Practicals on Computer Networks and Distributed Systems

Estimating the Internet path to destination host: The traceroute utility

Tracing the packets to an Internet host

The ICMP protocol messages can be used for varied purposes not limited to estimating IP connectivity, as we have done in the previous practical. For example, to trace the network pathway taken by IP packets at a given moment and towards a given destination host, ICMP message Time Exceeded in Transit offers valuable help. Originally, the Time Exceeded message was included in ICMP as a means for a router to inform a sender that some IP packet would have visited too many routers on its trip towards its destination, which opened the possibility that an internetwork loop might exist at the time. Loops prevent the packets that enter them from being delivered at their destinations altogether. Consequently, having the ability to detect them is crucial.

The IP packet field that helps detecting when a packet might have entered a loop is named Time To Live, or TTL. Every time an IP packet visits a given router, its value is checked for zero, otherwise it is decremented by one (At least by one). If the packet's TTL is zero, the packet is dropped and a ICMP Time Exceeded in Transit notification will be sent back to the sender.

The TTL field, as the packet progresses toward its destination will eventually reach the value 0 if a sufficient number of routers are visited by the packet on its trip to the destination host. IP prescribes that whenever an IP packet's TTL reaches value 0 at a router, the router should discard (Drop) the packet altogether, and it should as well send back an *ICMP Time Exceeded in Transit Message* to the sender of the IP packet. TTL was conceived as an upper bound to the number of routers an IP packet should cross on its trip to its destination, thus, in case of a network design or network management error caused a network loop, packets would have a deterministic guarantee not to loop in there forever. A looping packet would waste a variety of network and compute resources, and consequently it has to be avoided at all cost.

The TTL field of an IP packet can be used for estimating the sequence of routers that make up the *current* path from a source host to a given destination host. The traceroute utility aims to estimate the sequence of routers that comprise the network pathway to a destination host at the specific time the test is being carried out (Network pathways can change at any time, due to the varying degrees of Internet congestion and the availability of links.

The traceroute utility, when executed at a host, first sends an IP packet containing a null-payload UDP message from UDP port 33434 (By default) with TTL=1. This IP packet can be conveniently be named an IP **Probe** Packet. The probe, upon being received by the first IP router on the pathway that leads to some destination host chosen by the user, will get TTL=0 and the router will react by sending back an ICMP *Time Exceeded in Transit Message (TEM)* to the source host. The TEM message stands for the router if adequately interpreted by the source host after it receives it. Now, the source host may determine the identity of the first router since the received IP packet carries the IP address of the sending network interface of the identified router. Now, traceroute will send a probe IP packet containing TTL=2, which will elicit an ICMP *TEM* from the second router on the pathway. This process of sending probe packets and expecting an ICMP *TEM* to each continues until it is the destination host which sends the ICMP TEM, at which time, traceroute will finish.

Exercise 1. Estimating the network path to a host.

The following example illustrates the use of Linux traceroute to www.princeton.edu; option -n means that traceroute should not print host names but their IP addresses, in numeric format:

```
$ traceroute -n www.princeton.edu
traceroute to www.princeton.edu (104.18.5.101), 30 hops max, 60 byte packets
1  192.168.3.1  0.566 ms  0.704 ms  0.867 ms
2  192.168.1.1  3.176 ms  3.226 ms  3.358 ms
3  81.46.41.134  5.521 ms  7.297 ms  7.409 m
4   81.41.239.193  14.044 ms  12.819 ms  14.160 ms
5   * * *
6   * * 81.46.3.233  16.955 ms
7   216.184.113.248  16.383 ms 216.184.113.250  14.157ms 216.184.113.248 13.749 ms
8   81.173.106.39  14.331 ms 176.52.248.251  14.055 ms  12.374 ms
9   81.173.106.39  12.381 ms 188.114.108.7  10.903 ms
10  104.18.5.101  11.154 ms 172.70.58.2  11.315 ms  12.217 ms
```

a. Explain the meaning of line number 3 in the traceroute listing obtained above: The IP address and the ensuing three times given in milliseconds.

b. Test traceroute against this host: www.columbia.edu

c. Now, test the same host with ping and check out the results are coherent

Exercise 2. Search the RFC to the IP protocol and check the four fields essential to traceroute: Source IP, Destination IP, Protocol (Mux key) and TTL

a. What's the width of each of the mentioned IP packet fields?

   Source IP:

   Destination IP:

   Protocol:

   TTL:

b. Explain the semantics of each field

   Source IP:

   Destination IP:

   Protocol:

   TTL:

Exercise 3. Will all routers send back ICMP TEM (ICMP TTL Exceeded in Transit message)?

In the traceroute listing from the example in Exercise 1, some of the intervening network nodes will block the outbound ICMP traffic for security reasons, that is the reason why you will see some routers represented by asterisks (* * *). Explain the meaning of line number 6 (In italics and bold) on the listing in that example.

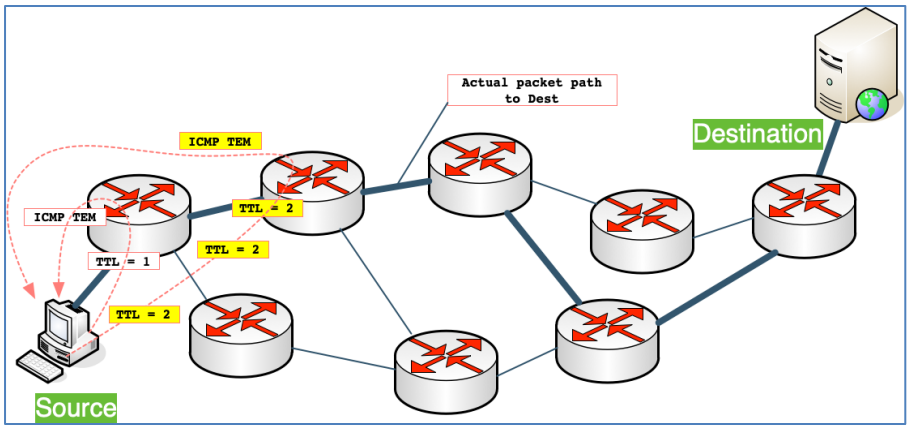

Figure 2.1. Source host sending two probe packets having TTL=1 and TTL=2 and receiving the respective TEM messages

Exercise 4. The internal operation of the Linux traceroute is based on the UDP and ICMP protocols

The Linux traceroute command line utility estimates the sequence of router that comprise the path to a destination host by sending UDP probes encapsulated into IP packets that have each a TTL value starting at 1 and typically finishing at 30. Every time an IP packet visits a router, it is decremented by at least 1 and, when the TTL reaches zero, the router should drop it and it must send back an ICMP TTL Exceeded in Transit message to the originating host. Note that the traceroute utility doesn't call the UDP service interface (Datagram sockets); instead, traceroute handcrafts each UDP datagram it sends and has each of them encapsulated into an IP packet that carries a given TTL value.

By convention, the port number included in the UDP datagram, must be a high one. The purpose of this artifact consists of allowing traceroute to control de TTL of each sent IP packet, which is the essence of traceroute functioning. In summary: traceroute creates a PF_INET/SOCK_RAW/protocol=1 raw socket and sets the raw socket's option IP_HDRINCL so that traceroute can set the TTL of each sent IP packet. traceroute at the source host collects the responses received from each identified router by using another PF_INET/SOCK_RAW/protocol=1 raw socket. The latter socket doesn't set option IP_HDRINCL, since it is not used for sending.

The protocol stack in Fig.3 might result misleading at first sight. Note that the UDP box means only that traceroute is complying with the specifications for the UDP protocol, that is, it is implementing that protocol's peer-to-peer interface. However, traceroute is not invoking UDP's service interface, which means that traceroute doesn't create a datagram socket for sending the necessary datagrams to the destination host.

When the IP packet makes it to the end host, the latter will complain with an ICMP message of type ICMP_DEST_UNREACH and code ICMP_PORT_ UNREACH. Observe the example that follows: In terminal 1 traceroute is executed against www.bbva.es and in terminal 2 at the same host, a tcpdump trace capturing the probes sent with growing TTL and the ICMP TTL Exceeded in Transit sent back from the routers at which TTL became 0.

```
$ traceroute www.bbva.es (Term 1)
```



```
traceroute to www.bbva.es (104.83.210.61), 30 hops max, 60 byte packets
 1  DD-WRT-NetworkingLab (192.168.1.1)  0.414 ms  0.644 ms  0.750 ms
 2  n101001.unileon.es (193.146.101.1)  3.874 ms  3.849 ms  3.857 ms
 3  n111003.unileon.es (193.146.111.3)  3.879 ms  3.829 ms  3.836 ms
 4  185.179.107.225 (185.179.107.225)  5.087 ms  5.149 ms  5.228 ms
 5  REDCAYLE.ETHTRUNK0-85.ciemat.rt2.mad.red.rediris.es (130.206.201.121)  8.074 ms  9.333 ms  9.399 ms
 6  ciemat.rt2.ae1-0.telmad.rt4.mad.red.rediris.es (130.206.245.2)  24.502 ms  19.537 ms  20.246 ms
 7  rediris-ias-geant-gw.mar.fr.geant.net (83.97.88.129)  21.432 ms  21.163 ms  21.144 ms
 8  ae8.mx1.gen.ch.geant.net (62.40.98.73)  48.979 ms  49.148 ms  49.046 ms
 9  ae2.mx1.mad.es.geant.net (62.40.98.66)  46.995 ms  47.078 ms  47.303 ms^C
```

$ tcpdump -i eno1 -n -vvv -S host 104.83.210.61 and udp or icmp[0]=11 *(Term 2)*

```
tcpdump: listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
13:32:00.488112 IP (tos 0x0, ttl 1, id 36858, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.45325 > 104.83.210.61.33434: [bad udp cksum 0xfcca -> 0xda5f!] UDP, length 32
13:32:00.488167 IP (tos 0x0, ttl 1, id 36859, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.40056 > 104.83.210.61.33435: [bad udp cksum 0xfcca -> 0xeef3!] UDP, length 32
13:32:00.488207 IP (tos 0x0, ttl 1, id 36860, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.43156 > 104.83.210.61.33436: [bad udp cksum 0xfcca -> 0xe2d6!] UDP, length 32
13:32:00.488245 IP (tos 0x0, ttl 2, id 36861, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.48830 > 104.83.210.61.33437: [bad udp cksum 0xfcca -> 0xccab!] UDP, length 32
13:32:00.488284 IP (tos 0x0, ttl 2, id 36862, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.44845 > 104.83.210.61.33438: [bad udp cksum 0xfcca -> 0xdc3b!] UDP, length 32
13:32:00.488321 IP (tos 0x0, ttl 2, id 36863, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.57335 > 104.83.210.61.33439: [bad udp cksum 0xfcca -> 0xab70!] UDP, length 32
13:32:00.488358 IP (tos 0x0, ttl 3, id 36864, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.35183 > 104.83.210.61.33440: [bad udp cksum 0xfcca -> 0x01f8!] UDP, length 32
13:32:00.488420 IP (tos 0x0, ttl 3, id 36865, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.39151 > 104.83.210.61.33441: [bad udp cksum 0xfcca -> 0xf276!] UDP, length 32
13:32:00.488459 IP (tos 0x0, ttl 3, id 36866, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.44732 > 104.83.210.61.33442: [bad udp cksum 0xfcca -> 0xdca8!] UDP, length 32
13:32:00.488504 IP (tos 0x0, ttl 4, id 36867, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.57690 > 104.83.210.61.33443: [bad udp cksum 0xfcca -> 0xaa09!] UDP, length 32
13:32:00.488512 IP (tos 0xc0, ttl 64, id 10315, offset 0, flags [none], proto ICMP (1), length 88)
    192.168.1.1 > 192.168.1.88: ICMP time exceeded in-transit, length 68
      IP (tos 0x0, ttl 1, id 36858, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.45325 > 104.83.210.61.33434: [udp sum ok] UDP, length 32
13:32:00.488542 IP (tos 0x0, ttl 4, id 36868, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.38623 > 104.83.210.61.33444: [bad udp cksum 0xfcca -> 0xf483!] UDP, length 32
13:32:00.488588 IP (tos 0x0, ttl 4, id 36869, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.55229 > 104.83.210.61.33445: [bad udp cksum 0xfcca -> 0xb3a4!] UDP, length 32
13:32:00.488626 IP (tos 0x0, ttl 5, id 36870, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.50747 > 104.83.210.61.33446: [bad udp cksum 0xfcca -> 0xc525!] UDP, length 32
13:32:00.488664 IP (tos 0x0, ttl 5, id 36871, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.55561 > 104.83.210.61.33447: [bad udp cksum 0xfcca -> 0xb256!] UDP, length 32
13:32:00.488707 IP (tos 0x0, ttl 5, id 36872, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.56919 > 104.83.210.61.33448: [bad udp cksum 0xfcca -> 0xad07!] UDP, length 32
13:32:00.488729 IP (tos 0x0, ttl 6, id 36873, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.42173 > 104.83.210.61.33449: [bad udp cksum 0xfcca -> 0xe6a0!] UDP, length 32
13:32:00.488803 IP (tos 0xc0, ttl 64, id 10316, offset 0, flags [none], proto ICMP (1), length 88)
    192.168.1.1 > 192.168.1.88: ICMP time exceeded in-transit, length 68
```

Figure 3.1. tcpdump trace of traceroute to host `104.83.210.61`

a. Run traceroute to www.princeton.edu and capture the UDP probes sent and the received ICMP TTL Exceeded in Transit message. Use tcpdump for the capture and provide an explanation of what you observe.

b. Observe the protocol stack to an execution of traceroute in fig. 3.1. Obtain the protocol stack active at a router standing on the path from sender to receiver when it receives an IP packet having TTL=1 and thereby reacting by sending back an ICMP TEM.

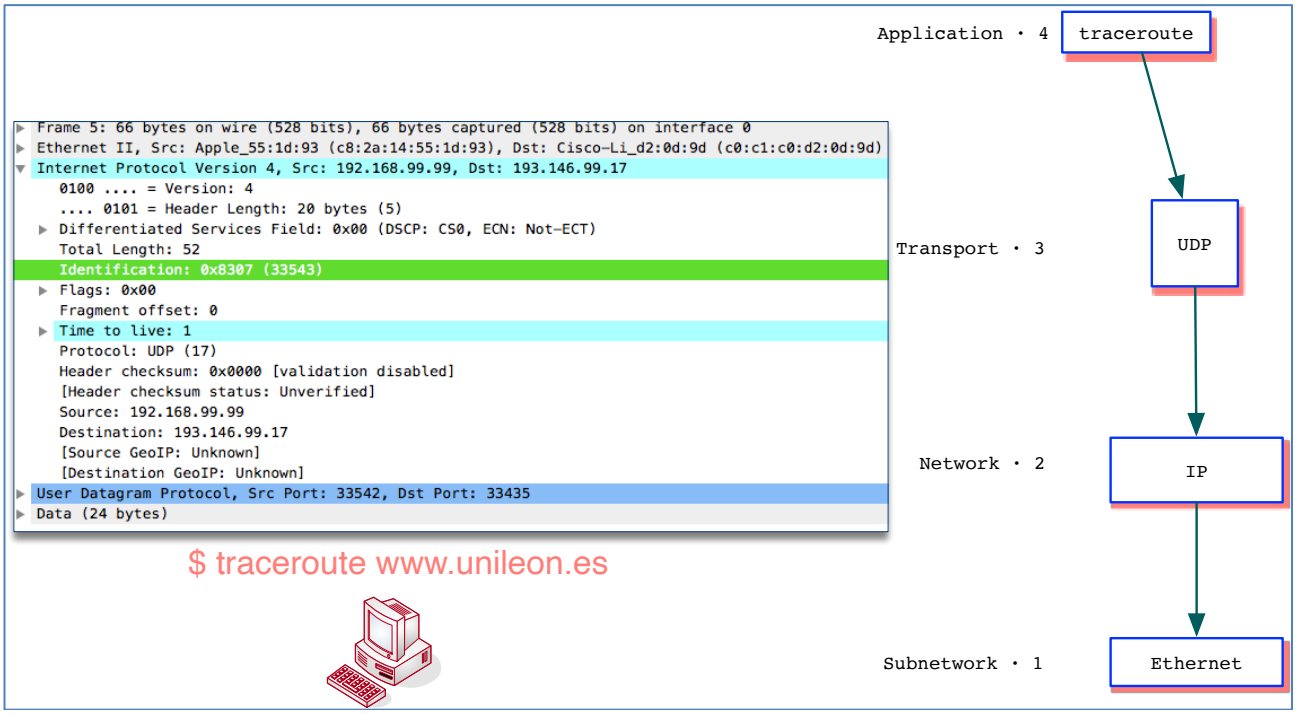

Figure 3.1. The protocol stack resulting from a traceroute probe packet (Right). The trace was obtained with Wireshark (Left)

