} = 2,4 \cdot 10^3$ baud/s

# Encoding - Signals and Symbols

- If we have an alphabet of M symbols, the number of bits that each symbol can represent is $\log_2(M)$
- For instance, if we encode bits with an electric tension over a wire, within a given voltage range, we can divide this range into
  - 2 levels => 2 symbols = 1 bit per symbol
  - 4 levels => 4 symbols = 2 bits per symbol
  - 128 levels => 128 symbols = 7 bits per symbol

→ "divido" la TENSIONE in base ai livelli che mi servono e così sto inviato $\log_2(N)$ bit in un colpo solo

- To transmit a tuple of bits, the encoder signals the symbol (i.e., sets the tension) corresponding to the given tuple. The decoder does the converse.

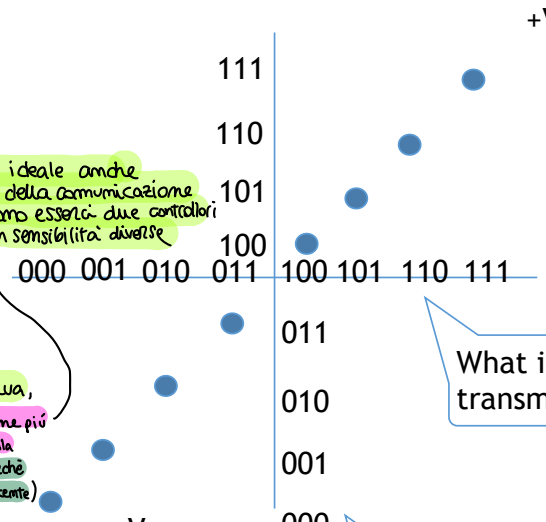


# Encoding - Symbols

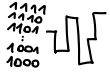
→ il computer lavora in base 2 e per inviarlo in segnali "divide" la tensione in vari intervalli (in base a quale base mi conviene, es. 8 o 16) e una volta ricevuto viene riconvertito in binario.
=> lo faccio per risparmiare simboli

- Example: a range -V — +V divided in 8 levels → associo ad ogni valore di tensione 3 bit
- Symbols are the 8 possible voltage levels; each carries 3 bits
- On an ideal channel, we could divide the signal range as much as we want, and stuff as many bits as we want on one symbol → si ma non è detto che il bit che mandi sia quello che arriva, a causa di DISTURBI e INTERFERENZE (infatti più "suddivido" la tensione più rischio errori nella trasmissione perché vado nella tacchetta adiacente)
  - Infinite capacity for everyone!

non esiste il canale ideale anche perché agli estremi della comunicazione possono esserci due controllori con sensibilità diverse




Tx → 1101 1010 1110 1111
13 10 14 15
↳ non vengono inviati sequenzialmente, bensì istantaneamente con i LIVELLI es.




converto in $b_{16}$ : D A E F e uso solo 4 simboli al posto che 16, poi in RX converto da $b_{16}$ a $b_2$

# Encoding - Noise on the channel

- But real channels have **NOISE**, which is a *random* signal added to the good one
  - Thermal noise on cables, distortions, interferences, technological limits on the precision of encoders and the discriminating power of decoders, etc.
- If the noise is too powerful, a symbol can be changed in the channel, and then misinterpreted by the decoder
  - eg. The tension can increase enough to fall in the next one
- The important factor is the **signal-to-noise ratio**, **SNR:** the ratio between the average signal power S and the average noise power N

$$SNR = S/N \quad \text{[pure number]}$$

# Encoding - Noise on the channel

- Often SNR is expressed in **decibel** (dB):

$$SNR = 10*\log_{10}(S/N) \quad [dB]$$

- We sometimes say that this is the "noise level".
- An increment of 3 dB corresponds to about doubling the S/N
- An increment of 10 dB corresponds to increasing the S/N ten times
- For instance
  - SNR=0 dB means S/N = 1, so S=N
  - SNR=10 dB means S/N = 10
  - SNR=20 dB means S/N = 100
  - SNR=30 dB means S/N = 1000
  - SNR= -20 dB means S/N = 0.01

# Encoding - Link Capacity and Shannon-Hartley Theorem

- The famous Shannon-Hartley theorem gives the upper bound to the capacity of a link in terms of bits per second (bps) as a function of SNR of the link and the available bandwidth B (in Hz).

$$C = B \log_2(1+S/N) = B \log_2(1+SNR) \quad [bps]$$

↳ il 2 per passare da Hz a bit; Hz; ↳ $\frac{S}{N}$

→ Se il segnale è debole rispetto al rumore allora $\frac{S}{N}$ SNR tende a 0 e quindi $B \cdot \log_2(1+0) \to 0$, non riusciamo a trasmettere indipendentemente dalla banda

- Ex: phone link: le frequenze vanno da 300 a 3300 Hz ⇒ perciò la banda (RANGE) è 3000 Hz
  - B = 3300 – 300 = 3000 Hz, S is the (average) signal power, N the average noise power.
  - If there is 30 dB (↳ SNR) of noise then S/N = 1000. (perché ricordando che $SNR = 10 \cdot \log_{10}(S/N)$ [dB])
  - Now C = 3000 x $\log_2(1001)$ = 30 kbps.

→ anche se la banda è piccola ma c'è poco rumore, io posso trasmettere tanta informazione

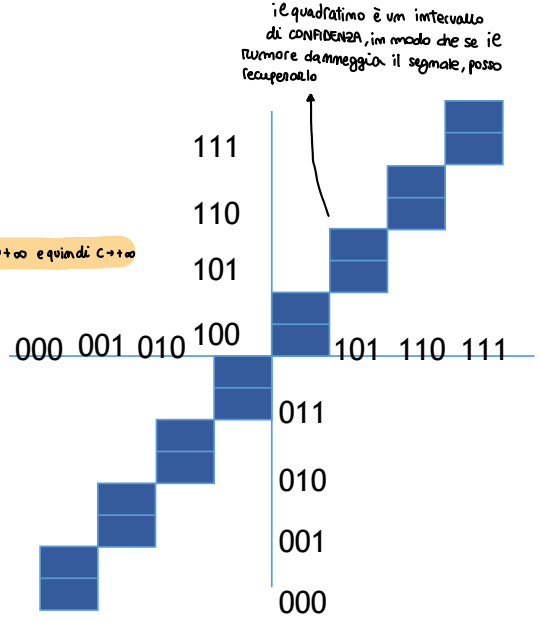
- To get a larger capacity, we need to improve the bandwidth, or the quality of the line (SNR), or both

↳ nel calcolo è $B \cdot \log_2(1+\frac{S}{N})$ (B: BANDA; $\frac{S}{N}$: QUALITÀ DELLA RETE)

→ se c'è troppo rumore devo diminuire il #simboli, se ce ne è poco posso aumentarli

# Encoding: Link Capacity and Shannon-Hartley Theorem

- Capacity is proportional to the bandwidth: the larger B, the faster the rate signals can be commuted (i.e. smaller time slot for sending a symbol)
- Dependency on $\log_2(1+S/N)$: the larger S/N, the more bits can be encoded on a signal level.
- Ex: N=S/7 => eight levels (4 positive, 4 negative) can be read by the decoder with no noise interference. Hence, 3 bits can be encoded on each level (noise: blue areas)



ES. ESAM 1

- 15 fonemi/s e ci sono 45 fonemi

qual è il throughput → $bit/_{fonema} = \log_2(45)$ → bit che rappresentano 1 fonema = 5,49 → per codificare 1 fonema uso 5,5 bit

il bit-rate $C = 5,49 \cdot 15 = 82,3$ bit/s
$\left[bit/_{fonema}\right]\left[fonema/_{s}\right]$

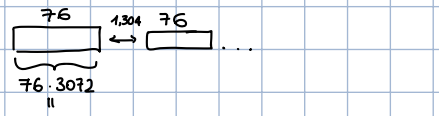
se SNR = 10 dB, qual è l'ampiezza di banda?

$B = ?$ $SNR = 10\,dB = 10 \log_{10}(S/N) \Rightarrow 10 = 10 \log_{10}(S/N) \Rightarrow 1 = \log_{10}(S/N) \Rightarrow \frac{S}{N} = 10$

$\Rightarrow 82,3 = B \cdot \log_2(1+10) \Rightarrow B = \frac{82,3}{\log_2(11)} = 23,8$ Hz

- ogni simbolo porta 3072 bit e dura 1,246 ms

FRAME ha 76 simboli e i frame sono separati da una pausa di 1,304 ms




↳ 1,246 · 76 = 94,7 ms + 1,3 = 96 ms per FRAME + PAUSA

$C = \frac{233472}{96} = 2,4 \cdot 10^6$ bit/s = 2,4 Mb/s

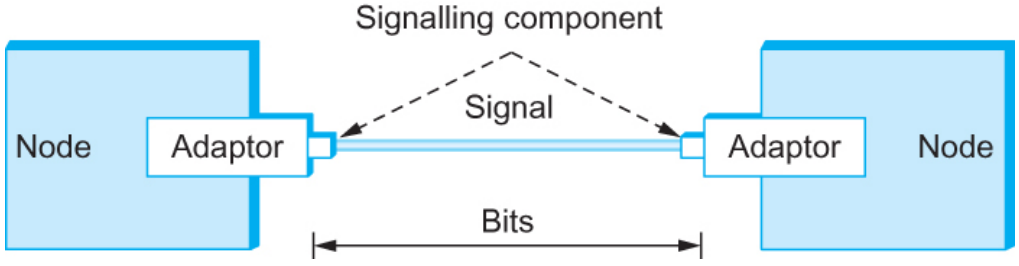
B = 1536 KHz (anche se la dab ha banda 160 Hz, me usa un piccolo intervallo)

$C = B \cdot \log_2(1+\frac{S}{N})$

$2,4 \cdot 10^6 = 1536 \cdot 10^3 \log_2(1+S/N)$ (1,536 · 10^6)

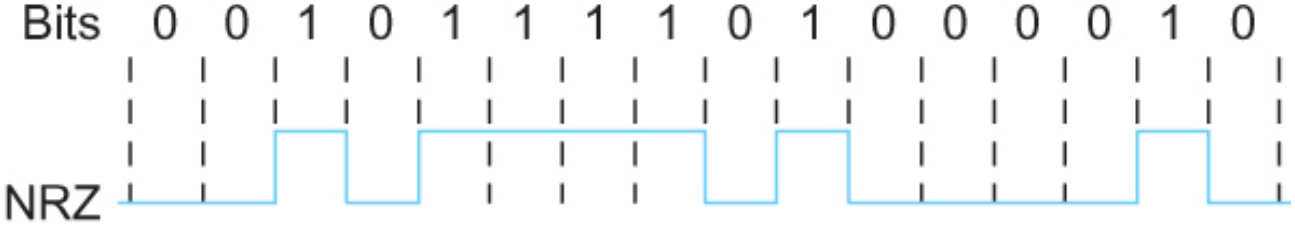
$\log_2(1+S/N) = \frac{2,4}{1,536} \Rightarrow 1 + S/N = 2^{\frac{2,4}{1,5}} \Rightarrow \frac{S}{N} = 2^{\frac{2,4}{1,5}} - 1$

# Encoding




Signals travel between signalling components; bits flow between adaptors

↳ alle estremità




NRZ encoding of a bit stream → i dati vengono passati direttamente come tali in uscita

# Encoding - Problems with NRZ

Come si fa a capire se è alto o basso?


il computer non sa dove è la "linea di mezzo" (fra 0 e 1), che riceve solo descrizioni istantanee

- **Baseline wander**
  - The receiver keeps an average of the signals it has seen so far (↳ VALORE MEDIO)
  - Uses the average to distinguish between low and high signal
  - When a signal is significantly low than the average, it is 0, else it is 1
  - Too many consecutive 0's and 1's cause this average to change, making it difficult to detect

↳ fa per capire più che altro come gestire con i picchi alti per distinguere poi quelli più bassi.

Tengo il valore di MEDIA e se un segnale è tanto più basso della MEDIA allora è 0, se è più alto è 1

↳ la media si alza o abbassa troppo e rischia di confondere valori. es. se la media → 1 e arriva 0,98 rischia, essendo sotto la media, di approssimare a 0.

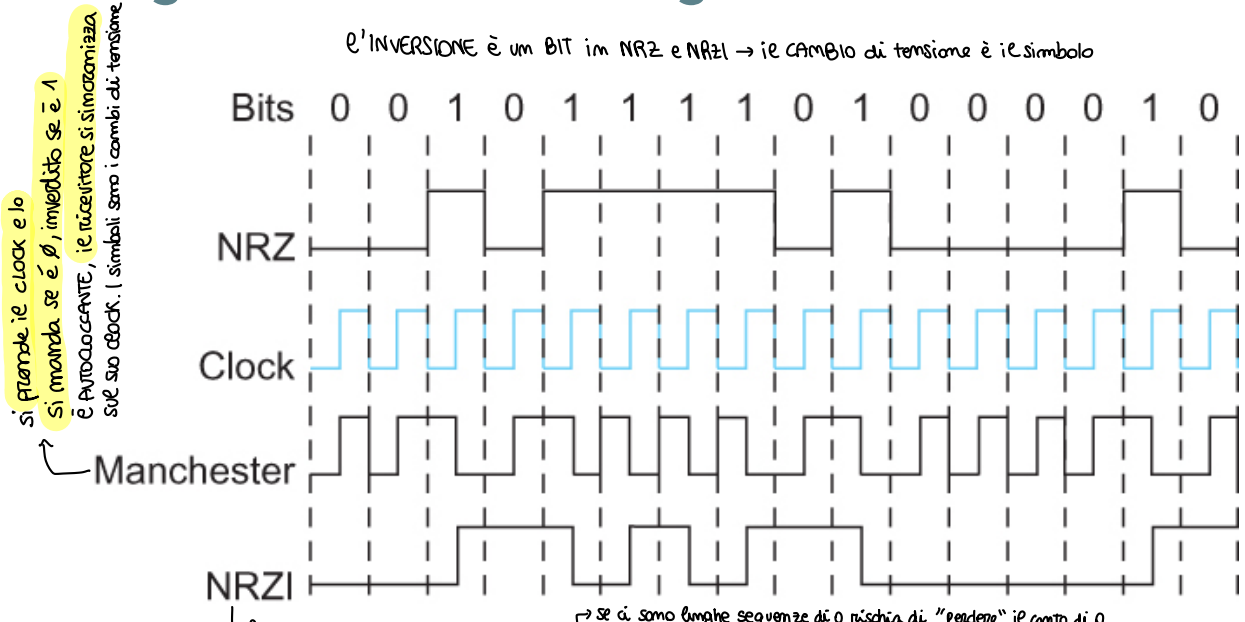
- **Clock recovery** → difficilissimo tenere sincronizzati due clock
  - Frequent transition from high to low or vice versa are necessary to enable clock recovery → quando hai tanti "su e giù" il clock si risincronizza, se hai lunghi periodi statici il clock tende a perdersi
  - Both the sending and decoding process is driven by a clock
  - Every clock cycle, the sender transmits a bit and the receiver recovers a bit
  - The sender and receiver have to be precisely synchronized

# Encoding - NRZI and Manchester

- **NRZI**: Non Return to Zero Inverted
  - Sender makes a transition from the current signal to encode 1 and stay at the current signal to encode 0
  - Solves for consecutive 1's, ma non quello degli 0 consecutivi (perché con lo ø è statico) → - Soffre di BASELINE WANDER (solo per lo ø, no 1) - Soffre di clock-sync
- **Manchester encoding** → non da problemi con lunghe sequenze di 0 e 1 (quindi non perde sincronismo)
  - Merging the clock with signal by transmitting Ex-OR of the NRZ encoded data and the clock
  - Clock is an internal signal that alternates from low to high, a low/high pair is considered as one clock cycle
  - In Manchester encoding
    - 0: low to high transition
    - 1: high to low transition

- NON soffre di BASELINE WANDER
- NON soffre di CLOCK-SYNC
- DIMEZZA la BANDA perché in un CICLO di CLOCK il ricevitore verifica 2 volte la direzione della salita/discesa in 1 istante

# Encoding - different encodings



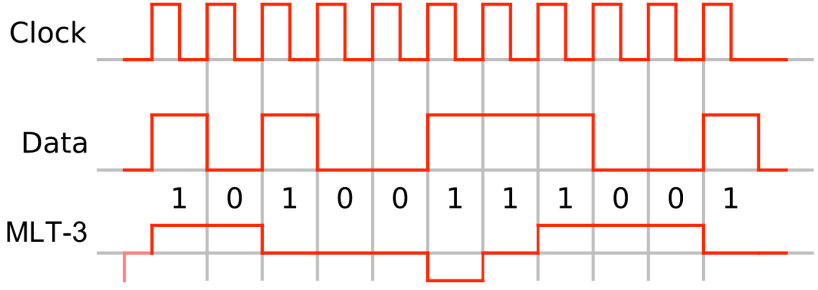

Different encoding strategies

# Encoding

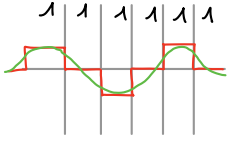
- Problem with Manchester encoding
  - Doubles the rate at which the signal transitions are made on the link
    - Which means the receiver has half of the time to detect each pulse of the signal
  - The rate at which the signal changes state is called the link's **baud rate**
  - If we consider the levels as state, in Manchester we need 2 states to represent one bit (low→high=0, high→low=1) thus **the bit rate is half the baud rate** (2 states = 1 bit)

# Encoding - MLT-3 → Ethernet si basa su questo principio

- MultiLine Transmission, with 3 levels: very similar to NRZI, but it uses 3 levels of signal: -1, 0, +1




se trasmetto solo 1111...




Se invece è NRZI con tutti 1 ottengo un clock

- Signal changes from one level to the next at the beginning of a "1" bit. No transition at the beginning of "0" bit
- When it reaches one extreme (1 or -1), it changes direction.
- Risk of losing synchronisation for long sequences of "0" → perché non cambia mai verso

# Encoding - 4B/5B encoding → rompe le lunghe sequenze di ø

- Insert extra bits into bit stream so as to break up the long sequence of 0's and 1's
- Every 4-bits of actual data are encoded in a 5- bit code that is transmitted to the receiver → in modo che non puoi mai avere più di 3 zeri consecutivi (guarda la tabella)
- 5-bit codes are selected in such a way that *each one has no more than one leading 0 (zero) and no more than two trailing 0's*.
- In this way, three consecutive 0's can never appear, even between two different 5-bit codes.
- Efficiency: 80%
- Delay: 5 bits (receiver needs to get 5 bits to decode 4 bits).
- Used often in conjunction with MLT-3 (Ethernet 100 Mbit)

# Encoding - 4B/5B encoding

FAST ETHERNET (100 Mb/s)
Usa 4B/5B, in realtà trasmessi 125 Mb/s

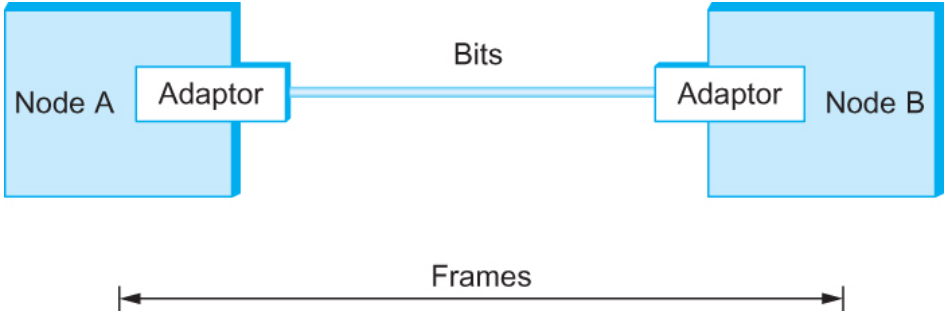
- 16 codes are left out from encoding
- 7 codes violate the rule *no more than one leading 0 (zero) and no more than two trailing 0's.*
  In particular:
  - 11111 – when the line is idle (quando mom sto trasmettendo niente)
  - 00000 – when the line is dead (la linea è scomessa)
- 6 of these are used for control
  - 00100 – to mean halt
  - 00111 - reset
  - …

| Data Sequence | Encoded Sequence | Control Sequence | Encoded Sequence |
|---|---|---|---|
| 0000 | 11110 | Q (Quiet) | 00000 |
| 0001 | 01001 | I (Idle) | 11111 |
| 0010 | 10100 | H (Halt) | 00100 |
| 0011 | 10101 | J (Start delimiter) | 11000 |
| 0100 | 01010 | K (Start delimiter) | 10001 |
| 0101 | 01011 | T (End delimiter) | 01101 |
| 0110 | 01110 | S (Set) | 11001 |
| 0111 | 01111 | R (Reset) | 00111 |
| 1000 | 10010 | | |
| 1001 | 10011 | | |
| 1010 | 10110 | | |
| 1011 | 10111 | | |
| 1100 | 11010 | | |
| 1101 | 11011 | | |
| 1110 | 11100 | | |
| 1111 | 11101 | | |

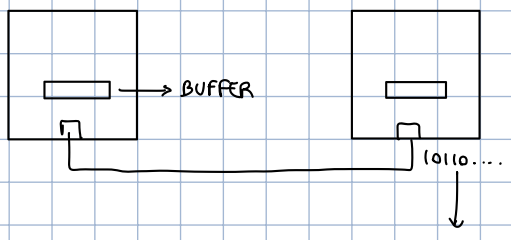
# Framing

ORA SIAMO nel livello 2 BIT→ FRAME + controlli d'errore
→ d'ora in poi pensiamo di trasmettere bit, escludiamo i dettagli elettrici

- We are focusing on packet-switched networks, which means that blocks of data (called *frames* at this level), not bit streams, are exchanged between nodes.
- It is the network adaptor that enables the nodes to exchange frames.




Bits flow between adaptors, frames between hosts





all'interno di questa sequenza

1. deve riconoscere dove inizia e dove finisce il frame
   ovvero frame header + data + data link trailer

2. vedere se questi dati sono corretti (se bit sono stati alterati)

=> in ricezione raggruppo i bit (che sono un flusso continuo) in byte e poi, ad es. con i caratteri sentinella, inizio a riconoscere il frame

# Framing

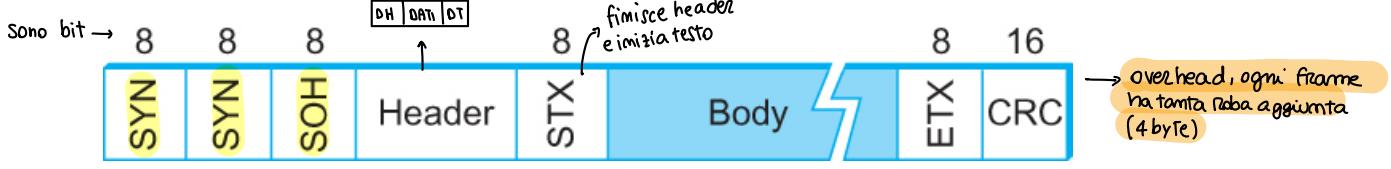
- When node A wishes to transmit a frame to node B, it tells its adaptor to transmit a frame from the node's memory. This results in a sequence of bits being sent over the link (which can be point-to-point or shared).
- The adaptor on node B then collects together the sequence of bits arriving on the link and deposits the corresponding frame in B's memory.
- Recognizing exactly what set of bits constitute a frame—that is, determining where the frame begins and ends—is the central challenge faced by the adaptor

# Framing

- **Byte-oriented Protocols**
  - To view each frame as a collection of bytes (characters) rather than bits
  - BISYNC (Binary Synchronous Communication) Protocol
    - Sentinel-based → caratteri sentinella che indicano l'inizio e la fine
    - Developed by IBM (late 1960)
  - DDCMP (Digital Data Communication Protocol)
    - Byte-count based
    - Used in DECNet
  - (but other variants are possible)

# Framing → caratteri sentinella che indicano l'inizio e la fine

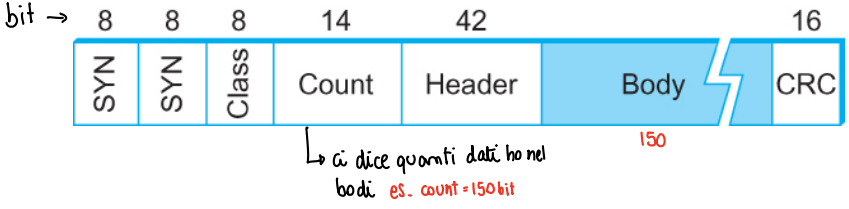
- BISYNC – *sentinel* approach (lavoro fatto in fase di decodifica)
  - Frames transmitted beginning with leftmost field
  - Beginning of a frame is denoted by sending a special SYN (*synchronize*) character
  - Data portion of the frame is contained between special *sentinel* character STX (start of text) and ETX (end of text)
    - SOH : Start of Header
    - DLE : Data Link Escape ("escaping"): Body vs. ETX → poi non confondere una parte del body con l'ETX
    - CRC: Cyclic Redundancy Check → trailer



# Framing

- Byte-counting approach: DDCMP
  - *count*: how many bytes are contained in the frame body
  - CRC: Cyclic Redundancy Check as before
  - If count is corrupted: Framing error

quanti dati ho nel body



# Framing

- Bit-oriented Protocol: frames as sequences of bits
  - Beginning and Ending Sequences
  - HDLC (IBM): High Level Data Link Control
    - Delimiting sequence: 0 1 1 1 1 1 1 0


HDLC Frame Format

  - And what if delimiting sequence occurs within the frame?

# Framing

- HDLC Protocol, on the sending side:
  - any time five consecutive 1's have been transmitted from the body of the message (i.e. excluding when the sender is trying to send the distinguished 01111110 sequence), the sender inserts 0 before transmitting the next bit ("bit stuffing")
  - Eg: 011111xy becomes 0111110xy

Rompo le sequenze di 1 (nel body) per evitare che vengano scambiati per delimitatori (01111110)

es. 011111 e qui metto uno 0 poi il valore che c'era (es. 1) e ci aggiungo un altro 0 => così non lo scambio per delimitatore

# Framing

- HDLC Protocol, on the receiving side:
- After having received 5 consecutive 1's, read next bit
  - If 0 : Stuffed, so discard it and proceed
  - If 1 : Either End of the frame marker, or Error has been introduced in the bitstream. To tell, look at the next bit
    - If 0 ( 01111110 ) then End-of-frame marker => frame ended
    - If 1 ( 01111111 ) then Error => discard the whole frame The receiver needs to wait for next 01111110 before it can start receiving again

# Framing

- **PPP (Point-to-Point Protocol)** is commonly run over Internet links


- Derived from HDLC, uses sentinel approach and bit stuffing
- Special start of text character denoted as **Flag**
  - 0 1 1 1 1 1 1 0
- Address, control : default numbers
- Protocol for demux : IP / IPX / …
- Payload : negotiated by LCP (default: 1500 bytes)
- Checksum : for error detection (CRC-16 or CRC-32)

↳ meglio avere frame grandi per ridurre overhead

meglio avere frame piccoli per ridurre la proprietà d'errore

# Error Detection

- Errors are introduced into frames due to *electrical interference* and *thermal noises*
  - The former gives rise to (deterministic) bursts -> sequences of bits destroyed
  - The latter to rare, supposedly independent one-bit spikes
- **Detecting** errors is a fundamental function of the datalink layer
- Approaches when the recipient detects an error:
  - **Drop** silently the frame - as it was never sent → il più semplice
    - Upper layers may recognise the packet loss, and take suitable countermeasures (if they need it)
  - **Notify** the sender that the message was corrupted, so the sender can send again. ↱ ritrasmissione dei dati a livello data link
    - If the error is rare, then the retransmitted message will (likely) be error-free
  - **Forward Error Correction**: Using some error correction algorithm, the receiver reconstructs the message even in presence of errors (up to some extent)

↳ il sender oltre a mandare informaz. per capire se è errato può anche mandare codici di correzione → OVERHEAD
↳ spesso quando la trasmissione è unidirezionale (es. antenna broadcast sky) dove non posso chiedere di rimandarmi i pacchetti

# Bit errors: simple examples

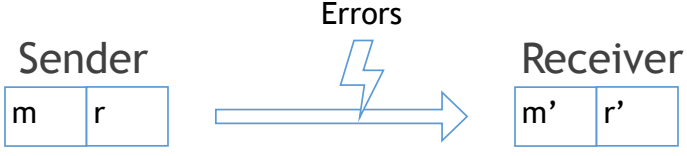
- Joint occurrence of two rare events A and B:
  - $P[A \cup B] = 1 - P[\sim(A \cup B)] = 1 - P[(\sim A) \cap (\sim B)]$
    $= 1 - (1-p_A)(1-p_B) = p_A + p_B - p_A p_B \approx p_A + p_B$
- At least one error in a 10 kb packet, $p_E \approx 10^{-7}$, bit independence approximation:
  - $P[\{\text{one error in 10 kb}\}] = 1 - P[\{\text{no error in 10 kb}\}] = 1 - (p(\text{bit corretti}))^n$
    $= 1 - (1-10^{-7})^{10000} = 0.0009995 \approx 10^4 \cdot 10^{-7} = 10^4 \cdot p_E$
- Two errors in the same packet, same error, same approximation:
  - $P[\{\text{two errors in 10 kb}\}] \approx (1-p_E)^{10000-2} \cdot (p_E)^2 \cdot C(10^4,2) =$
    $(1-10^{-7})^{9998} \cdot 10^{-14} \cdot 10^4 \cdot (10^4-1)/2 \approx 10^{-14} \cdot 5 \cdot 10^7 = 5 \cdot p_E$

# Error Detection

- Basic Idea of Error Detection:
  - **To add redundant information to a frame that can be used to determine if errors have been introduced**
- Extreme Case: transmitting two complete copies of data
  - Identical → No error
  - Differ → Error
- Poor Scheme?
  - n bit message, n bit redundant information
  - Error can go undetected → se un errore è presente nella stessa posizione in entrambi non verrà rilevato
- In general, we can provide strong error detection technique
  - n bits message, k redundant bits
  - Usually k is constant, and << n
  - (In Ethernet, a frame carrying up to 12,000 bits of data requires only 32-bit CRC)

# Error Detection

- Extra k bits are redundant: They add no new information to the message
- Derived from the original message using some algorithm
- Both the sender and receiver know the algorithm
- Result of the transmission:




- Receiver computes r'' using m' (and the same algorithm)
  - if r' = r'' → no error
  - if r' differs from r'' → error

# Error Detection

- **CRC** (Cyclic Redundancy Check)
  - Common technique for detecting transmission error
  - Used in DDCMP, HDLC, CSMA/CD, Token Ring, etc.
- Other approaches
  - **Parity**
  - **Two Dimensional Parity** (BISYNC)
  - **Checksum** (IP, UDP, TCP)

# One-dimensional parity

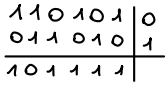
→ aggiungere 1 bit di parità

- "simple" (one-dimensional) parity involves adding one extra bit to a n-bit code to balance the number of 1s in the byte.
- For example, on 7-bit codes
  - Odd parity sets the eighth bit to 1 if needed to give an odd number of 1s in the byte
  - Even parity sets the eighth bit to 1 if needed to give an even number of 1s in the byte
- Each byte is checked separately from the others
- An even number of errors in a byte goes undetected → non rileva modifiche pari di bit

| | Parity bits |
|---|---|
| 0101001 | 1 |
| 1101001 | 0 |
| 1011110 | 1 |
| 0001110 | 1 |
| 0110100 | 1 |
| 1011111 | 0 |

↓
il frame viene letto a byte e quindi ad ogni byte viene aggiunto un bit di parità

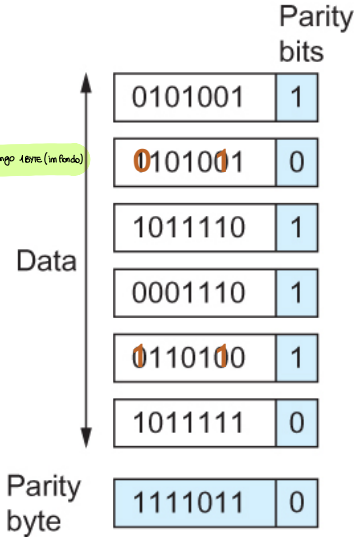
# Two-dimensional parity




anche i bit di parità possono essere modificati

- Two-dimensional parity does a similar calculation for each bit position across each of the bytes contained in the frame
- This results in an extra *parity byte* for the entire frame, in addition to a parity bit for each byte → controllo i bit di parità sia in orizzontale che verticale
- Two-dimensional parity catches all 1-, 2-, and 3-bit errors and most 4-bit errors
- 4-bit errors are not detected if there are "aligned" in same rows AND in the same columns → se si cambiano 4 bit a quadrato e allora non li rileva

→ aggiungo 1 BIT in fondo ad ogni BYTE ed inoltre aggiungo 1 BYTE (in fondo) per tutti i bit di controllo delle colonne




Four bit errors which are not detected by parity bits, nor by two-dimensional parity

# Internet Checksum Algorithm

→ sommo 16 bit alla volta, coppie di byte

↳ viene usato a livelli 3-4 ma non a livello 2

- Consider the data as a sequence of 16-bit integers
- Add them together using 16-bit ones-complement arithmetic (explained next slide) and then take the ones complement of the result.
  - The resulting 16-bit number is called the **checksum**
- Transmit the checksum together with the message

- The receiver performs the same calculation on the received data and compares the result with the received checksum → dal MESSAGGIO si calcola il CHECKSUM e se coincide con il CHECKSUM che ha ricevuto allora NO ERRORI
- If any transmitted data, including the checksum itself, is corrupted, then the results will not match, so the receiver knows that an error occurred
- Not used at the link level, but used at network and transport level

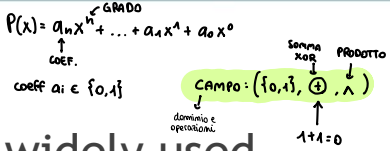
# Internet Checksum Algorithm

- In ones-complement arithmetic, a negative integer −x is represented as the complement of x; E.g.: 2 ⇔ 0010 iff -2 ⇔ 1101
- When adding numbers in ones complement arithmetic, a carryout from the most significant bit needs to be added to the result.
  - This happens because zero has two logical representations: 00..0 and 11..1: we need to skip the latter when the sum wraps around the represented domain
- Example: addition of −5 and −2 in ones complement arithmetic on 4-bit integers
  - +5 is 0101, so −5 is 1010; +2 is 0010, so −2 is 1101
  - If we add 1010 and 1101 ignoring the carry, we get 0111
  - Since there is a carry from the most significant bit causes us to increment the result, giving 1000, which is the ones complement representation of −7, as we would expect
- (The book provides a weird example: -8 cannot be represented in ones complement arithmetic!)

# Internet Checksum Algorithm

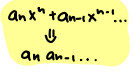
```
u_short cksum(u_short *buf, int count) {
   register u_long sum = 0;     \\ 16-bit checksum
   while (count--) {            \\ count 16-bit units
        sum += *buf++;          \\ 16-bit buffer units
        if (sum & 0xFFFF0000) {      \\ if carry
              sum &= 0xFFFF;         \\ erase carry
              sum++;                 \\ increment
              }
        }
   return ~(sum & 0xFFFF);      \\ erase last carry
}                               \\ and complement
```

# Cyclic Redundancy Check (CRC)




↳ usato ovunque

- Introduced by Wesley Peterson (1961), and widely used
- Reduce the number of extra bits and maximize protection
- Based on Modulo 2 **polynomial arithmetics**
- Given a bit string we can associate a polynomial P(x) for it
- E.g. 110001 corresponds to the polynomial of degree 5
  $1 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^5 + x^4 + 1$ → considero la stringa come somma




- Thus a bit string is just another representation for a polynomial
- A k-bit frame has a maximum degree of k-1
- Let M(x) be a message polynomial and C(x) be a given *generator polynomial*.

# Cyclic Redundancy Check (CRC)

SOMMA: $P_1(x) + P_2(x) = \sum_{i=0}^{n} (a_i \oplus b_i) x^i$

$P_1(x) + P_1(x) = \sum_{i=0}^{n} (a_i \oplus a_i) x^i = 0$ perché $1+1=0$

XOR → $-$: 0 1 / 0 0 1 / 1 1 0 ; $\oplus$: 0 1 / 0 0 1 / 1 1 0

PRODOTTO:

es. $P(x) = x^3 + x^2 + 1$ (1101)

$C(x) = x^2 + 1$ (101)

$P(x) \cdot C(x) = x^5 + x^3 + x^4 + x^2 + x^2 + 1 = x^5 + x^4 + x^3 + 1$ (1 1 0 0 1)

## Polynomial Arithmetic Modulo 2

- Polynomials can be added and multiplied (easy)
- Any polynomial B(x) can be **divided** by a divisor polynomial C(x) if B(x) is of higher degree than C(x) (note: true for any Modulo b polynomial arithmetic)
- Any polynomial B(x) can be divided **once** by a divisor polynomial C(x) if B(x) is of the same degree as C(x):
  if deg(B) = deg(C) then B(x)/C(x) = 1 + remainder
- In this case, the remainder is obtained by subtracting C(x) from B(x):
  B(x)/C(x) = 1 with remainder R(x) implies
  B(x) = 1· C(x) + R(x) and hence R(x) = B(x) – C(x).
- To subtract C(x) from B(x), we simply perform the exclusive-OR (XOR) on each pair of matching coefficients (no carry, same as summing):
  B(x) + C(x) = B(x) - C(x) = B(x) XOR C(x)

XOR

| ⊖ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| ⊕ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**SOMMA:** $P_1(x) + P_2(x) = \sum_{i=0}^{n} (a_i \oplus b_i) x^i$

$P_1(x) + P_1(x) = \sum_{i=0}^{n} (a_i \oplus a_i) x^i = 0$ perché $1+1=0$

**PRODOTTO:**

es. $P(x) = x^3 + x^2 + 1$ $(1101)$

$C(x) = x^2 + 1$ $(101)$ $\quad P(x) \cdot c(x) = x^5 + x^3 + x^4 + x^2 + x^2 + 1 = x^5 + x^4 + x^3 + 1$ $(1\ 1\ 1\ 0\ 0\ 1)$

**DIVISIONE:** $B(x) = Q(x) \cdot C(x) + R(x)$ (1 se $B(x)$ ha lo stesso grado di $C(x)$) $\quad$ grado $(R)$ < grado $(C)$ $\quad$ grado $(Q)$ = grado $(R)$ − grado $(C)$

B 110101 → faccio xor: R = 011100 $\quad$ => che sarebbe $x^5 + x^4 + x^2 + 1$ $(110101)$ / $x^5 + x^3 + 1$ $(101001)$

C 101001

alcuni bit swappati

io mando $P(x)$ e riceve $P'(x)$

$P'(x) = p(x) + E(x)$ $\quad$ es. TX: 100101...

RX: 110100...

e(x) = 010001...

resto $(P(x), c(x))$ = resto $(E(x), c(x))$

rileva gli errori facendo la divisione tra $E(x)$ e $C(x)$ => $\frac{E(x)}{C(x)}$

se il resto è 0 è molto probabile che non ci siano errori

resto è 1 allora c'è errore → scarto

NON RILEVA quando $E(x)$ è multiplo di $C(x)$

# Cyclic Redundancy Check (CRC)

- Let P(x)/C(x) leave a remainder R(x)=0.
- The receiver's viewpoint: error is a sequence of bits, i.e. a polynomial E(x)
- When P(x) is sent and P'(x) is received we have
  P'(x) = P(x)+E(x)
- The receiver compute R'(x), the remainder of P'(x)/C(x)
- R'(x) = remainder [P'(x)/C(x)] = remainder [E(x)/C(x)]
- If it nonzero, then an error (non divisible by C(x)) has occurred!

- The only thing the sender and the receiver should know is C(x).

# Cyclic Redundancy Check (CRC)

→ il controllo avviene a posteriori

- Let M(x) come from a frame with m bits and
- Let the generator polynomial C(x) comes from a frame having less than m bits: say, deg(C) = r.

$P(x) = M(x) \cdot C(x)$ (messaggio, generatore)

- The logical algorithm:
  send P(x) = M(x)C(x) (which is divisible by C(x), obviously)
  Receivers gets P'(x) = P(x) + E(x)
  Then, no errors are detected by the receiver iff
  R'(x) = reminder[P'(x)/C(x)] = reminder[E(x)/C(x)] = 0

- How to perform multiplication M(x)C(x)?

# Cyclic Redundancy Check (CRC)

The algorithm *in practice*:

SENDER

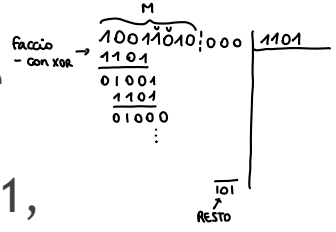
- Append r zeros to the low-order end of the frame, so it now contains m+r bits this gives the polynomial $x^r M(x)$. sto shiftando a SX di r perché moltiplico tutto per $x^r$
- Divide $x^r M(x)$ by C(x) using modulo 2 division (see algorithmic procedure next slide)
- Subtract the remainder R(x) (which is always r or fewer bits) from the string corresponding to $x^r M(x)$ using modulo 2 subtraction (immediate: R(x) XOR 0 = R(x))
- The result $P(x) = x^r M(x) - R(x) = x^r M(x) + R(x)$ is the checksummed frame to be transmitted. Notice that reminder[P(x)/C(x)] = 0

RECEIVER

- The receiver receives P'(x) = P(x) + E(x)
  and computes R'(x) = reminder[P'(x)/C(x)] = reminder[E(x)/C(x)]
  - if R'(x) = 0 then it discards r LSB's, i.e. it takes M(x) = x-r P'(x); otherwise there is an error

# Cyclic Redundancy Check (CRC)

- Example:
  M=10011010, $M(x)=x^7+x^4+x^3+x$
  C=1101, $C(x)=x^3+x^2+1$
  ↳ POL. GENERATORE
- deg(C(x))=3, so we add 3 zeros at the end of M (the CRC will have 3 bits)
- The remainder is $R(x)=x^2+1$, that is R=101
- $P(x)=x^3M(x)+R(x)$
  =10011010101




Se io scrivo il RESTO al posto dei 3 ZERI finali si annullano

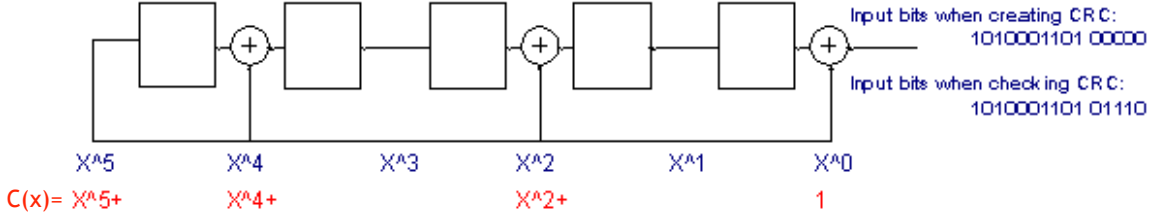



CRC Calculation using Polynomial Long Division

# Implementing the CRC algorithm in hardware

↳ con porte XOR ed è molto economico il circuito da costruire

- A r-long CRC can be easily computed by a circuit with r 1-bit registers and some XOR ports (these circuits are embedded directly in network adapters)
- The circuit is implemented as follows:
  - The register contains r bits, equal to the length of the CRC
  - There are up to r XOR gates
  - The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial C(X)
  - Bits are shifted in, from MSB to LSB
  - After all bits are shifted, the content of registers give the requested CRC
- Same circuit can be used to generate and to check the CRC
- Example:Message M=1010001101; C=110101: r=5; resulting CRC: R=01110



# Cyclic Redundancy Check (CRC)

Properties of Generator Polynomial

- Let P(x) represent what the sender sent (↳ messaggio) and P'(x) = P(x) + E(x) is the received string.
- E(x) = $x^i$ means that the i-th bit has been flipped in the corresponding position in the message P(x). (↳ messaggio alterato)
- We know that P(x)/C(x) leaves a remainder of 0, but if E(x)/C(x) leaves a remainder of 0, then either E(x) = 0 or C(x) is factor of E(x). (↳ POL. GENERATORE)
- When C(x) is a factor of E(x) we have problem; errors go unnoticed.
- If there is a single bit error then E(x) = $x^i$, where i determines the bit in error. If C(x) contains two or more terms it will never divide E(x) = $x^i$, so all single bit errors will be detected.

↳ perché se E ha 1 sola cifra e C ne ha più di una allora non si può fare la divisione

# Cyclic Redundancy Check (CRC)

→ Una codifica polinomiale con r bit di controllo può rilevare sequenze di errori di lunghezza ≤ r

- In general, it is possible to prove that the following types of errors can be detected by a C(x) with the stated properties
  - All single-bit errors, if the $x_k$ and $x_0$ terms have nonzero coefficients.
  - All double-bit errors, if C(x) has a factor with at least three terms.
  - Any odd number of errors, if C(x) contains the factor (x+1). → se E(x) contiene un numero di spari di termini (errori) allora non potrà mai essere diviso per (x+1) nel sistema a modulo 2
  - Any "burst" error (i.e., sequence of consecutive error bits) for which the length of the burst is less than k bits. (Most burst errors of larger than k bits can also be detected.)
- A polynomial C(x) can be factored as $C(x)=C_1(x)*\ldots*C_n(x)$, so if a message can be divided by C(x), it means that it can be divided by each $C_1(x),\ldots, C_n(x)$
- hence, if we want to ensure several properties *at the same time*, it is enough to use a generator polynomial which is the product of the polynomials representing each property

↳ quindi

# Cyclic Redundancy Check (CRC)

- Six generator polynomials that have become international standards are:
  - CRC-8 = $x^8+x^2+x+1$
  - CRC-10 = $x^{10}+x^9+x^5+x^4+x+1$
  - 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 = $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
- CRC-32 is used in 802.3 (Ethernet), 802.11 (WiFi), …
- PPP can use CRC-32 or CRC-16 with a different polynomial
- For many other examples see https://en.wikipedia.org/wiki/Polynomial_representations_of_cyclic_redundancy_checks

# Reliable Transmission

↳ il più usato a liv. D.L. per rilevare errori

- CRC is used to *detect* errors, not to correct them.
  - Some error codes are strong enough to correct errors up to a given limit (e.g. Hamming), but the overhead is typically too high.
- Corrupt frames must be discarded

↳ piuttosto che correggere è più economico richiedere l'invio, anche se non è sempre possibile farlo (es. satellite)

- An *unreliable* link-level protocol leaves to higher level protocols the decision to retransmit messages, if needed.
  - e.g. Ethernet, WiFi
- A *reliable* link-level protocol must recover from these discarded frames.
  - e.g. PPP
  - Obtained using a combination of two fundamental mechanisms: Acknowledgements and Timeouts

# Reliable Transmission

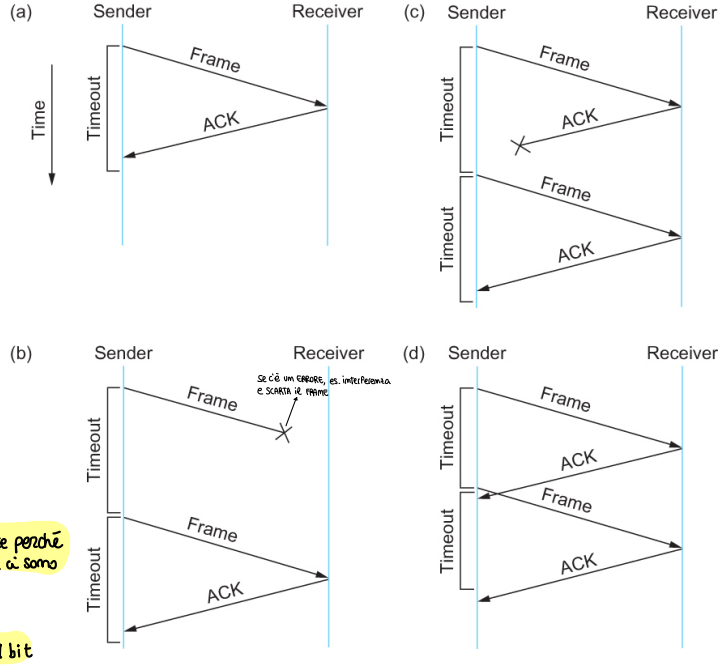
FRAME → messaggio che porta informazioni di controllo, "i dati che mi hai mandato fin'ora sono giusti"

- An *acknowledgement* (ACK) is a small control frame that a protocol sends back to its peer saying that it has received the earlier frame.
  - (A *control frame* is a frame with header only - no data).
- The receipt of an acknowledgement indicates to the sender of the original frame that its frame was successfully delivered.
- If the sender does not receive an acknowledgment after a reasonable amount of time, then it retransmits the original frame.
- The action of "waiting a reasonable amount of time" is called a **timeout**.
- The general strategy of using acknowledgements and timeouts to implement reliable delivery is sometimes called **Automatic Repeat reQuest** (ARQ).

# Stop and Wait Protocol

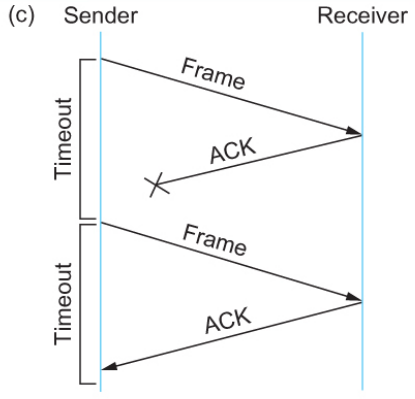
- Idea of stop-and-wait protocol is straightforward
  - After transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame.
  - If the acknowledgement does not arrive after a certain period of time, the sender times out and retransmits the original frame
- Timeline showing four different scenarios for the stop-and-wait algorithm.
  - (a)The ACK is received before the timer expires;
  - (b)the original frame is lost;
  - (c)the ACK is lost; → così ricevere 2 copie dello stesso pacchetto, come capisce perché effettivamente me ha ricevuti 2? eramo volontari oppure ci sono stati errore di commessiamo?
  - (d)the timeout fires too soon
- Cases (c), (d) are problematic

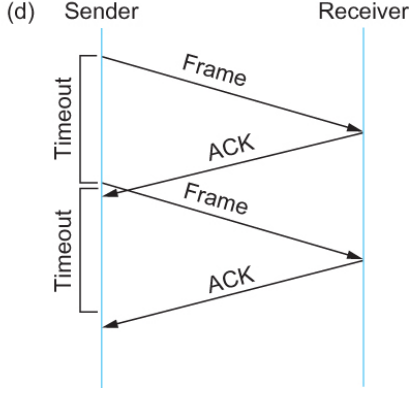
↓
RISOLVO con l'aggiunta di 1 bit



# Stop and Wait Protocol

- If the acknowledgment is lost or delayed in arriving
  - The sender times out and retransmits the original frame, but the receiver will think that it is the next frame since it has correctly received and acknowledged the first frame
  - As a result, duplicate copies of frames will be delivered



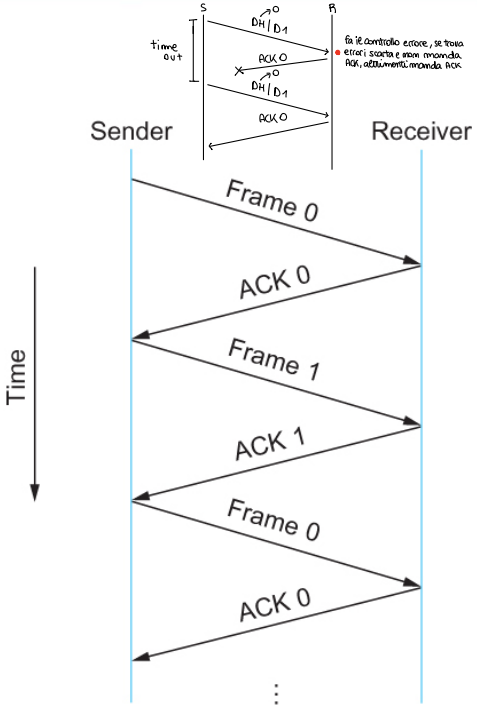
# Stop and Wait Protocol

→ è un sistema lento

- How to solve this
  - Use 1 bit sequence number (0 or 1)
  - Each frame is numbered (either 0 or 1)
    - This information is put in the header
  - In case of timeout, a frame is retransmitted with the same number
  - When the sender retransmits frame 0, the receiver can determine that it is seeing a second copy of frame 0 rather than the first copy of frame 1 and therefore can ignore it (the receiver still acknowledges it, in case the first acknowledgement was lost)

→ se viene flippato me ne accorgo con il CRC e scarto

non sono miei dati



# Stop and Wait Protocol

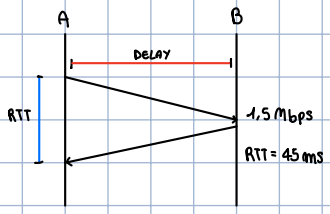
CAPACITA' ≠ THROUGHPUT
↓ a quanto trasmette al max
↳ a quanto trasmette effettivamente

- The sender has only one outstanding frame on the link at a time
  - This may be far below the link's capacity
- Consider a 1.5 Mbps link with a 45 ms RTT
  - The link has a delay × bandwidth product of 67.5 Kb ≅ 8 KB
  - Since the sender can send only one frame per RTT and assuming a frame size of 1 KB, the maximum Sending rate is:
    - Bits per frame ÷ Time per frame = 1024 × 8 ÷ 0.045 = 182 Kbps
    - About one-eighth of the link's capacity
    - The longer the RTT, the less we use the link
  - To use the link fully, then sender should transmit up to eight frames before having to wait for an acknowledgement

tempo ANDATA - RITORNO
quantità max di dati che posso mandare prima che il primo pacchetto arrivi
$D = 45 \cdot 10^{-3} \cdot 45 \cdot 10^{-3} = 67{,}5 \cdot 10^{3}\,\text{bit} \cong 8\,\text{KB}$
per sfruttare al massimo il canale quanto dovrei mandare

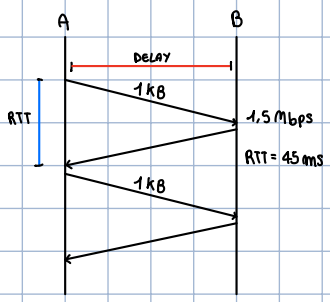
se posso inviare solo un frame vuol dire che sto andando a 182 kbps, contro gli 1,5 Mbps PREVISTI !! SPRECO

## STOP & WAIT (poco efficiente)




$D = 15 \cdot 10^6 \cdot 45 \cdot 10^{-3} = 67{,}5 \cdot 10^3 \text{ bit} \simeq 8\text{KB}$

↳ per sfruttare al massimo il canale quanto dovrei pompare




CAPACITÀ: quanto al massimo puoi trasmettere

$$C = \frac{\text{Dati}}{\text{Tempo}} = \frac{8000 \text{ bit}}{45 \cdot 10^{-3}} = 182 \text{ kbps}$$

(8000 bit = 1KB)

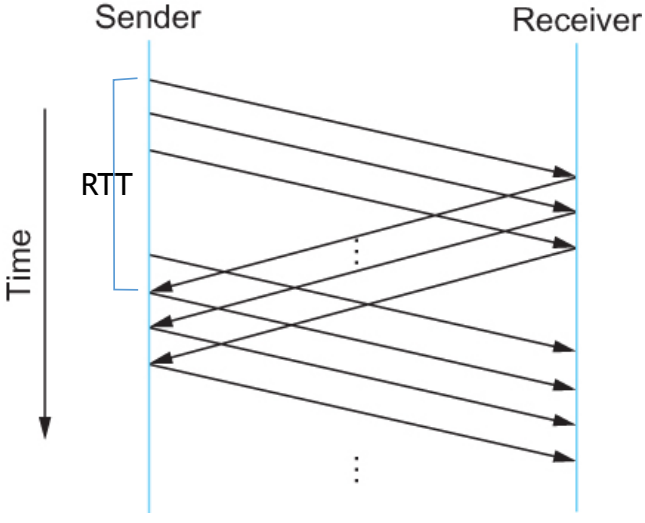
$D = 15 \cdot 10^6 \cdot 45 \cdot 10^{-3} = 67{,}5 \cdot 10^3 \text{ bit} \simeq 8\text{KB}$

↳ per sfruttare al massimo il canale quanto dovrei pompare

# Sliding Window Protocol ⇒ usato nel TCP

- The Sliding Window Protocol is a variant (better, a generalisation) of stop-and-wait protocol
- We allow the sender to send many frames during a *round* (RTT), even before receiving the first ACK, in order to fill the link capacity
- (Used sometimes at link level, more often at transport level)

oltre al bit abbiamo anche un numero di sequenza

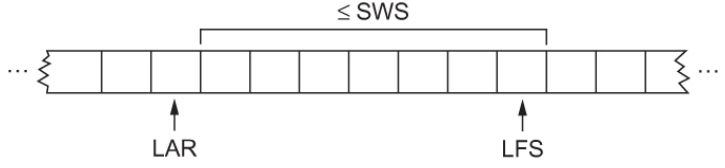



Timeline for Sliding Window Protocol

# Sliding Window Protocol

- Sender assigns a sequence number denoted as *SeqNum* to each frame.
  - Assume it can grow infinitely large
- Sender (LATO SENDER) maintains three variables
  - *Sending Window Size* (*SWS*): Upper bound on the number of outstanding (unacknowledged) frames that the sender can transmit (MAX # di pacchetti che posso spedire)
  - *Last Acknowledgement Received* (*LAR*): Sequence number of the last acknowledgement received (# dell'ultimo frame inviato e ricevuto correttamente dal receiver (ovvero l'ultimo ACK ricevuto))
  - *Last Frame Sent* (*LFS*): Sequence number of the last frame sent (# dell'ultimo frame che ho spedito)
- Sender also maintains the following invariant

$$LFS - LAR \leq SWS$$

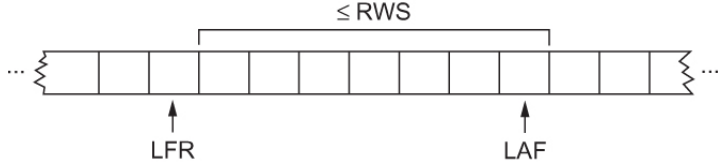


# Sliding Window Protocol

- When an acknowledgement arrives, the sender moves LAR to right, then goes in condition to transmit another frame
- Also, the sender associates a timer with each frame it transmits
- It retransmits the frame (with the same sequence number) if the timer expires before the ACK is received
- Note that the sender has to be willing to buffer up to SWS frames (why?)

# Sliding Window Protocol

- Receiver maintains three variables
  - *Receiving Window Size (RWS):* Upper bound on the number of out-of-order frames that the receiver is willing to accept
  - *Largest Acceptable Frame (LAF):* Largest acceptable frame sequence number
  - *Last Frame Received (LFR):* Sequence number of the last frame received
- Receiver also maintains the following invariant
  $LAF - LFR \leq RWS$



# Sliding Window Protocol


- When a frame with sequence number SeqNum arrives, what does the receiver do?
  - If LFR < SeqNum ≤ LAF ➡ Accept it
  - If SeqNum ≤ LFR or SeqNum > LAF ➡ Discard it (the frame is outside the receiver window)
- Now the receiver needs to decide whether or not to send an ACK
  - Let *SeqNumToAck* denote the largest sequence number not yet acknowledged, such that all frames with sequence number less than or equal to SeqNumToAck have been received
  - **cumulative acknowledgement:** the receiver acknowledges the receipt until SeqNumToAck even if high-numbered packets have been received
  - In this way, we can buffer out-of-order frames, and acknowledge them cumulatively
- The receiver then sets LFR = SeqNumToAck and shifts window: LAF = LFR + RWS

avviso mandando un ack fino al primo frame che mi manca, anche se magari ho già quelli dopo

# Sliding Window Protocol

- For example, suppose LFR = 5 and RWS = 4
  (i.e. the last ACK that the receiver sent was for seq. no. 5)
  - LAF = 9
- If frames 7 and 8 arrive, they will be buffered because they are within the receiver window
  - But no ACK will be sent since frame 6 is yet to arrive
  - Frames 7 and 8 are out of order
- Frame 6 arrives (it is late because it was lost first time and had to be retransmitted)
- Now Receiver:
  - acknowledges Frame 8
  - bumps LFR to 8
  - bumps LAF to 12

# Issues with Sliding Window Protocol

- When timeout occurs, the amount of data in transit decreases
  - Since the sender is unable to advance its window
- When the packet loss occurs, this scheme is no longer keeping the pipe full
  - The longer it takes to notice that a packet loss has occurred, the more severe the problem becomes
- How to improve this
  - Negative Acknowledgement (NAK)
  - Additional Acknowledgement
  - Selective Acknowledgement

# Issues with Sliding Window Protocol

- Negative Acknowledgement (NAK)
  - Receiver sends NAK for frame 6 when frame 7 arrive (in the previous example)
  - However this is unnecessary since sender's timeout mechanism will be sufficient to catch the situation
- Additional Acknowledgement
  - Receiver sends additional ACK for frame 5 when frame 7 arrives
  - Sender uses duplicate ACK as a clue for frame loss
  - Used in TCP
- Selective Acknowledgement
  - Receiver will acknowledge exactly those frames it has received, rather than the highest number frames
  - Receiver will acknowledge frames 7 and 8
  - Sender knows frame 6 is lost
  - Sender can keep the pipe full (at the price of additional complexity)

# Issues with Sliding Window Protocol

How to select the window size?

- SWS is easy to compute: Delay × Bandwidth
- RWS can be anything
- Two common setting
  - RWS = 1: No buffer at the receiver for frames that arrive out of order
  - RWS = SWS: The receiver can buffer frames that the sender transmits
- If RWS > SWS, the receiver buffer will be never filled
- If RWS < SWS, then sender could send frames which cannot be buffered and must be discarded and retransmitted later
- The case SWS=RWS=1 is the stop-and-wait ARQ.

# Issues with Sliding Window Protocol

- Finite Sequence Number
  - Frame sequence number is specified in the header field
  - Finite size
    - e.g., with 3 bit, eight possible sequence number: 0, 1, 2, 3, 4, 5, 6, 7
  - It is necessary to wrap around
  - How to distinguish between different incarnations of the same sequence number?
  - Number of possible sequence number must be larger than the number of outstanding frames allowed
    - Stop and Wait: One outstanding frame
      - 2 distinct sequence number (0 and 1)
    - Sliding window: Let MaxSeqNum be the maximum sequence number (usually $=2^n-1$ where n is the number of bits in the header field)
      - SWS ≤ MaxSeqNum
      - Is this sufficient?

# Issues with Sliding Window Protocol

- SWS ≤ MaxSeqNum
  - Is this sufficient? Depends on RWS
    - If RWS = 1, then sufficient
    - If RWS = SWS, then not good enough
- For example, we have eight sequence numbers, MaxSeqNum=7
  0, 1, 2, 3, 4, 5, 6, 7, and RWS = SWS = 7
  - Sender sends frames 0, 1, …, 6.
  - Receiver receives frames 0, 1,…, 6 and sends ACKs 0, 1,…, 6; now it is waiting for frames 7, 0, …, 5.
  - But all the ACKs are lost! So the sender retransmits 0, 1, …, 6.
  - Receiver accepts the frames 0, 1, …, 5, which are *mistaken for the new ones*, and put in the buffer after the slot for 7!

# Issues with Sliding Window Protocol

- To avoid this: if RWS = SWS then it must be
  $SWS \leq (MaxSeqNum + 1)/2 = 2^{n-1}$
  where n are the bits for the sequence number
- In this way, only half sequence numbers can be be outstanding at any time
- In the example above, let us set RWS = SWS = 4
  - Sender sends frames 0, 1, 2, 3, and stops waiting for acks
  - Receiver receives frames 0, 1, 2, 3 and sends ACKs 0, 1, 2, 3; now it is waiting for frames 4, 5, 6, 7 (it has up to 4 free slots).
  - But all the ACKs are lost, so the sender retransmits 0, 1, 2, 3.
  - Receiver was waiting for 4,…,7, so it recognises that the frames 0, 1, 2, 3 just received are copies of old ones: he does not accept these copies, but sends the acks again.

# Summary about Sliding Window Protocol

- Serves three different aims
  - Reliable → è AFFIDABILE, i messaggi arrivano
  - Preserve the order → nel caso dovessero arrivare in disordine, li riordina
    - Each frame has a sequence number
    - The receiver makes sure that it does not pass a frame up to the next higher-level protocol until it has already passed up all frames with a smaller sequence number
  - Frame control
    - Receiver is able to throttle the sender
    - Keeps the sender from overrunning the receiver from transmitting more data than the receiver is able to process