ECE 435 – Network Engineering
Lecture 13

Vince Weaver

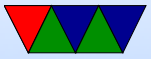
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 February 2026

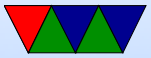
Announcements

- HW#4 due Friday (e-mail and DNS)

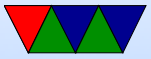


Unreliable, Connectionless – UDP

- User Datagram Protocol (RFC 768)
- Just an 8-byte header tacked onto the data packet
- No reliability, no rate control, stateless
  If you want these things you have to add at higher layer
- Error control optional
- Why none of those things? All add overhead.
  - Used when want packets to get through quickly.
  - Don't care about re-transmits, better for real-time (VOIP, streaming?)



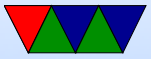
  - Easy to implement, for low-level stuff like bootp/dhcp
  - Good for broadcasting
- Provides process-to-process communication and per-segment error control
- Can send UDP packets to a destination without having to set up a connection first



UDP Header

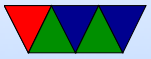
2 bytes	2 bytes
Source Port	Destination Port
Packet Length	Checksum

- 16-bits: source port (optional, says where it is coming from in case need to respond, 0 if unused)
- 16-bits: destination port
- 16-bits: length (in bytes, includes the header)
  min: 8, max: 65,515 (<64k to fit in 64k IP packet)
- 16-bits checksum (optional, 0 if unused, see below)
- data follows



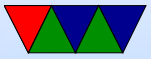
Port Number Review

- 16-bit, so 64k of them
- Can map to any you want, but there are certain well-known ones. Look in `/etc/services`
  For example. WWW is 80/tcp. DNS is 53/udp
- Most OSes, ports <1024 require root (why?)
- 1024 ... 49151 are registered IANA ports
- 49152 ... 65535 are ephemeral ports, dynamic for use by any service



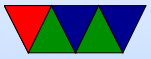
Uniquely identifying flow

- TCP/UDP on IPv4 represented by 104 bit socket pair 5-tuple
  - Source/destination addr
  - Source/destination port
  - protocol ID (TCP or UDP)
- IPv6 in theory has a specific field for this



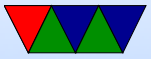
UDP checksum

- Find info on this in RFC768 and RFC1071
- If set to zero, ignored
- Receiver drops packet if invalid checksum
  Does not request resend, does not notify sender
  (yes, really, no error message if dropped)

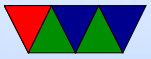


UDP checksum Algorithm

- Ones-complement sum of all 16-bit words in header and payload
- Padded with 0s to be multiple of 16-bits
- Also added to the checksum is the pseudo-header (Layering Violation)
  Enables receiver to catch problems (delivered to wrong machine) – why could this be a problem?
- The ones' complement of checksum is put in checksum field. That way when you checksum a valid packet the

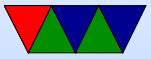


result will be 0.



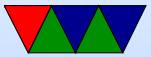
IPv4 Pseudo Header

- 96-bit pseudo header
- source IP (2*16)
- dest IP (2*16)
- protocol (padded to 16)
- length (16 bits)



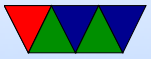
IPv6 Pseudo Header

- 128-bit src IP
- 128-bit dest IP
- 32-bit UDP len
- 24-bit 0
- 8-bit next/type (17 UDP)



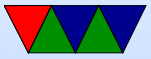
Ones' Complement Refresher

- Positive numbers are same as always (with high bit 0)
- Negative numbers are represented by the inverse (bit flipped) of positive. (no adding 1 (that's twos' complement)
- When adding, if there's a carry, it is wrapped around and added in to the low bit ("end-around carry")
- Subtraction is a bit more complicated
- There are two zeros, 0x0000 and 0xffff



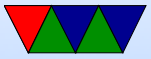
Checksum Corner Cases

- What happens if checksum is 0? Conflict with disabled checksum? Entered as 0xffff, which in ones complement is -0
- Checksum considered mandatory on IPv6 because IPv6 header not checksummed
- Why would you ever leave checksum out? Takes time to compute, might care about latency over errors [video?]



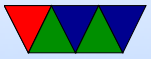
UDP example Packet

```
0x0000:  8875 563d 2a80 0030 18ab 1c39 86dd 6002  .uV=*..0...9..‘.
0x0010:  2618 0031 1140 2610 0048 0100 08da 0230  &..1.@&..H.....0
0x0020:  18ff feab 1c39 2001 4860 4860 0000 0000  .....9..H‘H‘....
0x0030:  0000 0000 8844
UDP starts at 0x36:
                       e239 0035 0031 9c0e 8657  .....D.9.5.1...W
0x0040:  0120 0001 0000 0000 0001 0377 7777 0465  ...........www.e
0x0050:  7370 6e03 636f 6d00 0001 0001 0000 2910  spn.com.......).
0x0060:  0000 0000 0000 00
```



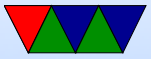
UDP example Packet Decoded

- What is source port? What is destination port? Size?
- How can you tell what high-level protocol it is?
  Can you make an educated guess from the ports?



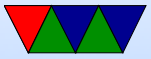
UDP checksum example (from prev slide)

- 16-bit sum of "virtual header" (two IPv6 addresses, protocol (0x0011) and length of udp packet/header (0x0031)) is 0x2'9f8c
- 16-bit sum of UDP header leaving off checksum is 0xe29f
- 16-bit sum of UDP data is 0x2'e1c0
- Add them get 0x6'63eb
- It's a 16-bit sum, so add 0x6 + 0x63eb = 0x63f1 ones complement is 0x9c0e, which matches the UDP checksum field



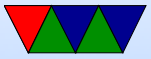
UDP and the Operating System

- Server: user binds to UDP socket
- OS sets up queue
- Network stack decodes packet, notes that it is UDP
- Runs checksum, drops it if invalid
- Finds port, looks to see if any processes listening for it
- If so, adds to queue
- If not, sends an ICMP "port unreachable" error message
- All UDP messages to that port, no matter who sends them, end up in the same queue.



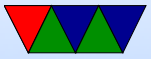
Writing UDP sockets code

- Use `SOCK_DGRAM` rather than `SOCK_STREAM`
- Can skip the listen/accept state, as no connection is there. Just receive the packets as they come in.
- Can't read then write, as no connection. For the server to write back to the client it needs to use `recvfrom()` which also provides ip/port
- To send a packet use `sendto()`



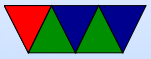
UDP Socket – Client code

```
// setup socket
socket_fd = socket(AF_INET, SOCK_DGRAM, 0);

// get dest address/port
dest=gethostbyname(DEFAULT_HOSTNAME);
memset(&dest_addr,0,sizeof(dest_addr));
dest_addr.sin_family=AF_INET;
memcpy(dest->h_addr,&dest_addr.sin_addr.s_addr,
        dest->h_length);
dest_addr.sin_port=htons(port);

sendto(socket_fd,buffer,strlen(buffer),0,
        (struct sockaddr *)&dest_addr, dest_len);
```



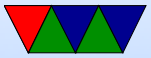
UDP Socket – Server code

```
// setup socket
socket_fd = socket(AF_INET, SOCK_DGRAM, 0);

// wait for incoming connection
bind(socket_fd, (struct sockaddr *) &server_addr,
        sizeof(server_addr));

// read data from socket, including client_addr info
recvfrom(socket_fd,buffer,(BUFFER_SIZE-1),0,
    (struct sockaddr *) &client_addr, &client_len);

// send reply
sendto(socket_fd,buffer,strlen(buffer),0,
    (struct sockaddr *)&client_addr, client_len);
```



Common UDP Services

- Obsolete: echo/discard/users/daytime/quote/chargen
- Nameserver
- bootp/tftp
- ntp (network time protocol)
- Old versions of NFS
- snmp
- Tunnel for QUIC



UDP real-time

- Real-Time Protocol (RFC1889)
- On top of UDP, multiplexes
- data streams
- timestamps