Due **July 9th by 14:00** in the course assignment boxes. **Be sure you place your solution in a box assigned to Cmpt 471.**

Soft copy should be submitted using the School's Course Management System The URL for the Course Management System is `https://courses.cs.sfu.ca`.

This assignment has two goals:

1. To give you additional experience using the `wireshark` protocol analyser to capture network traffic; and
2. to firmly ground you in the reality of assembling a packet and transmitting it on the network.

At the end of this assignment, you will know that anyone with sufficient knowledge and privileges can construct any packet or frame that they choose to create. To accomplish this, you will do a bit of low-level socket programming:

1. Construct an ICMPv4 echo message,
2. encapsulate it in an IPv4 packet,
3. encapsulate the IPv4 packet in an Ethernet V2 frame, and
4. transmit it.

To prove that you've gotten it right, use `wireshark` to capture your echo message and the resulting echo reply message from the target workstation. So that it'll be easy to recognise your ICMP message, the data portion should contain your name in ascii.

Chapter 21 of the text provides a general introduction to socket programming at the transport level (TCP and UDP protocols). For this assignment, however, you'll be working at a much lower level, with complete responsibility for assembling the ICMP message, IP packet, and Ethernet frame. The lower levels of the socket programming interface can be a bit intimidating. To keep the programming to a manageable level, for this assignment you will fill in a program skeleton with the code that constructs the ICMP echo message and encapsulates it in the IP packet and Ethernet frame. The skeleton code is posted with this assignment in the file as4skel.cpp. The skeleton code is written in C++, and can be adapted for plain C if you wish. Use one of these two languages for your solution.

The skeleton code is hardwired to use the `eth1` network interface. You will need to adjust IP and Ethernet addresses to match the network segment that you're using.

For explanations of the socket programming packet interface, the `man` pages for `packet`, `netdevice`, and `socket` are good places to start. They contain pointers to additional `man` pages.

The majority of the `#include` files that you'll need are found in `/usr/include/net` and `/usr/include/netinet`. You can choose to use the structures defined in these files for the Ethernet, IP, and ICMP headers, or you can simply define your own structures.

One issue to be aware of is the difference between host byte ordering and network byte ordering. 'Big-endian' CPUs store the most significant byte of a multibyte number in the the lower numbered memory location (big end first). 'Little-endian' CPUs store the most significant byte in the higher numbered memory location. Given the usual conventions for writing numbers (most significant digits to the left) and writing memory locations (lower numbered addresses to the left), multibyte numbers tend to look scrambled when printed out a byte at a time on a little-endian machine. Internet network byte ordering is defined to be big-endian. Intel CPUs are little-endian. You will

want to read the `man` page for the functions `ntohs`, `ntohl`, `htons`, and `htonl`, functions which translate between host and network byte order.

Your program should be named `a4soln.cpp` if you use C++ as the programming language, `a4soln.c` if you use C as the programming language. It does not have to have an elegant interface — a minimally acceptable program will run on one workstation, with hard-coded Ethernet and IP addresses. If you're feeling ambitious, you can experiment with supplying address information on the command line or obtaining it using system calls, but for this assignment it is not required. (A more flexible interface complicates the programming and distracts from the central goals of the assignment.) **This paragraph is not permission to write a poorly structured program!** Well-structured, well-documented code is required.

While not required for the assignment, you may find it interesting to experiment a bit with lying — try using a source IP or Ethernet address different from the workstation where the program is running, or mismatched IP and Ethernet destination addresses.

The deliverables for this assignment are:

- Hard copy of your program, with comments.
- Hard copy of the frames containing your ICMP message and the reply, as captured by `wireshark`,
- The usual general documentation describing how you did the assignment (code design, data capture, how to compile and run your program, *etc.*).
- A soft copy of the source code for your program. This will be compiled and tested in the Network Lab. Be sure to specify the workstation where the code should be built and tested!