

Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique



University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Lab 3

Socket Programming and the IP protocol v4

Objectives: (1) to know the importance of IP protocol; and (2) to get familiar with Network Socket programming, which will be needed in subsequent labs.

In this lab, students are required to make two programs using sockets to transmit IP packets from a client to a server.

What to submit: Submit your code (two Java classes)

The Scenario:

Client's IP: 192.168.0.3 (in this example, it will vary based on your client's IP).

Server's IP: 192.168.0.1 (in this example, it will vary based on your server's IP).

USER

User call the program as follows (if Python is used):

```
python packet_sender.py -server 192.168.0.1 -payload "COLOMBIA 2 - MESSI 0"
```

CLIENT PROGRAM (Encoder - packet_sender.py)

Converts the string to HEX:

434f 4c4f 4d42 4941 2032 202d 204d 4553 5349 2030

Encapsulates the data into an IP datagram (packet) and initializes the checksum to 0000.

4500 0028 1c46 4000 4006 **0000** C0A8 0003 C0A8 0001 **434f 4c4f 4d42 4941 2032 202d 204d 4553 5349 2030**

Calculates the checksum:

4500 0028 1c46 4000 4006 **9D35** C0A8 0003 C0A8 0001 **434f 4c4f 4d42 4941 2032 202d 204d 4553 5349 2030**

and sends it to the client.

SERVER PROGRAM (Decoder - packet_receiver.py)

*Receives the data stream and prints to the screen the data received with the following message:
 The data received from 192.168.0.3 is COLOMBIA 2 - MESSI 0
 The data has 160 bits or 20 bytes. Total length of the packet is 40 bytes.
 The verification of the checksum demonstrates that the packet received is correct.*

If the packet received is corrupted, the following message should be printed:

The verification of the checksum demonstrates that the packet received is corrupted. Packet discarded!

More details

Consider the stream used in our scenario above:

4500 0028 1c46 4000 4006 9D35 C0A8 0003 C0A8 0001 434f 4c4f 4d42 4941 2032 202d 204d 4553 5349 2030

- **45** corresponds to the first two fields of the header. 4 means IPv4 and 5 corresponds to the header length. Since header length is described in 4 byte words, the actual header length comes out to be $5 \times 4 = 20$ bytes. [fixed]
- **00** corresponds to TOS or the type of service. This value of TOS indicated normal operation. [fixed]
- **0028** corresponds to the total length field of the IP header. So in this case the total length of the IP packet is 40 bytes. The header is made of 20 bytes, the payload includes 20 bytes. [variable]
- **1c46** corresponds to the identification field. [variable]
- **4000** can be divided into two bytes. These two bytes correspond to the flags and fragment offset of IP header fields. [fixed for this lab]
- **4006** can be divided into '40' and '06'. The first byte '40' corresponds to the TTL field and the byte '06' corresponds to the protocol field of the IP header. '06' indicates that the protocol is TCP. [fixed]
- **9D35** corresponds to the checksum which is set at the source end (which sent the packet) [variable]
- The next set of bytes **C0A8 0003** (192.168.0.3) and **C0A8 0001** (192.168.0.1) correspond to the source IP address and the destination IP address in the IP header. [variable - it will depend on the address of the client and the server]
- 434f 4c4f 4d42 4941 2032 202d 204d 4553 5349 2030 corresponds to the payload (20 bytes in this case). [variable]

For simplicity: All fields marked as [fixed] will be hardcoded. No options are included in the packet.

The payload length + header length must be divisible by 8. If this is not the case, padding must be performed (add zeros to the payload).

How the checksum (16 bits) is calculated - ENCODING:

If we consider the header checksum field to be 0 and calculate the total of all 16 bits, it will be:

$$4500 + 0028 + 1c46 + 4000 + 4006 + 0000 + C0A8 + 0003 + C0A8 + 0001 = 262C8$$

Calculating further, adding carry: $2 + 62C8 = C2CA$

one's complement represented in hex: 9D35 (FFFF-C2CA)

Note that the checksum is calculated using only the header fields. That is, the payload bytes are not included in the calculation.

How the checksum (16 bits) is verified - DECODING:

$4500 + 0028 + 1c46 + 4000 + 4006 + 9D35 + C0A8 + 0003 + C0A8 + 0001 = 2FFFFD$

Calculating further, adding carry: $2 + FFFF = FFFF$

one's complement represented in hex: 0000 (No error)

Additional instructions:

- **You need to implement two programs**, a client and a server and two functions (methods), one to encode (at the client side), one to decode (at the server side).
 - o **The Client:** PacketSender.java or packet_sender.py
 - Reads the data from user (from the keyboard)
 - Sends the encoded stream to the server through socket
 - o **The Server:** PacketReceiver.java or packet_receiver.py
 - Acknowledges the client that the encoded stream has been received
 - Decodes the stream and prints it on the screen

You can code the two programs using Java or Python 3.

Use any necessary built-in functions for the ASCII->HEX conversions.

For more info:

0	4	8	16	19	31
Version	Header Length	Service Type	Total Length		
Identification			Flags	Fragment Offset	
TTL	Protocol		Header Checksum		
Source IP Addr					
Destination IP Addr					
Options				Padding	

Python Sockets:

<https://docs.python.org/2/howto/sockets.html>

Java Sockets:

<https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>

The following will be marking scheme used by the TA:

Component	Grade
1. Client communicates to a server through a Socket	20 points
2 Encoding operation works correctly	10 points*
3. Decoding operation works correctly	10 points*
4. README describes how to run the code	5 points
5. At the client side , data is read from the keyboard	5 points
Total	50 points

* The TA will run five different test cases. Using different payloads.