Network Programming Laboratory
Winter 2018/2019

Session 5:
Raw and Packet Sockets

Janne Riihijärvi

RWTH Aachen University - Institute for Networked Systems

Contents

1 Overview

In all the previous sessions the socket interface has been used to enable application data transport over UDP or TCP protocols. While vast majority of applications can be enabled this way, occasionally there is a need to use alternative transport or network layer protocols. In this laboratory session you will learn basic usage of *raw sockets* enabling arbitrary data to be communicated on top of IP, and will also be introduced to *packet sockets*, which can be used to flexibly implement protocols on layers 2–4 in applications.

2 Raw and Packet Sockets

In this section we describe briefly both raw IP sockets as well as packet sockets.

2.1 Raw IP Sockets

The description and semantics of raw IP sockets are given in detail on the manual page `man 7 raw`. The basic function call is

```
#include <sys/socket.h>
#include <netinet/in.h>
raw_socket = socket(AF_INET, SOCK_RAW, int protocol);
```

The third argument corresponds to the protocol number used in the IP header. Helper function `getprotobyname()` and others can be used to find protocol numbers using the name of the protocol, and the complete protocol number database can usually be found from `/etc/protocols` text file. There are also various macros defined for commonly needed protocol numbers. For example, the protocol number for ICMP messages is defined as `IPPROTO_ICMP`.

It is also possible to set the value of `protocol` in the above call to `IPPROTO_RAW`, which allows arbitrary IP headers to be sent, including the corresponding protocol numbers. However, this is rarely needed, and if this level of control becomes necessary packet sockets are often used instead (described in the next subsection).

For sending and receiving data with raw sockets, same function calls can be used as for datagram sockets. The address structure to be used is the same as well, although the interpretation of the port number field `sin_port` is different, corresponding to the IP header protocol number. For sending it should be set to zero in Linux.

2.2 Packet Sockets

The description and semantics of packet sockets are given in detail on the manual page `man 7 packet`. The basic function call is

```
#include <sys/socket.h>
#include <netpacket/packet.h>
#include <net/ethernet.h> /* the L2 protocols */

packet_socket = socket(AF_PACKET, int socket_type, int protocol);
```

The argument `socket_type` can be set to either `SOCK_RAW` or to `SOCK_DGRAM` depending on whether data received through the sockets should include the link layer header or not, respectively. Link layer header information is available in the corresponding socket address structure `sockaddr_ll`. The argument `protocol` corresponds to the IEEE 802.3 protocol numbering scheme. Macros for commonly used protocols are defined in `<linux/if_ether.h>`. When set to `htons(ETH_P_ALL)` all the higher layer protocols are received through the

socket.

As with using wireshark, opening packet sockets requires root privileges.

By default receiving from a packet socket will give you frames whose destination address matches to that of the interface in question, even in shared medium networks. The interface can, however, be set to so-called *promiscuous mode* causing all frames to be received. The process for this is explained in section "Socket Options" in the packet socket manual page.

Note that packet sockets are a powerful tool in Linux, but do not tend to be particularly portable to other operating systems, so use them with care. The `pcap` library offers a portable version of some of the packet socket functionality, but not all.

3 Programming Tasks

1. Make your own version of `ping` client that will send an ICMP echo request to a specified IP address, and waits for either a timeout or an ICMP echo reply. Your implementation should be based on raw IP socket or (much more challenging option) packet sockets.

   Hints: Review first the structure of ICMP echo request and reply, and double-check your understanding against Wireshark output generated using `ping`. Most of the fields in the ICMP header (message type, code, identifier and sequence number) are straightforward to deal with, with the checksum being slightly more complicated. However, functions to compute the Internet checksum are readily available, one of them being given in RFC1071.

   For constructing the ICMP message sent over a raw socket, the simplest and safest approach is to set it up as a character array, and set the field values one at a time (remember the issues with network and host byte order). An alternative is to define a suitable `struct`. However, in this case one has to be careful with lengths of the fields, as well as the padding compiler might insert between the fields.

2. Based on packet sockets, make a highly simplified command like version of wireshark, that upon startup receives all the packets from the network interfaces on your machine, and displays a single-line summary on the terminal. The summary should include at least the protocol types used at network and transport layer, and any additional data you might want to infer from the frames.