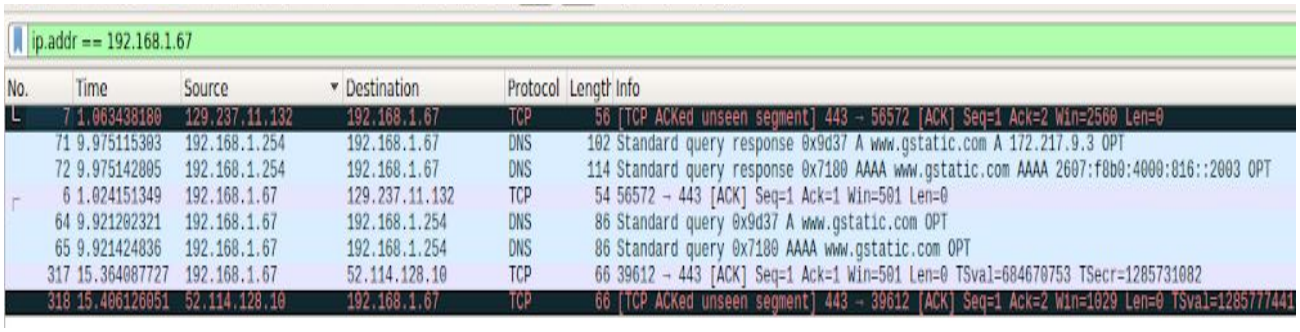
Jace Kline 2881618

EECS 565 Project 2 Report

Part 1:

Per the instructions posted on the website, I added the IP filter and began capturing packets. While capture was occurring, I navigated to my web browser and typed "www.google.com" into the URL bar and then hit ENTER. After navigating to Google, I went back to Wireshark and stopped the packet capture. The following picture shows the results of this sequence.



In this capture, my laptop's IP address was 192.168.1.67 and another website's host server was at address 129.237.11.132 (most likely Blackboard). Please disregard this address. The sequence of packet transfers to connect to Google occurred as follows: First, my laptop sent a DNS request through my network's router (IP = 192.168.1.254) in order to resolve the IP address for hostname "www.google.com". My router took my request and routed it to an external DNS server (external to the LAN). The external DNS server then received the query packets and generated a response of 2 UDP packets, which were sent back to my router. Because I requested a static site, the DNS response actually resolved the 'google.com' address to a static domain address 'www.gstatic.com', which serves to efficiently serve Google's static pages. My router then proceeded to direct the packets back to my laptop's IP address. Following this, my

laptop sent another DNS request, this time to resolve the 'www.gstatic.com' domain. This domain was ultimately resolved to IP address 52.114.128.18, which is evidenced by a TCP packet being sent from my device to port 443 at that address. In response, this address sent a packet to port 443 on my laptop. Since port 443 is used for HTTPS, we can see that the packets being exchanged were using HTTPS protocol (sent over TCP).

Part 2:

Deciphering LAN Devices:

In this first section, I would like to share my findings regarding the types of devices on my LAN. I would like to point out that, strangely, I only observed one DHCP packet transferred during the capture period, so I was ultimately unable to utilize the information provided by the DHCP packets to derive device information. Furthermore, the mDNS protocol packets captured did not provide meaningful device names (i.e. XXX.local) about the devices on the network. However, what I was able to do was to gather the MAC (Hardware) addresses of each of the Network Interface Cards (NICs) of the local devices to determine their manufacturer. Although this isn't information isn't very specific, it does provide some insights into the devices on the network. The following table illustrates my findings.

IP Address	MAC Address	NIC Manufacturer	Device Guess
192.168.1.67	e0:94:67:c4:3e:39	Intel	My laptop
192.168.1.162	88:78:73:62:c4:a6	Intel	Friend's laptop
192.168.1.158	7c:67:a2:39:cc:e5	Intel	Friend's laptop
192.168.1.148	fc:a6:67:b5:50:c2	Amazon	Amazon Fire TV

192.168.1.172	8c:dc:d4:42:b8:bd	Hewlett Packard	Friend's laptop
?	50:bc:96:a3:13:fd	Apple	Girlfriend's Mac
192.168.1.254	f8:f5:32:b2:11:b0	Arris	Router

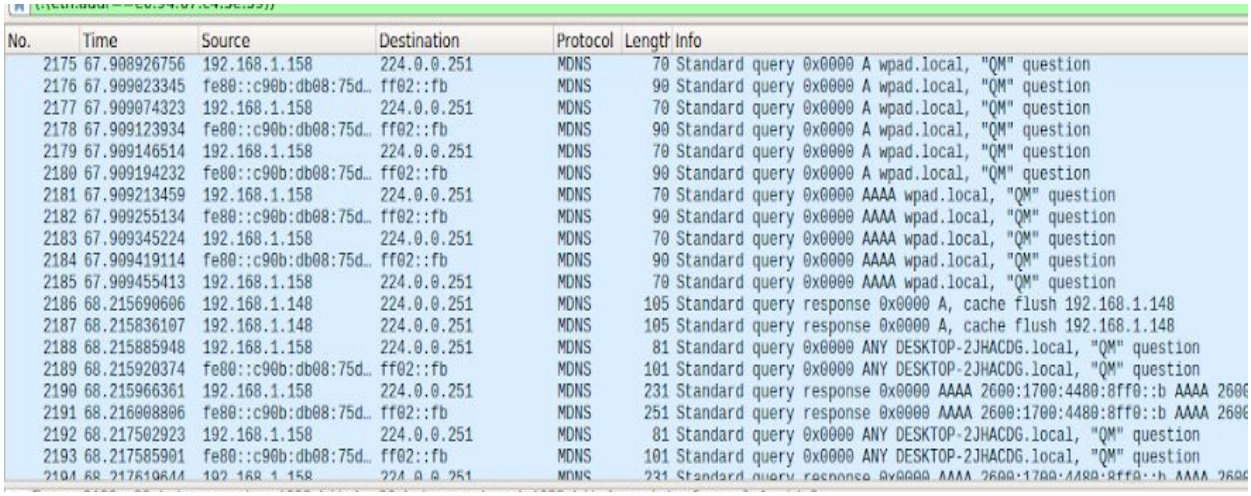
The reason I could decipher the manufacturers of the NICs from the MAC addresses is because the MAC address ranges of the first 4 (of 6) "chunks" can be resolved to a specific manufacturer. In fact, Wireshark automatically offers this conversion suggestion, illustrated by the following example: 50:bc:96:a3:13:fd => Apple_a3:13:fd.

Deciphering Unfamiliar Addresses:

When examining the packets, I kept coming across the following unconventional receiving addresses: 'ff02::fb', '224.0.0.251', and 'Broadcast'. In this section, I will discuss what I have found on the origin of each of these IP addresses.

As it turns out, the addresses 'ff02::fb' and '224.0.0.251' actually refer to the same network entity, where the former is the IPv6 address and the latter is the IPv4 address. These addresses actually point to the router's mDNS multicast interface for the LAN. All mDNS-capable host devices on the network will listen to this address by default. This interface is used for DNS-like resolution on a LAN scope. Suppose host 1 and host 2 are on the same LAN. Through mDNS, both host 1 and host 2 receive a mDNS name that will persist despite possibly changing dynamic IP addresses for these hosts. Let "host1.local" and "host2.local" be the mDNS names of the two hosts, respectively. If host 1 wants to send something to host 2, host 1 will send a service resolution request to the mDNS IP address (listened to by all mDNS capable devices) asking "who is host2.local?". Since host 2 is mDNS-configured and listening to the shared interface,

host 2 will respond via a UDP packet over the shared interface with its IP address, port number, and other identifying information. Not only will host 1 be able to store this information, but all other mDNS devices on the LAN will be able to update their resolution information associated with ‘host2.local’. In essence, mDNS is just a shared LAN interface for intra-LAN addressing resolution. In many ways, it resembles what occurs in normal DNS. The following screenshot shows an mDNS sequence.



No.	Time	Source	Destination	Protocol	Length	Info
2175	67.908926756	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM" question
2176	67.909023345	fe80::c90b:db08:75d…	ff02::fb	MDNS	90	Standard query 0x0000 A wpad.local, "QM" question
2177	67.909074323	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM" question
2178	67.909123934	fe80::c90b:db08:75d…	ff02::fb	MDNS	90	Standard query 0x0000 A wpad.local, "QM" question
2179	67.909146514	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM" question
2180	67.909194232	fe80::c90b:db08:75d…	ff02::fb	MDNS	90	Standard query 0x0000 A wpad.local, "QM" question
2181	67.909213459	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 AAAA wpad.local, "QM" question
2182	67.909255134	fe80::c90b:db08:75d…	ff02::fb	MDNS	90	Standard query 0x0000 AAAA wpad.local, "QM" question
2183	67.909345224	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 AAAA wpad.local, "QM" question
2184	67.909419114	fe80::c90b:db08:75d…	ff02::fb	MDNS	90	Standard query 0x0000 AAAA wpad.local, "QM" question
2185	67.909455413	192.168.1.158	224.0.0.251	MDNS	70	Standard query 0x0000 AAAA wpad.local, "QM" question
2186	68.215690606	192.168.1.148	224.0.0.251	MDNS	105	Standard query response 0x0000 A, cache flush 192.168.1.148
2187	68.215836107	192.168.1.148	224.0.0.251	MDNS	105	Standard query response 0x0000 A, cache flush 192.168.1.148
2188	68.215885948	192.168.1.158	224.0.0.251	MDNS	81	Standard query 0x0000 ANY DESKTOP-2JHACDG.local, "QM" question
2189	68.215920374	fe80::c90b:db08:75d…	ff02::fb	MDNS	101	Standard query 0x0000 ANY DESKTOP-2JHACDG.local, "QM" question
2190	68.215966361	192.168.1.158	224.0.0.251	MDNS	231	Standard query response 0x0000 AAAA 2600:1700:4480:8ff0::b AAAA 2600
2191	68.216008806	fe80::c90b:db08:75d…	ff02::fb	MDNS	251	Standard query response 0x0000 AAAA 2600:1700:4480:8ff0::b AAAA 2600
2192	68.217502923	192.168.1.158	224.0.0.251	MDNS	81	Standard query 0x0000 ANY DESKTOP-2JHACDG.local, "QM" question
2193	68.217585901	fe80::c90b:db08:75d…	ff02::fb	MDNS	101	Standard query 0x0000 ANY DESKTOP-2JHACDG.local, "QM" question
2194	68.217619644	192.168.1.158	224.0.0.251	MDNS	231	Standard query response 0x0000 AAAA 2600:1700:4480:8ff0::b AAAA 2600

Next, I would like to discuss the ‘Broadcast’ address. After research, I discovered that the broadcast address represents all devices on the network. Hence, if a packet or frame is sent via the broadcast address, all devices (not a subset) on the network will be transmitted that data. The broadcast address of a particular network typically occupies the last possible value that could be filled in the Node section of the IP address, prefixed by the network address. For example, if the network address is 192.168.1.0 and the subnet mask is 255.255.255.0, then the broadcast address would be 192.168.1.255. From my observations, broadcasting was used heavily with the ARP and 0x7373 protocols.

Jace Kline 2881618

Discussing Various Protocols:

While examining the various packets captured by Wireshark, I was exposed to many unfamiliar protocols. A few of these that I would like to discuss are ARP, ICMPv6, and IGMPv2.

The ARP protocol stands for “Address Resolution Protocol”. Its function is to resolve the MAC address of a given device from the device’s IP address. In the capture, it was interesting to observe how the host devices on the LAN would send out an ARP broadcast message that would ask something like “*Who has 192.168.1.XX? Tell 192.168.1.YY.*” Essentially, the hosts communicate information regarding the mappings between IP addresses and MAC addresses of other hosts on the LAN, and then the receivers store this information for later.

The ICMPv6 protocol is an important protocol within the IPv6 protocol suite that essentially is in charge of “meta” messages such as error messages and diagnostic functions across IP networks. In my Wireshark capture, the types of messages sent using this protocol consisted of “Neighbor Solicitation”, “Neighbor Advertisement”, and “Router Advertisement”. All of these messages fall under the scope of ICMPv6’s ‘Informational’ messages as opposed to ‘Error’ messages.

The IGMPv2 protocol is used for group multicasting and declaring a host to be a member of a particular group. Interestingly, the mDNS IP address was used as the destination address in each of these “Membership Report” messages sent via IGMPv2. In each case the host was declaring membership of itself to the mDNS multicast group using IGMPv2 as the message packet type.