CSE 123 Discussion 2

10/09/2018

Sliding Window Protocol
Common.h

• You can add your own data structures here.
• struct Sender_t
  • SWS – Sliding window size
  • LAR (Last Acknowledgement Received) - Sequence number of last acknowledgement received, defines lower bound of the sender window
  • LFS (Last Frame Sent)- Sequence number of the last frame sent, defines upper bound of the window
  • Window is from [LAR+1, LFS], that is all frames that have been sent but not yet Acked.
Frame Sequence Number in Window

CASE 1: Usual Case
LAR <= LFS

CASE 2: Sequence Number Wrap Around
LAR > LFS

In this case, we are not using the full window of 4.

Sender with SWS = 4, sequence number in [0,7]
Receiver_t

- RWS - Max receive window size
- NFE - Next Frame Expected
- LFR - Sequence number of largest consecutive frame received
- LAF - Sequence number of largest acceptable frame
- LFR = NFE - 1
- LAF = NFE + RWS - 1
Frame Sequence Number in Window

CASE 1: Usual Case
\[ NFE + RWS - 1 \geq NFE \]
\[ NFE + RWS - 1 \geq NFE \&\& \text{seqNo} > NFE \&\& \text{seqNo} \leq NFE + RWS - 1 \]

Remember NFE is just LFR + 1 and LAF is just NFE + RWS - 1.

Green sequence numbers are in window and grey are outside.

CASE 2: Sequence Number Wrap Around
\[ NFE + RWS - 1 < NFE \]
\[ NFE + RWS - 1 < NFE \&\& (\text{seqNo} > NFE \| \text{seqNo} \leq NFE + RWS - 1) \]

Receiver with RWS = 4, sequence number in [0,7]
Frame Not in Window On Receiver

• Send ACK with number NFE – 1
  • This tells the sender that receiver has successfully received all frames up to NFE – 1 = LFR

• Will happen when ACK is lost and needs to be re-sent
Circular Sender/Receiver Window

• Implement send and receive queue as circular array or list
• Index in to sender’s send queue using sequence number % SWS
• Index in to receiver’s receive queue using sequence number % RWS
• Take the codes from P&D as reference
Sequence Number Wrap Around

• You should NOT use more than 8 bits (unsigned char) for seq/ack numbers.
• You need to handle sequence number wrap around once the value reaches 255. Your seq/ack number should wrap back to 0.
• How to do this?
• Answer: % modulus
Homework Discussion

Due on 10/12 Friday in class
Automatic Repeat Request (ARQ) Protocol

Stop and Wait:

1. Sender transmits a data frame with a sequence number encoded

2. Receiver replies with an acknowledgement frame

3. Sender either (1) transmits a new data frame if it gets an acknowledgement of the previous frame before the timeout, or (2) retransmits the frame after the timeout.
Timeout = 10ms
Round Trip Time = 8ms
Question 5?
Question 1f?
Two-Dimensional Parity

Given below is a series of 7 7-bit items of data, with an additional bit each and an extra byte to account for parity.

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

What is the original data?

How many bits are actually transmitted?
Given below is a series of 7 7-bit items of data, with an additional bit each and an extra byte to account for parity. Odd parity is applied.

<table>
<thead>
<tr>
<th>First Byte</th>
<th>Parity Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 1 0 1 1</td>
<td>1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>0 0 0 0 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 0 0 0 1</td>
<td>0 1 0 1 0 0 0 1</td>
</tr>
</tbody>
</table>

Can we detect/correct all 1-bit flip?

Question 2.a?
Two-Dimensional Parity

Given below is a series of 7 7-bit items of data, with an additional bit each and an extra byte to account for parity. Odd parity is applied.

Can we detect/correct all 2-bit flips?

What if 2 flipped bits are in the same row, or the same column?

Question 2.b
Error checking using CRC

Sending side:
1. To detect up to $k$-bit burst errors, select a generator $G$ which has $k + 1$ bits. (Look up Table 2.3 on Page 102 of the textbook)
2. Shift data $D$ to the left for $k$ bits ($D << k = 2^k * D$)
3. Divide ($2^k * D$) by generator $G$ using Modulo-2 arithmetic, and get a remainder $r$
4. Actual data transmitted is ($2^K * D$) XOR $r$
Example:

Assume sender’s message is 1011 0011 0101 0110, and we decide to encode this message with CRC-8 polynomial. What is the actual bit-sequence that get transmitted?

\[
D = 1011 \ 0011 \ 0101 \ 0110
\]
\[
G = 1 \ 0000 \ 0111
\]
\[
k = 8
\]
Sending side:
1. To detect up to k-bit burst errors, select a generator G which has k + 1 bits. (Look up Table 2.3 on Page 102 of the textbook)
2. Shift data D to the left for k bits (D << k == 2^k * D)
3. Divide (2^k * D) by generator G using Modulo-2 arithmetic, and get a remainder r
4. Actual data transmitted is (2^K * D) XOR r

D = 1011 0011 0101 0110
G = 1 0000 0111  k = 8

Answer:
1011 0011 0101 0110 1101 0101
Error checking using CRC

Receiving side:
1. Check whether the received bit stream \( S (2^k \times D \oplus r) \) is divisible by the generator \( G \).

Message = 1011 0011 0101 0110 1101 0101

\( G = 1 \ 0000 \ 0111 \)

\( r = 0 \rightarrow \text{Error check PASSED} \)
Questions?