Panel 1

Last Time:

Data Link level

Framing + frame byte 01111110

stuffing / unstuffing

Error correction - redundancy to detect + recover errors

Error detection - parity bit

cyclic redundancy check CRC

coolprog.exe ⇐

CRC Checksum 57899785

MD value

1

Panel 2

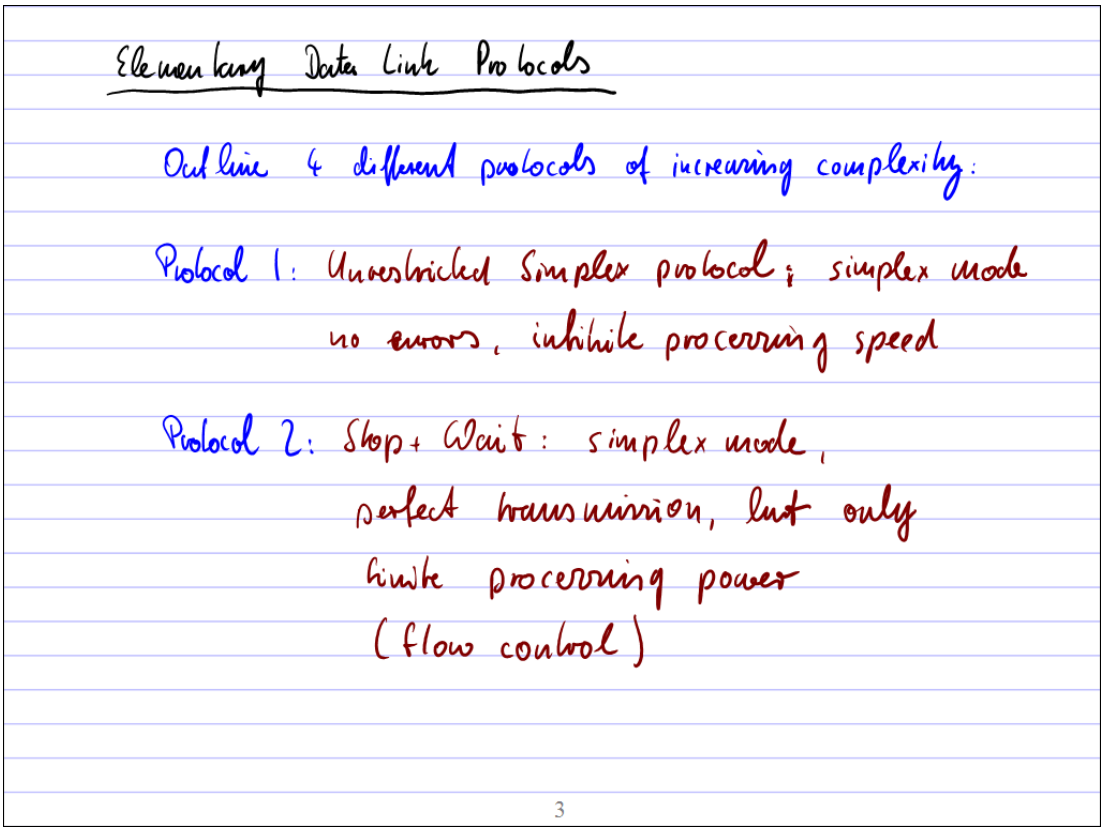
Even parity:

01111110 ⇒ 0101011100011111111111100101 1 01111110

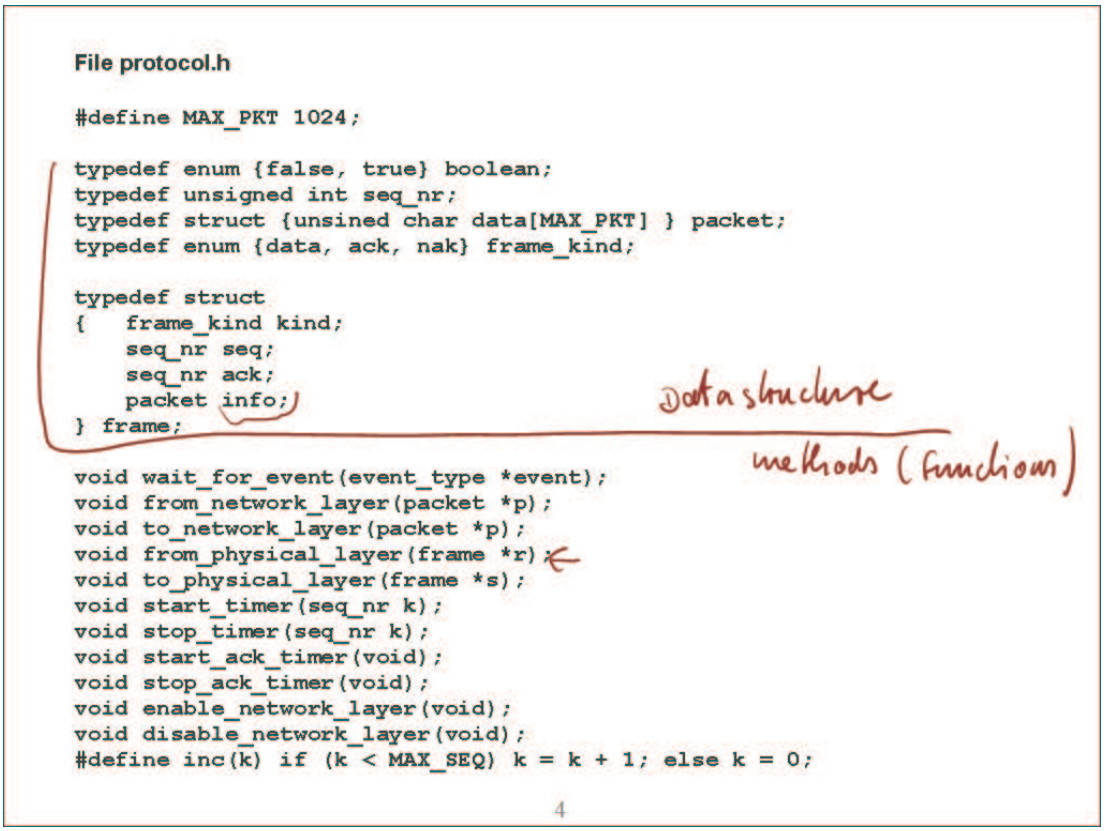
1. Parity bit
2. Stuff
3. Frame

2

Panel 3



Panel 4



Panel 5

Unrestricted Simplex Protocol: Sending

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

void sender(void)
{   frame s;
    packet buffer;
    while (true)
    {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
    }
}
```

Unrestricted Simplex Protocol: Receiving

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

void receiver(void)
{   frame r;
    event_type event;
    while (true)
    {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
    }
}
```

5

Panel 6

Stop and Wait Protocol: Sending

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

void sender(void)
{   frame s;
    packet buffer;
    event_type event;
    while (true)
    {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}
```

Stop and Wait Protocol: Receiving

```
void receiver(void)
{   frame r, s;
    event_type event;
    while (true)
    {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

6

Panel 7

Next refinement: still simplex, finite processing power, unreliable channel

Suggestion: use timer

if receiver gets undamaged packet, it sends back acknowledgement, else not

sender starts timer; if timer times out, resend packet

Question: what if ackn. gets lost?

=> leads to duplicate frames

7

Panel 8

More refined protocol needs:

timer : to recognize transmission errors

sequence #: to recognize duplicates

How many bits for sequence # (small as possible)

if frame is damaged, receiver will not ackn.

=> send again that frame

=> receiver must recog. frame m from m+1

get by with 1 bit: seq # is 0 or 1

8

Panel 9

Positive Acknowledgement with retransmission: Sending

```
#define MAX_SEQ 1;
typedef enum {frame_arrival, chsum_err, timeout} event_type;
#include "protocol.h"

void sender(void)
{
     seq_nr next_frame_to_send;
     frame s;
     packet buffer;
     event_type event;

     next_frame_to_send = 0;
     from_network_layer(&buffer);
     while (true)
     {
          s.info = buffer;
          s.seq = next_frame_to_send;
          to_physical_layer(&s);
          start_timer(s.seq);
          wait_for_event(&event);
          if (event == frame_arrival)
          {
               from_physical_layer(&s);
               if (s.ack == next_frame_to_send)
               {
                    from_network_layer(&buffer);
                    inc(next_frame_to_send);
               }
          }
     }
}
```

9

Panel 10

Positive Acknowledgement with retransmission: Receiving

```
void receiver(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);
        }
    }
}
```

10

Panel 11

Next protocol: duplex
limited processing (flow control)
unreliable channel

could do 2 simplex channels
- waste of bandwidth

Idea: instead of returning only ackn. to sender, will return frame with info + add ackn. piece to frame
"piggy backing"

Symmetric protocol, same code on both parties.

11

Panel 12

Protocol 4 (Sliding Window) is bidirectional, so only one function is needed:

```
#define MAX_SEQ 1
typedef enum {frame_arrival, chksum_err, timeout} event_type;
#include "protocol.h"

void slidingWindow(void)
{
   // Defining variables needed
   seq_nr next_frame_to_send = 0;
   seq_nr frame_expected = 0;
   frame r, s;
   packet buffer;
   event_type event;

   // Getting info from network layer and defining first sending frame
   from_network_layer(&buffer);
   s.info = buffer;
   s.seq = next_frame_to_send;
   s.ack = 1 - frame_expected;

   // sending out the frame and starting the timer
   to_physical_layer(s);
   start_time(s.seq);

   while (true)
   {  // waiting for event. Chksum_err or timeout events are ignored
      wait_for_event(&event);
      if (event == frame_arrival)
      {  // an undamaged frame arrived, so get it into r
         from_physical_layer(&r);

         if (r.seq == frame_expected)
         {  // handle inbound frame stream, i.e. give info to network layer
            to_network_layer(&r.info);
            inc(frame_expected);
         }
         if (r.ack == next_frame_to_send)
         {  // handle outbound frame stream, i.e. fetch new packet
            from_network_layer(&buffer);
            inc(next_frame_to_send);
         }
      }
      // construct outgoing frame
      s.info = buffer;
      s.seq = next_frame_to_send;
      s.ack = 1 - frame_expected;
      // send outgoing frame and start timer
      to_physical_layer(&s);
      start_timer(s.seq);
   }
}
```

try to figure out

12

Panel 13