**Question 1: Explain what is ARP (Address Resolution Protocol)**

Address Resolution Protocol or ARP is used to map the network device’s physical address with its logical address, most prominently used in IPv4 to map the MAC address to its IP address for an Ethernet device. This protocol is used by the IP layer to figure out which physical device it has to send the packets based on the available IP address. In networking terms the logical address or IP address can be re-assigned to a different device as network configurations change so the ARP is used as the bottom most protocol to get the physical address of the device and send the destination packet to the corresponding device. This protocol is simple yet very handy; the ARP packet contains the following fields:

* ARP header
  + Hardware type like Ethernet.
  + Protocol type like IP.
  + Hardware length which is 6 in case of MAC address.
  + Protocol length which is 4 in case of IPv4.
  + ARP operation, which is a request or a reply.
* 6-byte sender hardware address.
* 4-byte sender IP address.
* 6-byte target hardware address.
* 4-byte target IP address.

There are 2 types of operations that are possible:

* ARP REQUEST:
  + The sender device knows the target IP address but not the target physical address.
  + In this case the sender device creates an ARP packet with operation type as ARP REQUEST and target hardware address field with all 0s.
  + The sender device broadcasts this ARP packet in the LAN it is configured for.
  + The target IP address based device sends back an ARP REPLY with the target hardware address.
* ARP REPLY:
  + The original Target device receives the ARP requests and replies back with ARP operation as ARP REPLY.
  + The sender hardware address and sender IP address fields in the ARP packet will be the current device details.
  + The target hardware address and target IP address fields in the ARP packet will be the original request sender device details.
  + The packet is sent directly to the original request sender device.
  + The device receiving the ARP REPLY will use the sender hardware address from the ARP packet send the TCP or UDP packet data over the LAN.

**Question 2: Implement ARP in C language on linux machine using gcc. Send the C code back and explain the result after you run the program on your computer. Also, explain what you would expect the program to do.**

I have used the Ethernet hardware device to illustrate the ARP procedures. The ARP REQUEST and ARP REPLY procedures are demonstrated on a linux environment. Following are the details of implementation:

Files: arp.c , arp.h, arputil.c, arputil.h and constants.h.

* *Note: the interface device is hard coded to “eth0”, it can be changed in the code in constants.h file.*

Build environment: Ubuntu/GCC

Build command: from the source directory run the command “gcc arp.c arputil.c –o arp”. This will create the executable **arp**.

Execution details: This executable takes two types of argument options: -m (mode of operation) and –i (Target IP address). We will have to create two separate processes to illustrate the working of ARP; the following are the details on different modes of operations:

* ARP REQUEST/REPLY RECEIVE – BLOCKING MODE:
  + From command line > **./arp –m 2**
  + Starts the ARP to receive any incoming ARP REQUESTS or ARP REPLIES
  + The incoming ARP REQUEST is handled by replying back with our device’s hardware address.
  + The incoming ARP REPLY is handled by printing the target MAC address to std output.
  + This is a blocking mode where the Process listens to all incoming ARP packets and handles them accordingly.
  + Example output:
    - \*\*\*\*Processing ARP RESPONSE\*\*\*\*
    - SENDER MAC address: 00:50:56:EE:49:DD
    - TARGET MAC address: 00:0C:29:8E:7B:6E
* ARP REQUEST SEND – NON BLOCKING MODE:
  + From command line > **./arp –m 1 –i 192.168.29.2**
  + This command will start a new process to send ARP REQUEST to the target IP address 192.168.29.2.
  + This is a non-blocking mode where the Process sends the ARP REQUEST and exits.
  + The response is handled in the blocking mode process described above.
  + Example output:
    - **\*\*\*\*Processing ARP REQUEST SEND\*\*\*\***
    - **SENDER MAC address: 00:0C:29:8E:7B:6E**
    - **TARGET MAC address: 00:00:00:00:00:00**
* In addition to this execution we can use the tcpdump or wireshark to track the ARP REQUEST and ARP RESPONSE packets by capturing the log over “eth0”.

Implementation details: There are two main functions that handle the blocking mode and no-blocking mode.

* processARPRequest():
  + This function creates an AF\_PACKET based socket.
  + Retrieves the MAC address and IP address using ioctl SIOCGIFHWADDR and SIOCGIFADDR respectively.
  + Creates the Ethernet frame in the following format defined by using a local structure:
    - Ethernet header
    - ARP Packet
    - Padding data
  + Sends the packet buffer and exits using sendto().
* processARPReply():
  + This function creates an AF\_PACKET based socket.
  + Retrieves the device MAC address using ioctl SIOCGIFHWADDR.
  + Using the recvfrom() function blocks and waits for incoming ARP packets on Ethernet.
  + If the incoming packet is ARP REQUEST, reply back with the device hardware address retrieved.
  + If the incoming packet is ARP REPLY, display the MAC address received.

Future enhancements:

* Create a timer based dynamic ARP cache using hash map or structure arrays to locally store the MAC addresses once received to avoid re-sending the ARP REQUESTS frequently. This will avoid some extent of network flooding with ARP packet data in the LAN.
* Use the pthread libraries to create two separate working threads with the same process to handle ARP REQUEST SEND and ARP REQUEST/REPLY RECEIVE. This will avoid the use any system call based IPC to share the ARP cache data among processes.