NWEN 302 Lab Two

Kandice McLean
300138073

My implementation of the ARP component successfully meets the following requirements:

- The ARP table is filled within 30s (dependant on map size)
- ARP requests are only sent if the entry does not exist in the address table
- ARP packet header has standard header values as defined by *https://en.wikipedia.org/wiki/Address_Resolution_Protocol*
- The ARP table is updated whenever the nodes receive a message from another node that is not yet in the address table, whether it is reply or request
- ARP entries time out
- ARP table is of limited size

Creating the ARP packets and packet header was very easy, because it is similar to the last lab except this time I defined the packet myself to fit with the cnet definitions of addresses.

I didn't have any problems with implementing the fixed size as I designed the program to deal with that situation from the start. I used LRU queueing system to update the table.

Implementing the time-out function was the most difficult aspect of this lab. I first tried to implement this using a counter, but it resulted in uneven results across each node. After checking the API I found that each node contains a time variable in nodeinfo, so I used this, and created an EXPIRY definition in arp.h. If the current time of the node minus the time stored in the struct is more than/equal to the EXPIRY then the address is considered old and is deleted. The best way I could think of doing this was to check them each time a message is received, so the call to clear the table is in ethernet.c.

I defined two different data structures in this program. I made the structures as simple as possible, to reduce the possibility of serious errors in the program.

The map structure has three variables, a MAC address, ip address and a timestamp.

The arphdr structure is defined like a standard arp header, and the variables use data types that are the same size as the variable they are representing, such as using an unsigned short for HTYPE which is 2 bytes (unsigned as HTYPE will never be a negative number). Keeping the size of the arp header structure matched to the defined sizes in an arp header made casting packets into an arp header very simple.

I have allowed ip.c to drop packets it cannot find the M.A.C address for. This is how ARP is implemented on Linux systems in real life so I decided it was best to make the simulation match this.

The arp_accept function boolean logic could be simplified to make it cleaner to read and more efficient. Other than that I don't think this implementation needs any big improvements as everything works as it should in the cnet simulation.