

# Lab 8

*14 May 2009*

In this lab you'll put into practice the ideas from Chapters 3 and 4 of TSIC: you'll get exposed to UDP, the datagram (connectionless) transport-layer service, and you'll also see the use of a (fairly simple) application-layer protocol, for coordinating SMS-like data transmission.

## 1 UDP

The main difference between a program using TCP and one using UDP is that in the latter, we don't need to establish a connection before we can send and receive messages. We do still need to open a socket and bind it to a port in order to receive messages, but then we can loop around a call to `recvfrom`.

For outbound messages, the two transport protocols are even more similar: in UDP you create a socket and then call `sendto`, simply skipping the `connect` step.

These calls are described in somewhat more detail in Chapter 4 of TSIC, and in the API reference in the back of TSIC, as well as in the man pages for `recvfrom`, `sendto`, and `udp`.

## 2 Today's protocol

Each message that is sent via this protocol has three components: a phone number, a name, and a message body. The number and name both represent the sender; the phone number is a seven-digit number and the name is up to ten characters long. The message body is up to 112 characters. All characters are limited to printable ASCII.

The way that these three parts are laid out in the packet is as follows: the phone number is first, as a 32-bit int in network (big-endian) order. Next comes the name, always as eleven bytes, including a null-terminator and padded with unused bytes as necessary. The next byte of the message contains the number of characters in the message body; and the packet concludes with that many bytes, the message body. Note that the message

body is *not* null-terminated; the end of the string can be computed from the length byte. Under no circumstances should datagram packets using this protocol be longer than 128 bytes, but they can be shorter, and message bodies can be of zero length.

Slightly tricky thing: unlike a TCP stream, which is continuous, UDP packets are discrete entities that arrive all at once. That means you can't just read the number with one call, then read the name with another, and so on; the usual solution to this problem is to define a struct that matches up with the protocol, permitting a pointer to such a struct to be given as the `recvfrom` buffer.

### 3 Task

Write a receiver and a sender for this SMS protocol. They should be able to interoperate with the `smsrecv` and `smssend` that I've put in the course directory.

You might want to refer back to last week's `chatlogin` and `chatconnect` code (yours or mine) to remember how some of the TCP/IP calls were made; today's will be similar.

Hand in whatever you have done at the end of the lab period.