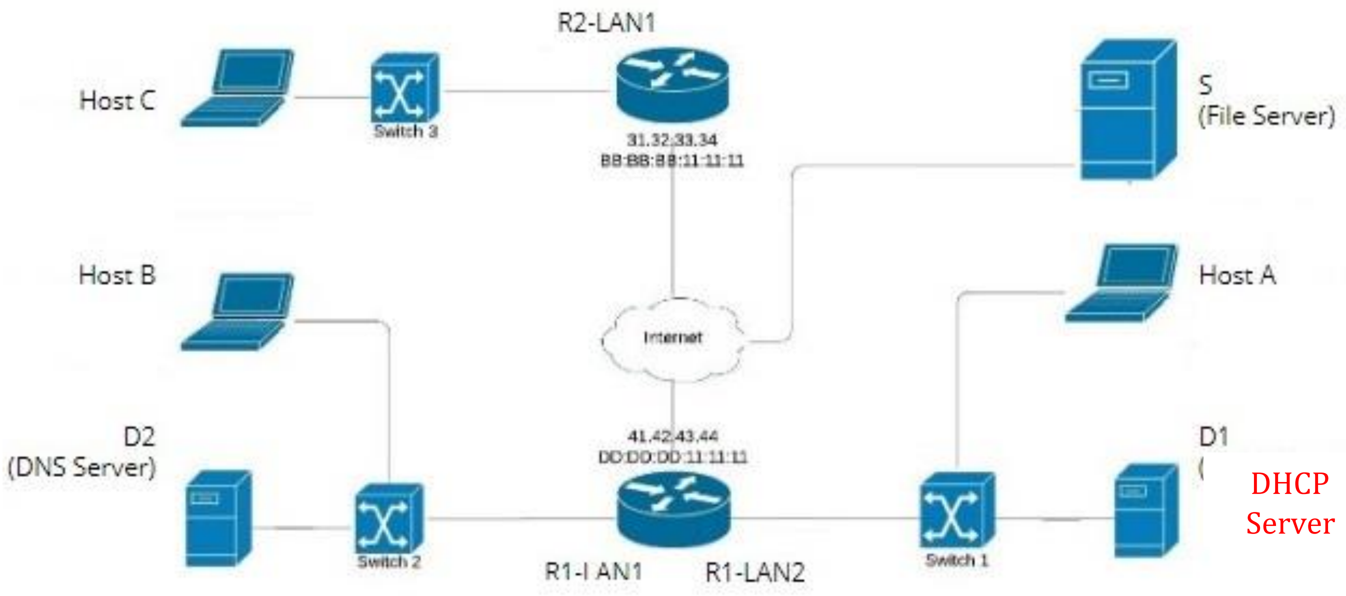
COMS3200/7201 Assignment 2

Due: 1st June ~~25th May~~ 2020, 20:00

100 marks total (20% weight for the final grade (100%))

Part A (30 marks total)

You are asked some questions on packet behaviours in the following network, which depicts three LANs (LAN1, LAN2 and LAN3) and a file server (S) connected to the internet.



Answer each of the following questions in the associated quiz on blackboard, following the specified instructions. Each student will receive different network details.

- All questions will be automatically marked.
- Each question follows on from the previous question – i.e.: for each question you can assume the tasks described in the previous question(s) have been fully completed.
- In any column that asks you for a protocol, the following options will be allowed: HTTP, FTP, Secure Shell (SSH), DNS, DHCP, UDP, TCP, ICMP, SNMP, ARP, OSPF, BGP.
- In any case where multiple protocols are used, pick the highest-layer protocol (e.g., if SSH was used, you would select it over TCP, even though SSH packets use TCP as a transport layer protocol).
- In any column that asks you for MAC address use upper case, two-digit hex format

(e.g., AB:CD:EF:01:10:20 **0**9:08:07:AA:BB:CC)

- In any column that asks you for IP address use the dotted decimal format (e.g. 1.2.3.4 255.255.0.0).
- Assume and only use 255.255.255.255 as the broadcast IP address of all the LANs depicted in this network.
- Assume that the broadcast MAC address of all the LANs is FF:FF:FF:FF:FF:FF.
- Assume all forwarding tables in the switches and routers are up to date.

Answer the following questions based on the network diagram and **customised** questions shown to you on Blackboard. Each student's diagram and question values will be different for each submission.

1. Suppose Host A has just joined the network and does not currently have an IP address. Complete the following table describing the four packets that will be transmitted through the network upon this event (in the order they were sent), assuming that all requests succeed, and that Host A ends up with the IP address [redacted] (i.e. customised information will be given here.)

Table 1.

Protocol	Opcode	Src MAC addr	Dst MAC addr	Src/Sender IPv4 addr	Dst/Target IPv4 addr

2. Now suppose the Host A wants to send a standard DNS query to D2. The Host A has already gotten the IPv4 addresses of D2 and its gateway router from D1. R1's ARP table contains the IP and MAC addresses of all hosts in the network, and all other ARP tables are empty. Complete the following table describing the first four packets that will be transmitted through the network upon this event, assuming that there are no other ongoing communications.

Table 2.

Protocol	Opcode	Src MAC addr	Dst MAC addr	Src/Sender IPv4 addr	Dst/Target IPv4 addr

3. The Host A now wants to establish a HTTP connection with a server (S) (file server). You may assume that the Host A knows S's IP address and that all ARP tables contain all required information to transmit any packet. Complete the following table describing the packets that are sent to transmit Host A's request to initialise the connection, assuming the request is being sent from TCP port [redacted] and the next available port number in R1's NAT table is [redacted]. Give the TCP flags as an 8-bit binary number representing the CWR to FIN flags.

Table 3.

Protocol	TCP Flags	Src TCP port	Dst TCP port	Src/Sender IPv4 addr	Dst/Target IPv4 addr
TCP					
TCP					

4. Suppose in addition to the connection described in (3), the Host B's port [redacted] is also currently connected to S. What are the contents of the NAT table in R1 while these connections remain alive? (if a new port is required, use [redacted]).

Table 4.

LAN IP	LAN Port	WAN IP	WAN Port

5. Suppose [redacted] sent a packet with destination IP address [redacted] and destination MAC address [redacted] What network devices (hosts, servers, routers, switches) in the above diagram would receive this packet?

6. Suppose S sends a packet with the destination IP address [redacted] What network devices (hosts, servers, routers, switches) in the above diagram would receive this packet?

Part B (70 marks)

In this part of the assignment you will be required to implement the network layer for a host running on a virtual IP network. You should be familiar with the typical TCP/IP stack, where Ethernet is used as a link-layer protocol and IPv4 used as a network layer protocol. The protocols we will be using will be ***virtual*** **-** that is, network layer addresses won't actually correspond to physical interfaces. To do this, the network stack will be redefined to that shown in the table below.

TCP/IP Stack	**Protocols**	**Virtual Stack**
		Network Layer
Transport layer	*UDP*	Link layer
Network layer	*IPv4*	
Link Layer	*Ethernet*	

Figure 1.

As described in Figure 1, UDP will be used as your virtual network's link layer, and you will be required to implement a virtualisation of IPv4 on top of this UDP-based link layer. Each UDP socket will correspond to a link layer interface, and localhost UDP port numbers will be used as the link layer addressing system of this virtual network.

In other words, your program is the simulation of a host on a virtual network running on your computer. UDP port numbers will be used as "MAC addresses". Your program should only connect to localhost, i.e. all **actual** packets should transmit across localhost.

By the end of this assignment, your implemented host program should be able to:

- Accept simple user commands through a basic command line interface (CLI)

- Send and receive messages across this virtual network layer
- Handle fragmentation of virtual IP packets

Program Invocation

Your program should be able to be invoked from a UNIX command line as follows. It is expected that any Python programs can run with version 3.6, and any Java programs can run with version 8. The IP-addr and LL-addr parameters correspond to the IPv4 address in CIDR notation (indicating the client's subnet) and link layer address (UDP port number) of your host program respectively.

Python	`python3 NetworkSim.py IP-addr LL-addr`

C/C++	`make`
	`./NetworkSim IP-addr LL-addr`

Java	`make`
	`java NetworkSim IP-addr LL-addr`

IMPORTANT: As the assignment is auto marked, it is important that the filenames and command line syntax exactly matches the specification. Specification adherence is critical for passing.

Your Task

1.1 Command Line Interface (5 marks)

To start with, you should implement a basic command line interface that will allow the user to supply basic information about the network. Your CLI should prompt users with a single > character, followed by a space. For the rest of the assignment we define anything wrapped in square brackets as a parameter or field that needs replacing (NOT as an optional parameter). The CLI should persistently prompt the user for another command until the program is terminated. **For full marks in this section your CLI needs to accommodate the following commands:**

- `gw set [IP-addr]`: set the gateway IP address of the subnet the client is a part of to [IP-addr] (overriding any existing gateway address)
- `gw get`: print the currently stored gateway IP address to stdout, or None if no gateway address has been specified
- `arp set [IP-addr] [LL-addr]`: insert a mapping from [IP-addr] to [LL-addr] in the host's ARP table (overriding any existing entries for [IP-addr])
- `arp get [IP-addr]`: print the currently stored link layer address mapped to [IP-addr] to stdout, or None if no mapping exists
- `exit`: terminate the program

Error-handling of user input is optional - you won't receive any marks for this, but you can choose to implement it provided it doesn't impact on the rest of the assignment.

1.2 Sending Messages (15 marks)

To receive marks in this section, your CLI should be able to handle the following additional command:

- `msg [IP-addr] "[payload]"`: send a virtual IPv4 packet to [IP-addr] with the given payload (which will be supplied as a string)

Any packets sent should have a meaningful identifier and a protocol number of 0. You may assume the that all payloads will be less than or equal to the MTU of the network's links, but the DF flag should still be set to 0. Your program should support any IP address, regardless of whether it is a part of your subnet or not. The subnet mask can be extracted from the client's IP in CIDR notation. When a packet is sent to an IP address outside of the subnet range, it should be sent to the gateway address.

If your program needs to send a packet to the gateway address and no gateway address has been specified, your program should print No gateway found to stdout. If no required address mapping can be found in the host's ARP table, your program should print No ARP entry found to stdout.

1.3 Receiving Messages (15 marks)

In addition to sending packets, your program should be able to receive them from other hosts. When your program receives an IPv4 packet with the protocol indicator set to 0, it should print the payload of that packet to stdout, in the below format ([IP-addr] should be replaced with the sender's IPv4 address and [message] should be replaced with the string encoding of the payload):

```
Message received from [IP-addr]: "[message]"
```

Note that there is only a single space after the colon. When your program receives an IPv4 packet with a non-zero protocol number, it should print the following message to stdout ([proto-num] should be the hexadecimal representation of the protocol formatted as 0x??):

```
Message received from [IP-addr] with protocol [proto-num]
```

You can assume all packets sent to your program are valid IPv4 packets. You should be able to receive messages at any time without blocking the CLI, and any messages should be printed cleanly (without any CLI prompts or responses disrupting the message contents). Remember to backspace the current prompt (the > character) before printing the output.

Hint: The escape character for backspace is "\b". For example, use `print('\b')` to backspace a previously printed character in Python.

1.4 IP Fragmentation (15 marks)

To receive marks in this section your program should be able to handle IP fragmentation. Your CLI should support the following extra commands:

- `mtu set [value]`: set the MTU of the network's links as the specified [value]
- `mtu get`: print the currently stored MTU (the default MTU should be 1500)

Virtual packets that are longer than the specified MTU (or the default MTU if none has been specified) should be fragmented before transmission. The length of a virtual packet is equal to the length of the IPv4 header added to the length of the IPv4 payload (i.e. you don't need to consider the length of the non-virtual headers). You can assume the value of the MTU will never be smaller than 100.

Your program should also be able to receive packets that have been fragmented (and display them as a single message).

1.5 Handling packet loss (20 marks)

To receive marks in this section your program should be able to handle packet losses. Your CLI should support the following extra commands:

- `frto set [value]`: set the FRTO (Fragment Reassembly TimeOut) of the network to the specified [value]
- `frto get`: print the currently stored FRTO (the default FRTO should be 5)

When fragmented packets are received these should be held in memory until all fragments for the given identification field (Id) have arrived, or until FRTO expires. Each Id should have its own FRTO timer, which starts with the arrival of a new packet Id. Once the FRTO expires, all fragments related to the FRTO should be discarded and a packet with a protocol number of 114 and the identification field set to Id should be returned to the sender.

When sending messages, you should keep a collection of fragmented packets for a maximum of 3 x FRTO (i.e. 3 times FRTO time), and resend the fragments related to the Id when you get a packet with the protocol number 114. Changing the FRTO value (using frto set) does not affect any existing timers in operation.

While printing non-zero protocol numbers remains unchanged, when the protocol number is 114, and the fragments being requested for are available (valid and not expired), your program should print the following message to stdout instead. ([fragmentation-Id] should be the hexadecimal representation of the identification field as 0x??)

```
Message-Resend received from [IP-addr] for id [fragmentation-Id]
```

Fragments older than 3 x FRTO are expired and should be deleted. If you get a packet with protocol 114 for an Id that is expired or unknown, these should be ignored, and the following message printed to stdout.

```
Message-Resend received from [IP-addr] for id [fragmentation-Id] BAD
```

Example CLI output

```
python3 NetworkSim.py 192.168.1.1/24 1024
> gw get
None
> gw set 192.168.1.30
> gw get
192.168.1.30
> msg 192.168.1.2 "hello"
No ARP entry found
> arp get 192.168.1.2
None
> arp set 192.168.1.2 2222
> arp get 192.168.1.2
2222
> msg 192.168.1.2 "hello"
> mtu get
1500
> mtu set 1600
> mtu get
1600
Message received from 192.168.1.2: "hello there, thank you for your message"
Message received from 192.168.1.3 with protocol 0x06
> frto get
5
> frto set 3
> frto get
3
Message-Resend received from 192.168.1.6 for id 0x00a4
Message-Resend received from 192.168.1.7 for id 0x00a5 BAD
> exit
```

Packet Formatting

Your IPv4 packets don't need to contain meaningful values for the differentiated services code point (DSCP)/Explicit Congestion Notification (ECN) fields, nor are you required to compute a correct checksum. You can assume packets will never contain any options and subsequently that the Internet Header Length (IHL) will always be 5. The TTL field should contain some meaningful value (i.e. to allow the packet to reach its destination).

Tips for Success

- Revisit the lectures and labs on the internet layer of the TCP/IP stack (in particular the lectures on IPv4).
- Make sure you fully understand the requirements of this assignment before commencing work.
- Ensure you always exercise good thread safety if using threads to implement concurrency.
- Frequently test your code on MOSS.
- Ensure your base functionality is working (such as the CLI) before attempting the more difficult tasks.
- Start early and ask any questions you might have early.

Library Restrictions

- The only communication libraries you may use are standard socket libraries which open UDP sockets.
- You can't use any libraries that aren't considered standard for your language (i.e. if you have to download a library to use it would be considered as non-standard).
- If you are unsure about whether you may use a certain library, please ask the course staff on Piazza.

Submission

Submit all files necessary to run your program. At a minimum, you should submit a file named `NetworkSim.py`, `NetworkSim.c`, `NetworkSim.cpp` or `NetworkSim.java`. If you submit a C/C++ or Java program, you should also submit a makefile to compile your code into a binary named `NetworkSim` or a .class file named `NetworkSim.class`.

IMPORTANT: If you do not adhere to this (e.g. submitting a C/C++/Java program without a Makefile, or a .class file instead of a .java file), *you will receive 0 for this part of the assignment. Do not submit executables.*

Marking

Your code will be automatically marked on a UNIX machine, **so it is essential that your program's behaviour is exactly as specified above.** Your program should complete all tasks within a reasonable time frame (for example a single packet should not take more than one second to construct and send) - there will be timeouts on all tests and it is your responsibility to make sure your code is not overly inefficient. It is expected that you will receive a small sample of tests before the submission deadline.

There are no marks for coding style, however you may lose marks in relevant sections for poor management of concurrency.

Academic Misconduct

Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism.

Late Submission and Extensions

Please view the ECP for policies on late submissions and extensions as well as updates based on COVID-19 situation.

Version History

12/05/2020	Initial release	1.0
14/05/2020	Extesion date and typos	1.5
15/04/2020	Corrected D1 as the DHCP server	1.6