Lecture 25: UDP Socket

User Datagram Protocol (UDP)

UDP service:
• “no frills,” “bare bones” extension of best-effort IP
• “best effort” service, UDP segments may be:
  • lost
  • delivered out of order to app
• connectionless:
  • no handshaking between UDP sender and receiver
  • each UDP segment handled independently of others

UDP [RFC 768]

UDP Socket

UDP socket identified by the tuple:
<dest IP address, dest port number>
(contrast with TCP’s four-tuple!)

When a host receives a UDP segment, it:
• checks the destination port number and
• directs the UDP segment to the socket with that port number
⇒ IP datagrams with different source IP addresses and/or source port numbers are directed to the same socket
Connectionless Demultiplexing

SP provides "return address"

UDP Socket

Similar to TCP's stream sockets, except:
• sockets created using SOCK_DGRAM instead of SOCK_STREAM
• no need for connection establishment and termination
• no connect-bind, listen, accept handshaking, but server must still always call bind()
• client doesn't need to call connect() though client may use connect() to tell kernel to "remember" the server's address and port#

Data Transmission

No "connection" between sender and receiver
• sender explicitly attaches IP address and port# of destination to each packet, by using sendto() instead of send()
  • send() can still be used if server's address and port# have been "registered" with kernel using connect()
• if receiver uses recvfrom() it can extract IP address and port# of sender from received packet
  • if these are not needed, recv() may be used instead

Transmitted data may be delivered out of order, or not delivered at all
Socket Addresses
Somewhere in the socket structure:

TCP Server:
- IP address
- Port#
  - bind(): match incoming packets' destination
  - connect(): copy to outgoing packets' destination

UDP Server:
- IP address
- Port#
  - bind(): match incoming packets' destination
  - To be filled in with sender's addr.
  - ephemeral port by kernel

TCP Client:
- IP address
- Port#
  - connect(): To be filled in with host's IP addr.
  - ephemeral port by kernel

UDP Client:
- IP address
- Port#
  - To be filled in with host's IP addr.

Socket Buffers
When receiver's socket receive buffer is full, incoming UDP packets will simply be dropped.

If sender's socket send buffer is smaller than the size of UDP data passed to send(), send() returns -1 and the system global variable errno is set to EMSGSIZE.

The APIs getsockopt() and setsockopt() are used to query and set socket options, including the SO_RCVBUF and SO_SNDBUF options.

Data Transmission
UDP packets have boundaries, not forming a byte-stream as in TCP, so recv() retrieves one message at a time, i.e., no need to call recv() in a loop:
- call to recv() returns the whole packet
- cannot retrieve only parts of a packet
- to inspect a packet call recv() with flags=MSG_PEEK
- same for recvfrom() and recvmsg()

Lab 5 Demo
Best-effort netimg with no flow-control and no error-control

Learn how to set send and receive buffer sizes

Play with different receive buffer sizes and observe effect of lack of flow control

Play with different drop rates and observe effect of lack of error control

Learn how to send and receive large data buffer using gather write (sendmsg()) and scatter read (recvmsg())
How to Send a Large File

Error recovery and correction both require associating data with sequence number (sqn).

How to attach sequence numbers to chunks of data to be sent (assuming 1K segment)?

Gather Write

Output data is gathered from multiple buffers

Scatter Read

Input data is scattered into multiple buffers

Out-of-Order Packets

Receive into here → i/o vector

Received from network:
Data Scatter/Gather

The socket APIs for scatter-gather I/O are 
recvmsg(sd, msg, flags) and 
sendmsg(sd, msg, flags)

TCP can also use sendto(), recvfrom(), sendmsg(), and recvmsg()

Lab5: Sequence Number

Sequence number is per byte, not per packet

The sequence number attached to a packet is the sequence number of its first byte

The sequence number of a byte is its byte offset from the start of image buffer

This enables out-of-order data to be placed in its right position in the image buffer

Lab5: Interoperability Testing

Home firewall may block UDP packets

Use "ssh -Y" to test client on CAEN eecs489 hosts if UDP blocked by home firewall

- on Windows, use MobaXterm or similar tools that supports X forwarding

VNC may have OpenGL compatibility issues and is slower