Data Link Layer

Mahalingam Ramkumar
Mississippi State University, MS

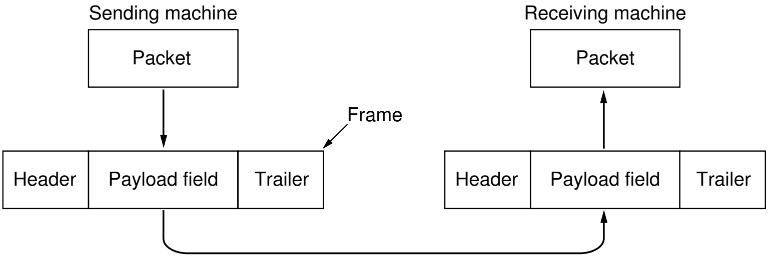
October 20, 2014

Outline

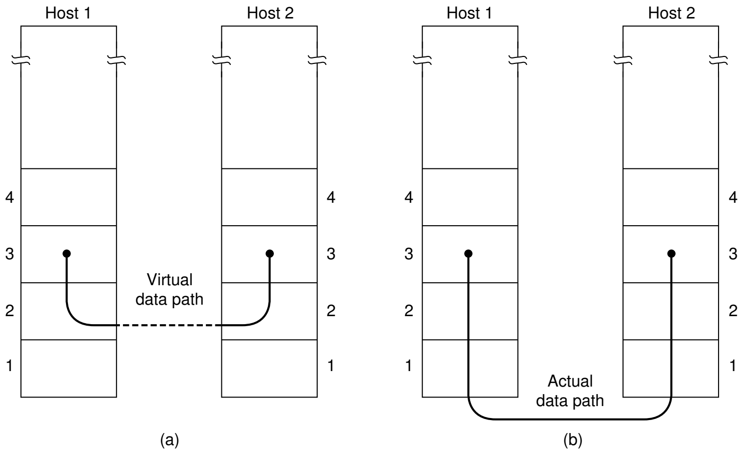
Data Link Layer

- Provide services to network layer
- Types of services
  - Unacknowledged connectionless service
  - Acknowledged connectionless service
  - Acknowledged connection-oriented service

Packet vs Frame



Virtual / Actual Communication Path



Issues

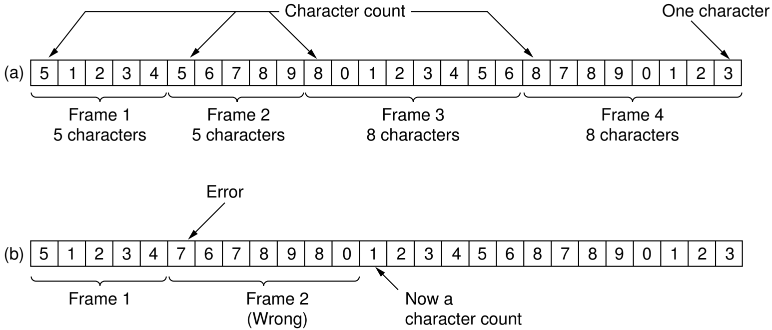
- Send a sequence of bits from one end of a wire to the other end.
- **Synchronization**: logical organization of frames
- **Integrity** of the data received (remember noise?)
- Type of service provided to higher layers
  - unreliable — connectionless, no acks
  - reliable — acks, connection oriented (multiple frames sent in a connection)
  - reliable and efficient delivery (ARQ protocols)
- For shared links two other issues: addressing and collisions
- Addressed by Medium Access Control protocols

Framing

- Organization of frames
- Methods
    - Character count
    - Byte stuffing
    - Bit stuffing

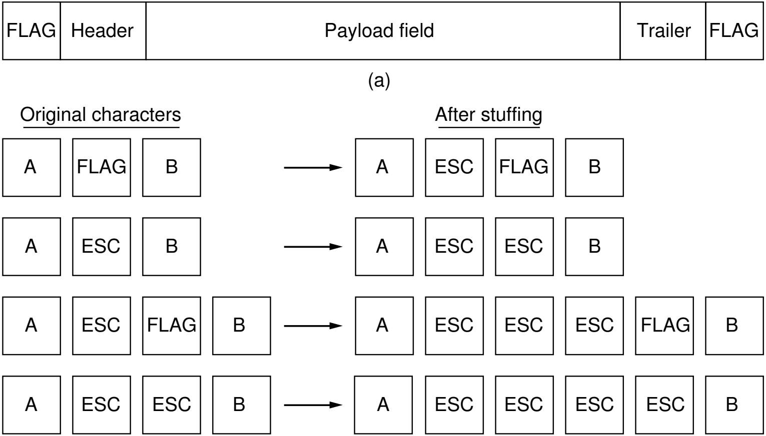
Character Count

Errors may result in loss of synchronization



Byte Stuffing

Used in PPP. Only for byte oriented communications



Bit Stuffing

Simple algorithm - 01111110 used as flag

At transmitter - after 5 consecutive ones insert a zero

At receiver - In a pattern of 111110 *anywhere* in the data, remove the trailing zero!

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Why do we need to stuff a 0 in 0111110?

Bit Stuffing

Why do we need to stuff a 0 in 0111110? (with 5 ones and a zero)
Even though flag has six consecutive ones? (01111110)
If we don't, notice the ambiguity

- 011111011 $\rightarrow$ 011111011 (unchanged)
- 01111111 $\rightarrow$ 011111011 (0 stuffed to separate 5th and 6th one)

If we do the ambiguity is resolved

- 011111011 $\rightarrow$ 0111110011
- 01111111 $\rightarrow$ 011111011

Errors Are Unavoidable

- Need to detect errors
- correct errors / request retransmission
- The key: **redundancy** — extra bits need to be transmitted for detecting / correcting errors
- Example: parity bits for error detection

Hamming Distance

- A metric for measuring "distances" between two sequences of bits
- $A = 1000110$
- $B = 1010100$
- How many bits need to be flipped to get B or A (or vice-versa)?
- Two bits — the *Hamming distance* between A and B is 2.

Error Detection with Parity Bit

- $A = 1000110$
- With even parity $A = 1000110\mathbf{1}$
- With odd parity $A = 1000110\mathbf{0}$
- For all subsequent discussions we will only use *even* parity
- If any of the eight (7+1) bits of A is flipped during transmission parity check will fail.
- What happens if two bits get flipped?
- Parity check fails to detect error...
- With one redundant bit we can only detect one-bit errors.
- Actually any odd number of bit-errors

Redundancy

- Need to transmit m bit symbols (2^m possible symbols)
- r redundant bits (for error detection / correction)
- $n = m + r$ bits actually transmitted.
- Efficiency is $\frac{m}{m+r} = \frac{m}{n}$
- In general, more the redundancy, more the errors that can be detected / corrected

A Simple-Minded Approach

- Just repeat every bit twice (efficiency 0.5) - any one bit error can be detected
- Parity bit (just add one extra bit) - efficiency $\frac{m}{m+1}$ is high for large m.
- How can we correct single bit errors automatically?
- Repeat each bit two more times (efficiency 0.33)
- Are there better ways?

Error Detection vs Correction

- Error correction needs more redundancy
- Error detection is much more crucial than error correction — why?
- Practical error detection techniques address the problem of detecting multiple-bit errors.
- Error correction is usually used only for correction of single bit errors.

Single Bit Errors are Statistically More Frequent!

- Let us assume that the probability that any bit may be erroneously received is $p = 10^{-9}$.
- If $N = 10^3$ bits are transmitted, probability that there may be a bit error is roughly one in a million.
- Probability that there may be two errors is roughly one in a trillion (million x million)
- We need to detect even rare errors — but not a big deal if we can not correct it (request re-Tx).

Our Focus

- **How do we correct one-bit errors?**
- If the receiver is able to correct errors, the sender does not need to retransmit
- **How do we detect multiple-bit errors?**
- If error is detected, the receiver can request the sender to retransmit
- Or simply *not* acknowledge reception

One-bit Error Correction

- Problem statement:
  - X is transmitted n-bit message
  - Y is received n-bit message
  - $H(X, Y) \leq 1$ (Hamming distance is at most 1)
- For any X there is a set consisting of $n + 1$ possible Ys
  - $Y = X$, or
  - other n cases where only one bit is flipped (first or second or $\cdots$ or n^{th})
- Example, $X = 1011101$ ($n = 7$), Y has eight possible values

$$Y \in \begin{bmatrix} 1011101 & 1011100 & 1011111 & 1011001 \\ 1010101 & 1001101 & 1111101 & 0011101 \end{bmatrix}$$

One-bit Error Correction

- For each X there is a Y-set ($n+1$ possible Ys)
- Each set includes X, and n values one Hamming distance away away from X
- What is the maximum number of non-overlapping sets? $2^n/(n+1)$.
- If X is a 7 bit number, then $2^n/(n+1) = 128/8 = 16$
- If X is a 10 bit numbers then $2^1 0/11 = 1024/11 = 93.09$ (or 93 non-overlapping sets)

One-bit Error Correction

- If $2^m \leq 2^n/(n+1)$ we have non overlapping set of Ys for every m-bit number
- For every m-bit number there is a set with i) an n-bit X and ii) n numbers one Hamming distance away from X
- The Xs are 2^m unique codes represented using $n = m + r$ bits.
- If any of the 2^m Xs are sent, we can readily identify the which one was sent even if a bit was flipped in the channel.
- To send an m-bit number over a noisy channel (in which not more than one bit can get flipped) we send a $n = m + r$-bit code

Efficient Choice of n and m

- If $2^n/(n+1)$ is a power of 2 then we have $2^m = 2^n/(n+1)$.
- Else we have to choose $2^m < 2^n/(n+1)$ (which is not efficient as we waste some usable codes)
- Efficient codes are represented as (m, n) (Hamming codes)
- $m = 1$; $n = 3$; $r = 2$; — an efficient code as $2^1 = 2^3/4$.
- $m = 2$; $n = 5$ $r = 3$; — *not* efficient as $2^5/6 = 32/6 > 2^2 = 4$
- $(m = 4, n = 7)$ $(r = 3)$ is an efficient code $(2^7/(7+1) = 128/8 = 16)$
- For efficient codes $n = 2^r - 1$, $m = n - r$
- $r = 8$, $n = 256 - 1 = 255$, $m = 255 - 8 = 247$: $(247, 255)$ code
- $r = 10$, $n = 1024 - 1 = 1023$. $m = 1023 - 10 = 1013$: $(1013, 1023)$ code

Hamming Codes

- $n = 2^r - 1$: efficient codes like $(4, 7), (11, 15), \ldots (1013, 1023), \ldots$
- $(4, 7)$ code: $r = 3$ parity bits, $n = 7$ transmitted bits, $m = 4$ information bits.
- Each code of the form $b_7\, b_6\, b_5\, \mathbf{b_4}\, b_3\, \mathbf{b_2}\, \mathbf{b_1}$
- The bold bits are $r = 3$ parity bits. Other $m = 4$ bits are the actual information bits.
- What is special about positions 1,2, and 4?
- Powers of 2. $(2^0, 2^1, 2^2)$.

Hamming Code (4, 7)

- $b_7\ b_6\ b_5\ \mathbf{b_4}\ b_3\ \mathbf{b_2}\ \mathbf{b_1}$
- b_1 is the parity bits for bits b_7, b_5, b_3 (all odd positions 7,5,3,1)
- b_2 is the parity for bits b_3, b_6, b_7 (positions 2,3,6,7)
- b_4 is the parity for bits b_5, b_6, b_7 (positions 4,5,6,7)
- Assume we want to send $m = 4$ bits 1011
- $b_7 = 1\ b_6 = 0\ b_5 = 1\ b_4 =?\ b_3 = 1 b_2 =?\ b_1 =?$
- $b_1 = 1,\ b_2 = 0,\ b_4 = 0$ are the correct parities
- The code for 1011 is 101**0**1**01**

Hamming Code Example

- The receiver checks parity bits and writes down the results of the check as $x_4 x_2 x_1$
- $x_4 = 0$ if parity b_4; $x_4 = 1$ if parity b_4 is wrong.
- If no bit is flipped by the channel then all parity bits will be correct (result 000)
- If b_7 is flipped all three parities will be wrong (result 111)
- If b_3 is flipped result is 011
- If b_5 is flipped result is 101
- See a pattern? The result gives the position of the erroneous bit.
- Go ahead and flip it back.

Hamming Code Example

- Code $X = 1010101$
- If $Y = 1010101$ (no error) result 000
- If $Y = 1010100$ (pos 1 error) result 001
- If $Y = 1010111$ (pos 2 error) result 010
- If $Y = 1010001$ (pos 3 error) result 011
- If $Y = 1011101$ (pos 4 error) result 100
- If $Y = 1000101$ (pos 5 error) result 101
- If $Y = 1110101$ (pos 6 error) result 110
- If $Y = 0010101$ (pos 7 error) result 111

(11, 15) Hamming Code

Code $b_{15} \cdots b_1$

b_1, b_2, b_4, b_8 are parity bits

Example 0 1 0 1 0 1 0 ? 0 0 1 ? 0 ? ? (11-bit information 01010100010)

b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
0	1	0	1	0	1	0	?	0	0	1	?	0	?	?
0	1	**0**	1	**0**	1	**0**	?	**0**	0	**1**	?	**0**	?	**1**
0	**1**	0	1	**0**	**1**	0	?	**0**	**0**	1	?	**0**	**0**	1
0	**1**	**0**	**1**	0	1	0	?	**0**	**0**	**1**	**1**	0	0	1
0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	0	0	1	1	0	0	1

Result written as $x_8 x_4 x_2 x_1$.

If bit 13 is flipped result is 1101

Syndrome Coding

- A syndrome is a "collection of symptoms."
- Each result of parity check is a symptom: each bit of the result (for example 1011 if $r = 4$ is a symptom)
- The syndrome should *unambiguously* indicate the error.
- With r redundant bits, we have 2^r unique syndromes
- We can detect $2^r - 1$ different "diseases" (one syndrome corresponds to "no illness").
- $n \leq 2^r - 1$

Burst Errors

- Burst errors affect a consecutive sequence of bits
- Power surges, explosions
- Hamming codes can only detect one-bit errors
- Can we handle burst errors?
- Yes: by *interleaving*

Interleaving

- 40 bits need to be sent
- Use (7,4) Hamming code to encode 4 bits at a time: 40 bits becomes 70 bits at positions $(1, 2, \ldots, 70)$
- 10 independent codes at positions $(1, 2, 3, 4, 5, 6, 7)$, $(8, 9, 10, 11, 12, 13, 14) \cdots (64, 65, 66, 67, 68, 69, 70)$
- Reorder the sequence of 70 bits by interleaving
- (1,8,15,... 64), (2,9,...,65), (3,10,...,66) ··· (7,14,... 70)
- At the receiver: de-interleave and then decode
- Any burst sequence up to 10 bits long will produce at most error in each of the ten codes

Error Detection

- Single error detection with parity bit
- Burst error detection: Interleaving

Multi-bit Error Detection

- Double errors
- arrange $n = w \times h$ bits as a matrix
- Add parity bit for each row (we now have $(w+1) \times h$ matrix)
- Parity bit for each column — we end up with a $(w+1) \times (h+1)$ matrix
- If 2 errors happen in the same row (column) they cannot be in the same column (row)

Cyclic Redundancy Checks (CRC)

- A polynomial - $y = P(x) = x^r + a_{r-1}x^{r-1} + \cdots + a_1 x^1 + a_0$
- If $P(x)$ is in the field of real numbers, $x, a_0 \ldots a_{r-1}, y$ can be any real number
- Any bit string can be seen coefficients of a polynomial in the finite field $\{0, 1\}$
- $P(x), x, a_0 \ldots a_{r-1}$ can only be 0 or 1
- Example: 110101 is $P(x) = x^5 + x^4 + x^2 + x^0$.
- Addition: $1 + 1 = 0 = 0 + 0$, $1 + 0 = 0 + 1 = 1$
- Subtraction is the same as addition!
- Multiplication: $1 \times 1 = 1$, $0 \times x = 0$.
- $P(0) = 1$. $P(1) = 1 + 1 + 1 + 1 = 0$.

Polynomial Arithmetic

- $P(x) = x^5 + x^4 + x^2 + 1$. $Q(x) = x^2 + x^1$
- $P(x) + Q(x) = x^5 + x^4 + x^1 + 1$
- $P(x) \times Q(x) = (x^5 + x^4 + x^2 + 1)(x^2 + x^1) = x^7 + x^6 + x^4 + x^2 + x^6 + x^5 + x^3 + x^1 = x^7 + x^5 + x^4 + x^3 + x^2 + x^1$.
- Polynomial division - what is $P(x)/Q(x)$

```
110 |110101 | 1001 (quotient)
     110
     ---
     000101
        110
        ---
        011  (remainder)
```

CRC

- Choose a generator polynomial $G(x)$ of degree r ($r+1$ bit number)
- Message to be sent $M(x)$
- Append r zeros to $M(x)$. The result is $x^r M(x)$.
- Evaluate the remainder of $x^r M(x)/G(x)$ (remainder will be a polynomial of degree *less* than r) - say $R(x)$
- Transmit $T(x) = x^r M(x) - R(x) = x^r M(x) + R(x)$
- Receiver receives $T(x)$
- In case of error receiver gets $T'(x) = T(x) + E(x)$
- Receiver checks if remainder of $T'(x)/G(x)$ is zero
- If remainder is not zero receiver decides that there is an error
- As $T(x)/G(x) = 0$, $T'(x)/G(x) = E(x)/G(x)$
- Errors where $E(x)/G(x)$ is zero goes *undetected*

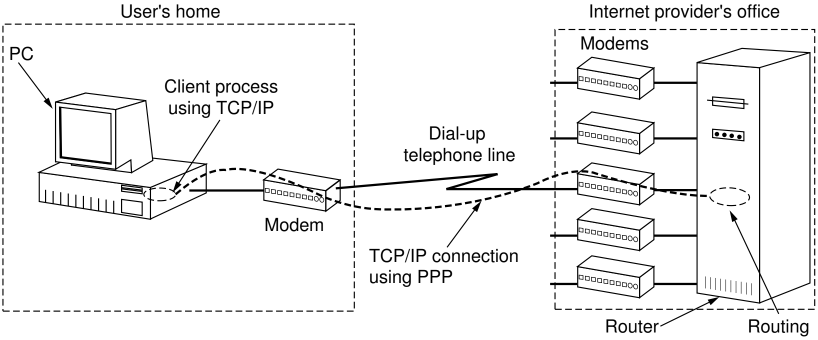
Good choice for $G(x)$

- Both highest and lowest order bits should be 1 $(1xx\cdots xx1)$
- If $G(x)$ has degree more than 1 all single bit errors will be detected - $E(x) = x^i$.
- Two isolated single bit errors $E(x) = x^i + x^j = x^j(x^{i-j} + 1)$: any $G(x)$ that does *not* have $x^k + 1$ as a factor (for all $k \leq m$) is sufficient
- If $x + 1$ is not a factor, then *all* odd number of errors can be detected.
- Any burst error of length $\leq r$ can be detected
- 32 bit polynomial specified in IEEE 802x

DL Layer Tasks

- Receive packets from NL
  - Add header
  - Add CRC
  - Frame the packet (bit/byte stuffing + flags at either end)
  - Hand over DL frame to MAC layer (if the medium is shared), or Physical layer
- Receive packets from PL/MAC layer
  - Unframe (remove stuffed bits/bytes and flags) the packet
  - Check CRC
  - Remove header and footer
  - Handover payload to higher layer

PPP in Dial-up Internet



Point-to-Point Protocol

- RFC 1662, 1663
- Features
  - Framing
  - Link Control Protocol (LCP)
  - Network Control Protocol (NCP)

Overview of PPP

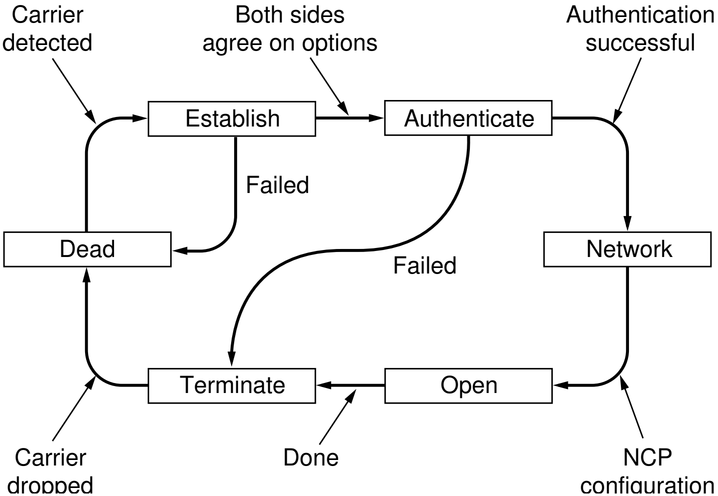
- Byte oriented protocol
- Byte stuffing for framing
- Physical layer is a telephone connection between two *modems* (client's modem makes a call to the modem of the ISP)
- Telephone connection established. PPP frames are sent instead of voice signals over the telephone line.
- Initially LCP packets sent in PPP frames for negotiation of physical layer parameters.
- Following this NCP packets are carried by PPP frames to establish network layer parameters (IP address)
- Now the host has access to the Internet. IP packets can be sent inside PPP frames.

PPP Frame Format

- Flag byte 01111110
- Address (not needed for point to point connection) always set to 11111111
- Control field: default value indicates no ACKs.
- During LCP parties can negotiate and decide to drop address and control bytes
- Protocol: specifies LCP, NCP and network protocols like IP, IPX, AppleTalk etc.
- Payload (LCP, NCP, IP packets)
- Checksum: 2 or 4 bytes (negotiated during LCP)

Bytes	1	1	1	1 or 2	Variable	2 or 4	1
	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110

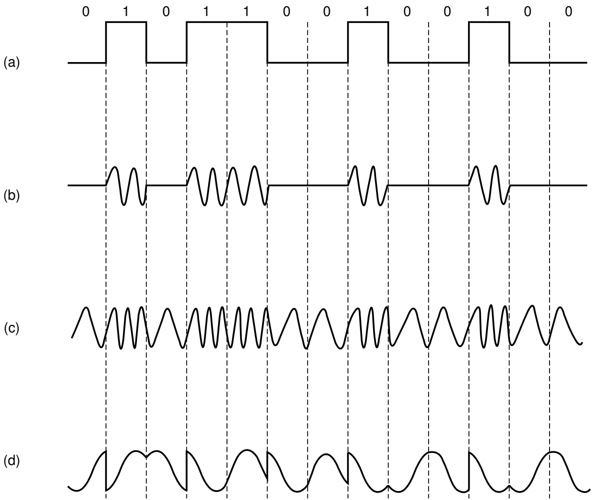
PPP Phases


Carrier detected
Both sides agree on options
Authentication successful
Establish
Authenticate
Failed
Dead
Network
Failed
Terminate
Open
Carrier dropped
Done
NCP configuration

Dial-up Internet

- Send bits over telephone cable
- Cable meant for carrying voice signals between 300-3000 Hz
- Done by modulating analog signals with bits.
- Modulation: carrier signal + information signal = modulated signal
- Demodulation: separating the carrier and information signal
- For our purposes the information signal consists of bits.
- The modulated signal should have characteristics similar to voice (30-3000 Hz bandwidth)
- Three types of modulation: amplitude, frequency, and phase modulation

Amplitude, Frequency and Phase Modulation



Dial-up Internet

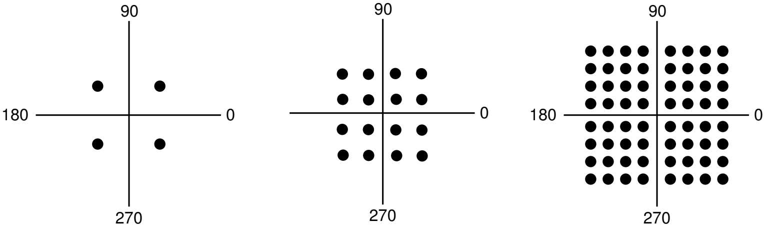
- Amplitude and phase modulation: bits are chunked and each chunk represented by a clipped sinusoidal pulse
- Bit rate and Baud rate
  - Baud rate is the number of sinusoidal pulses per second: fixed at 2400 per sec.
  - Each pulse represents k bits
  - The receiver needs to be able to differentiate between 2^k different pulses used to represent 2^k possible chunks.
  - If we use 8 different types of pulses we can send three bits per pulse ($8 = 2^3$)
  - We have managed to go up to 56 K with baud rate of 2400 (about 23 bits in each chunk)
  - How many different shapes? $> 2^{23} \approx 8$ million!

Modem

- **Mo**dulator - **dem**odulator
- Bits are chunked, converted to sinusoidal pulses (modulated)
- pulses sent over telephone connection
- The received signal is converted back to bits by demodulator
- Clipped sinusoid is the basic signal
- Variations achieved by modifying amplitude and phase
- Different signals can be represented on a *constellation* diagram

QPSK, QAM-16, QAM-64

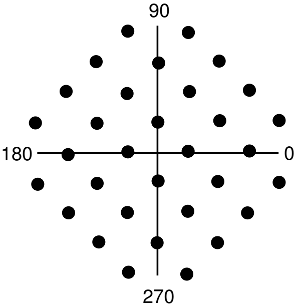
Quadrature Phase Shift keying (2 bits per sample), Quadrature Amplitude Modulation



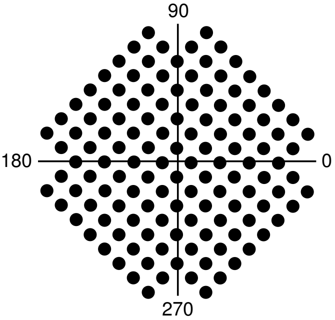
(a) (b) (c)

Trellis Coded Modulation

- Add more bits per sample
- Add extra bits to each sample for error correction
- V.32 (4+1, 32) - 9600 bps, V.32 bis (6 + 1, 128) 14,400 bps
- V.34 - 28,800 bps, V.34 bis - 36,600 bps



(b)



(c)

Modern Modems

- Full duplex: transmissions possible in both directions at the same time
- Half-duplex: Both directions, but not at the same time
- Simplex: Only one direction
- V.90, V.92 are full duplex standards
- Dedicated uplink and downlink channel
- 56 kbps downlink, 33.6 kbps uplink (V.90)
- 48kbps uplink (V.92) + facility to detect incoming calls while online

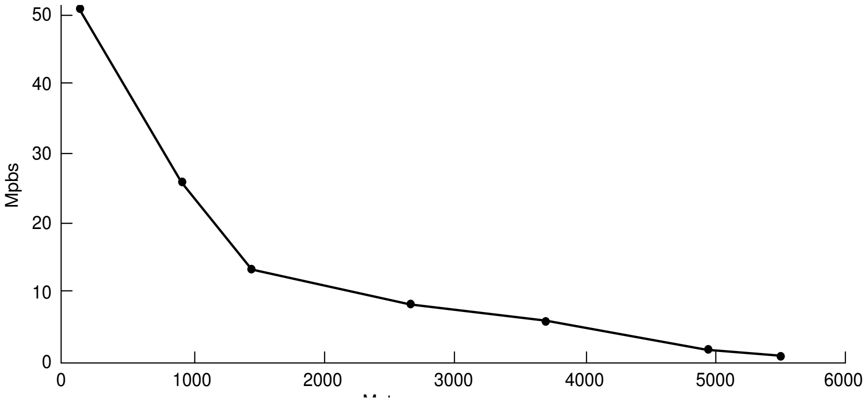
LCP in PPP

- The bits sent as pulses correspond to fax data or PPP frames
- Telephone connections can have great variation in quality
- Initially a base-line quality is assumed to send PPP frames carrying LCP packets.
- LCP is negotiation to agree on achievable bits per chunk and the appropriate constellation to use.

NCP in PPP

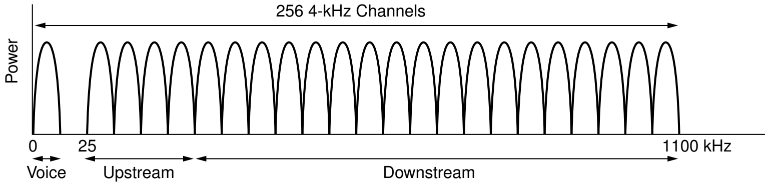
- After LCP an improved PPP link can be used to carry payloads
- Before IP packet can be sent NCP is needed
- NCP involves client authentication, issuing an IP address to the client, and conveying other useful parameters like net-mask and IP of a local DNS server.
- Only after NCP does the client become a “citizen of the Internet” capable of sending and receiving IP packets to any Internet host

Unused Bandwidth!

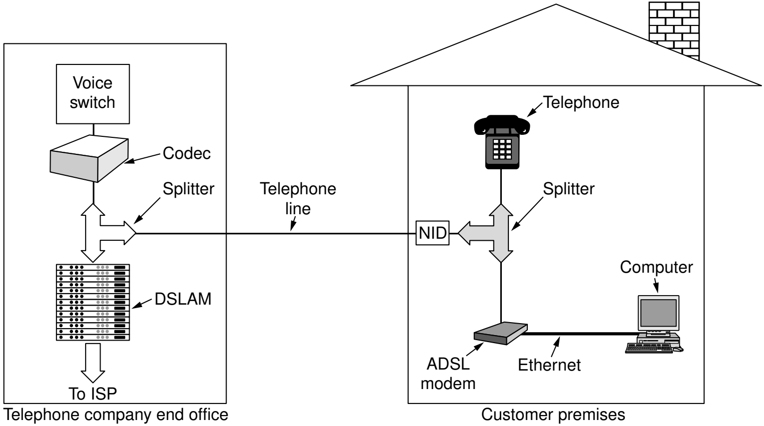


ADSL

- We have been sitting on a gold-mine!
- Telephones filter out higher frequencies to suppress noise
- 1.1 MhZ specturm divided into 256 channels - 4312 Hz each (DMT - Discrete Multitone)
- First channel used for POTS (plain old telephone service)

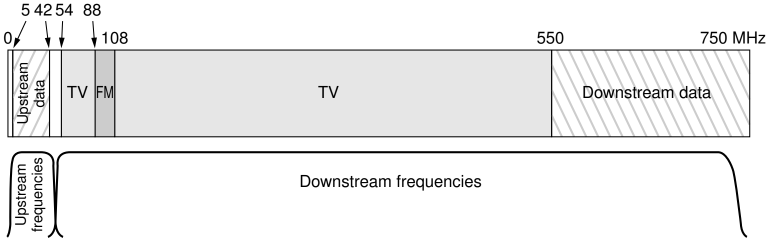


ADSL Equipment



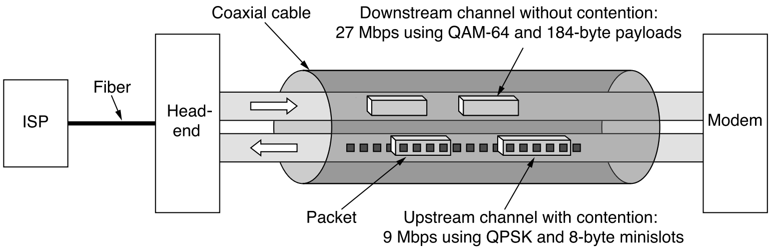
Cable Internet

- Usually the cable is a ring around the neighborhood (shared) connected to a head-end
- Over 750 MHz bandwidth - most used for TV channels



Upstream and Downstream

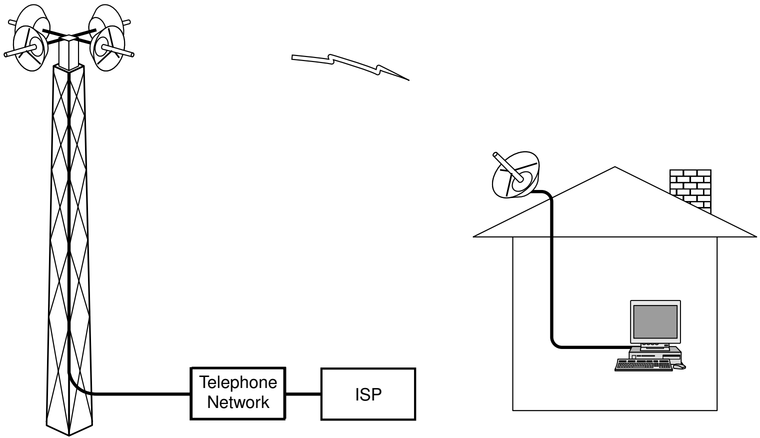
- *Contention* in upstream



ADSL vs Cable

- Bandwidth
- Peak performance
- Upstream / Downstream dynamics
- Contention
- Security
- QOS (Quality of Service)

Wireless Local Loops



Practical Limitations

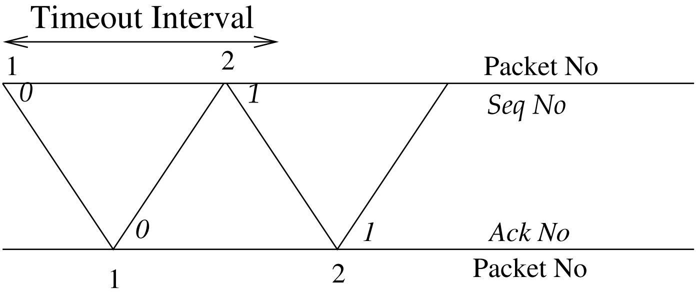
- Packet size (limited usually by channel error-rate): frame size F bits
- Packet duration depends on channel data rate. If data rate is R bps then packet duration is $P = \frac{F}{R}$ seconds
- Finite propagation time: depends on distance between source and destination, and velocity of propagation (electromagnetic waves) in the medium
- If c is the velocity, and L is the distance, propagation time is $\tau_c = \frac{L}{c}$.
- Typically $c \approx 2.5 \times 10^8$ m/s in copper wires.
- Processing time τ_r - a finite amount of time needed for processing packets

Practical Example

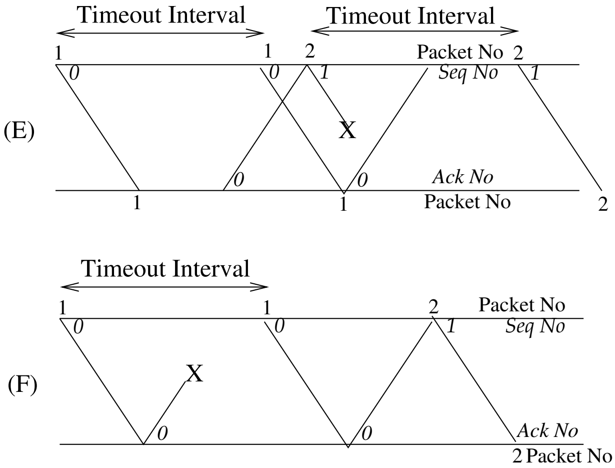
- 100 Mbps channel ($R = 10^8$), packet size 2000 bits
- $P = \frac{2000}{100000000} = 20\ \mu s$
- $L = 1000m.\ c = 2.5 \times 10^8$ m/s. $\tau_c = \frac{1000}{2.5 \times 10^8} = 4\mu s$
- Processing time - say $5\mu s$
- At $t = 0$ A starts transmission
- At $t = 4\mu s / t = 24\mu s$ B senses leading/trailing edge
- At $t = 29\mu s$ B starts ACK (assume 2000 bits)
- At $t = 33\mu s / t = 53\mu s$ A senses leading/trailing edge
- At $t = 58\mu s$ A begins to transmit *next* packet
- Round trip time (RTT) - $P + \tau_c + \tau_r + \tau_c + P + \tau_r$
- A used the channel for 20 μs and did not use the channel for 38 μs
- In a 100 Mbps channel A can only achieve 34 Mbps?

Alternating Bit Protocol (ABP)

(D)



ABP is Unambiguous!



Effective Data Rate

- 100 Mbps link $A \leftrightarrow B$ ($R = 100x10^6$), $L = 2000m$, packet size $F = 100$ bits, $\tau_r = 1\mu$s
- $P = \frac{100}{100000000} = 1\mu$s, $\tau_c = \frac{2000}{2.5\times10^8} = 8\mu$s
- What is the effective data rate between A and B when ABP is used?
- RTT $= 2P + 2\tau_c + 2\tau_r = 2 + 16 + 2 = 20$.
- In each 20 μs interval transmission is done only for 1 μs
- Effective data rate is 5 Mbps (P/RTT x R)

Improving Throughput

- Sender receives the ACK for the first packet after 20 μs
- Sender does nothing for 19 μs
- What if sender is allowed to send 20 packets before receiving the ACK for the first packet?
- After the ACK for the first packet is received the sender can send packet 21
- After the ACK for the second packet is received the sender can send packet 22, and so on
- Window size 20.

Pipelining

- The secret to higher throughput
- Assumes nodes can do “multiple” things at the same time.
- ABP needs only half-duplex channels, for pipelining we require full duplex channels (ACKs and packets cross each other)
- Nodes may need to buffer packets when things do not go well
- What is the buffer size needed?
- How do we number packets and acknowledgements ?

Selective Repeat Protocol

- Similar to ABP, with window size W
- Or ABP is SRP with window size 1
- Buffer size of W packets
- $2W$ distinct numbers. If $W = 4$, numbers 0,1,...7.
- P0 to P7 numbered 0 to 7 respectively
- P8 / P16 are also numbered 0; P9 / P17 are numbered 1...and so on.
- Sender window (size W): If i is the earliest outstanding ACK, window includes $i, i+1, \ldots i+w-1$. Sender can only send packets within the sender's window.
- Assume $W = 4$. If sender has received ACKs for $P0$, $P2$ and $P3$ (sender has received ACKs with numbers 0, 2 and 3), sender window includes $P1, P2, P3, P4$.

SRP: Receiver Logic

- Receiver window size $2W$. If j is the earliest *missing* packet
  - the receiver's pessimistic window includes W *past* packets $(j-1, j-2, \cdots j-W)$ and
  - the receiver's optimistic window includes W future packets $(j, j+1 \cdots j+W-1)$.
- Receiver will accept only packets within the two windows.
- Packets falling under the past window are ACKed and dropped (already received them).
- Packets falling under the future window are ACKed and stored.

SRP: Receiver Logic

- Example $W = 4$. Receiver who has (received and) acknowledged packets $P0 \cdots P4$ $(0, 1, 2, 3, 4)$ can receive (as the next packet) $P1 \cdots P8$ $(1, 2, 3, 4,\ 5, 6, 7, 0)$.
- Receiver has received packets $P0 \cdots P4, P5$ $(0, 1, 2, 3, 4, 5)$ has future window $(6, 7, 0, 1)$ and past window $(2, 3, 4, 5)$.
  - Can receive (as the next packet) $P2 \cdots P9$ $(2, 3, 4, 5,\ 6, 7, 0, 1)$.
  - If next packet has number 6 or 7 or 0 or 1 *store* and send ACK
  - If next packet has number 2 or 3 or 4 or 5 *ignore and send ACK.*
- Both sender and receiver require a buffer for W packets.

Negative Acknowledgements (NACK)

- Assume ACK for packet number 2 not received
- Normally sender would wait for a timeout period
- What if an ACK for 3 (while 2 is missing) is interpreted as a NACK for 2?
- ACK for 3 is (in general) received well before time-out for the ACK for 2.
- Typically NACKs lower the number of packets that need to be buffered.

Go Back N - GBN Protocol

- Window size W
- Sender logic is the same
- Receiver does not buffer packets (no window)
- Out of sequence packets are discarded
- Periodically sends acknowledgement for the last received packet number
- Example, if Rx gets packets 1,2,3,4,5,7,8,9,10 the ACK for 5 is sent. Sender gets no info about subsequent packets.
- Sender retransmits all packets 6 and above.