Panel 1

Last Time:

Switching / circuit & packet
Switches / cross bar
Switches / time division

Data Link Layer - reliable transmission

- Frame
- Error check
- Flow control

Formatting: flag style 0111110

Panel 2

Ex:

Data from network layer: 010011011011110

\( \text{data link layer: } \overline{010011011011110} \)

( to physical layer )

\( \text{to physical layer B} \)

\( \text{to physical layer C} \)

\( \text{data link layer} \overline{010011011011110} \)

( to network layer )

\( \text{to network layer} \overline{010011011011110} \)
Panel 3

Handling Errors

error: inverting a bit

(extra bit)

(error correction: build enough redundancy into a frame to detect & fix error.

Big overhead, used for unreliable simplex channel.

(error correction: detect an error but you don’t fix it; instead ask for retransmission.)

Panel 4

Simple Error Detection: Parity Bit

Even parity: add 0 or 1 to data of frame for even # of 1’s

Odd parity: add 0 or 1 to data of frame for odd # of 1’s

Recall that an even parity bit is computed so that the total number of 1’s in the data is even.

a. What is the parity of 101011110100110011001010011001001110100110010011111100?

b. Suppose a bit pattern has been constructed by (1) computing an even parity bit and appending it to the end of the data bits (2) stuffing by replacing all sequences of 1111 by 111110, and (3) framing by adding the flag byte 01111110 at the beginning and end of each frame. Then you receive the pattern:

011111101010110011001100101000010001101100000110111111011111101111110110001100000110111110110101001101111110

How many frames are there, which frame is invalid, and what is the original data in each valid frame?

Frame 1: valid
Frame 2: valid
Frame 3: invalid
Frame 4: valid
Frame 5: invalid
Panel 5

**Panel 5**

Parity bit: can detect single errors but not double errors
odd errors but not even ones

More powerful error detection: Cyclic redundancy check (CRC)

Senders and receivers agree on a generator polynomial $G(x)$
Each frame of $k$ bits is a $(k-1)$-degree polynomial
$10001 \sim x^4 + x^3 + x^0$

Add checksum so that new polynomial is divisible by $G(x)$, what you do is:

Divide original poly. $P(x)$ by $G(x)$ and find remainder

Panel 6

\[
\begin{align*}
3 & \mid 1235 \quad \text{is not divisible by 3 (R2)} \\
3 & \mid 1235 \\
\hline
12 & \mid 35 \quad \Rightarrow \quad 	ext{but } 1235 - 3 \times 411 = 2 \\
\hline
\end{align*}
\]

What type of error can you catch? Divisible by $G(x)$

If error occurs then instead of $T(x)$ you get

$T(x) + E(x) = \text{check } (T(x) + E(x))/G(x)$

$= \frac{T(x) \cdot X(x) + E(x)/G(x)}{G(x)}$

less remainder

If choose $G(x)$ carefully, you can detect most errors!
Panel 7

\[
\begin{align*}
\text{CRC-12:} & \quad G(x) = x^{12} + x^8 + x^3 + x + 1 \\
\text{CRC-16:} & \quad G(x) = x^{16} + x^{15} + x^2 + 1 \\
\text{CRC-CAN:} & \quad G(x) = x^{16} + x^{12} + x^5 + 1
\end{align*}
\]

CRC-16 catches: all single errors
all double errors
all errors with odd # of flipped bits
all burst errors length 16 or less
(99.997\% of burst errors of length 17
99.998\% of burst errors longer than 18)

All calculations are done via shift registers in
hardware.

Panel 8

Elementary Data Link Protocols

Outline 4 different protocols of increasing complexity:

Protocol 1: Unrelied Simplex: perfect transmission
Physical + network are infinitely fast always
ready.

Protocol 2:
Panel 9

```
File protocol.h

#define MAX_PKT 1024;

typedef enum {false, true} boolean;
typedef unsigned int seq_nr;
typedef struct [unsined char data[MAX_PKT] ] packet;
typedef enum {data, ack, nak} frame_kind;

typedef struct
{   frame_kind kind;
    seq_nr seq;
    seq_nr ack;
    packet info;
} frame;

void wait_for_event(event_type *event);
void from_network_layer(packet *p);
void to_network_layer(packet *p);
void from_physical_layer(frame *r);
void to_physical_layer(frame *s);
void start_timer(seq_nr k);
void stop_timer(seq_nr k);
void start_ack_timer(void);
void stop_ack_timer(void);
void enable_network_layer(void);
void disable_network_layer(void);
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0;
```