UDP and TCP

This lecture describes the two most common transport-layer protocols used by IP networks: the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).
After this lecture you should be able to:
predict the values of the following UDP and TCP header fields: source and destination port numbers, checksum, length, sequence number, acknowledgment, flags and window; predict the change of TCP state machine state as a result of a socket API call or receiving a frame with specific bits set; predict the maximum amount of data that will be sent in the next TCP segment based on the sequence numbers of queued data and the values in a received TCP header.

Introduction

So far we have studied some data link layer protocols such as PPP which are responsible for getting data from one end of a channel to the other, and the IP network layer protocol which is responsible for getting IP packets from one host to another. In this lecture we look briefly at two transport layer protocols that make it easier to write networking applications.

The transport layer serves to connect client applications on one host to the desired server processes on another. In addition, some transport protocols provide other services such as reliable, in-sequence message delivery and congestion control that are not provided by IP.

Applications that make use of transport layer services include clients such as web browsers or servers such as file servers.

The two transport-layer protocols in common use are:

UDP - sends a packet from a port on one machine to a port on another machine but, like IP, does not ensure delivery; relatively simple

TCP - use a "connection-oriented" model that provides reliable byte pipe from a process on one machine to process on another; relatively complex

Other abstractions, such as a reliable datagram protocol, would be possible but having only two transport protocols keeps things simple for implementers.

Transport Layer Functions

The most important features we might like to get from a transport protocol include:

- reliable delivery: sender receives an acknowledgment that the data was received
- sequencing: packets are guaranteed to arrive in the same order they were sent
- no duplication: data is not duplicated
- fragmentation: arbitrarily-long byte sequences can be broken down into smaller packets (called "segments" in TCP) and reassembled at the destination
- flow control: the receiver can be told (or can infer that it should) slow down the rate at which it sends data to a destination
- multiplexing: we can have multiple simultaneous connections between hosts, even to the same service

In the same way that IP addresses are used to specify a particular host, a server can provide multiple types of services. Each such service needs an "address" or "service access point" (SAP). In TCP/IP SAPs are defined by an associated 16-bit numbered "port". The Internet Assigned Numbers Authority (IANA) assigns port numbers to protocols (e.g. 80 for HTTP, 25 for SMTP, ...).

Each transport-layer IP connection is uniquely defined by the combination of 4 things: host and destination addresses and host and destination port numbers. Each different 4-tuple defines a different connection. This explains how, for example, multiple users on the same client can contact the same service on the same server.

Exercise 1: Two users on the same client connect to the same web server. Which of the addresses and ports are the same? Which are different?

UDP

The UDP (User Datagram Protocol),defined in RFC-768, is a simple[1] protocol that provides little more that is provided by IP. UDP does not offer flow control, reliable delivery, un-duplicated or in-sequence delivery. Instead,these features must be provided by the application. This allows more flexibility at cost of additional complexity in the application.

UDP Header

The following diagram, from RFC-768, shows the UDP header:

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...

     User Datagram Header Format
```

The header has four 16-bit words: source and destination ports, length, and a checksum. The main purpose of the header is to specify the destination SAP (port or service) that should handle the packet. For example, a DNS server "listens" on port 53 and a DHCP server on port 67. The source port is included so a client can listen for a UDP datagram sent as a response.

The length is the number of bytes in the UDP datagram, including this UDP header. The checksum is the regular IP 16-bit one's complement checksum applied to the header, data and three 32-bit words from the IP header (the IP "pseudoheader") as shown below:

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|          source address           |
+--------+--------+--------+--------+
|        destination address        |
+--------+--------+--------+--------+
|  zero  |protocol|   UDP length    |
+--------+--------+--------+--------+
```

Exercise 2: What are the minimum and maximum values for the UDP length field?

[1]The RFC is only 2 pages.

Examples of UDP-Based Applications

A typical UDP-based application is the DNS. A single UDP packet containing the query is sent to a DNS server and a single UDP response is received with the response. The application must handle delayed, duplicated or lost frames.

Another class of applications for UDP is interactive applications such as VoIP (voice over IP). Latency is very important for interactive applications – sometimes more important than avoiding loss of data. The reliable-delivery and congestion control features of TCP can sometimes result in long delays. Interactive applications often use UDP so they can implement their own error- and congestion-control algorithms.

TCP

Transmission Control Protocol (TCP) is the most commonly-used transport-layer protocol. It is defined in RFC 793[2].

Like UDP, TCP connections are established between ports that allow multiple simultaneous connections between hosts and also allow selection of the service that clients connect to.

Unlike UDP, TCP supplies a reliable byte pipe even though the underlying network can delete, reorder and even duplicate packets.

TCP is a full duplex protocol – it provides reliable byte streams in both directions.

A TCP connection has state that needs to be maintained for each connection (host/port addresses, sequence numbers, buffers, timers, ...). This state, stored in a "Transmission Control Block", needs to be initialized when the connection is set up and the memory used to store this state must be released when the connection is terminated.

To deal with highly variable network delays and data rates, TCP uses a "sliding window" ARQ protocol to:

- control *congestion* - excessive traffic on links which results in buffer overflows, and
- implement *flow control* - to prevent traffic being generated faster than it can be processed at the destination host.

It is important to understand that although the TCP frame format and the protocol state machine are

[2]TCP is much more complex than UDP, RFC 793 is 85 pages.

defined in RFC 793, there is considerable variation in TCP implementations because:

- TCP allows for options to be negotiated at the start of the connection; hosts may implement different sets of options, and
- the algorithms used to, for example, decide how long to wait before transmitting or retransmitting a frame are implementation-dependent.

TCP Header

The following diagram, from RFC-793, shows the contents of the TCP header:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The source and destination ports have the same meaning as in UDP.

The TCP protocol treats the data as two continuous sequences of numbered bytes flowing in the two directions. These sequences of bytes are divided into "segments," each of which is transmitted in a separate IP frame.

The Sequence Number represents the number of bytes already transmitted. This is a number relative to the initial Sequence Number sent in the starting SYN frame. This is typicallly a random value. It is also incremented by 1 by a SYN or FIN frame.

The Acknowledgment Number is the number of bytes received (including SYN and FIN), also relative to the starting sequence number.

The Window value can be interpreted to mean how many bytes of buffer space the receiver currently has available. The other host uses this to decide how much more data can be sent.

Note that Sequence number refers to the data being transmitted in the same frame while Acknowledgment and Window refer to the data flowing in the opposite direction (from the other host).

Sequence numbers are also incremented before the first byte (after SYN) and after the last byte (after FIN) even though no extra data bytes are sent. This allows acknowledgement of the start and end of a connection.

Note that sequence numbers do not normally begin with 0. The Sequence Number value transmitted with the SYN packet determines the starting value.

Exercise 3: The most recently received TCP packet for a connection had an Acknowledgement Number value of 1000 and a Window value of 64. Assuming 1024 bytes are ready to be sent, what will be the value of Sequence Number in the next packet transmitted for that connection? What will be the length of the IP packet?

The Data Offset the number of 32-bit words in the TCP header (including options).

The 6 bits in the flag field are used to manage the TCP connection:

- the URG and ACK bits indicate that the values in the Urgent Pointer and Acknowledgment Number fields are valid.
- the SYN and FIN bits are used to start and terminate a connection.
- The PuSH bits can be used to indicate the end of a message defined by a higher-level protocol.
- The RST bit is used to indicate a loss of synchronization (e.g. Acknowledgment of data not sent) and causes the connection to be closed.

The checksum is computed over the TCP header, an IP pseudo-header as with UDP, and the data.

The URG bit and the Urgent pointer were meant to allow high-priority data to be sent ahead of lower-priority data within the same TCP connection. It is rarely used.

TCP State Machine

The ASCII diagram below, taken from RFC-793, describes the TCP state machine. The text in each block labels that state. The labels on the state transition path state the action or received flag bits that cause the transition (above the line) and the bits that are set in the frame that is sent in response (below the line). Other labels are x (send nothing), active/passive OPEN (client/server socket creation), create/delete TCB (allocate/release data structures for the connection) and a Timeout (2 $\times$ 120 seconds).

Study of the diagram shows that setup and tear-down of a TCP connections requires a 3-way handshake { SYN, SYN+ACK, ACK } or { FIN, FIN+ACK, ACK }.

```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

TCP Connection State Diagram
Figure 6.

The above is a quick summary of how connections are set up and torn down. There are many more details which we do not have time to cover in this course.

Exercise 4: According to the TCP state transition diagram, what should happen if a host responds to an initial SYN frame by sending back a frame with only SYN set?

Also note that connections can be closed in one direction only.

Congestion Management In TCP

When more data arrives at a router than can be sent out, the data will build up in queues. This is called congestion. If the queues get too long then packets will have to be dropped. These frames then have to be retransmitted. This increases the load on the network which can then become unstable. Rather than using an explicit mechanism to monitor congestion, TCP checks for it indirectly by measuring packet loss and delay.

Since each packet is routed independently, the network layer (IP) cannot determine packet delay or loss. On the other hand, TCP sees acknowledgments and so it can measure delays and detect lost frames and take appropriate measures to deal with congestion. The transport layer is thus made responsible for congestion management and flow control.

One of the main purposes of TCP is to minimize the likelihood of dropped frames by reducing the rate at which frames are sent out when congestion is detected. The algorithms to do this are fairly complex and implementation-dependent. We will not cover them here.

Exercise 5: How can a host reduce the rate at which data is sent to it on a TCP connection?

Examples of TCP-Based Applications

Most internet applications use TCP. This includes web browsers (the HTTP protocol, server on port 80), e-mail transfer (the SMTP protocol, port 25), secure shell login (the SSH protocol, port 22) and many others.

Most of these protocols are text-based which simplifies writing and debugging the applications. For example, an an HTTP client first establishes a TCP connection and then sends a line starting with GET followed by the URL, the HTTP version and a CR character followed by headers. Here is an example of how we can emulate the behaviour of a web browser using the telnet utility from the command line:

```
telnet www.bcit.ca 80
Trying 142.232.77.1...
Connected to www.bcit.ca.
Escape character is '^]'.
GET http://www.bcit.ca/study/ HTTP/1.1
Host: www.bcit.ca
```

The server responds with some headers, a blank line and the page contents in the HTML markup language.

Many other Internet protocols work similarly. For example the SMTP protocol uses commands such as HELO, FROM, MAIL and DATA to transfer e-mail messages.

The Socket API

The "socket" library is commonly used to implement network client and server software that uses the TCP/IP protocol. An API (Application Programming Interface) is a set of functions that can be called by a program to implement the networking parts of the program. Although you do not need to be able to write such applications, it is useful to know how the API is accessed and you should know which of these calls cause changes of state in the TCP state machine. The main functions are:

socket() - allocates and initializes the required data structures (e.g. a TCB)

bind() - sets the local address (port) (typically used only on the server side)

listen() - enables incoming connections (on server) – the "passive open" described above

accept() - waits for incoming connections (on server)

connect() - waits for outgoing connections (on client) – the "active open" described above

send() write()] - sends bytes over the connection

receive()**read**() - reads bytes received over the connection

close() - shuts down the connection and releases memory – the "close" described above

Other library functions are often required. For example, the `gethostbyname()` function can be used by the application to convert a host name to an IP address. The method is implementation-dependent, but typically it's by querying the local host files and/or using the DNS protocols.