Introduction to LAN/WAN

Data Link Layer (Part II)
Topics

- Introduction
- Errors
- Protocols
  - simple
  - sliding window
- Modeling
- Examples
Protocols Purpose

- Agreed means of communication between sender and receiver
- Handle reliability
- Handle flow control
- We’ll move through basic to complex
Data Link Protocols

- Machine $A$ wants stream of data to $B$
  - assume reliable, 1-way, connection-oriented
- Physical, Data Link, Network are all processes
- Assume:
  - `to_physical_layer()` to send frame
  - `from_physical_layer()` to receive frame
  - both do checksum
  - `from_physical_layer()` reports *success* or *failure*
Frame

| kind | seq | ack | info |

- first 3 are control (*frame header*)
- *info* is data
- *kind*: tells if data, some are just control
- *seq*: sequence number
- *ack*: acknowledgements
- Network has *packet*, put in frame’s *info*
- Header is not passed up to network layer
Tanenbaum’s Protocol Definitions

```c
#define MAX_PKT 1024

typedef enum {false, true} boolean; /* boolean type */
typedef unsigned int seq_nr; /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet; /* packet definition */
typedef enum {data, ack, nak} frame_kind; /* frame_kind definition */

typedef struct {
  frame_kind kind;
  seq_nr seq;
  seq_nr ack;
  packet info;
} frame; /* frames are transported in this layer */
/* what kind of a frame is it? */
/* sequence number */
/* acknowledgement number */
/* the network layer packet */
```

Figure 3-9. Some definitions needed in the protocols to follow. These are located in the file protocol.h.

Continued →
Unrestricted Simplex Protocol

- Simple, simple, simple
- One-way data transmission (simplex)
- Network layers always ready
  - infinitely fast
- Communication channel error free
- “Utopia”
/* Protocol 1 (utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free, and the receiver is assumed to be able to process all the input infinitely quickly. Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
    frame s;                  /* buffer for an outbound frame */
    packet buffer;            /* buffer for an outbound packet */

    while (true) {
        from_network_layer(&buffer); /* go get something to send */
        s.info = buffer;              /* copy it into s for transmission */
        to_physical_layer(&s);       /* send it on its way */
    }
    /* Tomorrow, and tomorrow, and tomorrow, Creeps in this petty pace from day to day To the last syllable of recorded time - Macbeth, V, v */
}

void receiver1(void)
{
    frame r;
    event_type event;            /* filled in by wait, but not used here */

    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 */
    }
}
Simplex Stop-and-Wait Protocol

- One-way data transmission (simplex)
- Communication channel error free
- Remove assumption that network layers are always ready
  - (or that receiver has infinite buffers)
- Could add timer so won’t send too fast?
  - Why is this a bad idea?
- What else can we do?
/ 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. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;
  packet buffer;
  event_type event;
  /* buffer for an outbound frame */
  /* buffer for an outbound packet */
  /* frame_arrival is the only possibility */
  while (true) {
    from_network_layer(&buffer);
    s.info = buffer;
    to_physical_layer(&s);
    wait_for_event(&event);
    /* go get something to send */
    /* copy it into s for transmission */
    /* bye bye little frame */
    /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;
  event_type event;
  /* buffers for frames */
  /* frame_arrival is the only possibility */
  while (true) {
    wait_for_event(&event);
    from_physical_layer(&r);
    to_network_layer(&r.info);
    to_physical_layer(&s);
    /* only possibility is frame_arrival */
    /* go get the inbound frame */
    /* pass the data to the network layer */
    /* send a dummy frame to awaken sender */
  }
}
In parts (a) and (b) transmitting station A acts the same way, but part (b) receiving station B accepts frame 1 twice.
Simplex Protocol for Noisy Channel

- One-way data transmission (simplex)
- Remove assumption that communication channel error free
  - frames lost or damaged
- Damaged frames not acknowledged
  - look as if lost
- Can we just add a timer in the sender?
  - Why not? (Hint: think of acks)
- Positive Ack with Retransmissions (PAR)
ACKs must ALSO be numbered!!

PAR problem:
Ambiguities when ACKs are not numbered

Transmitting station A misinterprets duplicate ACKs
Simplex Protocol for a Noisy Channel

Figure 3-12. A Positive Acknowledgement with Retransmission protocol.
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;
    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
        }
        to_physical_layer(&s);
    }
}
Sliding Window Protocols

- Remove assumption that one-way data transmission
  - duplex
- Error prone channel
- Finite speed (and buffer) network layer
Two-Way Communication

- Seems efficient since acks already
- Have two kinds of frames (kind field)
  - Data
  - Ack (seq num of last correct frame)
- May want *data* with ack
  - delay a bit before sending data
  - *piggybacking* - add acks to data frames going other way
- How long to wait before just ack?
Sliding Window Protocols

- More than just 1 outstanding packet
  - “Window” of frames that are outstanding
- Sequence number is $n$ bits, $2^{n-1}$
- Sender has sending window
  - frames it can send (can change size): sent but no ACK
  - new packets from the Host cause the upper edge inside window to be incremented.
- Receiver has receiving window
  - frames it can receive (always same size)
  - ACKed frames from the receiver cause the lower edge inside window to be incremented
- Window sizes can differ
- Note, still passed to network layer in order!
Sliding Window, Size 1
1-Bit Sliding Window Protocol

```c
void protocol4 (void)
{
    seq_nr next_frame_to_send;
    seq_nr frame_expected;
    frame r, s;
    packet buffer;
    event_type event;
    next_frame_to_send = 0;
    frame_expected = 0;
    from_network_layer(&buffer);
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 - frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
    /* 0 or 1 only */
    /* 0 or 1 only */
    /* scratch variables */
    /* current packet being sent */
    /* next frame on the outbound stream */
    /* number of frame arriving frame expected */
    /* fetch a packet from the network layer */
    /* prepare to send the initial frame */
    /* insert sequence number into frame */
    /* piggybacked ack */
    /* transmit the frame */
    /* start the timer running */
```
while (true) {
    wait_for_event(&event);    /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) { /* a frame has arrived undamaged. */
        from_physical_layer(&r);    /* go get it */
        if (r.seq == frame_expected) {
            /* Handle inbound frame stream. */
            to_network_layer(&r.info);  /* pass packet to network layer */
            inc(frame_expected);        /* invert sequence number expected next */
        }
        if (r.ack == next_frame_to_send) { /* handle outbound frame stream. */
            from_network_layer(&buffer); /* fetch new pkt from network layer */
            inc(next_frame_to_send);     /* invert sender's sequence number */
        }
    }
    s.info = buffer;            /* construct outbound frame */
    s.seq = next_frame_to_send; /* insert sequence number into it */
    s.ack = 1 - frame_expected; /* seq number of last received frame */
    to_physical_layer(&s);      /* transmit a frame */
    start_timer(s.seq);         /* start the timer running */
}
Does it Work?

- Consider A with a too-short time-out
- A sends: seq=0, ack = 1 over and over
- B gets 0, sets frame_expected to 1
  - will reject all 0 frames
- B sends A frame with seq=0, ack=0
  - eventually one makes it to A
- A gets ack, sets next_frame_to_send to 1
- Above scenario similar for lost/damaged frames or acknowledgements
- But … what about startup?
Normal Startup

A sends \((0, 1, A0)\)

B gets \((0, 1, A0)\)
B sends \((0, 0, B0)\)

A gets \((0, 0, B0)\)
A sends \((1, 0, A1)\)

B gets \((1, 0, A1)\)
B sends \((1, 1, B1)\)

A gets \((1, 1, B1)\)
A sends \((0, 1, A2)\)

B gets \((0, 1, A2)\)
B sends \((0, 0, B2)\)

A gets \((0, 0, B2)\)
A sends \((1, 0, A3)\)

B gets \((1, 0, A3)\)
B sends \((1, 1, B3)\)
Abnormal Startup

A sends (0, 1, A0)
B sends (0, 1, B0)
B gets (0, 1, A0)*
B sends (0, 0, B0)

A gets (0, 1, B0)*
A sends (0, 0, A0)

A gets (0, 0, B0)
B gets (0, 0, A0)
B sends (1, 0, B1)

A sends (1, 0, A1)
B gets (1, 0, A1)*
B sends (1, 1, B1)

A gets (1, 0, B1)*
A sends (1, 1, A1)
B gets (1, 1, A1)
B sends (0, 1, B2)
Transmission Factors

- Assume a satellite channel, 500 msec rt delay
  - super small ack’s
- 50 kbps, sending 1000-bit frames
- $t = 0$, sending starts
- $t = 20$ msec frame sent
- $t = 270$ frame arrives
- $t = 520$ ack back at sender
- $20 / 520$ about 4% utilization!
- All of: long delay, high bwidth, small frames
- Solution?
Allow Larger Window

- Satellite channel, 500 msec rt delay
- 50 kbps, sending 1000-bit frames
- Each frame takes 20 msec
  - 25 frames outstanding before first ack arrives
- Make window size 25
- Called pipelining
- (See p.216, protocol 5)
  - added enable/disable network layer
  - MAX_SEQ - 1 outstanding
  - timer per frame
- Frame in the middle is damaged?
Go Back N

- Arbitrary window size:
  - send $w$ frames (not 1) before blocking
- If error, receiver discards all addtl frames
- Sender window fills, pipeline empties
- Sender times out, retransmits
- Waste of bandwidth if many errors
Selective Repeat

- Receiver stores all frames, waits for incorrect one
- Window size greater than 1

Diagram:
- Timeout interval
- Frames 0 to 13
- Acknowledgments
- Error
- Packets 2-8 passed to network layer
- Buffered by data link layer
Latest and Greatest: Non-Sequential Receive

- Tanenbaum, Protocol 6
- Ack latest packet in sequence received
- Acks not always piggybacked
  - Protocol 5 will block until return data available
  - *start_ack_timer*
  - How long ack timeout relative to date timeout?
- Negative acknowledgement (NAK)
  - damaged frame arrives
  - non-expected frame arrives
Problem?

Window size (MAX_SEQ) / 2

How many buffers are needed? MAX_SEQ?
Closing Thoughts...

- If constant round-trip propagation delay
  - set timer just slightly higher than delay
- If variable round-trip propagation delay
  - small timer has unnecessary retransmissions
  - large has many periods of idle network
  - same is true of variable \textit{processing} delay
- Constant, then “tight” timer
- Variable, then “loose” timer
  - NAKs can really help bandwidth efficiency
Topics

- Introduction
- Errors
- Protocols
- Modeling
  - complex specification and verification
- Examples
Examples

- HDLC
  - IBM SNA

- Internet
  - SLIP
  - PPP

- ATM
The Internet

- Point-to-Point on leased lines between routers
- Home user to Internet Service Provider (ISP) – PPP
Point-to-Point Protocol (PPP)

- Bit-based frame
  - resorts to character based over a modem
- Line control: up, down, options
  - Link Control Protocol (LCP)
- Network control options
  - NCP (Network Control Protocol)
  - Negotiate network layer options independent of particular network layer
  - Service for: IP, IPX, AppleTalk …