Previously on EITF25

Diagram showing the process of sending digital data from a sender to a receiver over a link. The sender sends digital data through a modulator, which converts it into an analog signal. The analog signal is transmitted over the link (or digital signal) and received by a demodulator, which converts it back into digital data.
Data Link Layer

- Medium Access Control
  - Access to network
- Logical Link Control
  - Node-to-node error and flow control
Link layer protocols

• Error detection
  – All errors must be detected

• Error correction
  – Receiver must get correct data

• Flow control
  – Receiver must not be overloaded
Internet: Data Link Layer

Logical Link Control Sublayer

- Error detection and correction
  \[\text{Forouzan ed.5 ch.10.1-5}\]
- Data link control, go-back-N
  \[\text{Forouzan ed.5 ch.11.1-2, ch.23.2}\]
- Point-to-point protocol
  \[\text{Forouzan ed.5 ch.11.4}\]

*[Kihl & Andersson: 4.1, 4.2, 4.3, 4.5]*
TCP/IP model and data units

Application layer
- Processes

Transport layer
- SCTP
- TCP
- UDP

Network layer
- IP and other protocols

Data link layer
- Underlying physical networks

Physical layer

PACKETS
DATAGRAMS
FRAMES
Framing

- Physical layer $\rightarrow$ bitstream
- Link layer $\rightarrow$ frames
- We need logical transmission units
  - Synchronisation points
  - Switching between users
  - Error handling
Error control

• Data assumed error-free by higher layers
  – Errors occur at lower layers (physical)
  – Job for LLC layer

• Extra (redundant) bits added to data
  – Generated by an encoding scheme from data
Error types

- Bit error

- Burst error
Error detection process

Sender

Encoder

Message

Generator

Message and redundancy

Unreliable transmission

Receiver

Decoder

Message

Checker

Correct or discard

Received information
Error detection schemes

- Simple parity-check code
- Cyclic Redundancy Check (CRC)
- Checksum
Simple Parity-Check Code

- Extra bit added to make the total number of 1s in the codeword
  - Even $\rightarrow$ even parity
  - Odd $\rightarrow$ odd parity

\[
\begin{array}{c}
dataword \\
10011100 \\
\end{array}
+ \begin{array}{c}
0 \\
\end{array}
= \begin{array}{c}
codeword \\
100111000 \\
\end{array}
\]

- Can detect an odd number of errors
Block coding

- Divide the message into $k$-bit blocks, called **datawords**.
- Add $r$ redundant bits to each block. The resulting $n$-bit blocks ($n=k+r$) are called **codewords**.
- The code rate is $R=k/n$. 
Cyclic Redundancy Check (CRC)

- Predefined shared *divisor* to calculate codeword
CRC: Polynomial representation

• The dataword of $k$ bits is represented by a polynomial, $d(x)$.
• The degree of the polynomial is $k-1$. 

![Diagram of binary pattern and polynomial]![Diagram of short form]
**CRC: The principle**

- **Objective**: Send a dataword $d(x)$ of $k$ bits represented by a polynomial of degree $k-1$.
- **Given**: Generator polynomial $g(x)$ of degree $m$.
- **Find**: Remainder polynomial $r(x)$ such that:
  \[ c(x) = d(x) \cdot x^m + r(x) \]
  can be divided by $g(x)$ without remainder.
- **Codeword $c(x)$** will then be sent to the receiver.
- **$r(x)$** has degree $m-1$ or less, and CRC has $m$ bits.
CRC: How it works

- **Sender:**
  1. Generate $b(x) = d(x) \cdot x^m$
  2. Divide $b(x)$ by $g(x)$ to find $r(x)$
  3. Send $c(x) = b(x) + r(x)$

- **Receiver:**
  1. Divide $c'(x) = c(x) + e(x)$ by $g(x)$
  2. Check remainder $r'(x)$ – if 0 data correct, $c(x) = c'(x)$
  3. Remove CRC bits from codeword to get dataword
Example: CRC derivation

- For dataword 1001, derive CRC using generator 1011.

- Data polynomial: \( d(x) = x^3+1 \)
- Generator polynomial: \( g(x) = x^3+x+1 \)
- Dividend: \( b(x) = d(x) \cdot x^3 = x^6+x^3 \)
- Codeword polynomial: \( c(x) = d(x) \cdot x^3 + r(x) \)
- CRC polynomial: \( r(x) = ? \)
Example: CRC derivation

Dividend: augmented dataword

Dataword: \(x^3 + 1\)

Divisor: \(x^3 + x + 1\)

Remainder: \(x^2 + x\)

Codeword: \(x^6 + x^3\) | \(x^2 + x\)
Error detection capabilities

- Single errors: $e(x) = x^i$ is not divisible by $g(x)$
- Double errors: $e(x) = x^j + x^i = x^i(x^{j-i}+1)$
  - Use primitive polynomial $p(x)$ with $deg = L$. Then if $n-1 < 2^L - 1$ it is not divisible and all double errors will be detected
- If $x+1 | g(x)$ all odd error patterns will be detected
- In practice, set $g(x) = (x+1) \cdot p(x)$
Some standard CRC polynomials

<table>
<thead>
<tr>
<th>Name</th>
<th>Polynomial</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-8</td>
<td>$x^8 + x^2 + x + 1$</td>
<td>ATM header</td>
</tr>
<tr>
<td></td>
<td>100000111</td>
<td></td>
</tr>
<tr>
<td>CRC-10</td>
<td>$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$</td>
<td>ATM, AAL</td>
</tr>
<tr>
<td></td>
<td>11000110101</td>
<td></td>
</tr>
<tr>
<td>CRC-16</td>
<td>$x^{16} + x^{12} + x^5 + 1$</td>
<td>HDLC</td>
</tr>
<tr>
<td></td>
<td>1000100000100001</td>
<td></td>
</tr>
<tr>
<td>CRC-32</td>
<td>$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^2 + x + 1$</td>
<td>LANs</td>
</tr>
<tr>
<td></td>
<td>1000001001100000010001110110110111</td>
<td></td>
</tr>
</tbody>
</table>
Checksum

• The checksum is used in the Internet by several protocols although not at the data link layer.

• The main principle is to divide the data into segments of n bits. Then add the segments and use the sum as redundant bits.
Checksum process

Sender

Message

m bits  m bits  ...  m bits

Generator

m bits  m bits  ...  m bits  m bits

Message plus checksum

Receiver

Message

m bits  m bits  ...  m bits  m bits

All 0’s

[yes]

Discard

[no]

m bits

Checker

m bits  m bits  ...  m bits  m bits  m bits

Message plus checksum
Example: Checksum

<table>
<thead>
<tr>
<th>Sender site</th>
<th>Receiver site</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>

Sum → 36
Wrapped sum → 6
Checksum → 9

Details of wrapping and complementing:

<table>
<thead>
<tr>
<th>Details of wrapping and complementing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0</td>
</tr>
<tr>
<td>1 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>

Packet:

<table>
<thead>
<tr>
<th>Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>7, 11, 12, 0, 6, 9</td>
</tr>
</tbody>
</table>

Receiver site:

<table>
<thead>
<tr>
<th>Receiver site</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 1</td>
</tr>
</tbody>
</table>

Sum → 45
Wrapped sum → 15
Checksum → 0

Details of wrapping and complementing:

<table>
<thead>
<tr>
<th>Details of wrapping and complementing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0 1</td>
</tr>
<tr>
<td>1 0 0 0</td>
</tr>
</tbody>
</table>
Error Correction

Two alternative ideas

- Forward Error Correction (FEC)
  - Send each bit multiple times
  - Decode according to majority decision
- Retransmission
  - Resend the entire frame
- In most communication systems, both error detection and error correction occur.
See you in 15’ :)  

- After the break  
  - Data link control protocols  
  - Point-to-point protocol
Error and flow control

• The basic principle in error and flow control is that the receiver **acknowledges** all correctly received packets.

![Diagram showing data transmission and ACK]

Data

ACK
The need for flow control

- The receiver must be able to handle all received frames. If the transmission rate is too high, the receiver may become overloaded and drop frames due to full buffers.
Data link control protocols

- Flow control
  - Send data
  - Wait for ACK
- Error control
  - Detect error
  - Retransmit
- Framing
Stop-and-wait ARQ

• Send and wait
  – Keep time
  – Wait for ACK
  – Retransmit

• Automatic repeat request
  – Frames (SEQ++)
  – Acknowledgements (SEQ+1)
  – Mismatch = problem!
Stop-and-wait ARQ flow diagram
Stop-and-wait ARQ inefficiency

• Too much waiting
• Solution
  – Keep the pipe full
  – But not too full

• Sliding window
  – Size matters
  – Window size < $2^m$
Sliding window

Send window, first outstanding frame \( S_f \)

Frames already acknowledged

0 1 2 3 4 5 6

Send window, size \( S_{\text{size}} = 2^m - 1 \)

Frames sent, but not acknowledged (outstanding)

Frames that can be sent, but not received from upper layer

Send window, next frame to send \( S_n \)

Frames that cannot be sent

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

a. Send window before sliding

b. Send window after sliding
Go-back-N ARQ window size

![Diagram showing Go-back-N ARQ window size with two scenarios:](image)

- **Scenario a. Window size < \(2^m\):**
  - Correctly discarded
  - Time-out

- **Scenario b. Window size = \(2^m\):**
  - Erroneously accepted
  - Time-out

2013-11-04 | EITF25 – Internet: Technology and Applications
Go-back-N ARQ flow diagram
Selective repeat ARQ

• Why?
  – Too many retransmissions
• What if?
  – Just send lost frames
• Higher efficiency
  – Higher receiver complexity
Windows again

Send window, first outstanding frame $S_f$

Frames already acknowledged

Frames sent, but not acknowledged

Frames that can be sent

Frames that cannot be sent

$S_{size} = 2^{m-1}$

Receive window, next frame expected $R_n$

Frames already received

Frames that can be received and stored for later delivery. Colored boxes, already received

Frames that cannot be received

$R_{size} = 2^{m-1}$
Selective repeat ARQ window size

- **Frame 0**: Sender sends frames 0 to 3.
- **Frame 1**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.
- **Frame 0**: Sender sends frames 0 to 3 again.

- **Correctly discarded**: frames 1 and 2 are correctly discarded.
- **Erroneously accepted**: frames 1 and 2 are erroneously accepted.

- **Window size**:
  - a. Window size = $2^{m-1}$
  - b. Window size > $2^{m-1}$

- **Time-out**: Receiver signals time-out when all frames are not received.

- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.
- **Frame 0**: Receiver indicates successful delivery.

- **Correctly discarded**: frames 1 and 2 are correctly discarded.
- **Erroneously accepted**: frames 1 and 2 are erroneously accepted.
Selective repeat ARQ flow diagram

Sender

Receiver

Initial

Request

Arrival

Frame 0

Frame 0 delivered

Frame 1

Lost

Frame 2

Frame 3

NAK 1

Frame 1 (resend)

ACK 4

ACK 1

Frames 1, 2, 3 delivered

Initial

Arrival
Note on ”Selective Repeat ARQ”

Forouzan 4th, pp. 332-339

- **ACKₙ**
  - Acknowledges frame n-1 and all earlier frames. Receiver says it is expecting frame n.

- **NAKₓ**
  - Negative acknowledgment for missing frame x. Receiver says it has not received frame x.

Forouzan 5th, pp. 720-726

- **ACKₙ**
  - Acknowledges frame n and frame n only. Receiver says it has received frame n.

- **NAKₓ**
  - There is no such thing as negative acknowledgment. Receiver does not request a missing frame x as long as the frames it receives fall inside the receive window.

= OUR LECTURE SLIDES!
Point-to-point protocol (PPP)

- Direct connection between two nodes
  - Internet access
  - Home user to ISP
    - Telephone line
    - Cable TV
State transitions in PPP

• We need more protocols
PPP frame format

- Support for several (sub)protocols
- Address & control not used
- Maximum payload 1500 bytes
Link control protocol (LCP)

- Establish
- Configure
- Terminate

<table>
<thead>
<tr>
<th>Code</th>
<th>Packet Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Configure-request</td>
<td>Contains the list of proposed options and their values</td>
</tr>
<tr>
<td>0x02</td>
<td>Configure-ack</td>
<td>Accepts all options proposed</td>
</tr>
<tr>
<td>0x03</td>
<td>Configure-nak</td>
<td>Announces that some options are not acceptable</td>
</tr>
<tr>
<td>0x04</td>
<td>Configure-reject</td>
<td>Announces that some options are not recognized</td>
</tr>
<tr>
<td>0x05</td>
<td>Terminate-request</td>
<td>Request to shut down the line</td>
</tr>
<tr>
<td>0x06</td>
<td>Terminate-ack</td>
<td>Accept the shutdown request</td>
</tr>
<tr>
<td>0x07</td>
<td>Code-reject</td>
<td>Announces an unknown code</td>
</tr>
<tr>
<td>0x08</td>
<td>Protocol-reject</td>
<td>Announces an unknown protocol</td>
</tr>
<tr>
<td>0x09</td>
<td>Echo-request</td>
<td>A type of hello message to check if the other end is alive</td>
</tr>
<tr>
<td>0x0A</td>
<td>Echo-reply</td>
<td>The response to the echo-request message</td>
</tr>
<tr>
<td>0x0B</td>
<td>Discard-request</td>
<td>A request to discard the packet</td>
</tr>
</tbody>
</table>
Authentication protocols (AP)

- Password authentication (PAP)
Authentication protocols (AP)

- Challenge handshake authentication (CHAP)
Network control protocols (NCP)

- Preparations for the network layer
  - IPCP for Internet

<table>
<thead>
<tr>
<th>Code</th>
<th>IPCP Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Configure-request</td>
</tr>
<tr>
<td>0x02</td>
<td>Configure-ack</td>
</tr>
<tr>
<td>0x03</td>
<td>Configure-nak</td>
</tr>
<tr>
<td>0x04</td>
<td>Configure-reject</td>
</tr>
<tr>
<td>0x05</td>
<td>Terminate-request</td>
</tr>
<tr>
<td>0x06</td>
<td>Terminate-ack</td>
</tr>
<tr>
<td>0x07</td>
<td>Code-reject</td>
</tr>
</tbody>
</table>

IPCP packet structure:
- Code
- ID
- Length
- Variable
- IPCP information

Flag | Address | Control | 0x8021 | Payload (and padding) | FCS | Flag

2013-11-04
IP datagram encapsulation in PPP
PPP session example
PPP session example (cont.)
Summary: Data Link Layer

Logical Link Control Sublayer

- Frames
- Error control
  - Detection and correction
- Flow control
  - Stop and wait, go back N, selective repeat
- Point-to-point protocol