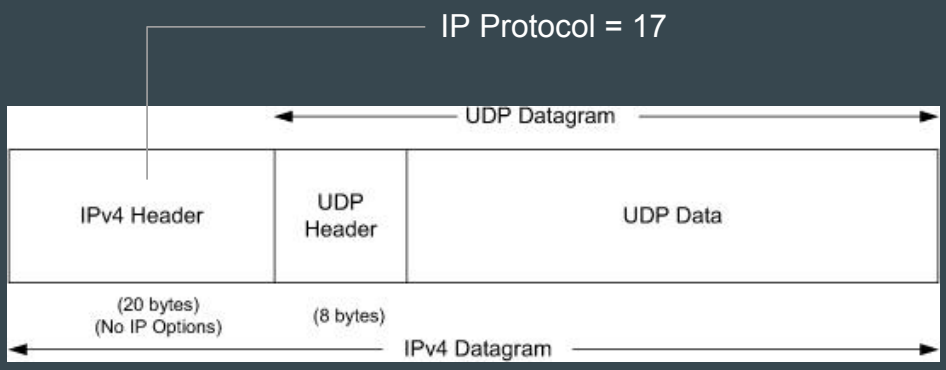
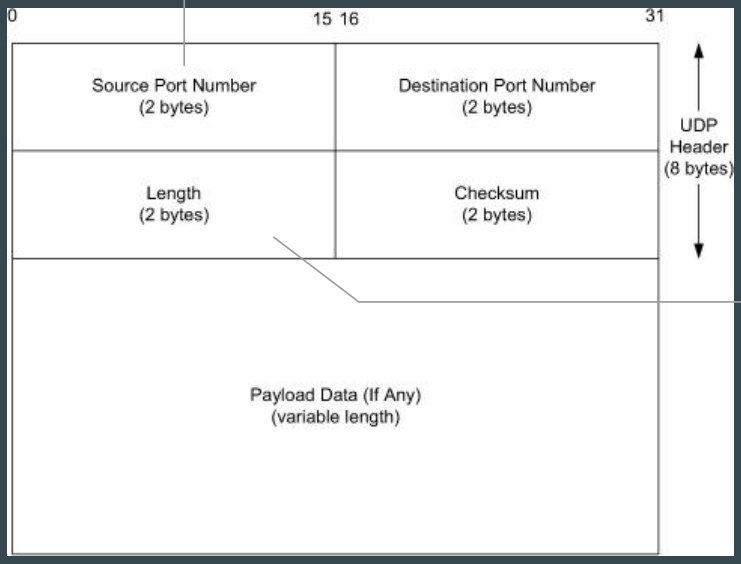
UDP and Related Things



UDP Characteristics

- Is a transport-layer protocol
  - End-to-end
- Connectionless, best-effort, no error correction, ...
  - Gives application control

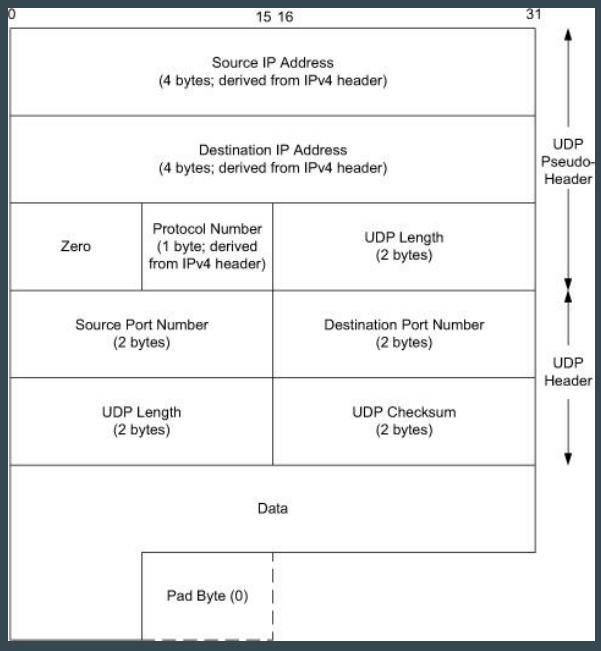


- Allowed to be 0
- Receiver demuxes based on ⟨destination-ip, destination-port⟩ only.

- Units = bytes
- Length of header + data ⇒ min = 8
- Redundant if encapsulated in IP

The header here is much simpler than it was for ip. The destination ip and destination port are the only bits of data required to get the packet to its destination. You can say that you have a source port of 0, but the destination might no allow it. The length field is the length of the header. Its completely redundant because the ip layer already has the length in it.

UDP Checksum



Scope of checksum

The checksum calculation is a bit wonky because the udp data might be odd lengthed. It is caclulated from the udp header, data, and a pseudo header. If this happens you add a byte for padding to fix this. Its a bit weird because the checksum is included in the calculation of the checksum.

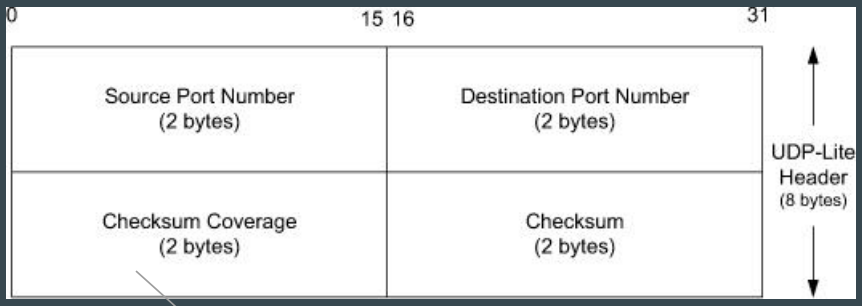
UDP Checksum - some details

- 1's complement of 1's-complement addition
- Pseudo-header used to guard against mistakes by IP layer.
- Checksum field of 0's used in checksum computation
- Padded with 0-byte for even # of bytes; not transmitted
- In UDP, checksum is optional in IPv4 - indicated with checksum = 00 00
  - To distinguish from legitimate checksum of all 0's, transmit ff ff
  - UDP checksum required in IPv6
- Checksum is used end-to-end, not hop-by-hop

We include a pseudo header that has duplicate information for the destination and source ip address in case there are bugs in the ip layer. When we are calculating the checksum we can use 0s in the checksum calculation. Some protocols don't even bother calculating the checksum because it already has it from the ip layer or it is weak. If the checksum works out to being 0 you can run into problems because it is used to denote that the checksum is unimportant. The udp checksum is not recalculated with each hop because it is a transport protocol. Problems can occur during network address translations.

If you fragment the udp datagram the header become separated from the data, but that is ok because they will be reassembled at the destination then the conversion to the udp layer happens.

UDP-lite



- 0 ⇒ entire UDP-lite datagram
- 1 - 7 - illegal values, header always covered

Protocol value in IP header = 136. So a different transport-layer protocol than UDP (17).

In udp lite we get rid of the length and use checksum coverage instead to say how much of you udp packet is covered in the calculation of the checksum. This transport layer is completely different than udp, it just shares a name. This was done for performance reasons.

Path MTU discovery with UDP

- UDP port 7 = echo service
- To test if MTU of, e.g., 1505, is supported, send UDP data
  - of 1477
  - to ⟨destination-ip, destination-port = 7⟩
  - Set DF (Do not Fragment) bit in IP header
- Response ⇒ supported
  - Is the converse necessarily true?

Udp is super simple and powerful. Traceroute uses it. It also has something called the echo service (tcp does as well, infact most of port numbers are shared between the two). This is a standard that says if you send a packet there it will return that exact packet back. We can use this to find the largest datagram you network can support (called maxmium transmission unit). We send a packet that is of a size we want to test minus 28 bytes for the header. We send this packet to the echo port with a no fragmentation flag. If we get a packet returned we know that this size is supported throughout the network. It is important to note that if you don't get a response that doesn't necissarily mean that the packet size is not supported (there a bunch of reasons there wasn't a response).

Programming quirk

- *sendto(), recvfrom()*
- Not required to support particular datagram size
- Some implementations adopt 512 as max
  - Every IP host required to support MTU ≥ 576
- Other issues may result in fewer bytes in a single *recvfrom()* call

UDP server design considerations

- Can *bind()* to “wildcard” local address
  - INADDR_ANY
- Server may want to know at which local address datagram was received
  - *getsockname()*
- Multiple servers can bind to same ⟨destination-ip, destination-port⟩
  - SO_REUSEADDR

There is another command called ioctl that is super shitty. Getsockname is a layer above it and works a bit better.

UDP traffic on the Internet

- No reliable statistics
- 10-40% of Internet traffic is UDP
  - RTP - Real Time Protocol
- % of fragments that are UDP is high: ≈ 65%
  - Overall only < 1% of Internet traffic is fragments
  - Why?
    - Packets in packets in packets
    - “Carelessness”

Most fragments are of type udp. A theory for this is that this is the result of encapsulation. As we encapsulate more layers the datagram gets bigger and thus is more likely to require fragmentation. Another theory is that people who write udp tend to be more careless.

Example applications

- DHCP
  - Initial requests have
    - source-ip = 0.0.0.0, source-port = 68
    - Destination-ip = 255.255.255.255, destination-port = 67
  - Response is unicast
- DNS
  - Used to resolve “names” to “numbers”
  - E.g., what is the IP address of ecelinux4.uwaterloo.ca
  - E.g., what is the mailserver associated with uwaterloo.ca?
  - E.g., what is the webserver associated with uwaterloo.ca?

Aside - IP fragmentation + ARP

- Your book discusses an interesting experiment:
  - Send an IP packet to destination that you know will be fragmented, e.g., into 4
  - How many ARP requests are generated?
    - Maybe up to 4
  - Problem (RFC 1122):
    The link layer SHOULD save (rather than discard) at least one (the latest) packet of each set of packets destined to the same unresolved IP address, and transmit the saved packet when the address has been resolved.

- Not just a problem when we have fragmentation.