CHAPTER 11

# Designing and Implementing Network Protocols

## 11.1. Introduction

This GINI toolkit includes a bare-bones all-software IP router. This chapter provides a set of projects that present protocols that can be built on top of the existing functionality provided by the GINI router. To do these projects, it is necessary to understand the basic structure of an IP router and gain intimate knowledge of certain portions of the GINI router's implementation.

## 11.2. Implementing the User Datagram Protocol

### 11.2.1. Overview

The *Internet protocol* (IP) provides a non-guaranteed datagram service across two machines that implement IP. The *user datagram protocol* (UDP) is a simple protocol that allows an application to use this service to send datagrams across two applications running on two different machines running on the Internet. The UDP is a very minimal protocol. It provides checksumming facility for the data payload and multiplexing mechanism so that datagrams can be delivered application-to-application. The multiplexing mechanism in UDP uses ports. The applications that use UDP are expected to handle issues such as out-of-order packet reception, lost datagrams, flow control, and congestion control.

### 11.2.2. Objective

Implement standards compliant UDP services on the GINI router, which already provides a network stack with many IP and ICMP services. The UDP service implementation should use the services already provided by the GINI router as much as possible. The UDP service should provide couple of funtions such as `send()` and `receive()` with appropriate arguments as the application interface.

### 11.2.3. Background Material

UDP is an extremely simple protocol. General information on UDP including packet formats and how it is structured with respect to IP can be obtained from most computer networking textbooks. However, for this implementation, it is best to read the requirements outlined in the following Internet RFCs.

**RFC 768** — User Datagram Protocol

**RFC 1122** — Requirements for Internet Hosts - Communication Layers

**RFC 1716** — Towards Requirements for IP Routers

### 11.2.4. Description

Figure 11.1 shows a suitable example network for demonstrating the UDP protocol implementation. Because the virtual machines are user-mode Linuxes, they already have a Internet standards compliant TCP/IP stack that includes a UDP service. The routers provided as part of the GINI toolkit do not have UDP service but provide almost complete IP and ICMP services. The UDP service implemented as part of this project will go on the routers. The UDP service essentially provides two functions: checksumming and multiplexing.

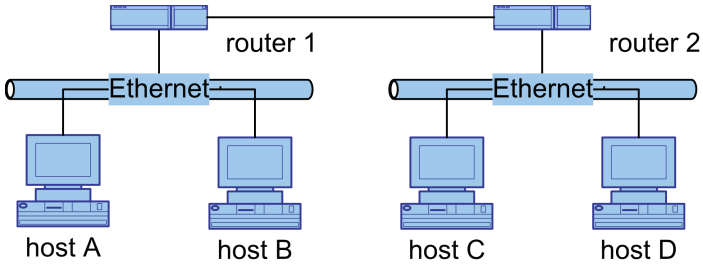


**Figure 11.1:** Example network topology for UDP experiments.

The UDP service implementation should handle packets with illegal UDP checksums according to the Internet standards. For instance, what should the UDP service do when it receives a UDP packet with illegal UDP checksum? Further, the UDP service performs packet multiplexing among the different applications using the notion called *ports*. If a packet arrives for a port that is not active (i.e., no known application is associated with the port), an error condition is created. This error condition should be handled according to the specifications on the Internet standards.

### 11.2.5. Useful Tools

This is an implementation project that requires the design and implementation of the UDP service. However, the following tools should be helpful in injecting test packets, establishing communication sessions, and visualizing the packets.

- `nc` — create arbitrary UDP connections
- `hping2` — create and inject arbitrary UDP packets
- `tcpdump` — a terminal-based packet visualizer
- `wireshark` — a graphically packet visualizer

### 11.2.6. Suggested Implementation Plan

1. Read the background material on the UDP protocol.

2. Design and implement the UDP service. The UDP service provides two basic functions: multiplexing and checksumming. The UDP service should implement the port concept to perform multiplexing. The UDP service should be able send or receive packets for given ports.

3. In the UDP service, if a packet is received for a unknown port, it should generate an error according to the Internet standards (check RFCs for detail). This feature is used by some application such as `traceroute` to carry out their function. Similarly, if the local UDP service sends a packet for a port that is not available on the remote machine, an error message will be received. Check whether it is necessary to handle this error message.

4. The UDP service should compute checksums on each outgoing and incoming packet. The UDP checksum includes a pseudo header of the IP packet to bind the payload to the header. Refer to the RFC documents for the exact checksum computation procedure. Routines provided in the GINI router for IP checksumming could be reused for this process.

5. Provide a simple application interface in the form of function calls for sending and receiving UDP datagrams. This application interface should be sufficient to build clients and servers. Suppose the application interface provides the following functions.

```
void send_udp(int dst, char *data, int len);
int recv_udp(int src, char *buf, int *len);
```

   A simple UDP client can be built using the above routines as follows:

```
while (1) {
   printf(``# '');
   scanf(``%s'', writebuf);
   send_udp(servaddr, writebuf, strlen(readbuf));
   recv_udp(servaddr, readbuf, &buflen);
   printf(``%s'', readbuf);
   ...
}
```

   The application interface shown above is merely a suggestion. The eventual application interface created as part of this project could be similar but it should be complete (handle both client and server needs) and simple.

6. Use the topology shown in Figure 11.1 to test the implementation. The UDP service runs on the routers. Use the packet injection tool `hping2` at `host A` to inject UDP packets and ensure that the UDP service running at `router 1` (`router 2` should work as well if the packet is forwarded) is processing them according to the specifications. Various different types of packets such as bad checksum packets, zero checksums, and undefined ports can be injected by `hping2`. Using `nc` run a UDP server at `host A` at port 9000.

```
nc -l -u -p 9000
```

   Connect to the server at `host A` from a client running at `router 1`. This client is a simple C program that is running at the router and using the application interface provided by the

UDP service. Once the client connection is working, run a UDP server at `router 1` at port 9050. Again the UDP server is a simple C program that is developed using the application interface provided by the UDP service. Connect to the UDP server running at the router from the machine using the following command.

```
nc –u router_1_IP_addr 9050
```

This should connect to the server and allow simple message communication between `host A` and `router 1`.

7. Run the traceroute from `host A` to `host C` in the UDP mode. The traceroute should provide the expected output.

### 11.2.7. Expected Deliverables

1. The UDP service built on top of the IP/ICMP services that are already part of the GINI router. The UDP service should interoperate with the one provided by the Linux networking stack.
2. An application programming interface for the UDP service in the form of function calls and their usage.
3. Demonstration of UDP service with applications such as traceroute (using UDP).
4. Simple UDP based telnet-like communication between a router and machine. The client or server at the machine can be implemented using `nc`.