Course on

Computer networks

Data link layer

Henk Corporaal

Leiden University
2000

Data link layer

Overview
- Design issues
- Error detection and correction
- Elementary protocols
- Sliding window protocols
- Protocol specification and verification
- Example protocols

Data link design issues

Purpose: ensure reliable, efficient communication between neighbor nodes

Four specific functions:
- Provide *services* to network layer
- Framing
- Error handling
- Flow control

Services provided to NL

Three types of service :

1. Connectionless, unacknowledged
2. Connectionless, acknowledged
3. Connection oriented

Connectionless unacknowledged service

- No connection is established
  No connection release
- No acknowledge of received frame
- No attempt to recover lost frames

Appropriate in case of:
- Real-time traffic: speech, video
  Short delay more important than 100% reliability
- Low error channels: leave error correction to higher layers

---

Connectionless acknowledged service

- Like previous one, except that sender expects acknowledge for each frame sent
- Resent frame if no acknowledge within certain time

Appropriate for unreliable channels as in wireless communication

Resending individual frames is much more efficient than resending whole messages by higher layers

Connection orientated service

- Connection established first and released afterwards
- Each frame numbered (sequence number)

  Each frame acknowledged
- Lost frames are resent, however they are presented to upper layer only once, and in the right order
- Reliable bit stream established

Used to connect WANs over leased lines, where data link layer is implemented in hardware (fig 3-1)

Framing

Physical layer accepts raw bit stream

Data link layer breaks this stream into frames

- checking number of bits sent
- knowing when an acknowledge has to be sent
- needed for error detection / correction

How to frame ?

- Time gaps
  - do not work; synchronisation problems occur
- Character count
  - No resynchronisation possible after error (see fig 3-3)
- Character flags with stuffing
- Bit flags with stuffing
- Physical layer violation
  - E.g. when 2 physical bits per logical bit are used

Character flags with stuffing

Each frame starts and ends with 2 ASCII control characters

- start: DLE STX
- end: DLE ETX

DLE = Data Link Escape (code # 16)

STX = Start of TeXt (# 2)

ETX = End of TeXt (# 3)

stuffing needed: insertion of DLE before each DLE in data stream (fig 3-4)

Disadvantage: oriented towards 8 bits ASCII stream

Bit flags with stuffing

Each frame starts and ends with special bit pattern, called flag : 0111.1110

Bit stuffing: insert ‘0’ after each occurrence of five consecutive ‘1’-s

Example: see fig 3-5

Physical layer violation

Using 2 physical bits per 1 logical bit

Physical	Logical
01	1
10	0
11	violation, used to start as marker
00	violation, used as end marker

Note: overhead of 50 %

- other approach: replace e.g. 4 bits by 5 bits

Error control

Problem: Receiver may get corrupted frames: some kind of receiver feedback needed

In acknowledged services the following can be used:

- Acknowledged correct and incorrect receival of frame

Problem:

- Acknowledges and frames may be lost $\Rightarrow$ use time-out, and retransmit the same frame

New problem:

- Distinguish duplicate frames from originals $\Rightarrow$ use sequence numbers for each frame

---

Flow control

Problem: Receiver may run out of buffer space

Rules needed: *when may sender transmit ?*

- E.g., use flow window with a size of n frames ($n \geq 1$)

Error handling

Errors usually occur in bursts

- advantage : fewer frames are corrupted
- disadvantage: burst errors harder to detect and correct

Two approaches:

- Detection
- Correction

Both require extra bits per data word or message

Error handling

Idea: add extra bits to frame

- Add r bits to data
- not all 2^{m+r} codes are legal (only 2^m codes are legal)

n = m + r bits code word

m : #bits for data

r : #bits added for error control

frame: | m | r |

Hamming distance

The hamming distance (HD) of 2 code words a and b is the number of bit positions in which a and b are different, i.e. :

HD (a,b)= # ones of (a exor b)

Note that HD (a,b) bit errors are required to convert a into b (or vice versa)

Hamming distance

- guarantee a certain minimal hamming distance HDmin between any two codes

If HDmin = d + 1 : d bit errors can be detected

HDmin = 2d + 1 : d bit errors can be corrected

Hamming distance

Example 1: parity

Add (parity) bit to every data word such that total numbers of ones is even (or uneven)

⇒ HDmin = 2

⇒ single bit errors are detectable

Hamming distance

Example 2:

4 data words with HDmin = 5

⇒ double errors can be corrected

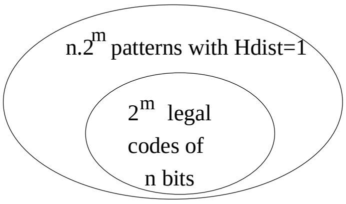
0000.0000.00
0000.0111.11
1111.1000.00
1111.1111.11

If 0000.0001.11 arrives, what is the original?

Hamming distance

How many redundant (r) bits do we need?

Assume single error correction (n = m+r bits)



Hamming distance

Total number of patterns $2^n \geq (n+1)2^m$

n = m+r ⇒

$$m + r + 1 \leq 2^r$$

m	r
8	4
16	5
1000	10

Hamming distance

Lower limit on r (for single error correcting code) can be achieved (see fig 3-6)

Add check bits at positions 1, 2, 4, 8, ...

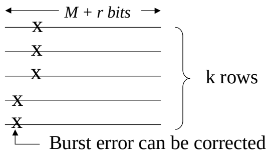
Bit 11d = 1011b is checked by bits at positions 1, 2, and 8; bit 12 by bits 4 and 8, etc.

If after receival precisely bits 1, 2, and 8 turn out to be wrong, we know that bit 11 has to be inverted.

Burst errors

How to correct burst errors of length ≤ k ?

Arrange k words (to be send) as sequence of k rows (including check bits) and transmit this column wise



Burst error can be corrected

Error-detecting codes

Error detection with retransmission often much more efficient than error correction

Example:

- error rate 10^{-6}
- block size 1000 bits
- single bit error correcting requires 10 bits / block
- single bit error detection requires 1 bit / block + retransmission of 1 block (= 1001 bits) per 1000 blocks

---

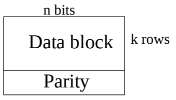
Error-detecting codes

Problem with parity:

probability of detecting burst error is 0.5

This is unacceptably low!

Solution: sent block as a matrix of n x k bits of data



n bits

Data block

k rows

Parity

Transmit one row at a time

Probability of accepting bad block = 2^{-n}

(any of the n columns should have the correct parity by accident)

Cyclic Redundancy Check: CRC

An even better method is to use CRCs

View bit sequence as polynomial, e.g. (degree 5):

110001 $\Leftrightarrow$ $x^5 + x^4 + x^0$

Idea: add r bits such that resulting frame is dividable by a certain polynomial; if not an error occurred

Polynomial Arithmetic (modulo 2):

- Add and Subtract are identical: use bitwise Exor (no carry/borrow needed)
- Div : just like binary division, except of using the above subtract

---

Cyclic Redundancy Check

How to calculate the checksum (or CRC) ?

Assume:

m bits data frame $\Leftrightarrow$ polynomial M (x)

degree r generator polynomial G (x) ; $r < m$

Steps (see fig 3.7)

1. Frame ++ 0^r $\Leftrightarrow$ x^r M (x)
2. Calculate R (x) = x^r M (x) rem G (x)
   Note that R (x) has at most r bits
3. Transmit T (x) = x^r M (x) - R (x)

Cyclic Redundancy Check

Property: T (x) rem G (x) = 0

Receiver checks this equation

Which errors are detected?

Assume T (x) + E (x) has been received

E (x) representing the error

All errors for which E (x) rem G (x) unequal zero are detected!

Conclusion: choose G(x) such that many errors are detected

E(x) should not be dividable by G(x)

Cyclic Redundancy Check

Which errors are detected?

1. Single bit errors: $E(x) = x^i$ (error in bit i)
   - are detected if G (x) contains two or more terms; then E(x) rem G(x) $\neq 0$
2. Double bit errors: $E(x) = x^i + x^j = x^j (x^{i-j} + 1)$
   - If G (x) is not divisible by x a sufficient conclusion to detect double errors is that G (x) does not divide $x^k + 1$ for any $k \leq$ frame length (i.e. $x^k + 1$ rem G(x) $\neq 0$)
   - E.g. G (x) = $x^{15} + x^{14} + 1$ does not divide $x^k + 1$ for any k < 32,768

Cyclic Redundancy Check

3. All odd-number bit errors are detected if G (x) contains factor x + 1
   - reason: E(x) rem (x+1) ≠ 0 for all odd E (i.e. all E with odd number of terms)

4. If degree G (x) = r, all burst errors of size k≤ r are detected
   - $E(x) = x^i(x^{k-1}+...+1)$
   - if G(x) contains $x^0 = 1$ it will not have x^i as factor, therefore E(x) rem G(x) ≠ 0 if k-1<r

Cyclic Redundancy Check

5. Burst error of length r+1 is accepted with probability $p=(½)^{r-1}$
   - this means that E(x) and G(x) have to be equal
   - they contain r+1 bits of which the first and last are certainly equal (namely 1); the other r-1 bits have a 50% change of being equal.

6. Longer ones are accepted with probability $(½)^r$

CRCs can easily be constructed and checked in hardware using a *linear feedback shift register* (LFSR)

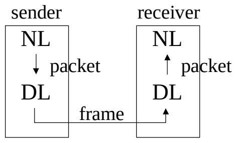
Cyclic Redundancy Check

Standard polynomials (see page 190) :

– CRC-12	$= x^{12}+x^{11}+ x^3+ x^2+ x^1+1$
– CRC-16	$= x^{16}+x^{15}+x^2+1$
– CRC-CCITT	$= x^{16}+x^{12}+x^5+1$

Example: CRC-16 and CRC-CCITT catch:
- all single and double errors
- all odd errors
- all burst errors with length ≤ 16
- 99.997 % of all 17-bit burst errors
- 99.998 % of all longer burst errors

Elementary DL Protocols



Frame = packet + header

Assumptions:
- Simplex traffic: a long datastream from sender to receiver
- Checksum errors are signalled by the physical layer
- NL, DL, PL are independent processes
- Connection-oriented, reliable service needed at NL

Elementary DL Protocols

General definitions “protocols.h” (fig 3-8)

Three protocols:

1. *Unrestricted Simplex* (Utopia; fig 3-9)
   no error and flow control; infinite buffer space at receiver side
2. *Stop-and-Wait Simplex* (fig 3-10)
   Including flow-control to prevent flooding of receiver
3. *Noisy Channel Simplex* (fig 3-11)
   Positive acknowledgement with retransmission (PAR)
   Sequence number needed (0 or 1) to distinguish retransmitted frames

Advanced DL Protocols

We add:

1. *Full duplex communication*
   Sender and receiver in one program (integrated)
2. *Piggybacking*
   Acknowledgement can be put in header of next packet to be sent
   How long should we wait for next packet?
3. *Use sliding window* (fig 3-12)
   Sender (receiver) keeps a set of sequence numbers (= window) of frames it is allowed to sent (receive)

Advanced DL Protocols

1. *One bit sliding window*
   Similar to previous protocol
2. *Go back n*
   Receiver requires only single buffer
3. *Selective repeat*
   Buffers on both sides required

Protocols 2 and 3 use pipelining

Why Pipelining ?

Consider: Channel b bits/sec
Frame of L bits
Round-trip delay R sec

Channel utilization = L / (L + bR)

This can be very low

Example: 50 kbps satellite channel, 500 msec round-trip delay; 1000 bit frames
Utilization = 4 %

One bit sliding window (fig 3-13)

- Bidirectional
- Network layer always ready to accept or send packets
- 3 types of event
  - frame_arrival (undamaged) from physical layer
    - check sequence number; if okee send frame contents to network layer, otherwise ignore frame
    - check ack number; if okee get new packet from network layer, otherwise send same frame again
  - cksum_err:
    - ignore incoming frame and send same frame again
  - timeout:
    - no acknowledge received in time
    - send same frame again

One bit sliding window (cont’d)

Normal operation

- A sends (0,1,data A0)
  - frame #0, ack #1
- B receives this and sends (0,0, data B0)
  - B acknowledges frame 0 to A
- A receives this and sends (1,0, data A1)
- B receives this and sends (1,1, data B1)
- A receives this and sends (0,1, data A2), etc.

- Small problem if both sides start to send at the same time (fig 3-14b)
  - occurs if transfers cross each other
  - frames are send multiple times

Features of ‘go back n’ (fig 3-16)

- Pipelined communication
- Multiple outstanding frames
  - upto MAX_SEQ frames without waiting for an ack
  - buffering needed at sender side (see fig. 3-15a)
  - receiver does not need buffering for go back n!
- Network layer does not always have a packet
  - generates ‘network_layer_ready event when there is a packet

---

Features of ‘go back n’ (cont’d)

- 4 types of event
  - network_layer_ready
    - buffer it
    - send a frame with proper seq and ack numbers
    - as side effect of this send operation a timer is started for sent seq#
  - frame_arrival
    - get frame from physical layer
    - if right seq# send to network layer
    - check ack# is in right range; if so, stop its timer, reduce window, and increment (ack_expected)
  - cksum_err
    - just ignore bad frames (as if they are never send)
  - timeout
    - retransmit all outstanding frames (see fig 3-15a)
- If #buffers ≥ MAX_SEQ disable_network

Features of ‘go back n’ (cont’d)

- Note: there are MAX_SEQ + 1 sequence numbers, however only MAX_SEQ frames may be outstanding

Example of what can go wrong

  - take MAX_SEQ = 3
  - sender sends frames 0-3 with seq# 0-3
  - an (piggybacked) ack of 3 comes (eventually) back
  - sender sends next frames 4-7, with seq# 0-3
  - again ack#3 comes in
- Question did receiver get frames 4-7?
- Answer: *we don’t know* (if frame 4 got lost, receiver keeps acknowledging frame 3)

Multiple timers needed

- Timer needed for each outstanding frame (or seq#)
- Problem: Expensive
- Solution: simulate multiple timers in software using only one real timer
- see fig 3-17

Selective repeat (fig. 3-18)

- Previous protocol works well if errors are rare, otherwise a lot of bandwidth is wasted

- The selective repeat algorithm:
  - accepts frame out of order
  - however pass them to network layer in order
  - receiver buffers a window of acceptable sequence numbers
  - this window has a fixed size, unlike the senders window (there is a dedicated buffer for each seq#)
  - each frame sent has associated timer; when it goes of only that frame is resent, not all outstanding frames, as in the previous protocol

Features of selective repeat

In the selective repeat algorithm the following features are included:

- Control network traffic
- Range of seq. number > 2 x Window-size
  - (see fig 3-19 for an example of what can go wrong)
- #buffers = Window-size
- Extra ack-timer: avoid piggyback when there is no traffic
  - this timer should implement a much shorter timeout compared to the timer used for timing out data frames
- Send NAK (neg. ack.) when receiving damaged frame
  - request a retransmission

Features of selective repeat (fig 3-18)

- 5 types of events
  - network_layer ready
    - expand window and send frame
  - frame_arrival
    - if data frame, check seq#; if wrong sent nak; if right accept frame, pass frame(s) to network layer and start ack-timer
    - if nak frame (and in proper range) resent frame
    - stop timers of all acknowledged frames
  - cksum_err
    - send nak frame
  - timeout
    - resent oldest frame
  - ack_timeout
    - send separate ack frame (no piggybacking possible)
- Again if buffer may overflow disables network layer

Protocol Specification and Verification

Real protocols become extremely complex

We have to verify them through

- simulation
- formal verfication methods

Two verification models are outlined:

1. Finite State Machine (FSM)
2. Petri Net

Protocol Specification and Verification

Finite State Machine of a protocol

FSM = (S, M, I, T)

S : set of states (of processes and channels)
M : set of frames that can be exchanged over the channel
I : set of initial states (one for each process)
T : set of state transitions

Protocol Specification and Verification

FSM checks:

- reachability: can a certain state be reached
- is action stated for each state
- is there progress possible for each state (no deadlock)

Example in fig 3-20 :

- 16 states: 2x2x4
  - 2 per sender & receiver (frame sent is 0 or 1)
  - 4 for the channel: 0, 1, A (= ack), - (= empty)
  - only 9 states are shown (others unreachable)
- Normal operation: transitions 1, 2, 3 and 4

Protocol Specification and Verification

Petri Net

Four elements: places, transitions, arcs and tokens (see fig. 3-22)
Places correspond to states

Firing rule:

- all inputs contain at least one token
- remove one token from each input
- add result token to each output
- non-deterministic firing

Protocol Specification and Verification

Petri Net Example: fig. 3-23

- no composite states: separate states for sender, receiver and channel
- transition 1: BD → AC normal emit of frame 0
- transition 2: timeout of frame 0
- transition 3, 4 similarly for frame 1
- transition 5-7: loss of frame 0, ack, and frame 1 resp.
- transition 8, 9: wrong seq# arrives
- transition 10,11: correct seq# received

- Similar checks possible (like deadlock, reachability) as for FSMs

Example Data Link Protocols

1. HDLC : high-level data link control
   Common in X.25 networks
   Derived from IBM’s SNA (system network arch.)
   Bit-oriented, bit-stuffing (see fig. 3-24)

2. IP point-to-point protocols (no routing needed):
   SLIP : serial line IP
   PPP : point-to-point protocol

3. ATM data link layer

SLIP - Serial Line IP

Used for dial-up line using modem (fig 3-26)

- flag byte 0xC0 at each end of frame
- character stuffing
  - in data 0xC0 is replaced by 0xDB, 0xDC; 0xDB is also stuffed

Problems:

- no error control
- supports IP only
- IP addresses of both sides to be known in advance
  - no dynamic address change
  - every user can not have an unique address
- no authentication

PPP - Point-to-Point Protocol

Solves all SLIP shortcomings

Used almost everywhere

PPP provides:

- Framing with error detection
- LCP (link control protocol) for bringing up lines, testing, negotiating options
- NCP (network control protocol) for each supported network layer

PPP - Point-to-Point Protocol

PPP frame format (fig 3-27)

- flag byte: 0111.1110 / character stuffing
- address field (default = 1111.1111)
- control field (default = 0000.0011 indicates unnumbered frames with unreliable transmission; no seq#)
- protocol field: indicates packet type
  - 0x indicates IP, IPX or XNS
  - 1x indicates LCP, NCP or others
- payload: 1500 bytes default length
- checksum of 2 (default) or 4 bytes

PPP - Point-to-Point Protocol

- Typical scenario of home user calling up (see also fig. 3-28)
  - Modems establish physical connection
  - PC sends router series of LCP packets
    - Select appropriate PPP parameters
  - Send a series of NCP packets to configure network layer
    - needs IP address (dynamic assignment)
  - Send IP packets (PC is on internet now)
- Finishing
  - Use NCP packets to tear down the network layer connection and free IP address
  - Use LCP to shut down the data link layer connection
  - Hang up modem

Data link layer in ATM

- There are no physical layer characteristics specific to ATM
  - ATM cells are carried by SONET, or other transmission systems
- TC (transmission convergence) sublayer corresponds to DL layer
- Transmission
  - add HEC (header checksum)
  - convert into bitstream
  - match speed to underlying physical transmission system by inserting empty or maintenance (OAM) cells
- Reception: the reverse

Data link layer in ATM

- Transmission: each cell of 53 byte includes 5 byte header
  - 4 byte identification, etc
  - 1 byte header checksum (CRC)
  - no checksum for payload
  - on fiber p(undetected error) = 10^{-20}
  - on OC-3 (= 155 Mbps): one error in 90k years !! (per channel of course)

Data link layer in ATM

- Reception of a cell
  - locate cell boundaries
  - verify header
  - discard cells with invalid headers
  - process maintenance cells
  - pass data to ATM layer

Data link layer in ATM

- How to detect cell boundaries (fig 3-30) (there are no marks !!)
  - trick: constantly check HEC field after each incoming bit
  - use 40-bits shift register
  - check δ headers in a row
  - change of accidental correct synchronisation = $2^{-8\delta}$
  - ‘deSynch’ after α headers
- A malicious user might spoof TC sublayer by putting a pattern in successive payloads that imitates HEC synchronisation: scramble payload
- Note: TC sublayer uses header of ATM field for synchronisation: no proper separation of layers
  - E.g. changing header format for ATM affects TC layer

Exercises

Make the following exercises of chapter 3:

- 1, 2, 3, 6, 7, 9, 10, 11, 17, 22