CSMC 417
Computer Networks
Prof. Ashok K Agrawala

© 2002 Ashok Agrawala
Set 6

Data Link Layer

- Transmit bits to the destination so that they can be handed over to the network layer
- Transmission in terms of FRAMES
- Services provided to the Network Layer
  - Unacknowledged connectionless service
    - Low error rate
    - Real Time traffic
    - Most LANs use this
  - Acknowledged connectionless service
    - Suitable for wireless channels
  - Acknowledged connection-oriented service

Phases of Connection Oriented Service

- Connection Establishment
- Data Transfer
- Connection Release

Framing

- Physical Layer moves raw bit stream
  - Bits with errors - numbers may be more or less
- Data Link Layer organizes bits in frames and adds a checksum for each frame
- How to break bit stream into frames?
  - Have time gaps
  - Network unreliability makes this approach infeasible
- Four Techniques used
  - Character Count
  - Starting and ending characters - with character stuffing
  - Starting and ending flags - with bit stuffing
  - Physical layer coding violations

Character Count

- Use a field in the header to indicate the number of bytes
- What is the number field has an error?
Character Stuffing

- Special Characters at the beginning and the end
- Stuff with additional character when special characters appear in the text
  - closely tied to ASCII character set
  - Have to embed ASCII character DLE

Bit Stuffing

- Special Code - Flag - 0111110
- Add a 0 after 5 consecutive 1's

Physical Layer Coding

- Only applicable when physical medium contains some redundancy
  - Some LANS encode 1 bit of data using two physical bits
    - 1 is high-low
    - 0 is low high
    - Easy to locate bit boundaries - transition in the middle
    - high-high and low-low are not allowed
  - IEEE 802 uses this scheme

Error Detection and Correction

- Error Characteristics
  - Depends on the media
  - Usually bursty
- Approach
  - Error Correcting Codes
    - Use redundancy and specially designed codes
    - Example - Hamming code for single bit error
    - Send 11 bits for 7 bits of information
  - Error Detecting Codes
    - Much easier to construct than error correcting codes
    - Example - parity bit, CRC

Polynomial Code

- Treat bit strings as representation of polynomials with coefficients 0 or 1
  - k bit frame - polynomial with terms x^0 to x^(k-1)
  - Generator Polynomial G(x) - High and low order bits are 1
    - Let r be the degree of G(x). Append r zeros to the low-order end. So Frame now becomes x^r M(x)
    - Divide G(x) into x^r M(x) using Mod 2 arithmetic
    - Subtract the remainder from x^r M(x). Send the result as T(x)
    - On receiving divide T(x) by G(x). If result is zero there are no errors
CRC Codes

- **CRC-12**
  - $x^{12} + x^{11} + x^3 + x^2 + x + 1$
  - Good for 6 bit characters

- **CRC-16**
  - $x^{16} + x^{15} + x^2 + 1$
  - Good for 8 bit characters
  - Catch all single and double bit errors, errors with odd number of bits

- **CRC-CCITT**
  - $x^{16} + x^5 + 1$
  - Catch all single and double bit errors, errors with odd number of bits

Structure of Data Link Layer

**Network Layer**
- Packet
  - Information
  - Header

**Physical Layer**
- Frame

DLL Functions

- Interface with Network Layer
  - To_Network_Layer
  - From_Network_Layer
  - Other Functions

- Interface with Physical Layer
  - To_Physical_Layer
  - From_Physical_Layer
  - Other Functions

- Local Operations
  - Buffer management
  - Error Checking
  - Timers
  - Sequence Numbering

Simple Operations

Unrestricted Simplex Protocol

- One direction data transfer
- Transmitting and receiving network layers always ready
- Zero processing time
- Infinite buffer space
- Perfect channel - does not damage or loose frames
- Use no sequence numbers or acknowledgements
- Only use info field in the frame

Protocol 1

- Frame transmit and receive is one direction only, from sender to receiver.
- The communication system is assumed to be error free, and the receiver is assumed to be a buffer in a process in the network layer, two processes. The receiver processes the frames one at a time.
- From the sender to the receiver, a packet enters the network layer, toPhysicalLayer() is called. The packet is then converted to a byte array and sent to the network layer.
- When the packet enters the network layer, fromPhysicalLayer() is called. The packet is then converted to a byte array and sent to the sender.
- When the packet enters the sender, fromNetworkLayer() is called. The packet is then converted to a byte array and sent to the network layer.
**Simple Stop and wait protocol**

- **Relax the assumption**
  - Ability of the receiving network layer to process incoming data infinitely fast — OR
  - Having infinite buffer space in the receiving data link layer

- **How to prevent sender from flooding the receiver**
  - If receiver requires $t$ sec to execute From _Physical_ layer and To _Network_ layer, the sender must transmit at an average rate less than one frame per $t$ sec.
  - Sender may slow down — time synchronize — may be too slow

- **Receiver Feedback**
  - Acknowledgement

---

**Simplex Protocol for Noisy Channel**

- **Relax the assumption**
  - Channel is perfect

- **Frames received may be**
  - damaged — Checksum
  - lost — Sequence number

- **Automatic Repeat reQuest (ARQ)**
  - Cumulative Ack
  - Selective Ack

---

**Sliding Window Protocols**

- When transit time is large Stop and Wait gives very low line utilization

- Should be able to have more frames in transit

---

**Protocol 2**

/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time, the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled.

```c
typedef enum fframe-arrivall event-type;
#include "protocol.h"
void sender2(void)
{
    frame s; /* buffer for an outbound frame
    packet buffer; /* buffer for an outbound packet
    event-type event; /* frame-arrival is the only possibility
    while (true) {
        from-network-layer(&buffer); /* go get something to send
        s.info = buffer; /* copy it into s for transmission
        to-physical-layer(&s); /* bye bye little frame */
        wait for-event(&event); /* do not proceed until given the go ahead
    }
}
void receiver2(void)
{
    frame r, s; /* buffers for frames
    event type event; /* frame-arrival is the only possibility
    while (true) {
        wait-for-event(&event); /* only possibility is frame-arrival
        from-physical-layer(&r); /* go get the inbound frame */
        to-network-layer(&r.info); /* pass the data to the network layer
        to-physical-layer(&s); /* send a dummy frame to awaken sender */
    }
}
```

---

**Protocol 3**

/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel.

```c
#define MAX-SEQ 1 /* must be 1 for protocol 3
typedef enum Iframe-arrival, cksum-err, timeout) event-type;
#include "protocol.h"
void sender3(void)
I
    seq-nr next frame-to-send; /* seq number of next outgoing frame
    frame s; /* scratch variable */
    packet buffer; /* buffer for an outbound packet
    event-type event;
    next-frame-to-send = 0; /* initialize outbound sequence numbers
    from-network-layer(&buffer); /* fetch first packet
    while (true) {
        s.info = buffer; /* construct a frame for transmission
        s.seq = next-frame-to-send; /* insert sequence number in frame
        to-physical-layer(&s); /* send it on its way */
        start-timer(s.seq); /* if (answer takes too long, time out
        wait-for-event(&event); /* frame-arrival, cksum-err, timeout
        if (event == frame-arrival)
            from-network-layer(&buffer); /* get the next one to send
            inc(next-frame-to-send); /* invert next-frame-to-send
    }
}
void receiver3(void)
{
    seq-nr frame-expected;
    frame r, s;
    event-type event;
    frame-expected = 0;
    while (true) {
        wait-for-event(&event); /* possibilities: frame-arrival, cksum-err
        if (event == frame-arrival) /* a valid frame has arrived. */
            from-physical-layer(&r); /* go get the newly arrived frame
            if (r.seq == frame-expected) /* this is what we have been waiting for. */
                to-network-layer(&r.info); /* pass the data to the network layer */
            inc(frame-expected); /* next time expect the other sequence nr
            to-physical-layer(&s); /* none of the fields are used */
    }
```
**Sliding Window**

- Sender is authorized to send any packet with sequence number between S1 and S2.
- Receiver receives any packet in the range R1 – R2.
- If a packet outside this range is received – discard
- Acknowledge any correct packet received in this range.
  - What information to send as acknowledgement

**Sliding Windows**

- Acknowledgement – Seq No. N
  - Packet N received correctly
  - All packets numbered N or lower received correctly
  - Receive sequence
  
  1 2 3 4 5 6 7 8 9

- Packet received in error
  - Discard packet – treat as if never received
  - Header received correctly
  - Send NAK with seq no of packet

- What information to send as acknowledgement

**Sliding Window Protocol**

- Sender
  - Advance S1 and S2
  - Reflecting packets received correctly
  - Time out
- Receiver

**Two Scenarios**

- One bit window

  - A sends (0, 1, A0)
  - A gets (0, 0, B0)
  - A sends (1, 0, A1)
  - B sends (0, 1, B0)*
  - B gets (0, 1, A0)*
  - B sends (1, 1, B1)
  - B gets (1, 0, A1)*
  - B sends (0, 0, B2)
  - B gets (1, 0, A3)*
  - B sends (1, 1, B3)
  - A gets (1, 1, B1)*
  - A sends (0, 1, A2)
  - A gets (0, 0, B2)*
  - A sends (1, 0, A3)

- Time(a)

**Error Recovery**

- Go back N
- Selective Repeat
Go Back N

Software Timers

Window Operations

Data Link Layer Protocols

HDLC

HDLC Frame Types

- Bit Oriented
  - SDLC
  - ADDCP
  - HDLC
  - LAP and LAPB
- Byte Oriented
  - SLIP
  - PPP
HDLC Frame Types

- Type 0 - Acknowledgement - Receiver Ready
- Type 1 - NAC - Reject
- Type 2 - Receiver Not Ready
  - Acks up to but not including Next
  - Sender must stop sending
- Type 3 - Selective Reject
  - Retransmit only the specified frame

SLIP and PPP

SLIP

- Send raw IP packet
- Provide Framing
  - Byte Oriented
  - Use Byte Stuffing
- Limitations
  - No Error Detection
  - Only works with IP
  - Must know IP address before starting
  - No Authentication
  - Not an approved standard

PPP

- Works with many protocols
- Provides three things
  - Framing Method and error detection
  - Link Control Protocol (LCP)
    - Brings line up/down
    - testing
    - Negotiate options
  - Network Control Protocol for each higher layer protocol
    - To negotiate options

PPP Frame Format

<table>
<thead>
<tr>
<th>Bytes</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1 or 2</th>
<th>Variable</th>
<th>2 or 4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flag</td>
<td>01111110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>11111111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>00000011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>Payload</td>
<td>Checksum</td>
<td>Flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11111110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Line Management

- Carrier detected
- Both sides agree on options
- Authentication successful

- Failed
- Open
- NCP configuration
- Done