ROSE-HULMAN INSTITUTE OF TECHNOLOGY
Department of Electrical and Computer Engineering

ECE 312 Communication Networks Winter 2016-2017

Due Thursday, February 2, 2017

Project 2 – Packet Encapsulation

In this project we will learn about packet encapsulation by implementing two protocols which will operate above the transport layer. We will use sockets as we did in project 1, but this time we will use datagrams (UDP sockets). The first protocol we will call Rose-Hulman Protocol (RHP). It is a very simple protocol which sits directly above the transport layer and uses UDP services. The protocol format is defined below. The second protocol we will call the Rose-Hulman Message Protocol (RHMP). It sits directly above RHP and invokes the services of RHP to send messages to its peers. Its protocol definition is also provided below. In an actual protocol stack implementation, RHP and RHMP would be implemented as separate processes. There could then be multiple instances of RHMP, each using a single instance of RHP, but with a unique port ID. For simplicity, you may implement this project with a single process, but use modular code to separate the functionality of the two layers.

You can find an example program for a client which uses UDP sockets (SOCK_DGRAM) to send a message to a server on Moodle. A server application will be running on a campus workstation which is accepting messages of the RHP protocol. The IP for the server is 137.112.38.47, and the port is 1874. Modify the example program to create a client application which sends a message to the server using the RHMP protocol. You will need to implement the construction of the message to meet the protocol definition. Keep in mind that the RHMP message should be encapsulated in an RHP packet. Your program should send three messages: one RHP control message with the string "hello", one RHMP message of type *Message_Request*, and one RHMP message of type *ID_Request*. It should wait for and accept a response message after each request. For each response, it should verify it meets checksum and extract the values for each field. For your source port value, use your campus mail number (CM#), without the CM. You can use either project partner's CM#. The server will send you a unique message and ID. Display the value for ***each*** field of the received messages, including the RHP fields. Print this information and submit it along with your source code and a cover page. The output of your program may look similar to:

Message received:
RHP type: 1
dstPort: 4000
. . .

For troubleshooting purposes, the server will send a response to every received packet. For any valid RHP control message received, it will send an RHP packet in return with an ASCII string message "Message received". For a valid RHMP message it will return a response message if the packet was a request message, otherwise it will provide the same RHP acknowledgement. For invalid format messages, it will return an RHP control packet with an appropriate message. Your program should compute the 16-bit internet checksum (as defined in Chapter 6.4) and insert it in the checksum field of the RHP header. It should also verify the checksum of each received RHP packet, and discard any packet that fails the check. Note that the RHP packets must have an even number of octets, and should insert a buffer of 8 zeros before the checksum if there are an odd number of bytes in the packet.

Protocol header formats (assumes lsb first transmission)

Rose-Hulman Protocol (RHP)

type	dstPort/length	srcPort	payload	buffer	checksum
8	16	16	variable	8 (or 0)	16

field	*size (bits)*	*description*
type	8	RHP message type (payload)
dstPort	16	Destination RHP port (length of payload for *type 1* message)
srcPort	16	RHP port of source
payload	variable	RHP SDU (dependent on type)
buffer	8 (optional)	Optional 8-bit buffer of zeros to ensure even number of octets in packet
checksum	16	16-bit internet checksum

defined types: 0 – RHMP message
1 – control message (ASCII string)

Rose-Hulman Message Protocol (RHMP)

type	commID	length	payload
6	10	8	variable

field	*size (bits)*	*description*
type	6	RHMP message type (payload)
commID	10	Communication identifier
length	8	Length of payload (bytes)
payload	variable	RHMP payload data (dependent on type)

defined types:		
	1 – *Reserved*	Empty payload (length = 0)
	2 – *ID_Request*	Request identifier response (no payload)
	4 – *ID_Response*	32-bit unsigned integer identifier
	8 – *Message_Request*	Request ASCII message response (no payload)
	16 – *Message_Response*	ASCII character string

All communications for this project should use *commID* 312.

Use *dstPort* 105 for messages sent to the server.

Programming note: in C, storage size of data types can depend on the hardware and compiler that is used. To ensure correct operation, you may want to make use of the following data types:

uint8_t	unsigned 8-bit integer
uint16_t	unsigned 16-bit integer
uint32_t	unsigned 32-bit integer

You may also find the following function useful:

```
void *memcpy(void *dest, const void *src, size_t n);
```