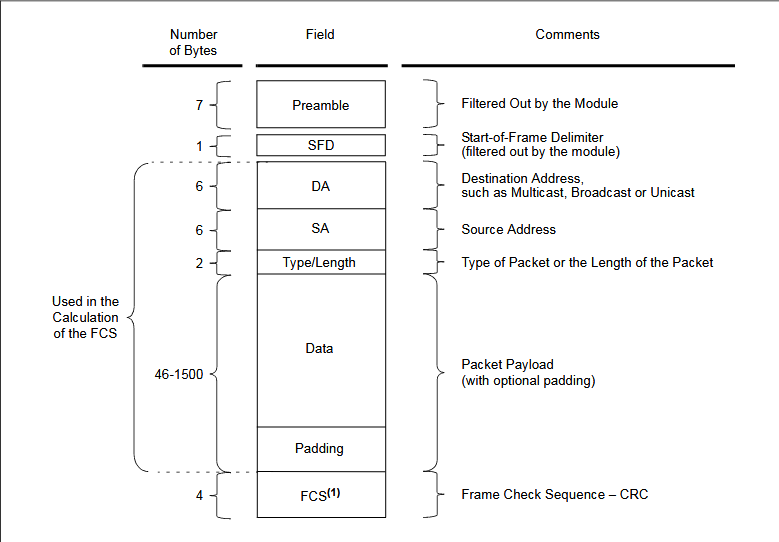
s

A buffer of size 500 bytes is created in the main webserver file.

**Ethernet Packet Format:**



**Function Make\_eth\_ip\_new: (not implemented)**

This function assigns Destination MAC address and source MAC address to the ethernet packet header.

The MAC address of the destination determines broadcast, unicast or multicast type of packet distribution.

///////////////////////////////////////////////////////////////////////////////////////////////////

The destination address field is a 6-byte field filled withthe MAC address of the device that the packet is directed to. If the Least Significant bit in the first byte of the MAC address is set, the address is a Multicast destination. For example, 01-00-00-00-F0-00 and 33-45-67-89-AB-CD are Multicast addresses, while 00-00-00-00-F0-00 and 32-45-67-89-AB-CD are not.

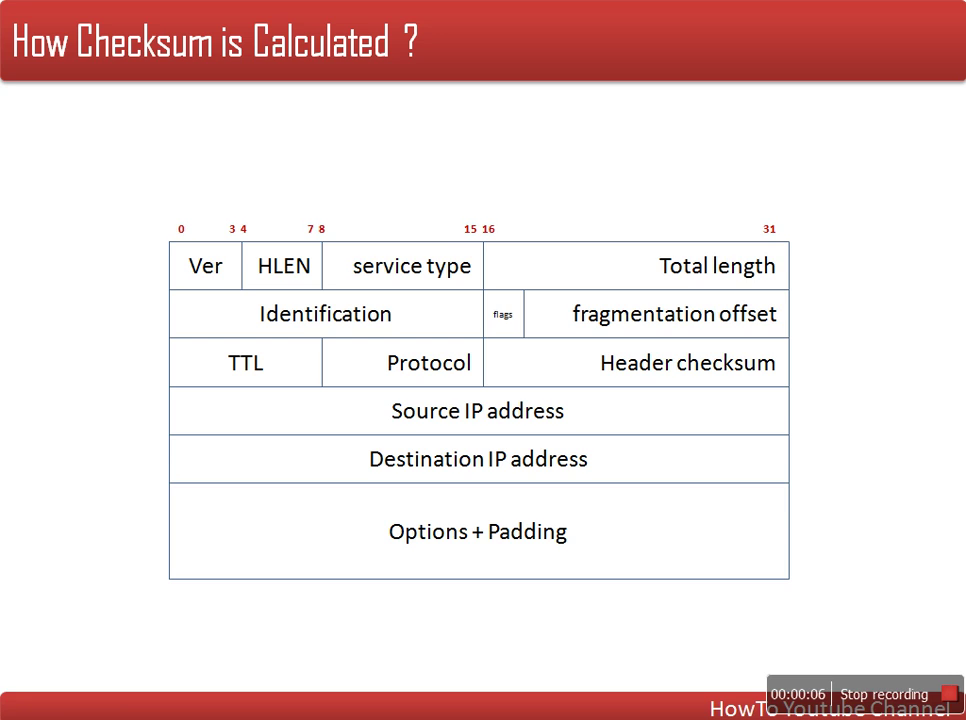
//////////////////////////////////////////////////////////////////////////////////////////////////////

Also values are set in the position pointed by ETH\_TYPE\_H\_P and ETH\_TYPE\_L\_P in the Ethernet type field of two bytes.

**Function Make\_eth**

1. The function copies the source MAC address to destination MAC address field of the buffer.
2. Assigns the userdefined MAC address the the source MAC address field of the buffer.

**IP Packet Structure Format:**



**Function Make\_ip**

1. Copies IP source field to IP destination field.
2. Assigns IP source field to user assigned ip address.
3. Fills IP header checksum.

**Function Fill\_ip\_hdr\_checksum (needs more description)**

1. Clears the 2 byte checksum.
2. Sets other buffer values???
3. Calculates checksum.
4. Assigns the calcualted checksum to the IP\_CHECKSUM\_P field.

**Function Checksum**

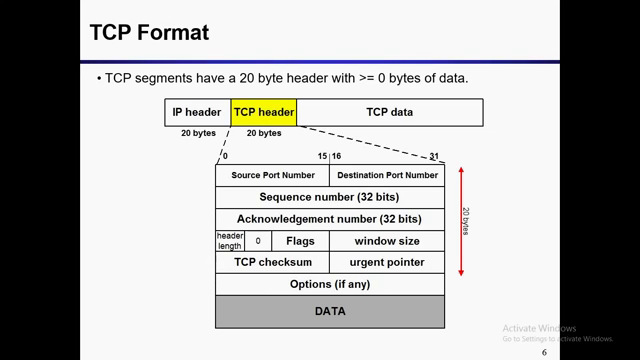
Why ANDing to 0xFFFF???

1. Divides the total header into length of 16 bits and adds them.
2. If there is a byte left then it is shifted left 8 bits and then added to the sum.
3. The bits from the MSBof the sum are added to the sum itself until the sum is 16 bits long.
4. Returns the complement of the sum.

**Function Init\_ip\_arp\_udp\_tcp**

1. Initializes the mymac, myip and the wwwp variables.

**TCP Packet Structure Format**



**Function Make\_ip\_tcp\_new**

1. Sets TCP header details into the buffer.
2. Sets source and destination IP.
3. Generates and sets the checksum field of the TCP packet.

**Function Eth\_type\_is\_arp\_and\_my\_ip**

1. Checks if argument buffer packet is ARP and the dest IP matches the stored IP or not.
2. Returns 0 if not and returns 1 if true.

**Function Eth\_type\_is\_ip\_and\_my\_ip**

1. Checks if the arguent buffer packet is IP and the destination IP matches the stored IP or not.
2. Also checks if the IP header is equal to 20 bytes or not.
3. Returns 0 if false and 1 if true.

**Function Init\_len\_info**

1. Retrives the total data length from the buffer.
2. Subtracts the length of ip header length from the result.
3. Retrives the total header length from the data.
4. Subtracts the total header length from the result.
5. If result is <=0 then the result is assigned to 0 else the result remains as it is.

**Function Get\_tcp\_data\_pointer**

1. If tcp data length of the buffer is greater than 0 then return the sum of TCP\_SRC\_PORT\_H\_P and info\_hdr\_len.
2. Else return 0.

**Function Tcp\_get\_dlength (Not Implemented)**

1. Returns the length of the data of tcp packet.
2. Algorithm same as that of the **Init\_len\_info** function.

**Function Fill\_tcp\_data**

1. Fills data passed in the character pointer s to the buffer pointed by the pointer pos.
2. Returns the current position of data in the data field.

**Function Fill\_tcp\_data\_p**

1. Same as **Fill\_tcp\_data** function.

**Function Make\_tcphead**

1. Sets tcp head buffer fields with ip, port, acknowledge num, sequence num, header length and so on.

**Function Make\_udp\_reply\_from\_request (Not implemented)**

Since the server uses only the tcp protocol so no study is done on this protocol.

**Steps for Communication Establishment**

The while loop basically checks the incoming packets and acts accordingly. There is no step wise communication rule. Inside while loop the following checks are done.

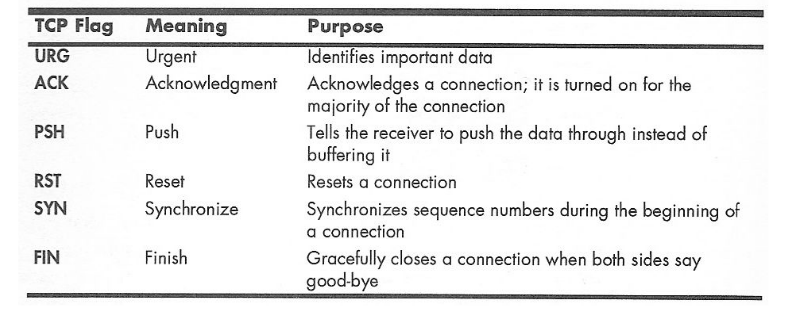
1. After physical connection the server will wait for packet from the client (indicated in while loop of the **LanTask** function).
2. If the packet length is 0 ie no packet is received the function will return.
3. If packet is received then the packet is checked if it is of type ARP or not and the ip matches or not.
4. If the packet if ARP type then ARP answer from the request is made and sent.
5. If the packet is not of ARP type then the while loop is continued.
6. If the received packet if of type IP and its destination IP matches our IP address then the test falls through, else the loop is continued again.
7. If the packet is for this node, ie the destination IP of the packet matches our IP then if the IP is of ICMP type then the ICMP request is sent. If the IP packet is of type TCP then a sync acknowledgement packet is sent to signal packet successfully received condition. (Pinging an ip address).

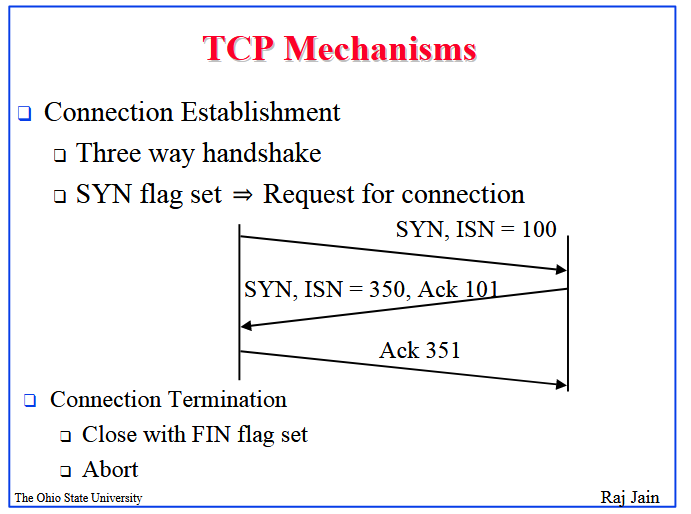
Types of acknowledgement requires more study (ack with data, ack with no data...)

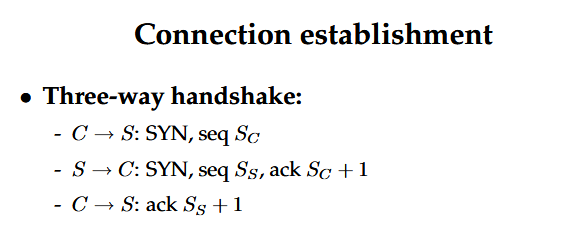
TCP\_FLAGS\_SYN\_V use?????????????????

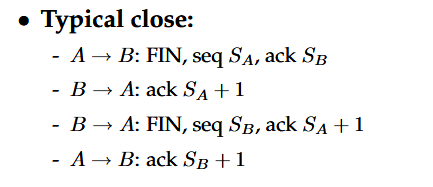
study of TCP flags needed.

1. If received IP packet is of type TCP and tcp destination port matches mywwwport then if the tcp flags are set to TCP\_FLAGS\_SYN\_V then a tcp synack\_from\_syn reply is sent. But if the flags are ser to TCP\_FLAGS\_ACK\_V then if there are no data in the packet ack packet with no data is sent. (how?)

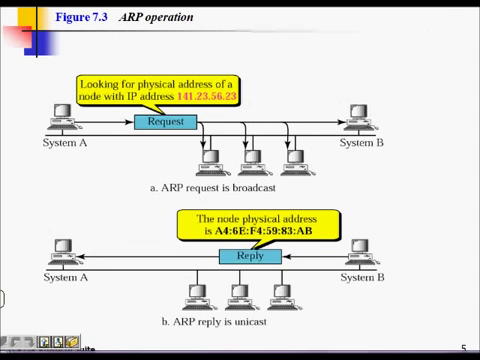
****

****

****

****

**Why ARP?**



In order to send the ethernet frames to the desired destination node the mac address of that node should be known. In this scenario, only the ip address of the receiver node is known. But the ethernet frame requires the physical address of the destination. So the destination IP needs to be resolved into the physical address of the receiving node. This resolution is done with the Address Resolution Protocol (ARP).

Following this protocol, the sender first broadcasts an ARP request. The request is checked by each node connected to the network. But only the actual receiver replies with its MAC address on the packet. Hence the sender can know the MAC address of the destination node with ARP.

In the case of our server, it aspects a broadcast ARP request from the network (maybe router or any other node). If the broadcast packet contains its ip address then it replies with a unicast packet that contains its MAC address. Hence the other node can then send the ethernet frame to this particular MAC address.

**Discussions:**

The code inside ip\_arp\_udp\_tcp.c file contains lots of codes that are not implemented. Since the server uses tcp protocol for communication, related to udp protocol are not necessary. This is because udp and tcp falls on the same layer or are alternate to eachother. But if we want to upgrade the server to support udp then codes in the webserver.c file needs to be updated to handle udp packets.