# CS441 G1T2 Project Report

Bryan Lee Min Yuan, Ko Hui Ning, Jolene Loh Kar Inn, Jen Leng

# 1. Quality of Code

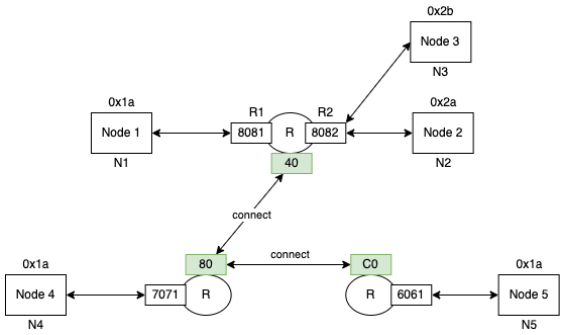
## 1.1 Architecture

The system is split into multiple components to represent the physical, link, network, and app layers. Threads are used to listen for asynchronous events such as user input and data reception.

The observable pattern is used to pass data up from lower layers to higher layers via callback functions, similar to interrupt-driven socket I/O.

With the exception of the user input handler, data is always passed between components as raw bytes to best simulate real networking code. Data classes represent data objects such as Ethernet frames and IP packets, with serialization functions to convert to and from raw bytes.

The demonstration network is configured as:




## 1.2 Public Routing

A distinction is made between private and public networks to simulate more complex routing protocols. The simplest solution matching the project requirements (1-byte IP addresses with nodes ***0x1a***, ***0x1b***, and ***0x2a***) is to assign 2 bits to the public address (a subnet mask of ***11000000***). IP addresses in the range ***0x00 – 0x3f*** are private and routed internally.

This leaves ***0x40***, ***0x80***, and ***0xc0*** in the public IP address space. This is sufficient to demonstrate IP traceback. For simplicity, NAT is not implemented. The fully qualified address is derived from the bitwise addition of public and private addresses.

### 1.2.1 Routing protocol

A simple Bellman-Ford algorithm is used to populate the routing table. For simplicity, routers are identified only by their public IP addresses (first 2 bits in the address).

### 1.2.2 Router network configuration

Routers are connected physically to form networks. To simulate this behavior, every router listens for TCP connections on "network interface" ports – each connection represents a wire on a local area network (LAN). For simplicity, only tree networks are supported.

### 1.2.3 Open category

***ARP Spoofing*** An additional protocol field is reserved on the 5th byte of the Ethernet frame header to support ARP payloads.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   src_mac   |   dst_mac   | prtcl | payload
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Each node maintains an ARP cache. Implementing ARP allows for the simulation of ARP spoofing.

Running the `arp_spoof` command on a node lets the node reply to all ARP requests for the spoofed IP with its own MAC address, thereby poisoning the requestor's ARP cache.

***Denial of Service (DOS)*** An artificial rate limit is added to the router on each message to simulate packet processing time. A fixed-size FIFO queue simulates router buffers. If packets arrive with an interarrival time faster than processing time, the buffers will fill and packets will be dropped when full. This enables DOS attacks on the routers as resources can be overloaded to drop packets and reduce availability.

***IP Traceback*** Supporting public routing enables an implementation of node sampling IP Traceback because packets can now traverse multiple routers. A miscellaneous field is reserved on the 5th byte of the IP packet header to be marked by routers. Given a packet, a router has a 50% chance of marking the packet with its public IP address.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| srcip | dstip | prtcl | size  | misc  | payload
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A DOS attack can be used to generate the high volume of packets required for node sampling

before IP Traceback is used to reconstruct the attacker's route. For example, packet traffic from Node 3 to Node 5 in the demonstration network will reconstruct the path ***0x40→0x80→0xc0***.

# 2. Learning Points

## 2.1 Network Infrastructure

Through this project, our team better understood the encapsulation and decapsulation of packets at the network and link layers. Implementing the data structures for message passing helped us realize the intricacies of designing a packet structure to be both predictable and flexible.

Implementing the network and link layers provided a new perspective on the pros and cons of separating the concerns between the OSI layers e.g. the link layer has no knowledge of the packet IP addresses and broadcasts frames based on their MAC addresses. Conversely, the IP handler has no knowledge of the frame MAC addresses.

## 2.2 MAC Address Resolution for Router

In the initial design of the router controller, it was assumed that the behavior of a router and network node are distinct enough to warrant different implementations. Consequently, the router was implemented with a central message queue for message passing while the network node was implemented with observables. The initial implementation of the Address Resolution Protocol (ARP) is built on the observable system for network nodes. However, it was later noted that the router also needed an implementation of ARP. This simple fact had slipped our mind and in implementing this project, we were reminded that routers also require ARP. To reduce code duplication, the router was refactored to use observables instead and share its ARP implementation with network nodes.

## 2.3 Network Attacks

***IP Spoofing*** Our team was under the impression that IP spoofing would allow packets to be sent back to the attacker. Thus in our initial implementation of IP spoofing, a spoofing node would reply to a ping request to avoid detection. However, after further research, we realize that IP spoofing does not prevent an impersonated victim within a LAN from receiving the original request due to a shared medium. Furthermore, the impersonated network device will receive replies to the spoofed address in a mechanism known as backscatter. This can be used to detect large-scale spoofing attacks.

***Denial of Service (DOS)*** In implementing DOS, we needed to rate-limit each router to magnify the effects of processing time for packet routing. On the first simulated DOS attack, the router's queue is congested with packets as expected and is unable to process benign packets. However, this behavior was unpredictable and the result of unsafe threading. To improve the simulation of our router, we added a thread-safe FIFO buffer to manage packet congestion by consistently dropping packets when the buffer is full. This simulates typical router behavior when incoming traffic arrives at a greater rate than outgoing traffic.

***IP Traceback*** The implementation of IP traceback better allowed us to appreciate how the probabilistic marking of packets allows for path reconstruction. With our own implementation, we are able to experiment with different numbers of routers and packet quantities to determine how they affect the accuracy of path reconstruction. We noted that with more routers, a higher number of packets would be necessary for accurate path reconstruction.

***Sniffing Attack*** To implement sniffing attacks, we first had to figure out how to make a node promiscuous. Our implementation uses a boolean value to indicate whether a node is promiscuous or not. This allows us to easily toggle sniffing on and off. If a node is promiscuous, it would not drop any packets, including those not intended for it. By observing the traffic when sniffing is turned on, we learned more about the flow of the packets, which aided us in further debugging.