CPSC 558 — Adv Networking — Project Part 1 – F-8 Protocol

P1: Data Link

Due Sunday, 20 March, by 11pm.

Introduction

In this project part, your team will implement the F-8 protocol over a tiny network. The host and link implementation of the Simple Data Link lab assignment should be used. The F-8 frame format is described below.

Implementation Language: C++ (or C)

F-8 Frame Format

Version: 0/0..15 // "F8"
Type: 0/16..18 // Type of frame; default 1 == Data.
TTL: 0/19..23 // hop down-count, do not forward (discard) a packet at 0.
DLen: 0/24..31 // Data length, in bytes; min 32, max 255.
Src_addr: 1/0..31
Dest_addr: 2/0..31
Protocol: 3/0..7 // Higher-level Ptcl: min 1; 17 for UDP.
F_Ident 3/8..15 // min 32; ID for all frag parts, for reassy; else 32.
F_Seq: 3/16..23 // min 32; frag seq # for frag reassy.
H_Checksum: 3/24..31 // 1 byte, on above header.
Data: <= 255 bytes
M_Checksum: 1 byte, on above frame.

The F-8 was patterned after that of IPv4. The F-8 frame header consists of 4 words, 16 bytes. The F-8 Version byte is "F8". The 3-bit Type field is used to indicate the type of frame, which for this project part should be set to the default. The TTL field is the number of suggested data link transmission hops before the frame is discarded as having gone too far. The Dlen byte indicates the length of the Data in bytes, max 255. This field should be decremented the by the link sending side. The Src_addr and Dest_addr fields are 32 bits each and identify the original network sender and intended recipient. The Protocol field indicates the higher-level protocol, if any. The F_Ident and F_Seq fields indicate that a higher-level message has been split into fragments if the F_Ident is >32, else the frame data contains the entire message. The F_Seq field has the fragment sequence number, beginning with 32. The H_Checksum field contains a checksum (a simple sum, with overflow allowed) on the prior header bytes. Ditto for the M_Checksum on the prior message frame. (The reason for the funny minimum field values is to avoid a byte of zero, so that the entire frame can be easily printed as a C string.)

Checksum

F-8 includes a (primitive) checksum error detection mechanism. This is caculated by simply summing successive bytes into a standard 32-bit two's-complement integer, and then extracting the bottom byte (i.e., low-order 8 bits). This extraction can be accomplished via an XOR with 0xFF. There are two checksums. (The M_Checksum includes the prior checksum byte in its caculation.)

F-8 Functions

In order to transmit an F-8 protocol message, augment each host Node with snd_f8() function that takes a message string argument, and which wraps the message in one or more F-8 frames and passes each frame to the Link's xmt() function. The snd_f8() function should take a second argument which, if not 0, is the maximum count of data bytes to use in each F-8 frame.

For extra credit, implement a snd_f8r() function which duplicates snd_f8() but sends fragments in reverse

sequence order.

Alter the host Node rcv() function to check the first two bytes of an incoming frame, and if it is "F8", then the Node's rcv_f8() function should be called with the frame as an argument. Also augment each host Node with a rcv_f8() function which extracts the message from one or more F-8 frames.

Given a text message on one host, you will wrap it in one or more F-8 frames and transmit it to its target host. Augment each host with a 32-bit network address slot where the upper 3 bytes (in octets) is 196.168.2 and the bottom byte is 1 for the sender and 2 for the receiver. Thus the sender host address is 196.168.2.1 and the receiver is 196.168.2.2.

Use a TTL value of 31, the maximum. Use a protocol of 1. Be sure to check the checksums on the receiving side; however, we will introduce no errors into the frames in Project Part 1.

The new functions should also output the string (message) prefaced by their node or link name and and "sent_f8" or "rcvd_f8", similar to the old functions.

Topology

Create the following (star) network topology with nodes and links, via their attach() functions.

Node 196.168.2.1 <-L1-> Node 196.168.2.2 <-L2-> Node 196.168.2.3
Node 196.168.2.2 <-L3-> Node 196.168.2.6

Host Maps

Augment each host Node with a host map. The map will tell the host: given Dest_addr X, use Link Y. Here are two of the host maps for the above topology:

196.168.2.1 Map:
196.168.2.2, use L1
196.168.2.3, use L1
196.168.2.6, use L1

196.168.2.2 Map:
196.168.2.1, use L1
196.168.2.3, use L2
196.168.2.6, use L3

Your host code should first check whether the Dest_addr is its own, and if not then use its host map look up which link to use and then forward the message over that link (i.e., call that Link's xmt() message.

Implementation

As usual, instantiate the network from int-main.

Testing

Test (from int-main) by using the following 57-byte host message, not including the quotes.

```
"CPSC 558 -- Adv Networking -- Project Part 1 -- Data Link"
```

The test should be done in two ways. Firstly, send the entire message in one frame. Secondly, send the text but limit each frame to at most 30 bytes.

To test your extra credit snd_f8r() function, call it with the above message and a 30 byte limit to send the second (short) fragment first. This test will exercise your program's ability to reassemble the fragments correctly after they have been sent out of sequence.

Readme File

You should provide a README.txt text file that includes the class and section, your (team) name, the project/program name, instructions for building, instructions for use, any extra features, and any known bugs to avoid. Be clear in your instruction on how to build and use the project by providing instructions a novice would understand. If there are any external dependencies for building, the README must also list them and how to find and incorporate them. Usage should include an example invocation.

A README would cover the following:

- Program name
  - Your Name (authors)
  - Contact info
  - Class number, Section (eg, 01),
  - "Project Part" and its number
- Intro (see the Introduction section, above)
- External Requirements
- Build, Installation, and Setup
- Usage
- Extra Features
- Bugs

Sample Run

Include a printout of sample runs for the test, showing the text output in a sample.txt file.

Academic Rules

Correctly and properly attribute all third party material and references, lest points be taken off.

Submission

Put the README, sample, and your source files (and any other text or pdf files) in a folder. Name the folder with the class number and project number (eg, "`p1`") or lab assignment number (eg, "`a1`") as a prefix (eg "`343-p1-`") and with a suffix of your team name. (Include no executable or "object" files.) Then zip up this folder. Name the .zip file the same as the folder. Thus, if your team name is RAX and this is for project #1, your folder and zip file would have this name: "`343-p1-RAX`".