RW354

Principles of Computer Networking

A.E. Krzesinski and B.A. Bagula

Department of Computer Science

University of Stellenbosch

Last updated: 20 July 2003

The material presented in these slides is used with permission from

- *Larry L. Peterson and Bruce S. Davie. Computer Networks: A Systems Approach (Second Edition). Morgan Kaufmann Publishers. ISBN 1-55860-577-0.*
- *William Stallings. Data and Computer Communications (Sixth Edition). Prentice-Hall Inc. ISBN 0-13-571274-2.*
- *Andrew S. Tannenbaum. Computer Networks (Fourth Edition). Prentice Hall Inc. ISBN 0-13-349945-6.*

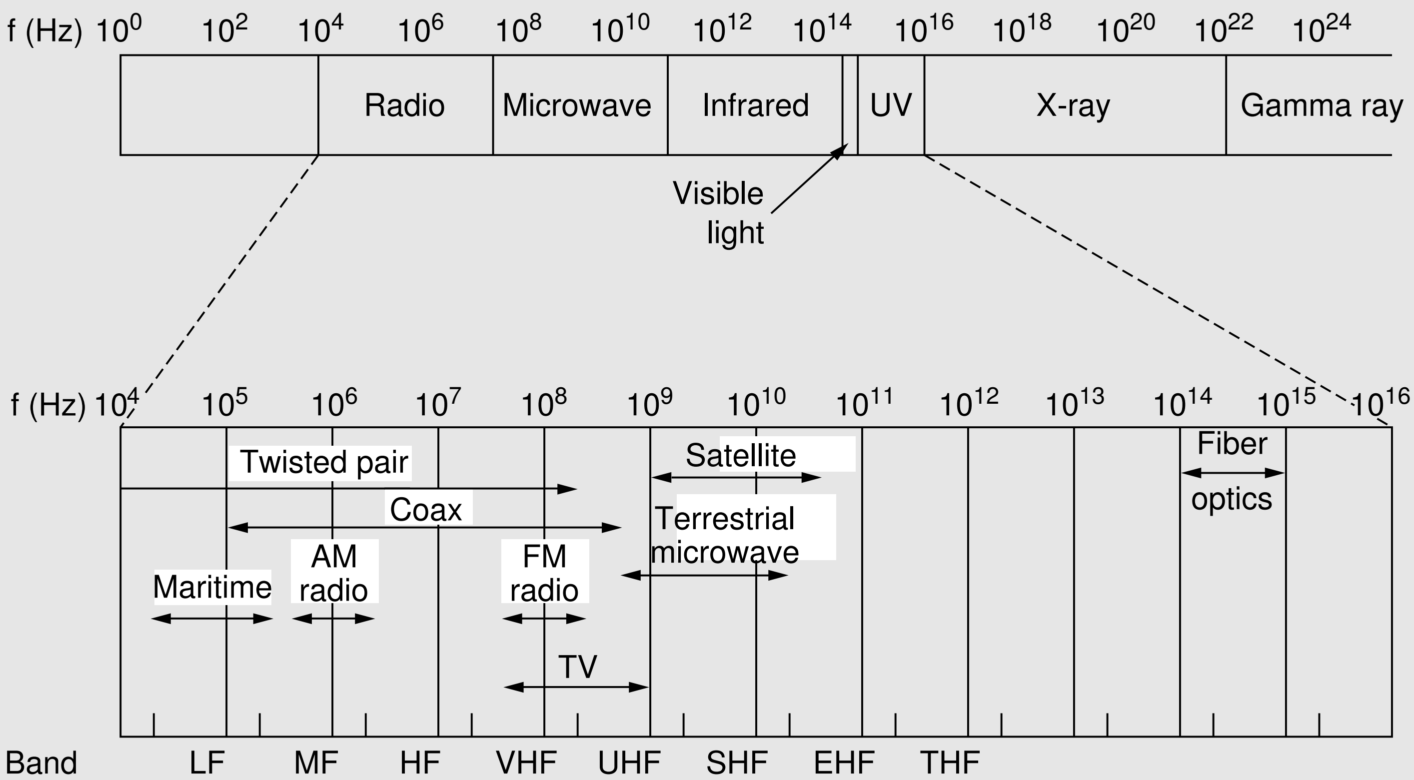
Nodes

Consider two nodes connected by a physical link. The following issues must be addressed for the nodes to successfully exchange data

- *data encoding*
- *frame delimitation*
- *error detection*
- *error correction*
- *media access control.*

Links - electromagnetic spectrum

The radio, microwave, infrared, visible & UV portions of the spectrum are used to transmit information by modulating the amplitude, frequency or phase of the waves.



ITU names: Low, Medium, High, Very, Ultra, Super, Extremely, Tremendously High Frequency Bands.

Links - cables

Sometimes you install your own links

Category 5 twisted pair	*10-100Mbps*	*100m*
50-ohm coax (ThinNet)	*10-100Mbps*	*200m*
75-ohm coax (ThickNet)	*10-100Mbps*	*500m*
Multimode fiber	*100Mbps*	*2km*
Single-mode fiber	*100-2400Mbps*	*40km*

Links - leased lines

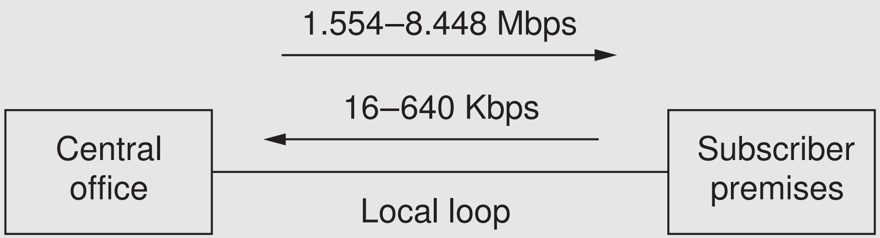
Sometimes the links are leased from the phone company

Service to ask for	Bandwidth you get
ISDN	64 Kbps
T1	1.544 Mbps
T3	44.736 Mbps
STS-1	51.840 Mbps
STS-3	155.250 Mbps
STS-12	622.080 Mbps
STS-24	1.244160 Gbps
STS-48	2.488320 Gbps

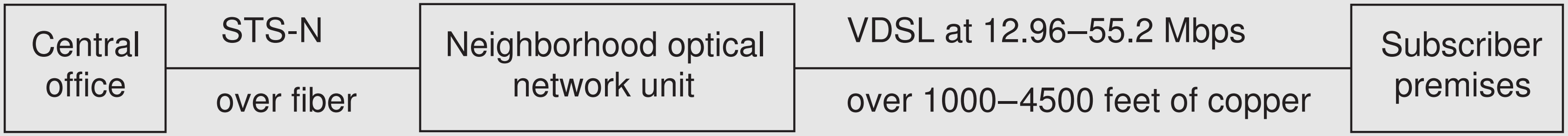
STS: Synchronous Transfer Signal.
STSN is sometimes called OCN: Optical Carrier.

Last-mile links

Asynchronous digital subscriber line (ADSL) connects the subscriber to the central office via the local loop.



Very-high-rate DSL (VDSL) connects the subscriber to the optical network that reaches the neighbourhood.



Shannon's theorem

The maximum channel capacity C in bits/second is

$$C = B\log_2(1 + S/N)$$

where B is the bandwidth of the channel in hertz and S/N is the signal to noise ratio. S/N is expressed in decibels

$$dB = 10\log_{10}(S/N).$$

For a telephone line $B = 3300 - 300 = 3000$Hz and $dB = 30$ so that $S/N = 10^3$. Then

$$C = 3000\log_2 1001 \sim 30\textit{Kbps}.$$

Modern modems provide 56 Kbps thanks to better S/N ratios & the use of clever coding & compression methods.

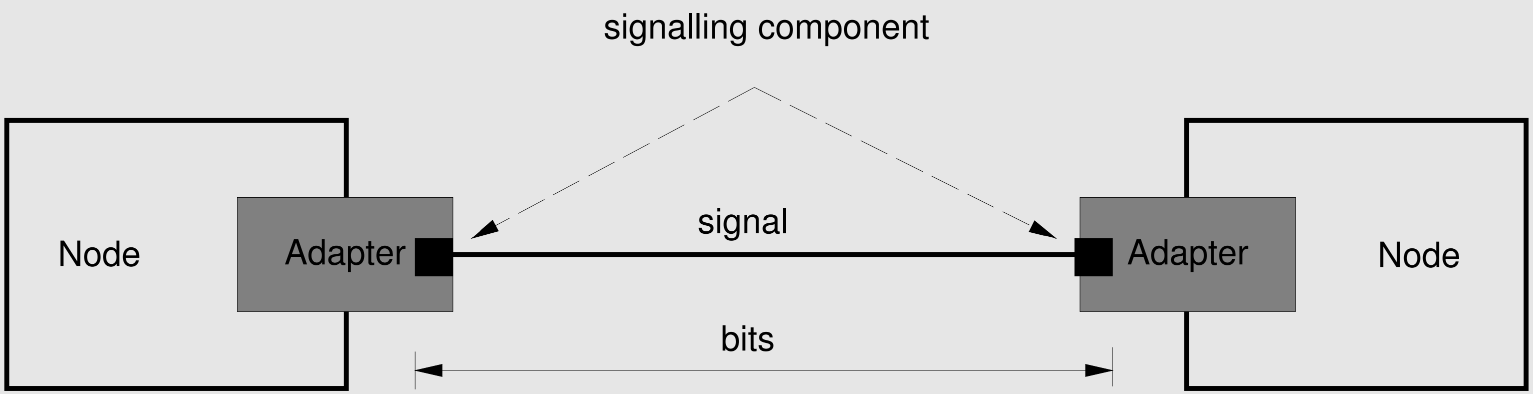
Encoding: overview

Signals propagate over a physical medium

- *digital signals*
- *analog signals.*

Data can be either digital or analog: we are interested in digital data.

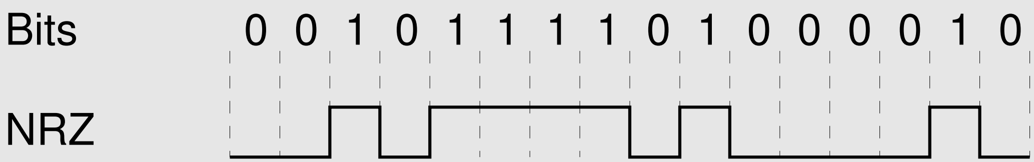
Problem: encode the binary data that the source node wants to send to the destination node into the signal that propagates over the medium.



Encoding: Non-Return to Zero (NRZ)

The most common way to transmit digital signals is to use two different voltage levels for the two binary digits.

The voltage level is constant (non-return to zero) during a bit interval.



Problem: consecutive 0's and 1's

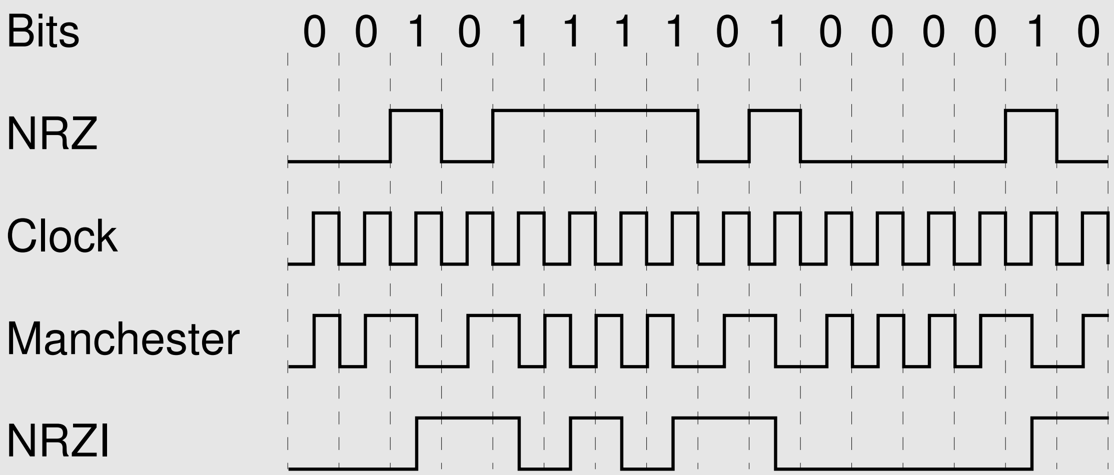
- *long sequences of 1's give rise to a dc-component which necessitates physical coupling of the transmission components (no dc-component allows ac-coupling via a transformer)*
- *unable to recover a clock signal.*

Encoding: NRZI

Non-return to Zero Invert on ones (NRZI). The data are encoded by the presence/absence of a signal transition at the beginning on the bit interval.

Make a transition from the current signal to encode a 1, and stay at the current signal to encode a 0. This solves the problem of consecutive 1's.

NRZI is an example of a differential encoding scheme.

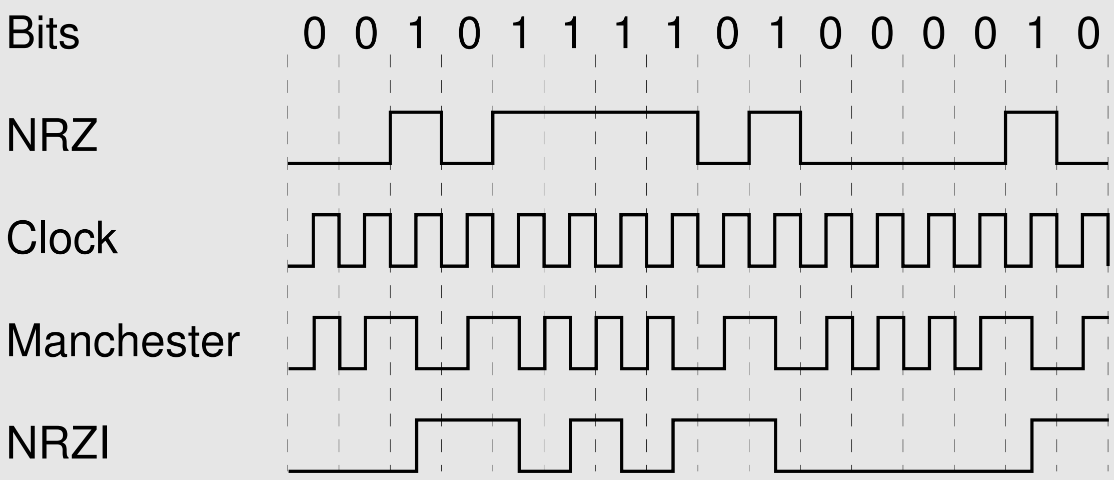


Encoding: NRZI

Problems: a dc-component can be present, and we are unable to recover a clock signal.

Modulation rate (signals per bit)

- *all 0's: 0*
- *101010. . . : 0.5*
- *all 1's: 1.0*

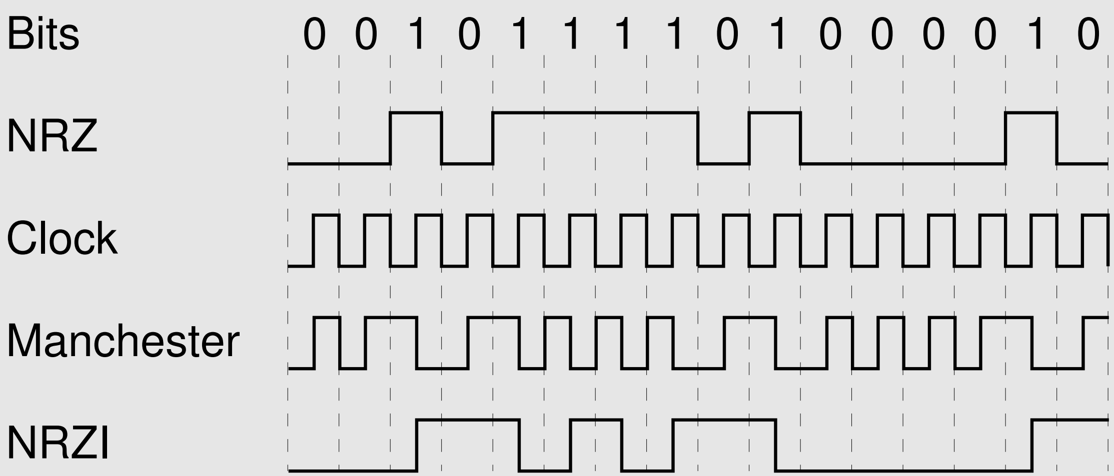


Encoding: Manchester

0: transition from low to high in the middle of a bit interval.
1: transition from high to low in the middle of a bit interval.

Advantages

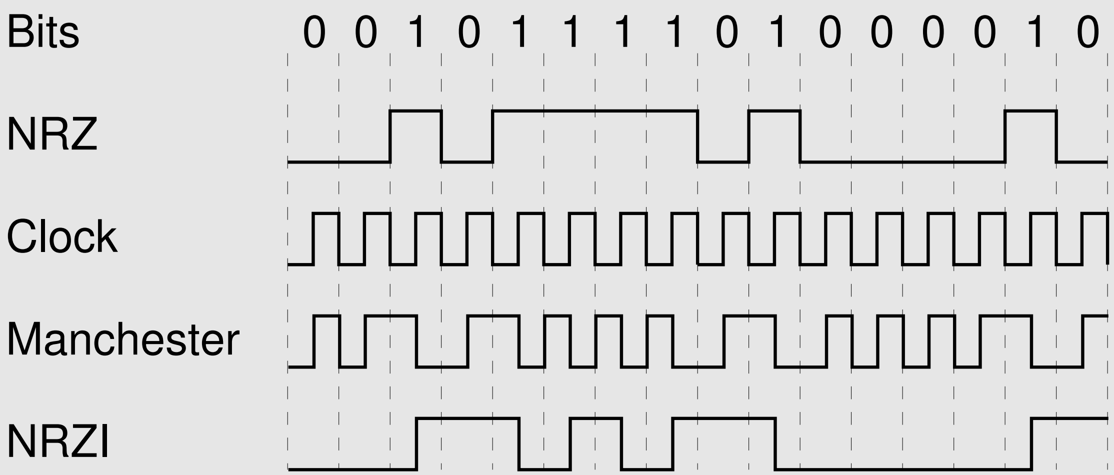
- *no dc-component*
- *a transition is present in the middle of each bit interval: the clock can be recovered.*



Encoding: Manchester

Modulation rate (signals per bit)

- *101010. . . : 1.0*
- *all 0's: 2.0*
- *all 1's: 2.0*



Encoding: 4B/5B

- *4 bits of data are encoded into a 5-bit code. The 5-bit codes are selected to have no more than one leading 0 and no more than two trailing 0's.*
- *Two concatenated 5-bit codes never have more than three consecutive 0's.*
- *The resulting 5-bit codes are transmitted using the NRZI encoding.*
- *There are no problems with consecutive 1's.*
- *This achieves 8/10 = 80% modulation efficiency.*

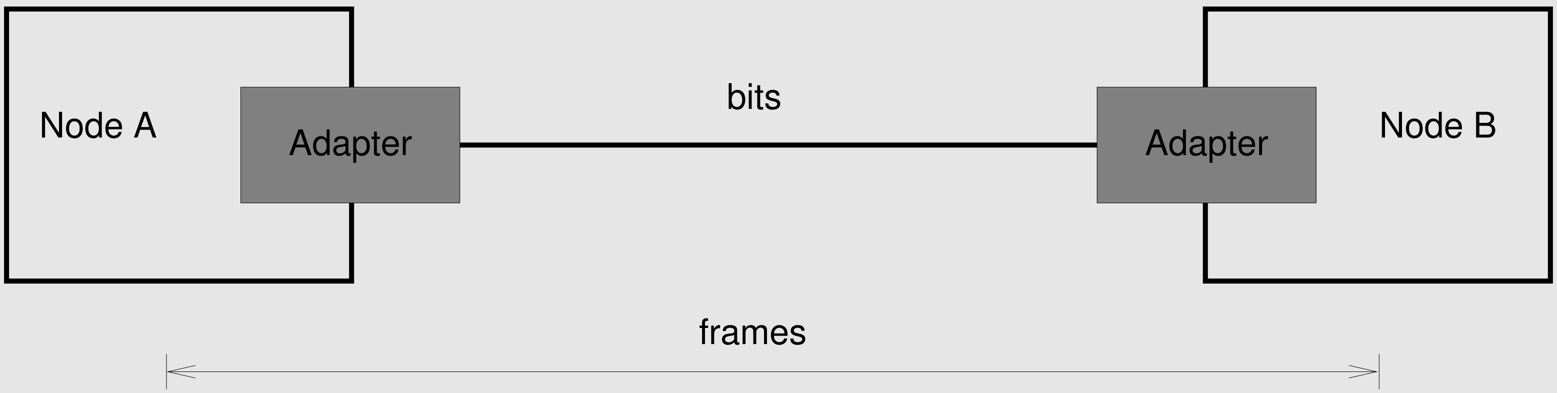
Encoding: 4B/5B

4-bit Data	5-bit Code	4-bit Data	5-bit Code
0000	11110	0001	01001
0010	10100	0011	10101
0100	01010	0101	01011
0110	01110	0111	01111
1000	10010	1001	10011
1010	10110	1011	10111
1100	11010	1101	11011
1110	11100	1111	11101

Framing: Overview

A sequence of bits is assembled into a frame

- *determine the first and last bit of the frame*
- *this is typically implemented by the network adapter*
- *the adapter fetches (deposits) frames out of (into) the host memory.*



Bits flow between adapters, frames flow between hosts.

Framing: byte-oriented protocols

The sentinel approach as used in BISYNC. The control characters STX & ETX delimit the data portion of a frame.

8	8	8		8		8	16
SYN	SYN	SOH	Header	STX	Body	ETX	CRC

- *Problem: the ETX character might appear in the data portion of the frame.*
- *Solution: escape the ETX character with a DLE character.*

Framing: byte-oriented protocols

The byte counting approach as used in DDCMP

8	8	8	14	42		16
SYN	SYN	Class	Count	Header	Body	CRC

- *Problem: the count field is corrupted - a framing error.*
- *Solution: detect when CRC fails.*

The Cyclic Redundancy Check (CRC) is used to detect errors.

Framing: bit-oriented protocols

HDLC: High-Level Data Link Control (also SDLC and PPP)

Delineate the frame with a special bit-sequence: 01111110

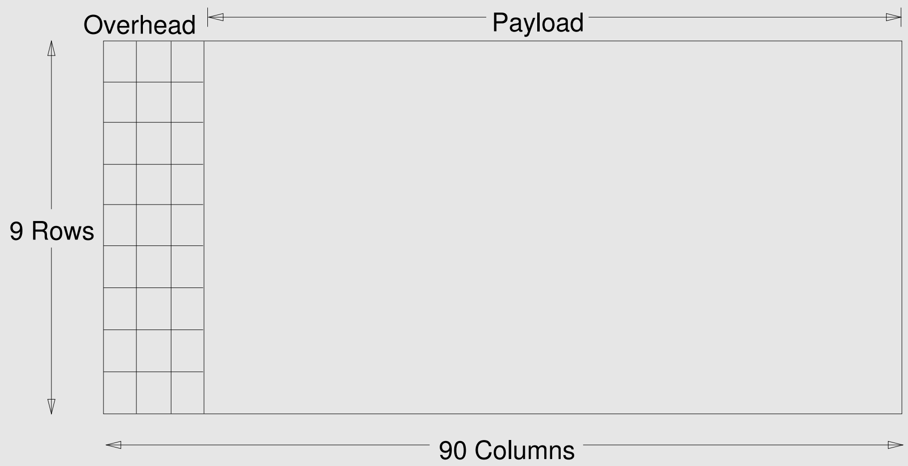
8	16		16	8
Beginning Sequence	Header	Body	CRC	Ending Sequence

Bit stuffing

- Sender: if five consecutive 1's are transmitted in the body of the message, append a 0.
- Receiver: if five consecutive 1's arrive
  - if the next bit is 0: remove it
  - else
    - if the next bits are 10: end-of-frame marker
    - else if the next bits are 11: error.

Framing: SONET

- *SONET: Synchronous Optical Network*
- *ITU standard for transmission over fiber*
- *the basic SONET building block is the STS-1 frame of 810 octets sent once every 125μs: 51.84 Mbps*

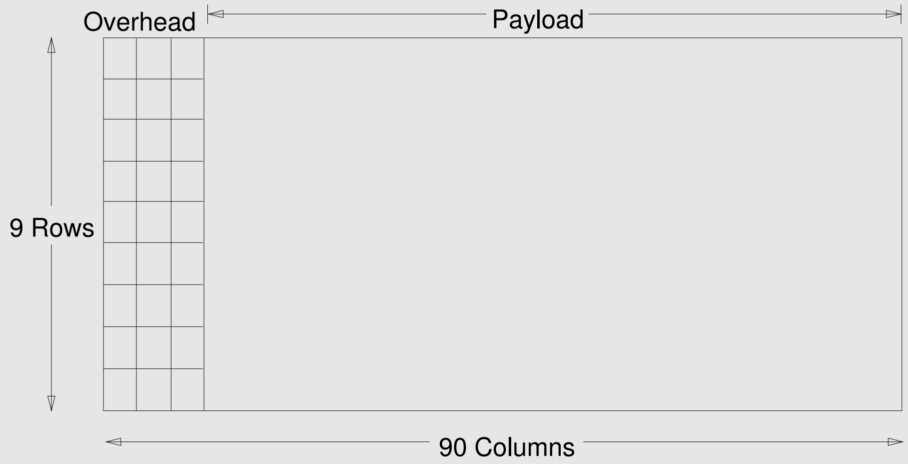


The frame can be viewed logically as a matrix of 9 rows & 90 columns.

Framing: SONET

SONET		SDH	Data rate (Mbps)		
Electrical	**Optical**	**Optical**	**Gross**	**SPE**	**User**
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-9	OC-9	STM-3	466.56	451.008	445.824
STS-12	OC-12	STM-4	622.08	601.344	594.432
STS-18	OC-18	STM-6	933.12	902.016	891.648
STS-24	OC-24	STM-8	1244.16	1202.688	1188.864
STS-36	OC-36	STM-12	1866.24	1804.032	1783.296
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728
STS-192	OC-192	STM-64	9953.28	9621.504	9510.912

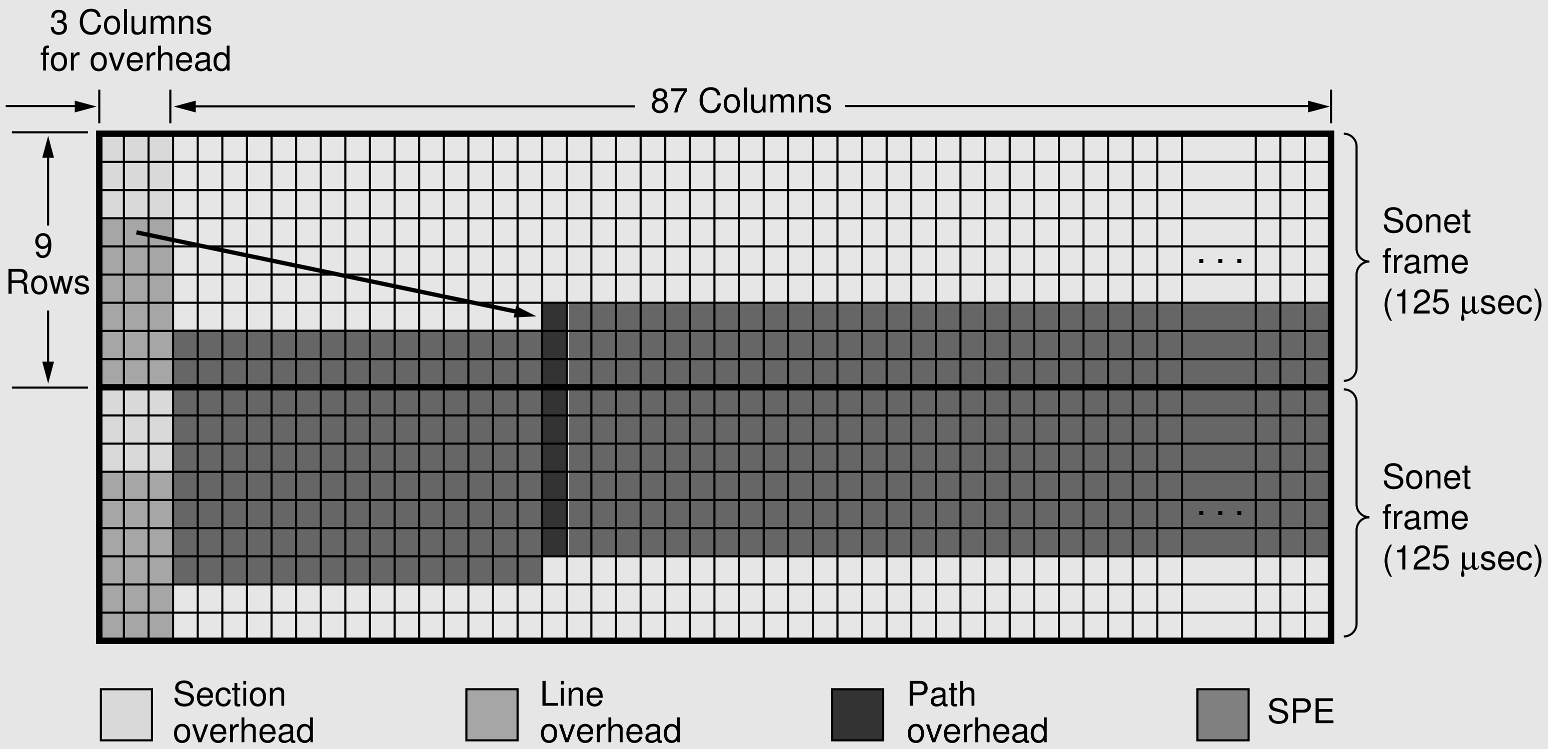
Framing: SONET



- *the first 3 columns of each frame are reserved for system management information*
- *the first 3 rows contain section overhead*
- *the next 6 rows contain line overhead*
- *the first 2 bytes of a frame are a synchronization pattern.*

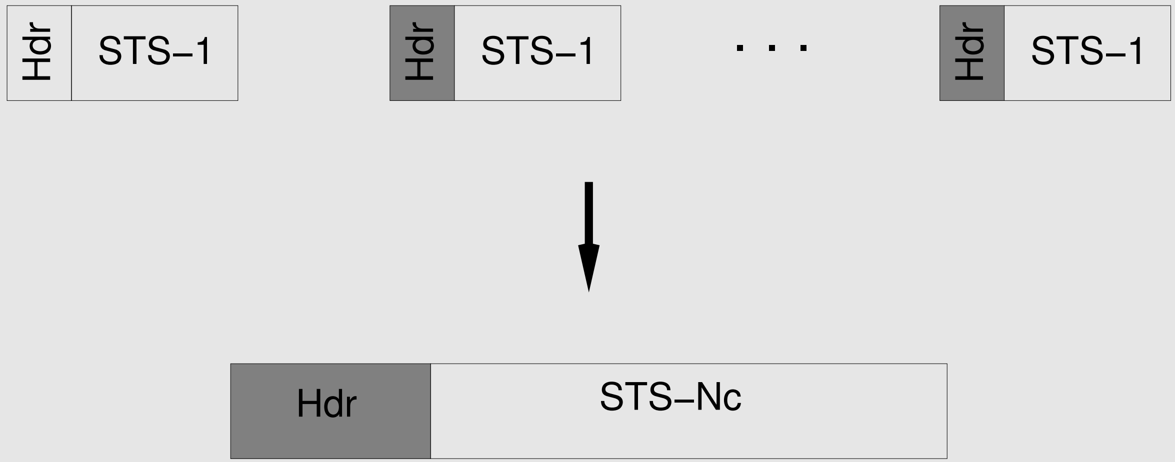
Framing: SONET

- *the remaining 87 columns contain 50.112Mb of user data: the Synchronous Payload Envelope SPE*
- *a pointer in the line overhead points to the SPE which can begin anywhere within a frame & can span frames*



Framing: SONET

- *overhead bytes are encoded using NRZ*
- *payload bytes are scrambled*
- *byte-interleaved multiplexing*
- *STC-Nc*



- *each frame is 125μs long.*

Errors: two-dimensional parity

		parity bits
data	0101001	1
	1101001	0
	1011110	1
	0001110	1
	0110100	1
	1011111	0
parity byte	1111011	0

2-D parity finds all 1-, 2- and 3-bit errors & most 4-bit errors.

Errors: Internet checksum algorithm

The checksum algorithm is based on addition.

The message is viewed as a sequence of 16-bit integers. Add these integers together using 16-bit ones complement arithmetic, and then take the ones complement of the result. That 16-bit number is the checksum.

```
u_short
cksum(u_short *buf, int count) {
    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) { /* carry occurred in top 16 bits */
            sum &= 0xFFFF;      /* bottom 16 bits */
            sum++;              /* add carry to result */
        }
    }
    return ~(sum & 0xFFFF);
}
```

Errors: Cyclic Redundancy Check

- *Add k bits of redundant data to an n-bit message where $k << n$.*
- *Represent an $n+1$-bit message as an n degree polynomial. Thus MSG=10011010 corresponds to $M(x) = x^7 + x^4 + x^3 + x^1$.*
- *Let k be the degree of some divisor polynomial $C(x)$. For example $C(x) = x^3 + x^2 + 1$, $k = 3$.*
- *Transmit the polynomial $P(x)$ that is evenly divisible by $C(x)$, and receive the polynomial $P(x) + E(x)$. $E(x) = 0$ implies no errors.*
- *The recipient divides $P(x) + E(x)$ by $C(x)$. The remainder will be zero if $E(x)$ was zero (there was no error), or $E(x)$ is exactly divisible by $C(x)$. Choose $C(x)$ to make second case extremely rare.*

Errors: CRC sender

- *Form $T(x) = M(x) \times x^k$. In our example we get $T(x) = x^{10} + x^7 + x^6 + x^4$ (10011010000).*
- *Divide $T(x)$ by $C(x)$ (1101). The CRC $E(x)$ is the remainder (101).*

```
                          11111001
Generator ---> 1101 ) 10011010000  <--- Message
                      1101
                       1001
                       1101
                        1000
                        1101
                         1011
                         1101
                          1100
                          1101
                             1000
                             1101
                              101  <--- Remainder
```

- *Send $T(x) - E(x)$ (10011010000 - 101 = 10011010101) which is exactly divisible by $C(x)$.*

Errors: CRC

We want to ensure that $C(x)$ does not divide evenly into the polynomial $E(x)$. The following errors can be detected

- *All single-bit errors, as long as the x^k and x^0 terms have non-zero coefficients.*
- *All double-bit errors, as long as $C(x)$ has a factor with at least three terms.*
- *Any odd number of errors, as long as $C(x)$ contains the factor $(x + 1)$.*
- *Any burst error (a sequence of consecutive errored 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.*

Errors: CRC

Common polynomials for $C(x)$ *are:*

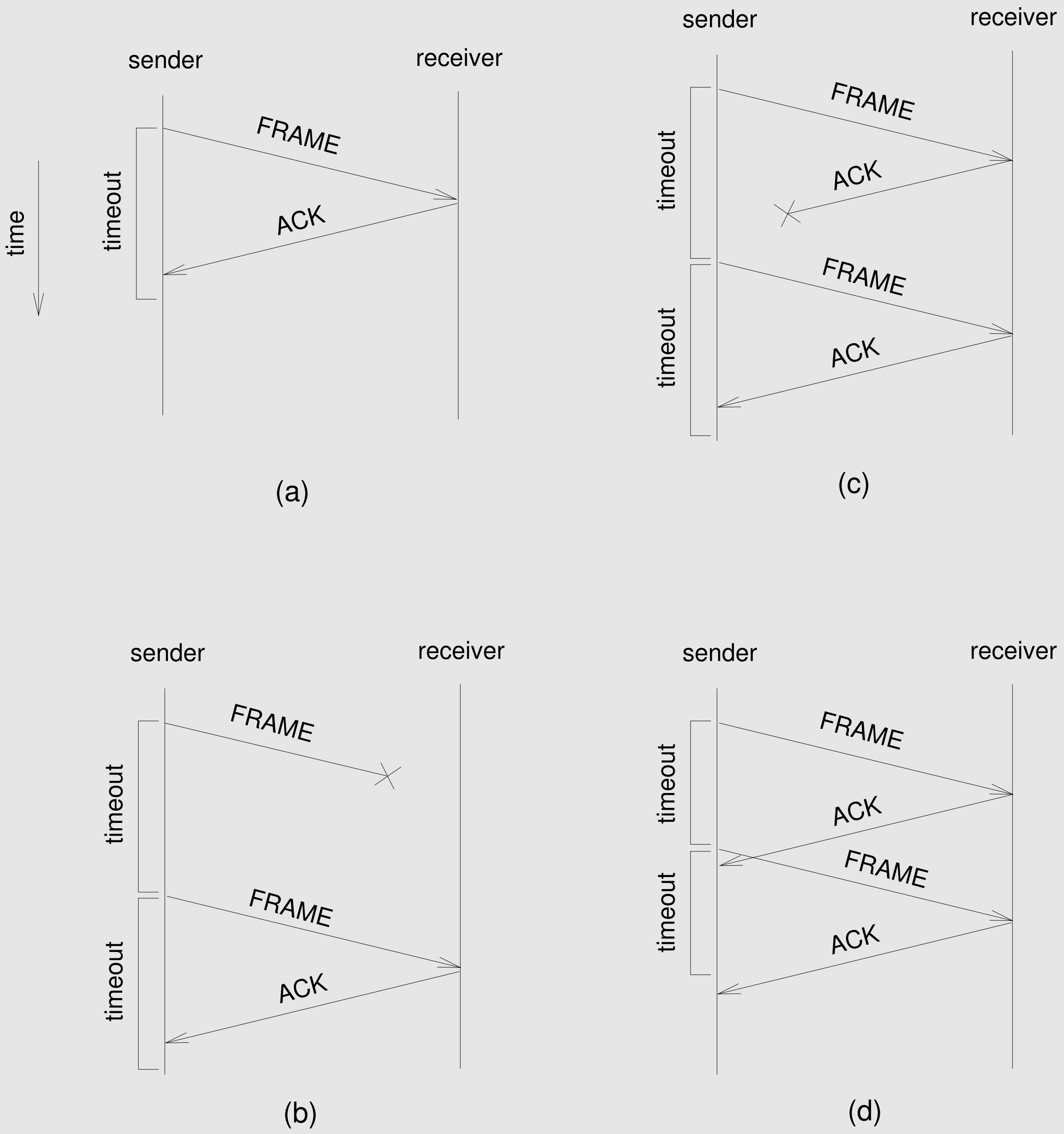
CRC	$C(x)$
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} +$
	$x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Reliability: overview

Recover from corrupt frames

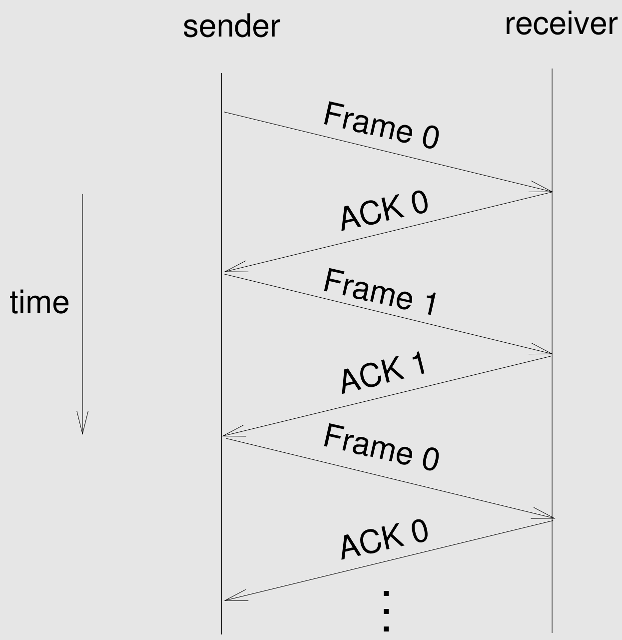
- *Forward error correction: the frames contain error correction codes (ECC) which are used to recover from transmission errors.*
- *Automatic Repeat reQuest: acknowledgements & timeouts are used to detect & re-transmit lost frames & frames with errors.*

Reliability: ARQ



Reliability: Stop-and-Wait

Stop-and-wait uses a 1-bit sequence number.

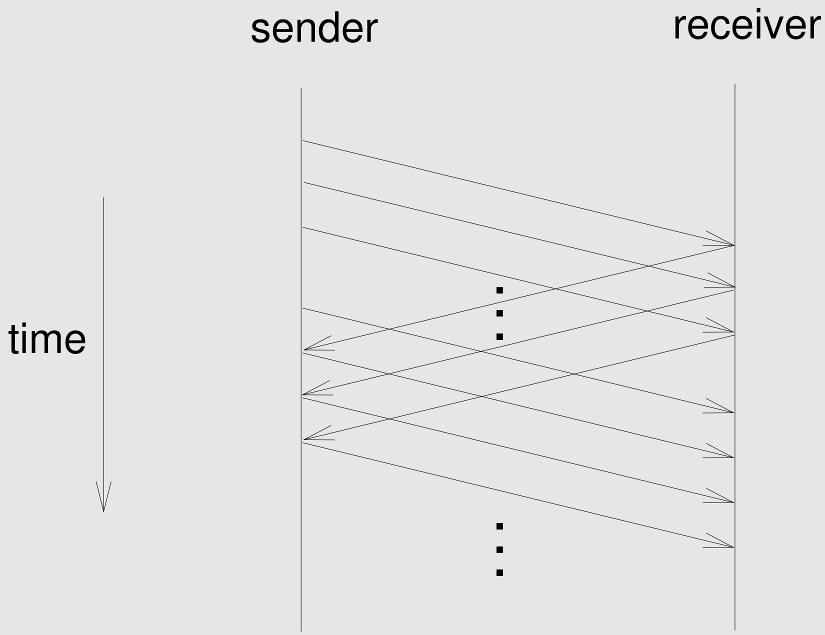


Problem: only 1 frame is sent per RTT – the pipe is not full.

Example: 1.5Mbps link $\times$ 45ms RTT = 67.5Kb (8KB). Assuming a frame size of 1KB, stop-and-wait uses about one-eighth of the link's capacity. We want the sender to transmit up to 8 frames before having to wait for an ACK.

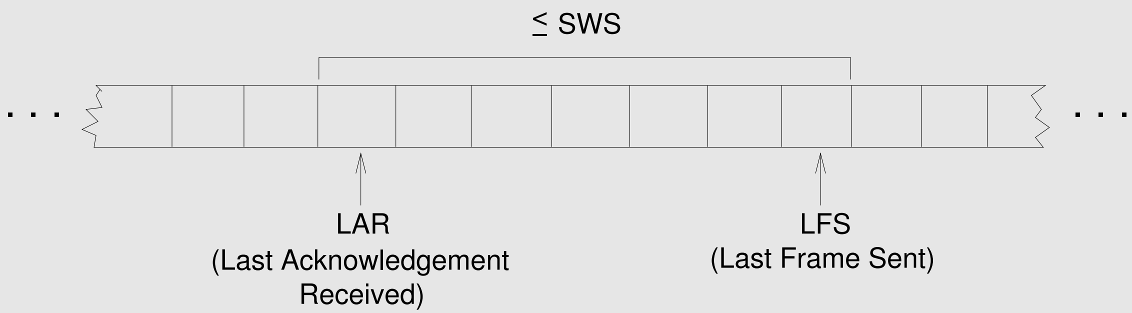
Reliability: sliding window

Allow the sender to transmit several frames before receiving an ACK, thus keeping the pipe full. There is an upper limit on the number of outstanding (un-ACKed) frames allowed.



Reliability: sliding window sender

- *Assign a sequence number to each frame:* `SeqNum`
- *Maintain three state variables*
  - *the send window size:* `SWS`
  - *the last acknowledgment received:* `LAR`
  - *the last frame sent:* `LFS`
- *Maintain the invariant:* $\mathtt{LFS} - \mathtt{LAR} + 1 \leq \mathtt{SWS}$



- *When the ACK arrives, advance* `LAR` *thereby opening the window*
- *Buffer up to* `SWS` *frames.*

Reliability: sliding window receiver

- *Maintain three state variables*
  - *the receive window size:* `RWS`
  - *the last frame acceptable:* `LFA`
  - *the next frame expected:* `NFE`
- *Maintain the invariant:* $\mathtt{LFA} - \mathtt{NFE} + 1 \leq \mathtt{RWS}$

Reliability: sliding window receiver

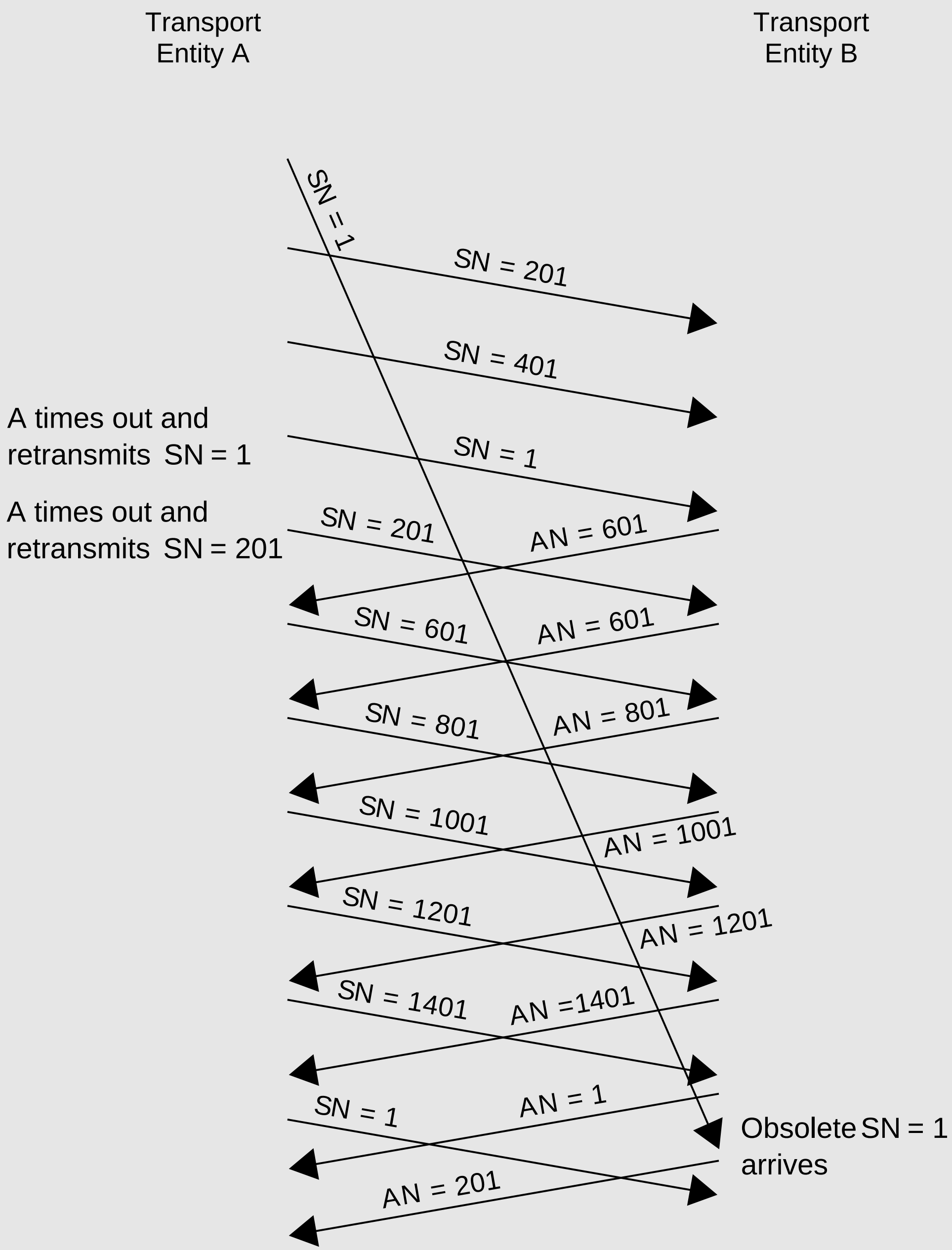
- *Frame* `SeqNum` *arrives*
  - if NFE $\leq$ SeqNum $\leq$ LFA then accept
  - if SeqNum $<$ NFE or SeqNum $>$ LFA then discard
- *Send a cumulative ACK for the highest numbered frame received in order*
- *Variations*
  - *selective acknowledgements*
  - *negative acknowledgements (NAK)*

Reliability: sliding window

The sliding window protocol serves three different roles

- *reliable delivery: the reliable delivery of frames over an unreliable link*
- *ordered delivery: frames are delivered in the correct order*
- *flow control: the receiver can throttle the sender.*

Reliability: incorrect duplicate detection



Reliability: sequence number space

- $\texttt{SeqNum}$ *field is finite; sequence numbers wrap around*
- *The sequence number space must be larger than the number of outstanding frames*
- $\texttt{SWS} \leq \texttt{MaxSeqNum} - 1$ *is not sufficient*
  - *assume a 3-bit* $\texttt{SeqNum}$ *field* $0 \ldots 7$
  - $\texttt{SWS} = \texttt{RWS} = 7$
  - *sender transmits frames* $0 \ldots 6$
  - *they arrive successfully, but the ACKs are lost*
  - *the sender retransmits* $0 \ldots 6$
  - *the receiver expects* $7, 0 \ldots 5$ *but receives the second incarnation of* $0 \ldots 5$
- $\texttt{SWS} < (\texttt{MaxSeqNum} + 1)/2$ *is the correct rule*
- *Intuitively* $\texttt{SeqNum}$ *"slides" between the two halves of sequence number space.*

Reliability: concurrent logical channels

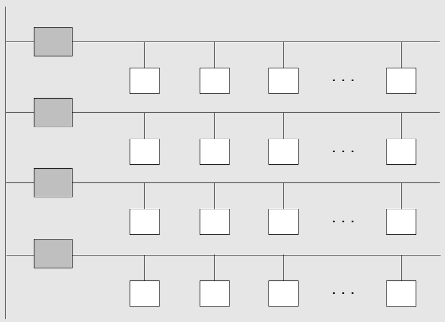
- *Multiplex several logical channels over a single point-to-point link. Run the stop-and-wait protocol on each logical channel.*
- *Maintain three bits of state for each logical channel*
  - *boolean: the channel is/not currently busy*
  - *sequence number: frames sent on the channel*
  - *next sequence number to expect on the channel*
- *ARPANET supported eight logical channels over each ground link (16 over each satellite link).*
- *Header for each frame included a 3-bit channel number and a 1-bit sequence number, for a total of 4 bits; same number of bits as the sliding window protocol requires to support up to eight outstanding frames on the link.*
- *Separates reliability from flow control and frame order.*

Ethernet: overview

- History
  - developed by Xerox PARC in mid-1970s
  - roots in the Aloha packet-radio network
  - standardized by Xerox, DEC, and Intel in 1978
  - similar to IEEE 802.3 standard.
- CSMA/CD
  - carrier sense
  - multiple access
  - collision detection
- Bandwidth: 10Mbps, 100Mbps and 1Gbps
- Problem: distributed algorithm that provides fair access to a shared medium

Ethernet: physical properties

- Classical Ethernet (thick-net): also called 10Base5
  - maximum segment of 500m
  - transceiver taps are at least 2.5m apart
  - connect multiple segments with repeaters
  - no more than 4 repeaters between any pair of nodes (2500m total)
  - maximum of 1024 hosts; Manchester encoding



- 10Base2 (thin-net): 200m, daisy-chain configuration
- 10BaseT (twisted-pair): 100m, star configuration.

Ethernet: frame format

64	48	48	16		32	8
Preamble	Dest Addr	Src Addr	Type	Body	CRC	Postamble

Ethernet addresses

- unique 48-bit unicast address assigned to each adaptor
- example: `8 : 0 : 2b : e4 : b1 : 2`
- broadcast: all 1s
- multicast: first bit is 1.

Ethernet: frame format

The adaptor receives all frames. It accepts & passes to the host

- *frames addressed to its own unicast address*
- *frames addressed to the broadcast address*
- *frames addressed to any multicast address it has been programmed to accept*
- *all frames when in promiscuous mode.*

Ethernet: transmitter algorithm

If the medium is idle

- *send immediately*
- *upper bound message size of 1500 bytes*
- *must wait 51μs between back-to-back frames.*

If the medium is busy

- *wait until idle and transmit immediately*
- *called 1-persistent which is a special case of p-persistent: when the line becomes idle transmit with probability p.*

If a collision occurs . . .

Ethernet: transmitter algorithm

If a collision occurs

- *jam for 512 bits, then stop transmitting the frame*
- *the minimum frame is 64 bytes : header + 46 bytes of data = 512 bits so that the frame is long enough for a collision to be detected*
- *delay and try again: exponential backoff*
  - *1st time:* $U(0, 51.2)\mu s$
  - *2nd time:* $U(0, 102.4)\mu s$
  - *3rd time:* $U(0, 204.8)\mu s$
  - *give up after several tries, usually 16.*

where $U(0, x)$ *is a random number uniformly distributed in the range* $(0, x]$.

Ethernet: experiences

Observed in practice

- *10-200 hosts, not 1024*
- *length shorter than 2500m, RTT closer to 5μs than 51μs*
- *packet length is bimodal*
- *high-level flow control and host performance limit the carried load.*

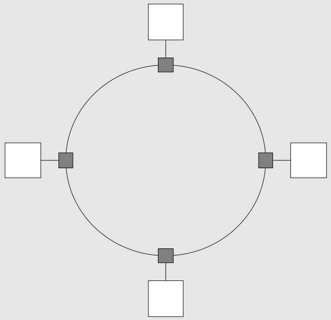
Recommendations

- *do not overload, 30% utilization is about max*
- *implement controllers correctly*
- *use large packets*
- *get the rest of the system right (broadcast, retransmission).*

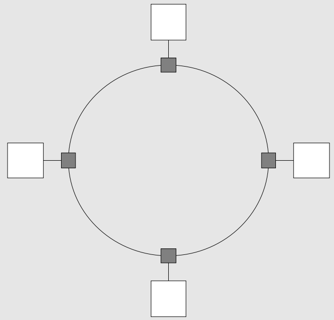
Token ring networks: overview

Token ring networks

- PRONET: 10Mbps and 80 Mbps rings
- IBM: 4Mbps token ring
- 16Mbps IEEE 802.5 token ring
- 100Mbps Fiber Distributed Data Interface (FDDI)



Token ring networks: overview

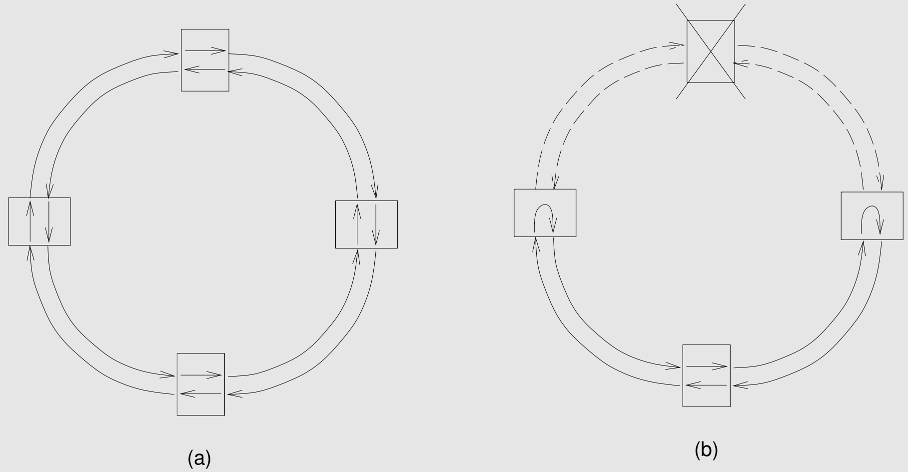


The basic idea

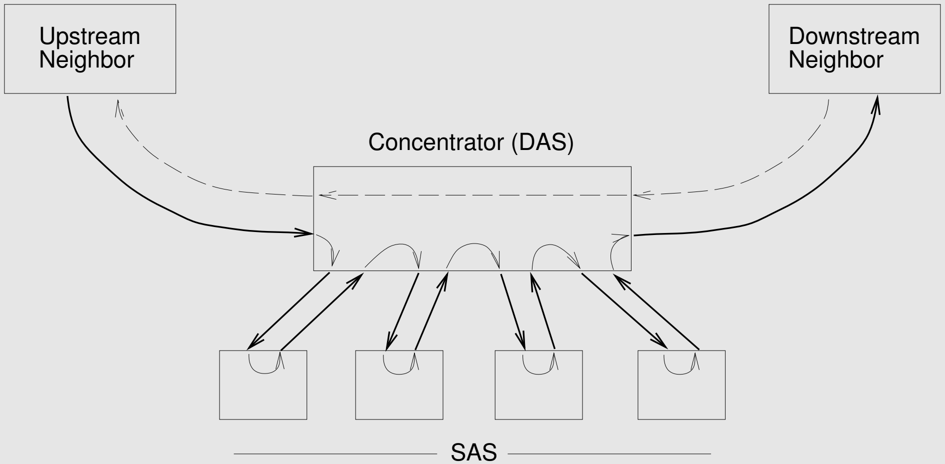
- *frames flow in one direction: upstream to downstream*
- *a special bit pattern (token) rotates around the ring*
- *a station must capture the token before transmitting*
- *a station releases the token after transmitting*
  - *early or delayed release*
- *station removes its frame when it comes back around*
- *stations get round-robin service.*

FDDI: physical properties of FDDI

Dual ring configuration



Single and dual attachment stations



FDDI: physical properties of FDDI

- *each station imposes a delay (e.g., 50ns)*
- *a maximum of 500 stations*
- *an upper limit of 100km (200km of fiber)*
- *uses 4B/5B encoding*
- *can be implemented over copper (CDDI).*

FDDI: timed token algorithm

- *the Token Holding Time (THT): the upper limit on how long a station can hold the token.*
- *the Token Rotation Time (TRT): how long it takes the token to traverse the ring.*

  $$TRT \leq ActiveNodes \times THT + RingLatency$$

- *the Target Token Rotation Time (TTRT): an agreed-upon upper bound on the TRT.*
- *Algorithm . . .*

FDDI: timed token algorithm

- *each node measures the TRT between successive arrivals of the token: the MTRT*
- *if MTRT $>$ TTRT then the token is late: don't send data*
- *if MTRT $<$ TTRT then the token is early: hold the token for TTRT$-$MTRT and send data*
- *define two classes of traffic*
  - *synchronous data: can always send*
  - *asynchronous data: can send only if token is early*
- *worse case: 2$\times$TTRT between seeing token*
- *not possible to have back-to-back rotations that take 2$\times$TTRT time*

FDDI: token maintenance

- Lost token
  - no token when initializing ring
  - bit error corrupts the token pattern
  - node holding the token crashes.
- Monitoring for a valid token
  - should see valid transmission (frame or token) periodically
  - maximum gap = ring latency + max frame $\leq$ 2.5ms
  - set timer at 2.5ms and send claim frame if it expires
- Generating a Token (and agreeing on TTRT) . . .

FDDI: token maintenance

- *generating a token (and agreeing on TTRT)*
- *execute when join ring or suspect a failure*
- *each node sends a special claim frame that includes the node's bid for the TTRT*
- *when a node receives a claim frame, update the bid and forward it*
- *if your claim frame makes it all the way around the ring*
  - *your bid was the lowest*
  - *everyone knows the TTRT*
  - *you insert a new token.*

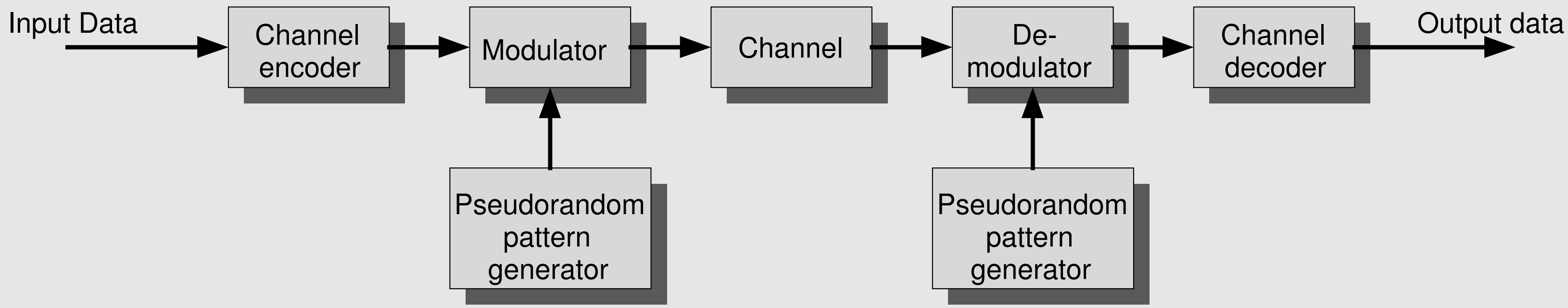
FDDI: frame format

8	8	48	48		32	8	24
Start of Frame	Control	Dest Addr	Src Addr	Body	CRC	End of Frame	Status

- *Control Field*
  - *1st bit: asynchronous (0) or synchronous (1) data*
  - *2nd bit: 16-bit (0) or 48-bit (1) addresses*
  - *last 6 bits: demux key (includes reserved patterns for token & claim frame)*
- *Status Field*
  - *from receiver back to sender*
  - *error in frame*
  - *recognized address*
  - *accepted frame (flow control)*

Wireless: 802.11

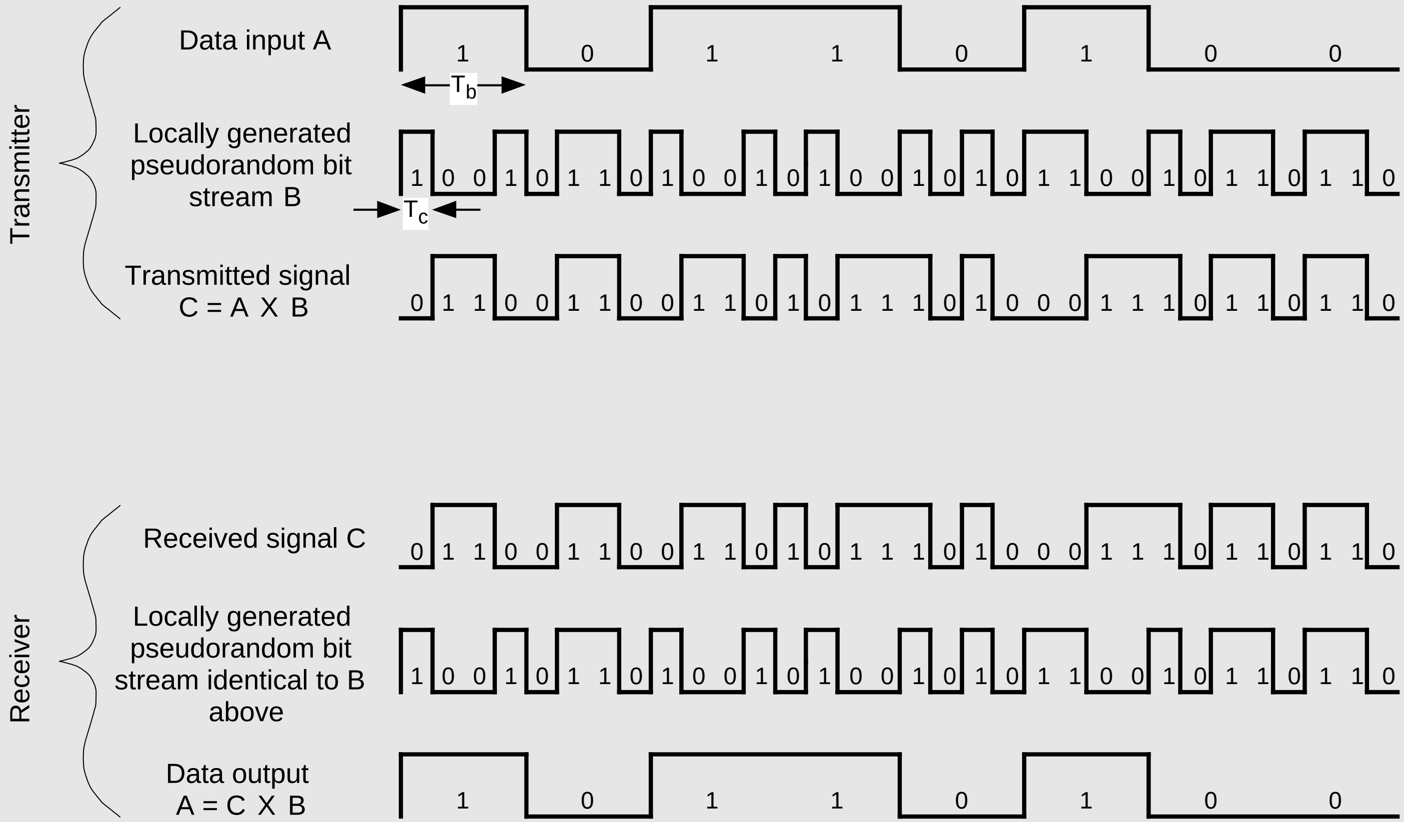
Spread spectrum spreads the signal over a wide frequency band to minimize interference from other devices



- *frequency hopping transmits the signal over a pseudo-random sequence of frequencies.*

802.11

- *direct sequence represents each bit in the original signal by multiple bits in the transmitted signal: the chipping code.*

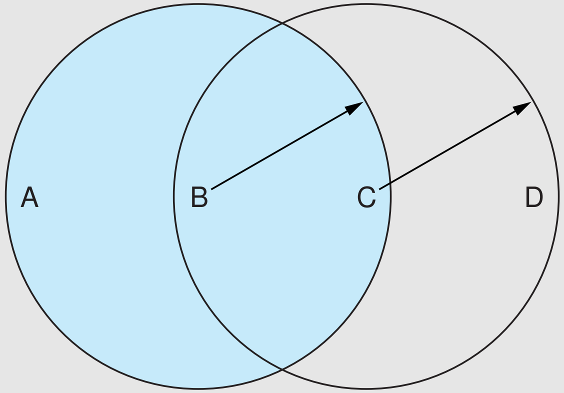


802.11

802.11 defines the physical layer in the 2.4Ghz band

- *frequency hopping: 79 1-Mhz wide frequency bandwidths*
- *direct sequence: 11-bit chipping code*
- *infrared: 10m range.*

802.11: Collision Avoidance

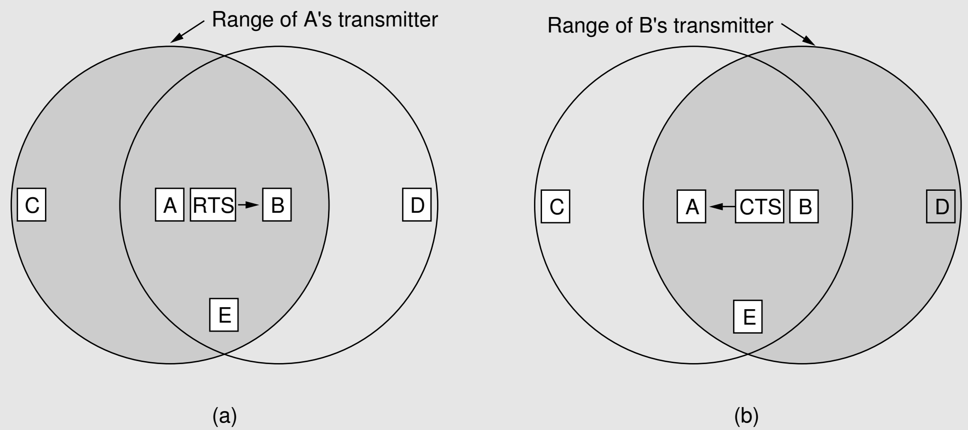


A and C wish to communicate with B. Their frames will collide at B. A and C are *hidden nodes* *with respect to each other – they are not aware of the collision.*

B sends to A. C is an *exposed node**: it is aware that B is sending to A, yet C can send to D.*

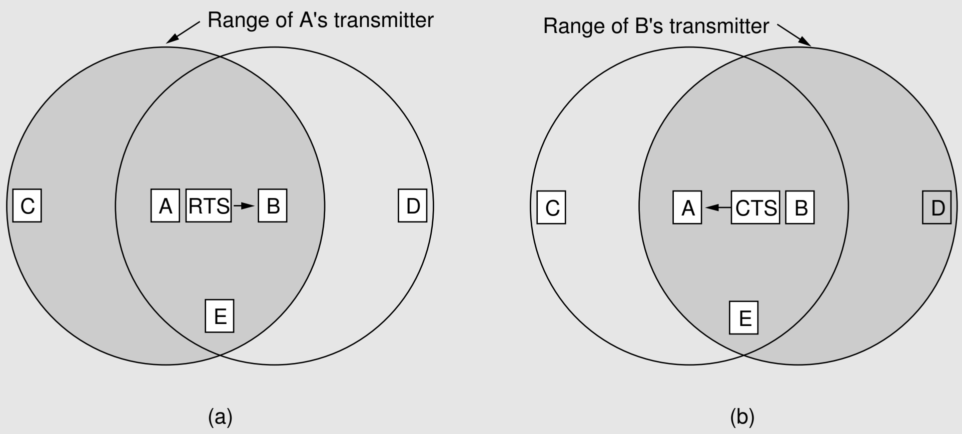
802.11: Multiple Access with Collision Avoidance

MACA: the sender & receiver exchange control frames before the sender transmits data.



- *A sends an RTS frame to B*
  - *the RTS frame specifies the* `length` *of the data frame to be transmitted: A will hold the medium for an amount of time* $T_{\texttt{length}}$
- *B replies with a CTS frame which contains* `length`.
- *A starts transmitting when it receives the CTS frame from B.*

802.11: MACA



- *A sends an RTS frame to B*
- *B replies with a CTS frame which contains* `length`
  - *D receives the CTS frame: D cannot transmit for an amount of time* $T_{\texttt{length}}$
  - *C receives the RTS frame but not the CTS frame: D can transmit after the CTS frame arrives at A.*

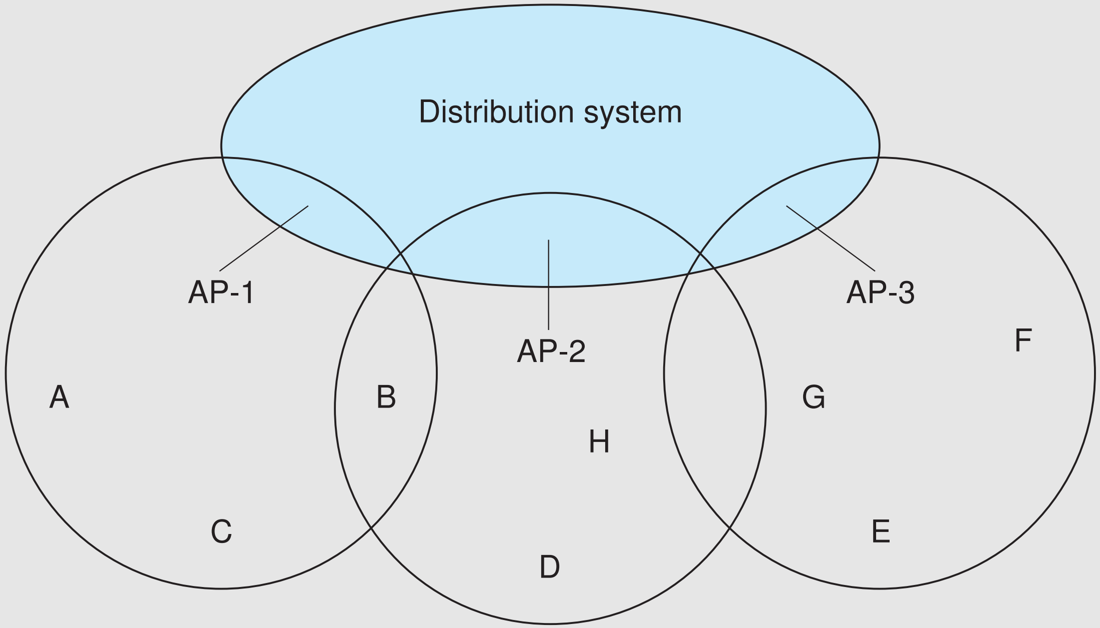
802.11: MACA for Wireless

- *Carrier sense: the sender listens to the medium – if the medium is idle the sender transmits else the sender waits until the medium is idle.*
- *If two or more senders find the medium to be idle their RTS frames may collide in which case the senders will not receive their CTS frames within a timeout period: they each wait a random amount of time (binary exponential backoff) before trying again.*
- *Stop-and-wait: each successfully received frame is ACKed.*

802.11: Carrier Distribution

Some nodes can roam. Some nodes are access points (APs) which are connected to a wired infrastructure.

APs are connected by a distribution system.



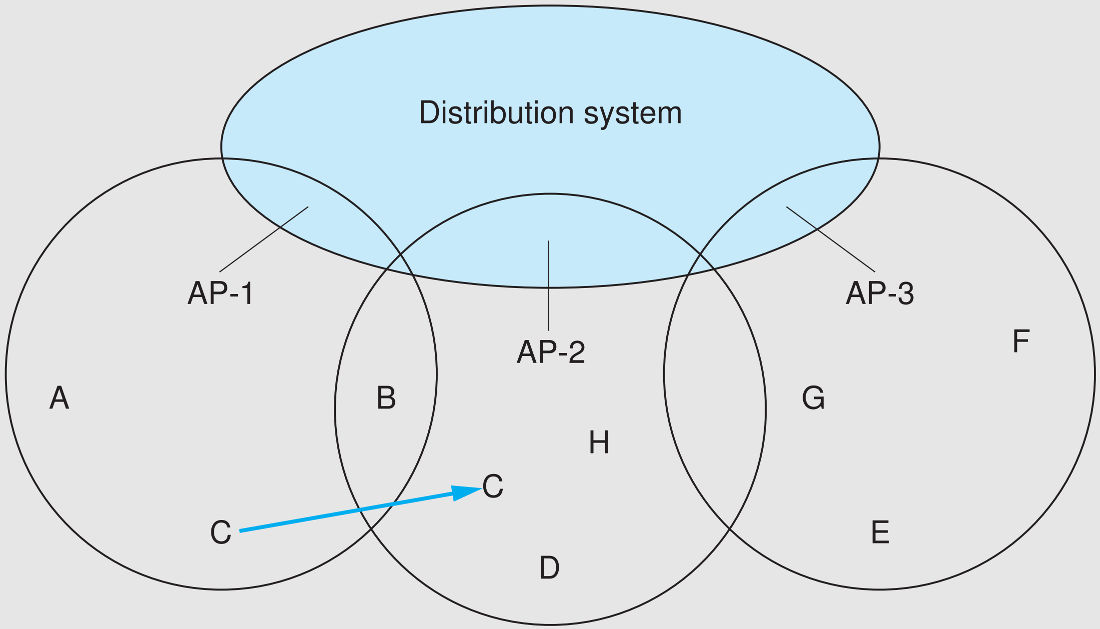
Each node is associated with one AP. When node A communicates with node E: A $\rightarrow$ AP-1 $\rightarrow$ AP-3 $\rightarrow$ E.

802.11: Active Scanning

When a node joins the network, or when a node decides to change its AP

- *the node sends a* `Probe` *frame*
- *all APs in reach reply with a* `Probe Response` *frame*
- *the node selects one of the APs & sends that AP an* `AssociationRequest` *frame*
- *the selected AP replies with an* `Association Response` *frame.*

802.11: Active & Passive Scanning



APs periodiocally send Beacon frames that advertise the capabilities of an AP.

802.11: Frame format

16	16	48	48	48	16	48	0–18,496	32
Control	Duration	Addr1	Addr2	Addr3	SeqCtrl	Addr4	Payload	CRC

- `Control`
  - *6-bit* `Type`*: data, CTS, RTS, scanning*
  - *1-bit* `ToDS`, `FromDS`
- *four addresses*
  - $\mathtt{ToDS} = \mathtt{FromDS} = 0$*: the source & destination are in the same cell*

    $\mathtt{Addr1} = \mathtt{source},\ \mathtt{Addr2} = \mathtt{dest}$
  - $\mathtt{ToDS} = \mathtt{FromDS} = 1$*: the source & destination are in different cells*

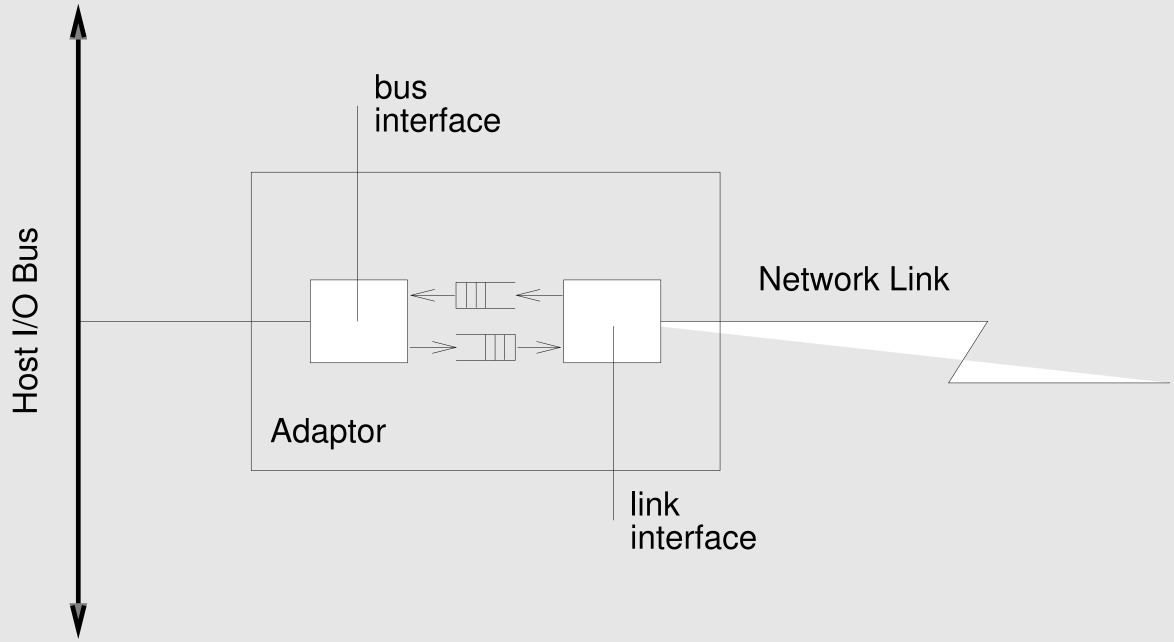
    $\mathtt{Addr1} = \mathtt{dest},\ \mathtt{Addr2} = \mathtt{Dest\ AP}$

    $\mathtt{Addr3} = \mathtt{source\ AP},\ \mathtt{Addr4} = \mathtt{source}.$

Network Adaptor: Overview

Typically where data link functionality is implemented

- *framing*
- *error detection*
- *media access control (MAC)*



Network Adaptor: Host Perspective

Control Status Register (CSR)

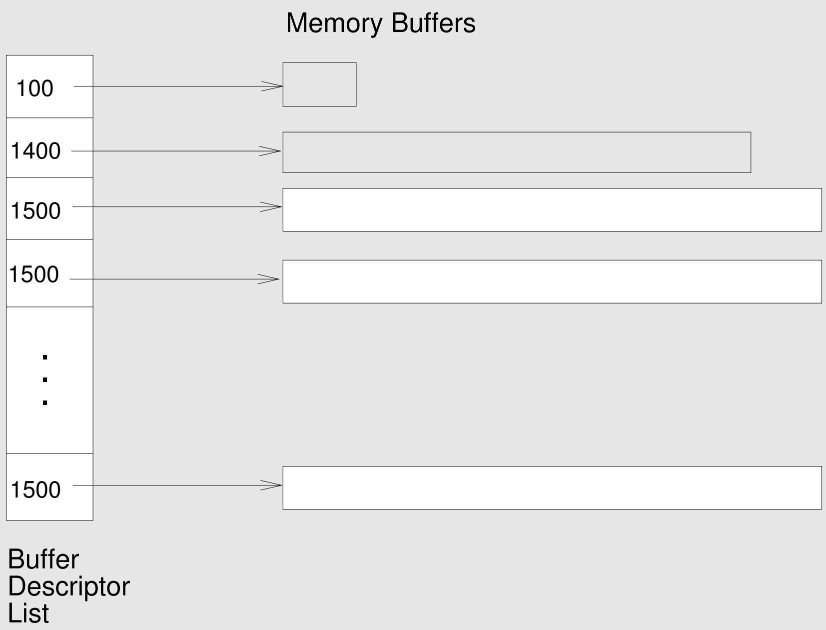
- *located at some memory address*
- *the CPU can read/write from/to the CSR*
- *the CPU writes to the CSR to instruct the adaptor (e.g. to transmit or receive)*
- *the CPU reads the CSR to learn the status of the adaptor (e.g. a receive error)*

Example

```
LE_RINT  0x0400  Received packet Interrupt (RC)
LE_TINT  0x0200  Transmitted packet Interrupt (RC)
LE_IDON  0x0100  Initialization Done (RC)
LE_IENA  0x0040  Interrupt Enable (RW)
LE_INIT  0x0001  Initialize (RW1)
```

Moving Frames Between Host and Adaptor

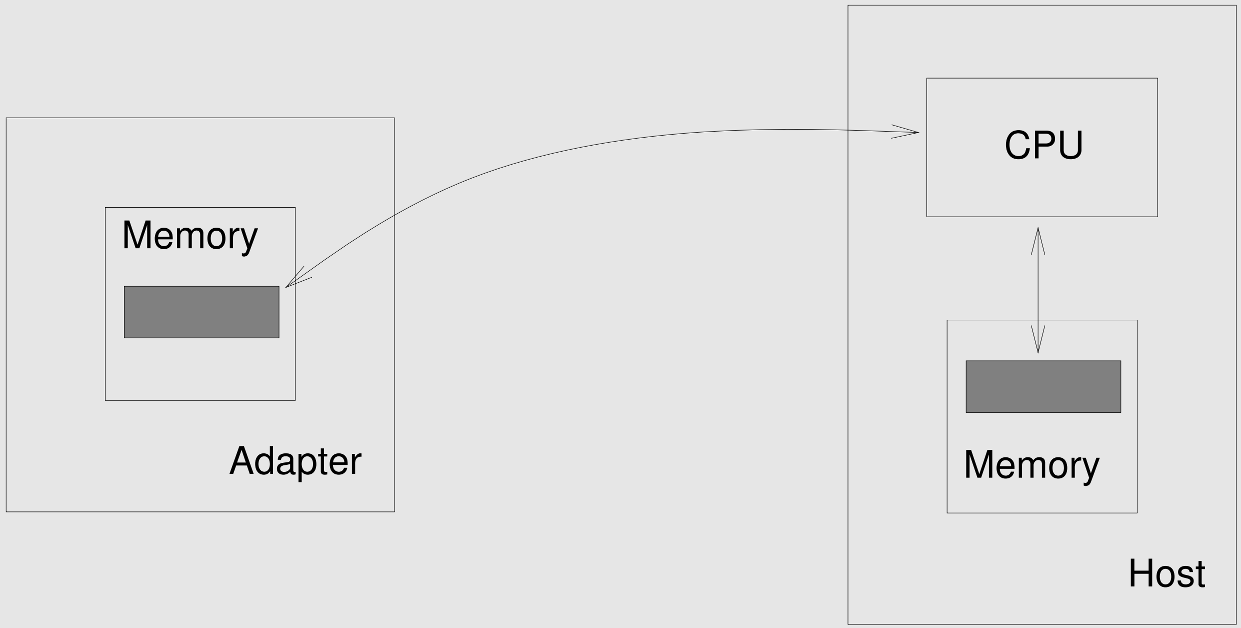
Direct memory access (DMA)



This illustrates scatter-read and gather-write where a frame is scattered over multiple buffers.

Moving Frames Between Host and Adaptor

Programmed I/O (PIO)



Network Adaptor: Device Driver

Interrupt handler

```
interrupt_handler() {
    disable_interrupts();
    /* some error occurred */
    if (csr & LE_ERR) {
        print_and_clear_error();
    }
    /* transmit interrupt */
    if (csr & LE_TINT) {
        csr = LE_TINT | LE_INEA;
        semSignal(xmit_queue);
    }
    /* receive interrupt */
    if (csr & LE_RINT) {
        receive_interrupt();
    }
    enable_interrupts();
    return(0);
}
```

Network Adaptor: Transmit Routine

```
transmit(Msg *msg) {
    char *src, *dst;
    Context c;
    int len;

    semWait(xmit_queue);
    semWait(mutex);
    disable_interrupts();
    dst = next_xmit_buf();
    msgWalkInit(&c, msg);
    while ((src = msgWalk(&c, &len)) != 0)
        copy_data_to_lance(src, dst, len);
    msgWalkDone(&c);
    enable_interrupts();
    semSignal(mutex);
    return;
}
```

Network Adaptor: Receive Interrupt Routine

```
receive_interrupt() {
   Msg  *msg,  *new_msg;
   char *buf;

   while (rdl = next_rcv_desc()) {
   /* create process to handle this message */
       msg = rdl->msg;
       process_create(ethDemux, msg);

   /* msg eventually freed in ethDemux */
   /* now allocate a replacement */
       buf = msgConstructAllocate(new_msg, MTU);
       rdl->msg = new_msg;
       rdl->buf = buf;
       install_rcv_desc(rdl);
   }
   return;
}
```