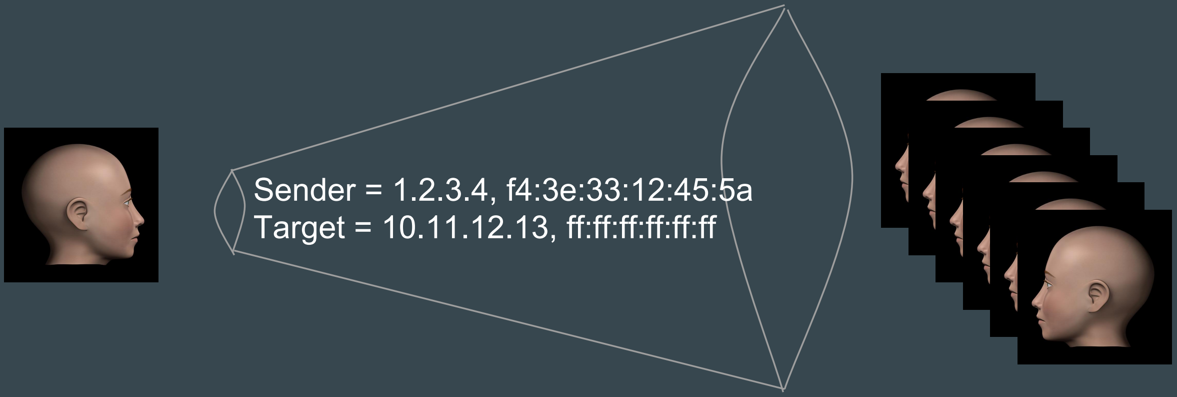
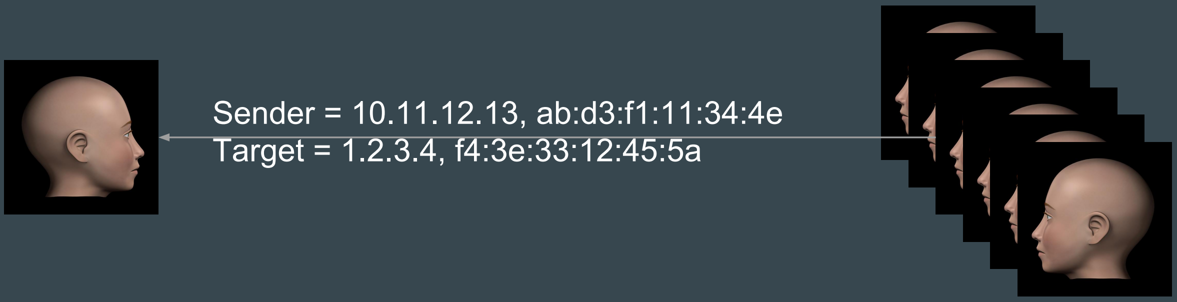
ARP


Sender = 1.2.3.4, f4:3e:33:12:45:5a
Target = 10.11.12.13, ff:ff:ff:ff:ff:ff

ARP allows other people to respond for you. This is considered a feature, but from a security standpoint this is not great. So how we we confirm that someone is listening for your packets? They can do this by setting their ethernet card in permiscuous code. This means that when a packet that comes to you that is not meant for you don't drop it like you should.

ARP



Sender = 10.11.12.13, ab:d3:f1:11:34:4e
Target = 1.2.3.4, f4:3e:33:12:45:5a

Sender address in Ethernet header does not have to be the same as the “sender’s HW address” in ARP response.

Send a packet where the ethernet address is not for you, but the ipaddress is. The first layer of the onion is the ethernet layer and you should drop it if its not for you, but evesdroppers will ignore that. Then they get to the ip layer which looks like it is for you. The evesdropper sees that the ip address is theirs so they respond. In real life this doesn't really work. A common software for eavesdropping is tcpdump.

Possible way to detect an eavesdropper

Destination IP address = eavesdropper
Destination Ethernet address ≠ eavesdropper

This is a basic address data structure.

IP forwarding - algorithm

Let D be destination IP from packet header.
Let d_j be destination in j^{th} entry of forwarding table.
Let m_j be mask in j^{th} entry of forwarding table.

- Find all entries with: $(D \& m_j) = d_j$
- From amongst those, pick entry with most 1's in m_j
- > 1 best match $\Rightarrow$ use some tie-breaking rule
- 0 matches $\Rightarrow$ "host unreachable."

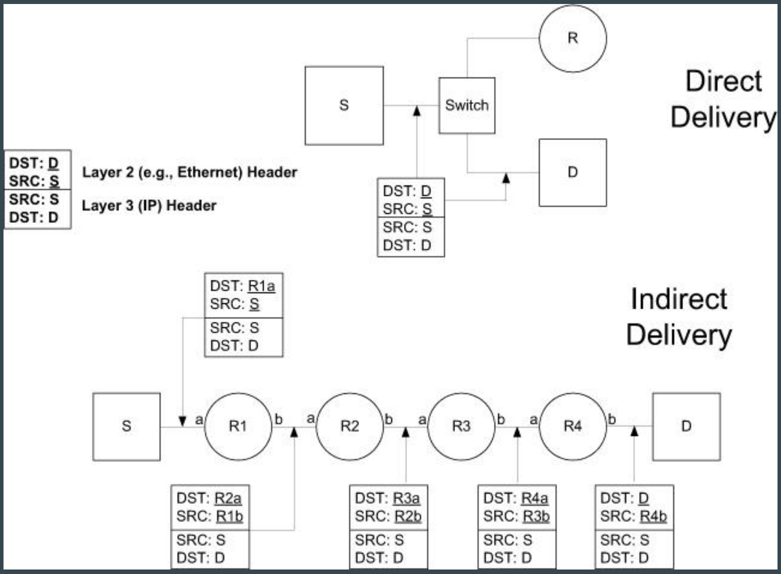
IP forwarding - examples

Destination	Mask	Next hop	Interface
0.0.0.0	0.0.0.0	10.0.0.1	10.0.0.100
10.0.0.0	255.255.255.128	10.0.0.100	10.0.0.100

- 10.0.0.19 matches both entries. 2nd entry is best-match.
- 178.162.3.4 matches 1st entry, not 2nd
- 10.0.0.131 matches 1st entry, not 2nd

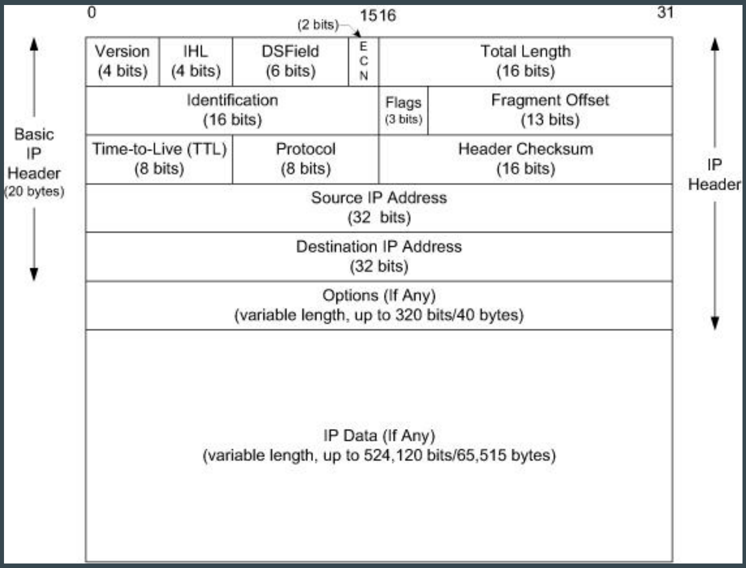
Test the ip against every entry in the table. Where the destination IP anded with its mask matches the ip address you have a match. Amongst those options pick the entry with the most 1s (this is the best match). If we have multiple equal matches use some tie breaking rule. If you have no matches something broke.

IP forwarding - “direct” vs. “indirect” delivery



If you put a sniffer on the line S to switch you'll see packets with source s and destination d. If we want to get from s to d on the bottom line we need a bunch of arp requests. S does not know about d, instead it issues an arp request to r1:a which then does a look up in its table and passes it on, so on and so forth. Gotta have a quick check to see if the destination is you, else look for next hop.

IP packet format

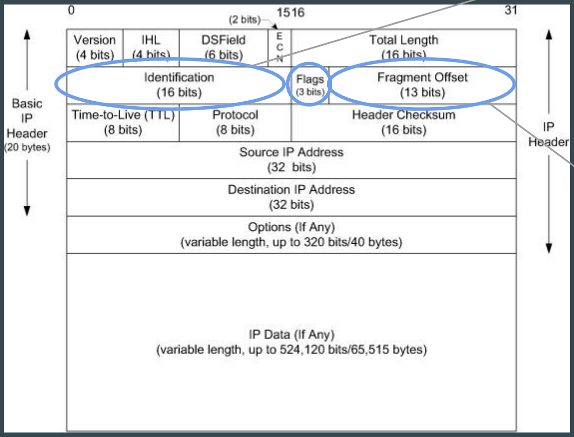


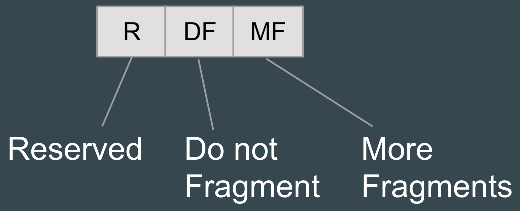
0
(2 bits)
1516
31
Version (4 bits)
IHL (4 bits)
DSField (6 bits)
ECN
Total Length (16 bits)
Identification (16 bits)
Flags (3 bits)
Fragment Offset (13 bits)
Time-to-Live (TTL) (8 bits)
Protocol (8 bits)
Header Checksum (16 bits)
Source IP Address (32 bits)
Destination IP Address (32 bits)
Options (If Any) (variable length, up to 320 bits/40 bytes)
IP Data (If Any) (variable length, up to 524,120 bits/65,515 bytes)
Basic IP Header (20 bytes)
IP Header

The version we are looking at is 4. The header is mostly unused, so is the ds field and congested network flag. This results in most ip packets starting wtih `4500`. Uusually the length is 540 ish (udp is frequently like this), but it can be anything up to 16 bits. Blah blah, time to live, header checksum to be explained later, blah. Source and Desination IP are self explanitory. Then some optiosn which is varibale length. The ip data can actually be empty because so much is already in the header.

Fragmentation - the IP Header

All fragments of a datagram have the Same ID.

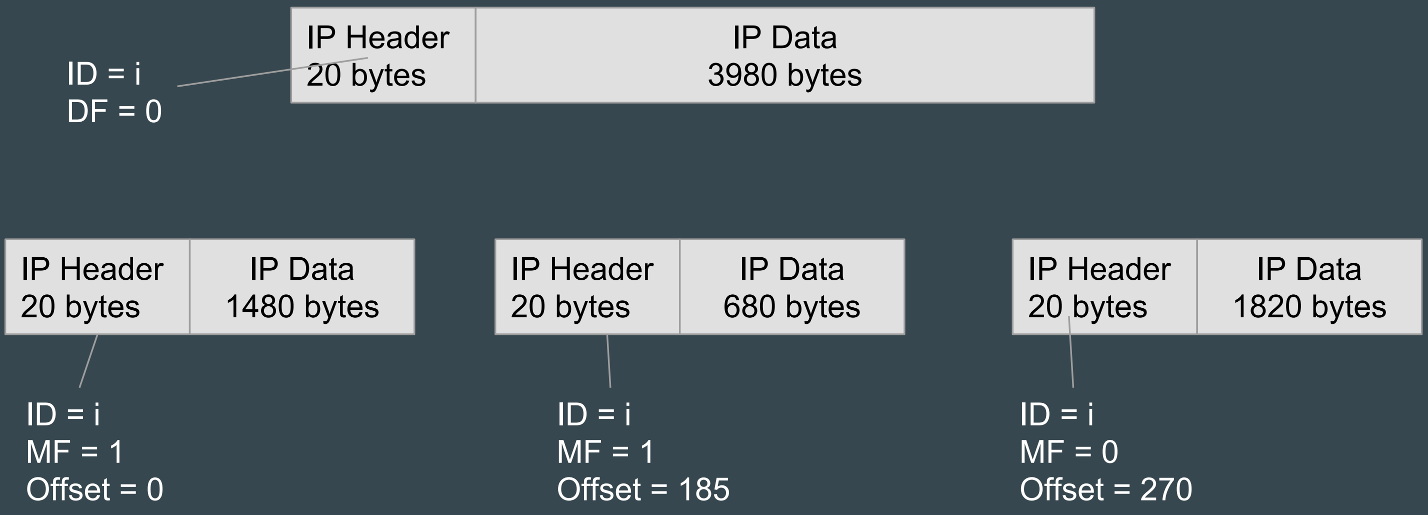

0
(2 bits)
15 16
31
Version (4 bits)
IHL (4 bits)
DSField (6 bits)
E C N
Total Length (16 bits)
Identification (16 bits)
Flags (3 bits)
Fragment Offset (13 bits)
Time-to-Live (TTL) (8 bits)
Protocol (8 bits)
Header Checksum (16 bits)
Source IP Address (32 bits)
Destination IP Address (32 bits)
Options (If Any) (variable length, up to 320 bits/40 bytes)
IP Data (If Any) (variable length, up to 524,120 bits/65,515 bytes)
Basic IP Header (20 bytes)
IP Header



Offset from start of data.
8-byte units.

A very small number of ip packets out there are actually fragments of packets. All fragments will have the same id which is how we know to stick them together. Usually this doesn't happen until all of the pieces reach their destination (assume this even though it doesnt always work that). There are three bits for flags, reserved, do not fragment, and more fragments. The do not fragment flag tells all downstream hops not to fragment this further. If a downstream hop wants to fragment a packet with a do not fragment (and vice versa), just drop it. Most of the time we want to not fragment shit. More fragment bit tells us if there are more fragments to come. These flags are followed by the length of the offset. Most fragments have a offset before their data begins (because we have fragmented the data you need to know where this chunk goes) and this value defines it.

Fragmentation - how it works via example



All fragments preserve the id. The first and second have the more fragment bits set. Once we receive the last packed (where the more fragment bit is 0) we can tell how many packets to expect and see if we have all of them. The order of arrival is not guaranteed.

Fragmentation - issues (3)

- Identification field may not be long enough at 16 bits.
    - ID does not repeat for 120 seconds @ 1500 byte packets ⇒ 6.5 Mbps throughput
    - Throughput 1 Gbps @ 1500 byte packets ⇒ ID space exhausted in < 1 second
    - Proposed solution RFC 6864: receiver relies on ID field for fragments only

Header Checksum

- "1's complement of 1's complement 16-bit sum."

```
Checksum ← 00 00
Perceive header as chunks of 16 bits = 2 bytes
Checksum ← Checksum + (each of those 2 bytes)
Add carry back into Checksum
Checksum ← bit-complement of Checksum
```

Start with some variable called check sum which is initialized to 0. Divide header into bytes, add them, handle the carry, and then take the bit compliment.

Header Checksum, example

- E3 4F 23 96 44 27 99 F3

00 00	E3 4F	1 06 E5	1 4B 0C	E4 FF
+ E3 4F	+ 23 96	+ 44 27	+ 99 F3	+ 00 01
= E3 4F	= 1 06 E5	= 1 4B 0C	= 1 E4 FF	= E5 00

Checksum = 1A FF

You can see this iterating through each byte, summing as it goes. The last step is it adding the overflow of 1 back into it.

Checking the checksum

- Compute checksum of entire header (including checksum)
- Result should be 00 00
- E.g.,
    - (E3 4F) + (23 96) + (44 27) + (99 F3) + (1A FF) = 1 FF FE
    - 1's complement: (FF FE) + (00 01) = FF FF
    - Complement = 00 00

The checksum of the entire header is computed and then you take its compliment, if this works out to 0 we know that there has been no errors.

TTL - Time To Live

- Use to limit datagram lifetime in the network
- Decremented by 1 @ each hop
- ⇒ header checksum must be recomputed @ every hop

As the packet gets bounced around every router recomputes the checksum because the time to live field has changed each time. Because of this we want this algorithm to be super fast.

TTL - Time To Live

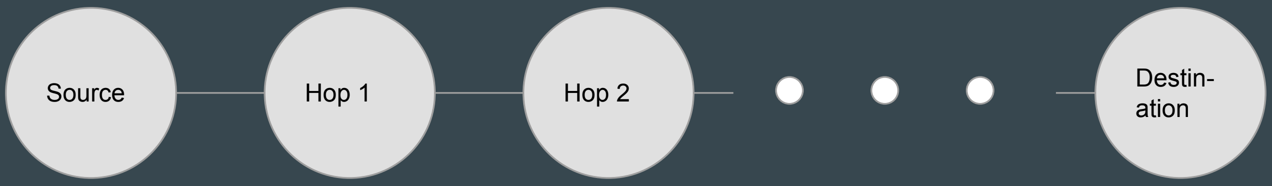
- Use to limit datagram lifetime in the network
- Decremented by 1 @ each hop
- ⇒ header checksum must be recomputed @ every hop

Traceroute will show you how many hops are required to get from your machine to some ipaddress. This is often used to figure out what value the time to live should be set to.

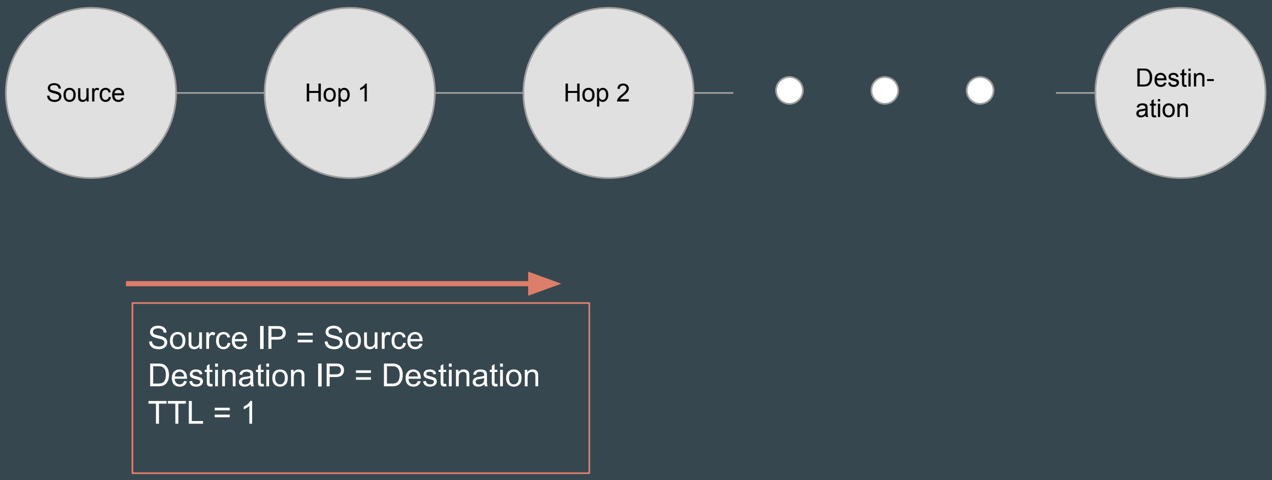
Traceroute - a nifty use of TTL

- Objective - want to find path from here (source) to destination
- Method - leverage TTL field

Traceroute - how it works, via example

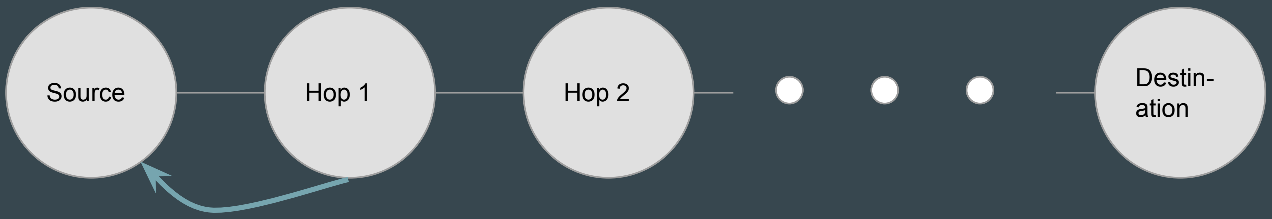


Traceroute - how it works, via example, Step 1



Source IP = Source
Destination IP = Destination
TTL = 1

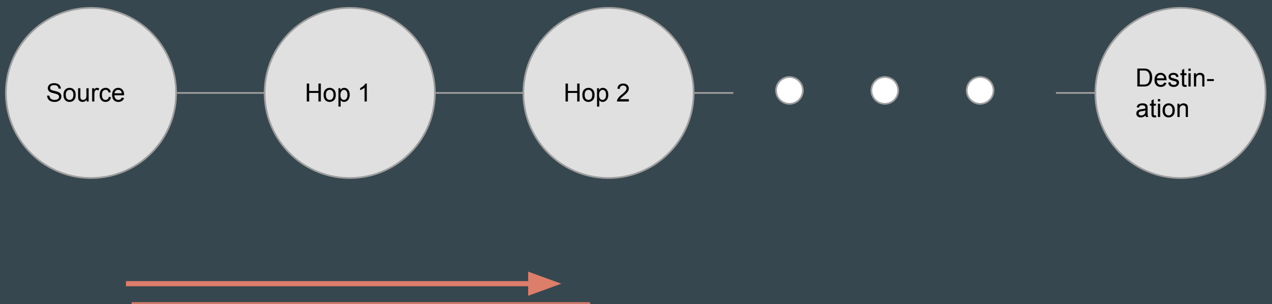
Traceroute - how it works, via example, Step 1



Source IP = Hop 1

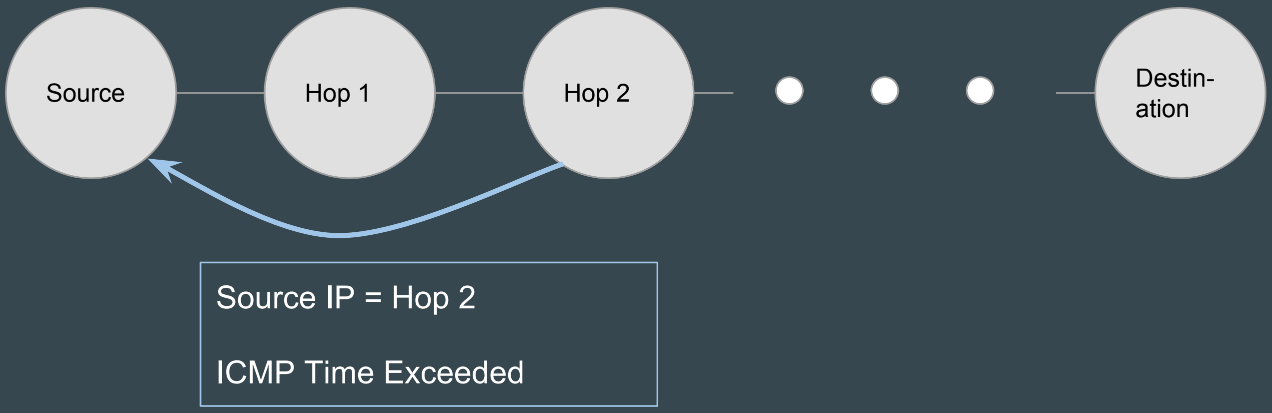
ICMP Time Exceeded
IP header + 8 bytes of data
from original datagram

Traceroute - how it works, via example, Step 1



Source IP = Source
Destination IP = Destination
TTL = 2

Traceroute - how it works, via example, Step 1



Source IP = Hop 2

ICMP Time Exceeded

We send out the packet with a time to live of 1. It then runsout of time and returns with a special error code saying the time has been exceeded. Then we increment the time to live and try again.

Question

It so happens that an IP packet sent by S to D incurs no errors in transit.

- True or false:

  The value of the header checksum that D sees is the value that S put in the checksum field.

The answer to this is false since the checksum field has been recomputed by each router. The exception to this is if the two end points are right next to each other.