CSE 123 : Computer Networks

Winter 2022 : Discussion-3

P1 TA : Shreyas Anantha Ramaprasad
Overview

1. Implementing communication between two or more hosts
2. Hosts implemented as Threads
3. Network link is simulated
4. 2 types of hosts → Senders and Receivers
5. Sender hosts must transmit messages typed in at the command line to a corresponding receiver host.

6. **Messages can be dropped**
7. **Messages can be corrupted!**
8. **Improve performance using SWP.**
Overview

• Tasks Completed in Project 1a
  • Framing and Message Partitioning
  • Acknowledgements
  • Retransmissions in case of dropped frames
  • Stop and Wait

• Tasks to do in Project 1b
  • Error detection
  • Instead of stop and wait, implement Sliding Window protocol
Error Detection - CRC

- -c option with ./tritonlink allows you to corruption probability
- Your task: Implement CRC:
  - Sender
    - When constructing the frame (header+payload), set CRC field to 0.
    - Calculate the CRC for the frame and set the CRC field
    - Convert frame to char array
    - Append to outgoing frames list
    - Check for corruption in ACKs
  - Receiver
    - Check for corruption.
    - If corrupted, drop. Otherwise, send ACK
- Remember, both messages and acknowledgements can be corrupted
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
Selective Repeat or Go Back N - Either is okay

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 Ack-ed.
Frame Sequence Number in Sender

**CASE 1: Usual Case**
LAR \leq LFS

LAR \leq LFS \&\& \text{seqNo} > LAR \&\& \text{seqNo} \leq LFS

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

**CASE 2: Sequence Number Wrap Around**
LAR > LFS

LAR > LFS \&\& (\text{seqNo} > LAR \text{ || seqNo} \leq LFS)

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

• struct Receiver_t
  • **RWS** - Max receiver 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 Receiver

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

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

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

Green sequence numbers are in window and grey are outside.

Receiver with RWS = 4, sequence number in [0,7]
Sender Buffer/Window

• Sender needs to maintain a window (buffer) while sending packets
• The structure is as follows
  ```c
  struct sendQ_slot {
    struct timeval* timeout;
    Frame* frame;
  } sendQ[SWS];
  ```
• Index in to the sender buffer using (sequence number % SWS)
• Max SWS = 8 for this project
• Don’t use SWS = 1 as it becomes equivalent to stop and wait
Receiver Buffer/Window

• Similarly, the receiver needs to maintain a window too
• The structure is as follows
  ```c
  struct recvQ_slot {
    Frame* frame;
  } sendQ[RWS];
  ```
• Index in to the receiver buffer using (sequence number % RWS)
• Maintain a separate recvQ_slot for each sender
• Try out different RWS, but use an RWS>=2 in your final submission (to avoid submission timeouts). Recommended : RWS = 8