CSC 361 Lab Session 9

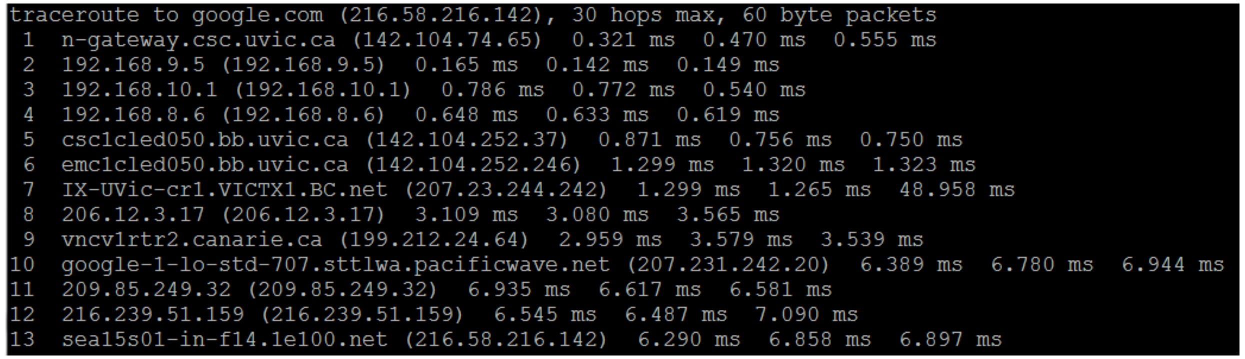
Dawood Sajjadi
Maryam Tanha

Dept. of Computer Science
University of Victoria

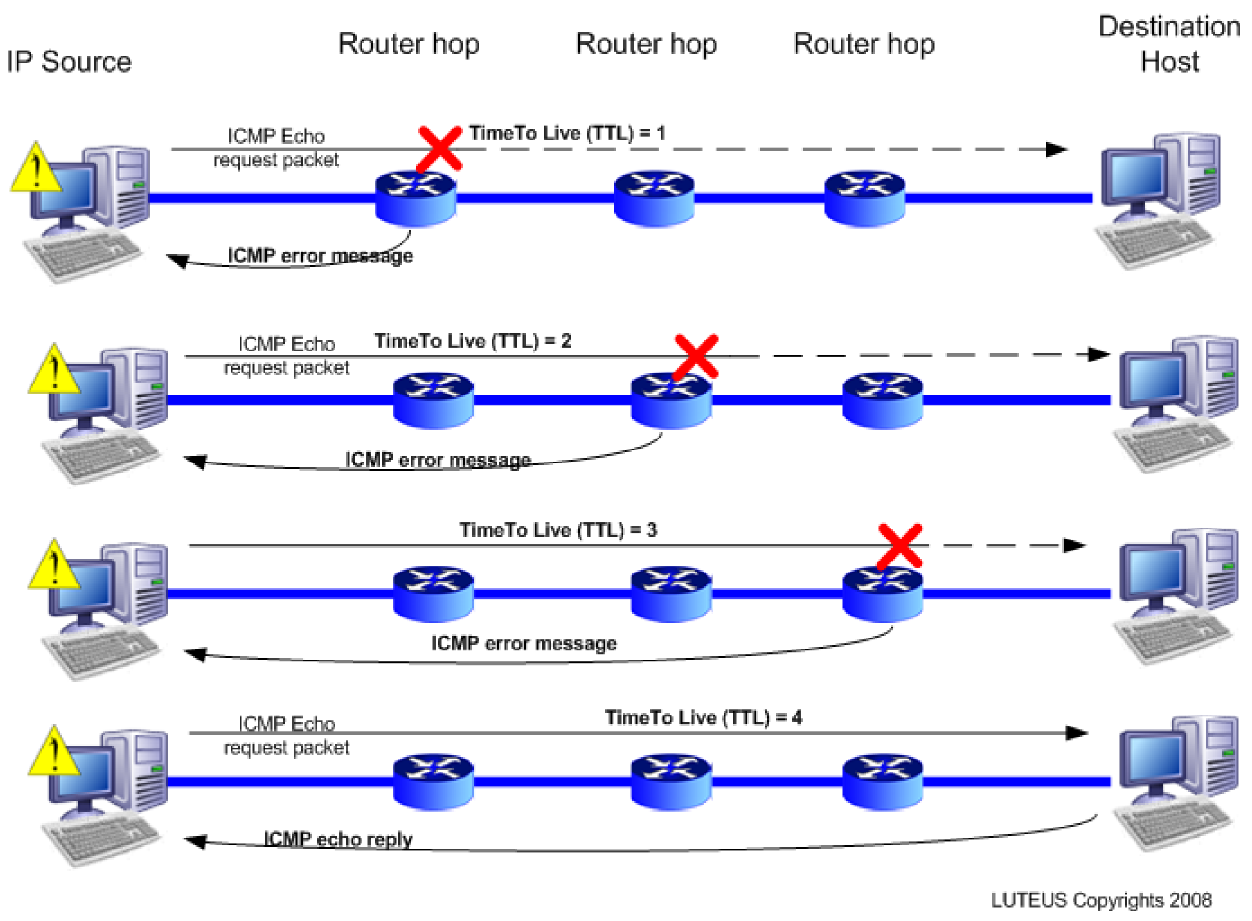
Summer 2016

Assignment 3 (recap)

PC$ traceroute google.com



```
traceroute to google.com (216.58.216.142), 30 hops max, 60 byte packets
 1  n-gateway.csc.uvic.ca (142.104.74.65)  0.321 ms  0.470 ms  0.555 ms
 2  192.168.9.5 (192.168.9.5)  0.165 ms  0.142 ms  0.149 ms
 3  192.168.10.1 (192.168.10.1)  0.786 ms  0.772 ms  0.540 ms
 4  192.168.8.6 (192.168.8.6)  0.648 ms  0.633 ms  0.619 ms
 5  csc1cled050.bb.uvic.ca (142.104.252.37)  0.871 ms  0.756 ms  0.750 ms
 6  emc1cled050.bb.uvic.ca (142.104.252.246)  1.299 ms  1.320 ms  1.323 ms
 7  IX-UVic-cr1.VICTX1.BC.net (207.23.244.242)  1.299 ms  1.265 ms  48.958 ms
 8  206.12.3.17 (206.12.3.17)  3.109 ms  3.080 ms  3.565 ms
 9  vncv1rtr2.canarie.ca (199.212.24.64)  2.959 ms  3.579 ms  3.539 ms
10  google-1-lo-std-707.sttlwa.pacificwave.net (207.231.242.20)  6.389 ms  6.780 ms  6.944 ms
11  209.85.249.32 (209.85.249.32)  6.935 ms  6.617 ms  6.581 ms
12  216.239.51.159 (216.239.51.159)  6.545 ms  6.487 ms  7.090 ms
13  sea15s01-in-f14.1e100.net (216.58.216.142)  6.290 ms  6.858 ms  6.897 ms
```



Traceroute sends Datagram Packets to the destination.
It uses Time-To-Live (TTL) field of the IP header.
If the TTL reaches **0**, the router returns an ICMP message (type 11 **ICMP TTL-exceeded**) to the sender.

The sender that runs **traceroute** can learn the identities of the intermediate routers (between itself and the destination) by looking at the source IP addresses in the datagrams containing the **ICMP TTL-exceeded** messages

Assignment 3 (sample output)

```
The IP address of the source node: 192.168.1.12
The IP address of ultimate destination node: 10.216.216.2
The IP addresses of the intermediate destination nodes:
      router 1: 24.218.01.102,
      router 2: 24.221.10.103,
      router 3: 10.215.118.1.

The values in the protocol field of IP headers:
      1: ICMP
      17: UDP


The number of fragments created from the original datagram is: 3
The offset of the last fragment is: 3680

The avg RRT between 192.168.1.12 and 24.218.01.102 is: 50 ms, the s.d. is: 5 ms
The avg RRT between 192.168.1.12 and 24.221.10.103 is: 100 ms, the s.d. is: 6 ms
The avg RRT between 192.168.1.12 and 10.215.118.1 is: 150 ms, the s.d. is: 5 ms
The avg RRT between 192.168.1.12 and 10.216.216.2 is: 200 ms, the s.d. is: 15 ms
```

If there are multiple fragmented datagrams, you need to output the above information for each datagram.

Assignment 3 (sample output)

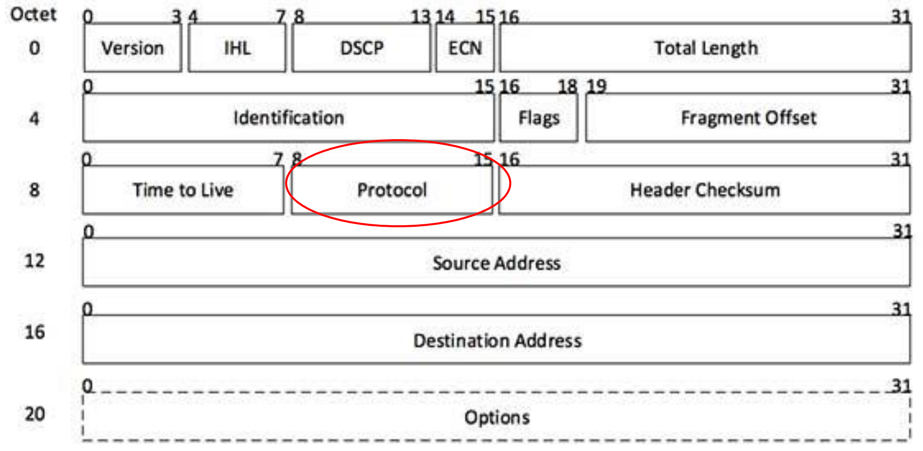
TTL	Average RRT in trace 1	Average RRT in trace 2	Average RRT in trace 3	Average RRT in trace 4	Average RRT in trace 5
1	0.5	0.7	0.8	0.7	0.9
2	0.9	1	1.2	1.2	1.3
3	1.5	1.5	1.5	1.5	2.5
4	2.5	2	2	2.5	3
5	3	2.5	3	3.5	3.5
6	5	4	5	4.5	4

Some intermediate router may only send back one "ICMP TTL exceeded" message for multiple fragments of the same datagram. In this case, please use this ICMP message to calculate RTT for all fragments. For example, Assume that the source sends Frag 1, Frag2, Frag 3 (of the same datagram, ID: 3000). The timestamps for Frag1, Frag2, Frag3 are t_1, t_2, t_3, respectively. Later, the source receives one "ICMP TTL exceeded" message (ID: 3000). The timestamp is T. Then the RRTs are calculated as: $T - t_1$, $T - t_2$, $T - t_3$.

Assignment 3

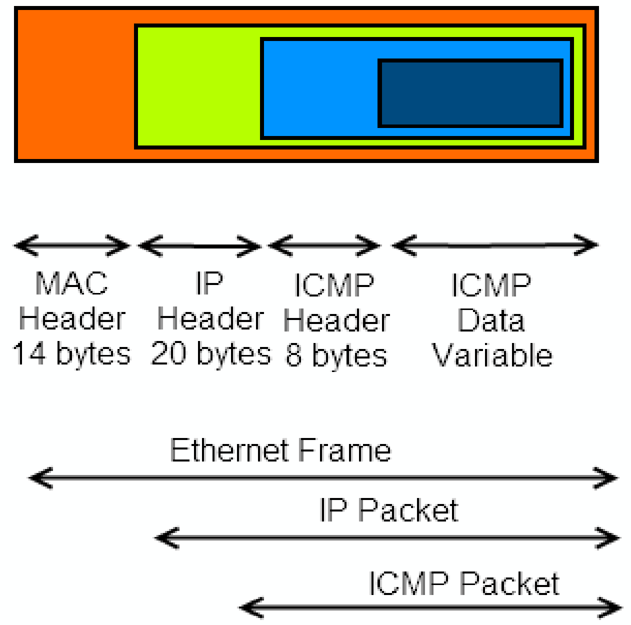
Decimal	Hex	Keyword	Protocol
0	0x00	HOPOPT	IPv6 Hop-by-Hop Option
1	0x01	ICMP	Internet Control Message Protocol
2	0x02	IGMP	Internet Group Management Protocol
3	0x03	GGP	Gateway-to-Gateway Protocol
4	0x04	IP-in-IP	IP in IP (encapsulation)
5	0x05	ST	Internet Stream Protocol
6	0x06	TCP	Transmission Control Protocol
7	0x07	CBT	Core-based trees
8	0x08	EGP	Exterior Gateway Protocol
9	0x09	IGP	Interior Gateway Protocol (any private interior gateway (used by Cisco for their IGRP))
10	0x0A	BBN-RCC-MON	BBN RCC Monitoring
11	0x0B	NVP-II	Network Voice Protocol
12	0x0C	PUP	Xerox PUP
13	0x0D	ARGUS	ARGUS
14	0x0E	EMCON	EMCON
15	0x0F	XNET	Cross Net Debugger
16	0x10	CHAOS	Chaos
17	0x11	UDP	User Datagram Protocol
18	0x12	MUX	Multiplexing
19	0x13	DCN-MEAS	DCN Measurement Subsystems
20	0x14	HMP	Host Monitoring Protocol

IPv4 Header



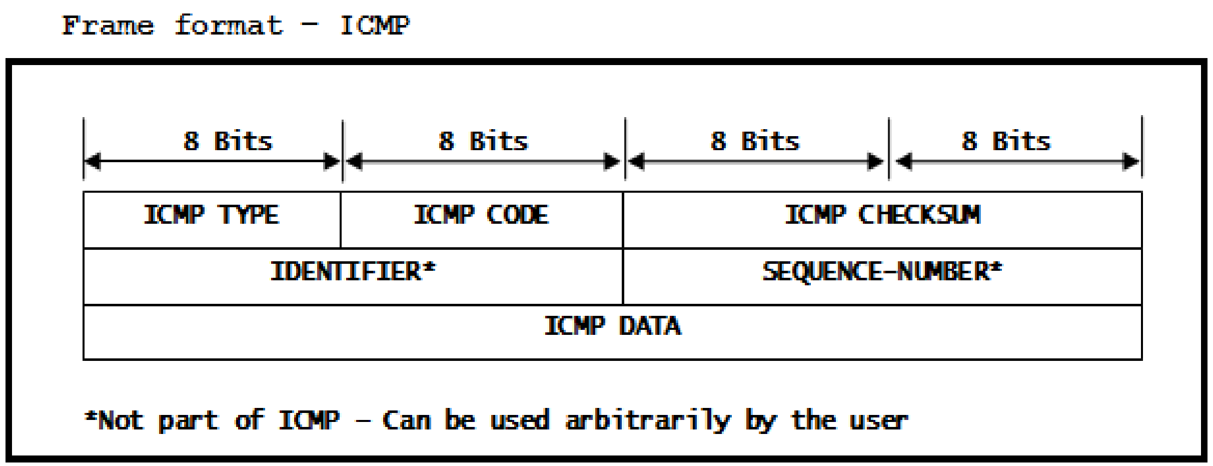
[Image: IP Header]

ICMP Packet Overview



Assignment 3

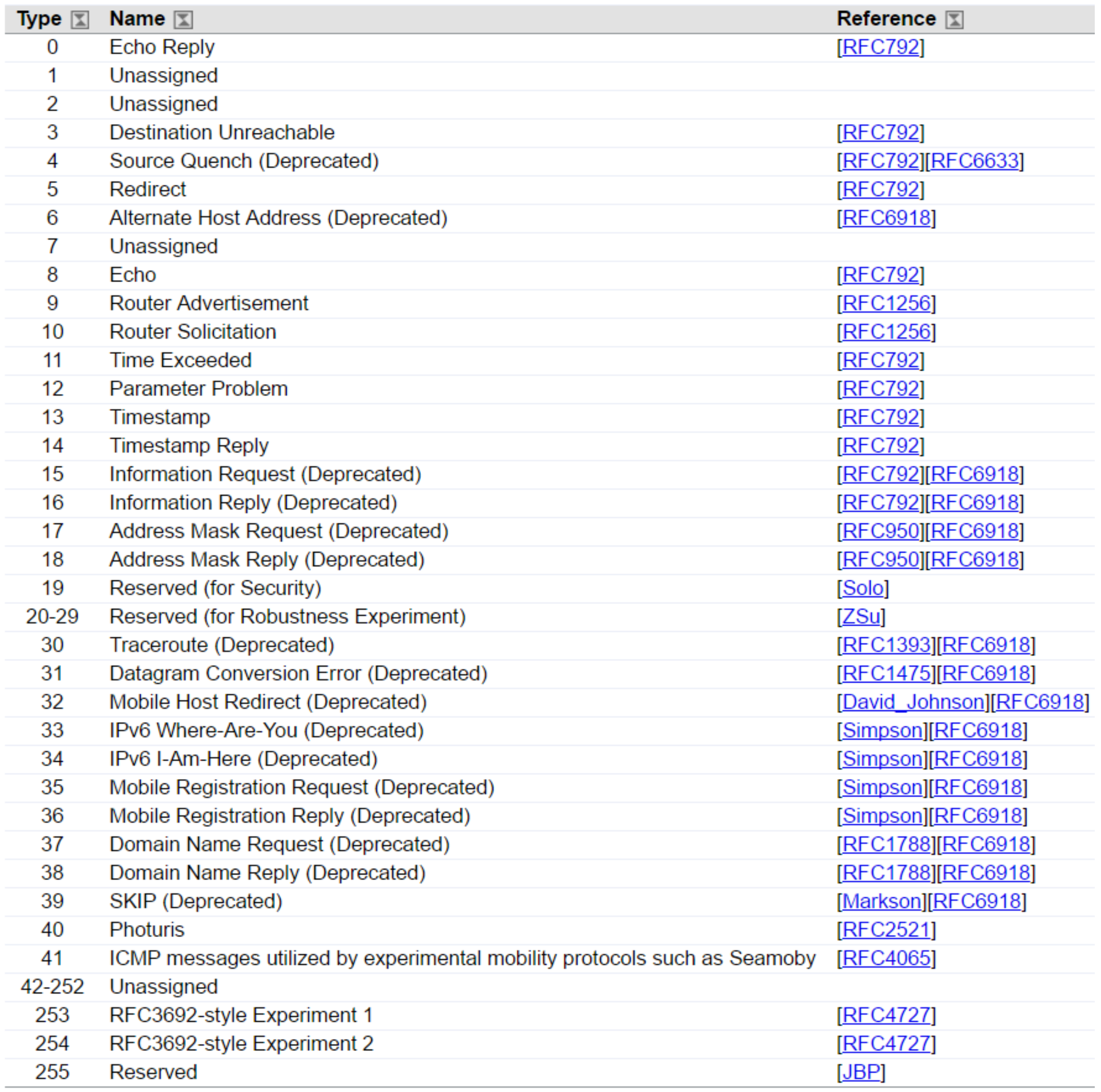
ICMP Header



Identifier and **Sequence Number** can be used by the client to match the timestamp reply with the timestamp request.

In Linux, **Source port number** included in the original UDP can be used to match against the ICMP error message.

ICMP Types



Type	Name	Reference
0	Echo Reply	[RFC792]
1	Unassigned	
2	Unassigned	
3	Destination Unreachable	[RFC792]
4	Source Quench (Deprecated)	[RFC792][RFC6633]
5	Redirect	[RFC792]
6	Alternate Host Address (Deprecated)	[RFC6918]
7	Unassigned	
8	Echo	[RFC792]
9	Router Advertisement	[RFC1256]
10	Router Solicitation	[RFC1256]
11	Time Exceeded	[RFC792]
12	Parameter Problem	[RFC792]
13	Timestamp	[RFC792]
14	Timestamp Reply	[RFC792]
15	Information Request (Deprecated)	[RFC792][RFC6918]
16	Information Reply (Deprecated)	[RFC792][RFC6918]
17	Address Mask Request (Deprecated)	[RFC950][RFC6918]
18	Address Mask Reply (Deprecated)	[RFC950][RFC6918]
19	Reserved (for Security)	[Solo]
20-29	Reserved (for Robustness Experiment)	[ZSu]
30	Traceroute (Deprecated)	[RFC1393][RFC6918]
31	Datagram Conversion Error (Deprecated)	[RFC1475][RFC6918]
32	Mobile Host Redirect (Deprecated)	[David_Johnson][RFC6918]
33	IPv6 Where-Are-You (Deprecated)	[Simpson][RFC6918]
34	IPv6 I-Am-Here (Deprecated)	[Simpson][RFC6918]
35	Mobile Registration Request (Deprecated)	[Simpson][RFC6918]
36	Mobile Registration Reply (Deprecated)	[Simpson][RFC6918]
37	Domain Name Request (Deprecated)	[RFC1788][RFC6918]
38	Domain Name Reply (Deprecated)	[RFC1788][RFC6918]
39	SKIP (Deprecated)	[Markson][RFC6918]
40	Photuris	[RFC2521]
41	ICMP messages utilized by experimental mobility protocols such as Seamoby	[RFC4065]
42-252	Unassigned	
253	RFC3692-style Experiment 1	[RFC4727]
254	RFC3692-style Experiment 2	[RFC4727]
255	Reserved	[JBP]

Assignment 3

Main Body Sample

```
int main (int argc, char *argv[]) {
    pcap_t *pcap;
    char err_buff [PCAP_ERRBUF_SIZE];
    const unsigned char *packet;
    struct pcap_pkthdr header;
    struct router routers [MAX_HOPS];
    struct outgoing times [MAX_HOPS];
    int protocols [MAX_STR_LEN];
    // A single argument is expected
    if (argc != 2) {
        printf("Usage: <trace_file>\n");
        exit(1);
    }
    // Initialize pcap
    pcap = pcap_open_offline(argv[1], err_buff);
    // Check the file
    if (pcap == NULL) {
        printf("Error: could not read file: %s\n", err_buff);
        exit(1);
    }
    // Loop through packets
    while ((packet = pcap_next (pcap, &header)) != NULL ) {
        // Extract data from packet
        if (dump_packet(packet, header.caplen, routers, protocols, header.ts, times)) {
            break;
        }
    }
    print_results (routers, protocols, times);
    return 0;
}
```

```
int dump_packet (const unsigned char *packet, unsigned int capture_len,
    struct router routers[MAX_HOPS], int protocols[MAX_STR_LEN],
    struct timeval ts, struct outgoing times[MAX_HOPS]) {

    struct ip *ip;
    unsigned int IP_header_len;

    // Skip Ethernet header
    packet += sizeof (struct ether_header);
    capture_len -= sizeof (struct ether_header);

    // Copy data to IP struct
    ip = (struct ip*) packet;
    IP_header_len = ip->ip_hl * 4;

    // Skip IP header
    packet += IP_header_len;
    capture_len -= IP_header_len;

    // Analyze contents of packet
    if (analyze_packet (ip, packet, routers, protocols, ts, times)) {
        return 1;
    }
    return 0;
}
```

Assignment 3

```
int analyze_packet (struct ip *ip, const unsigned char *packet,
    struct router routers[MAX_HOPS], int protocols[MAX_STR_LEN],
    struct timeval ts, struct outgoing times[MAX_HOPS]) {

    struct icmphdr *icmp;
    struct udphdr *udp;
    uint16_t port;
    unsigned short temp, id, offset;
    int mf;

    // Get ID of packet
    temp = ip->ip_id;
    id = (temp>>8) | (temp<<8);
    // Packet is ICMP
    if (ip->ip_p == 1) {
        icmp = (struct icmphdr*) packet;
        // Add protocol
        add_protocol(protocols, 1);
        // Packet timed out
        if (icmp->code == 11) {
            // Add intermediate router to list
            add_to_list(routers, packet, ip, protocols, ts, times);
        // First packet sent in trace route
        } else if ( (icmp->code == 8) && (ip->ip_ttl == 1) && (first_id == 0) ){
            // Set source and ultimate destination addresses
            ult_dst = ip->ip_dst.s_addr;
            src = ip->ip_src.s_addr;
            // Record time packet was sent
            add_time(ip, id, ts, times);
            // Set ID of first packet
            temp = ip->ip_id;
            first_id = (temp>>8) | (temp<<8);
            // Get MF flag value
            mf = (ip->ip_off & 0x0020) >> 5;
```

```
// If MF is set, increment total number of fragments
            if (mf == 1) {
                fragments++;
            }
        // Packet is a fragment of the first packet sent in traceroute
        } else if ( (first_id == id) ) {
            // Get MF flag value
            mf = (ip->ip_off & 0x0020) >> 5;
            // Increment total number of fragments
            fragments++;
            // Get offset value
            temp = ip->ip_off & 0xFF1F;
            offset = (temp>>8) | (temp<<8);
            // Calculate value of offset if there are no more fragments
            if (mf == 0) {
                last_frag = offset * 8;
            }
            // Record time packet was sent
            add_time(ip, id, ts, times);
        // Packet is outgoing, record time sent
        } else if (icmp->code == 8) {
            // Record time packet was sent
            add_time(ip, id, ts, times);
        // Packet signifies that the destination has been reached
        } else if ( (icmp->code == 0) || (icmp->code == 3) ) {
            add_to_list(routers, packet, ip, protocols, ts, times);
            list_index--;
            return 1;
        }
```