Transportation Layer Part 1

cbf6yd sjc2gg

February 25, 2020

Introduction

The transportation layer resides between the application layer and the network layer. It provides communication services directly to application processes running on different hosts.

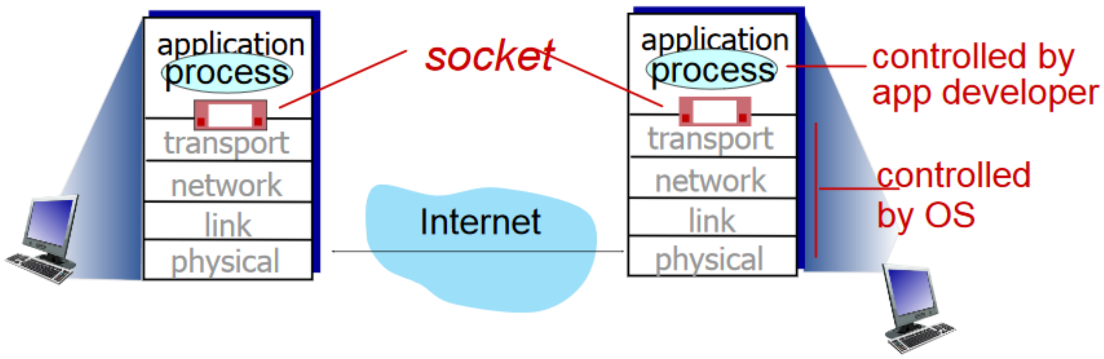

Figure 1: Layer Diagram

Socket Programming

To understand the functionality of the transport layer we must understand how it interacts with the application layer and network layer. In this case we will focus on the application layer.

1. UDP: Unreliable datagram
    - UDP stands for User Datagram Protocol.

- There is no connection between client and server with UDP.
- There is no handshaking before sending data.
- Relative to the application UDP provides unreliable transfer of datagrams between client and server.
- An example in python of a UDP server is shown on slide 8.

2. TCP: Reliable datagram
   - TCP stands for Transmission Control Protocol.
   - The client must contact the server with TCP.
     - Server must have created a socket (door) that welcomes the client's contact.
   - TCP converts IP's unreliable service between end systems into a reliable data transport services.

Transport Services and Protocols

The transport layer provides logical communication between app processes running on different hosts.

- Logical communication means from an application's prospective it is as if the hosts running the processes were directly connected.

Transport protocols run in end systems and not in the network routers. This is shown below in Figure 2.

When contacted by the client the TCP server creates a new socket to communicate with said client.

- This solves the congestion problem and allows the server to talk with multiple clients at the same time.
- What distinguishes these clients are port numbers.
  - If you have taken CS3240 you may remember that you had to specify a port in your browser's address bar to connect to your locally hosted website.
- An example in python of a TCP server is shown on slide 13.

It should be noted that the transport layer should not be confused with the network layer. Just remember that the network layer facilitates logical communication between hosts while the transport layer facilitates logical communication between processes.

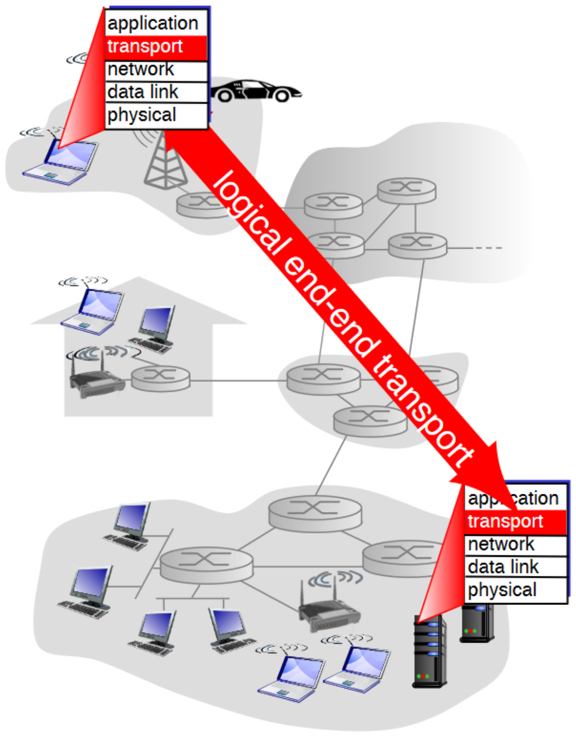

Figure 2: Logical End-End Transport

Multiplexing and Demultiplexing

In the transport layer the sender multiplexes in order to handle data from multiple sockets. The sender during this process adds a transport header that is later used for demultiplexing at the receiver. This process is visualized in Figure 3 below.

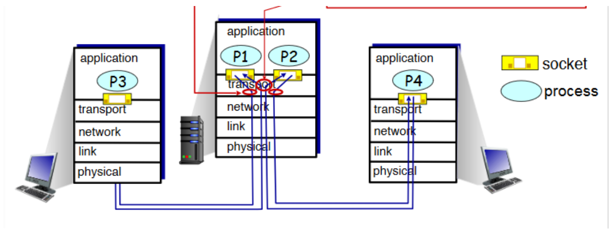

Figure 3: Transport Layer Multiplexing

Demultiplexing occurs at the host and begins when the host receives IP datagrams. In each datagram there is a source IP address and a destination IP address. Furthermore, each datagram also contains one transport-layer segment and each segment has within it a source, destination, and port number.

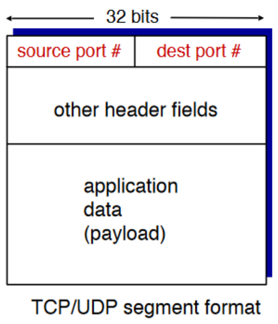

Figure 4: Datagram Demultiplexing

Demultiplexing is not limited to only TCP datagrams, rather demultiplexing is relevant for UDP datagrams as well. This is called connectionless demultiplexing. For TCP datagrams it is referred to as connection-oriented demultiplexing.

UDP: User Datagram Protocol

UDP can be described as a very bare bones internet transfer protocol. It is referred to as a "best effort" service and some segments may or may not be lost or delivered out of order.

As mentioned before UDP is a connectionless protocol that doesn't perform a handshake between the sender and the receiver. Each UDP protocol is handled independently of others.

Examples of UDP being used in real life are as follows:

- Streaming multimedia platforms
  - Streaming is naturally loss tolerant and rate sensitive. This means UDP is good for applications where dropping packets is better than waiting.
- DNS: Domain Name System
- SNMP: Simple Network Management Protocol

An example of a UDP segment header is shown below in Figure 5.

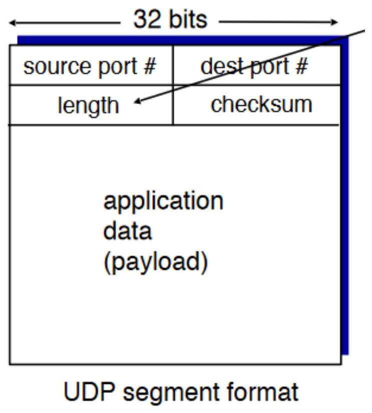

Figure 5: UDP Segment Header

Despite UDP's unreliability it does have a checksum that enables error detection.

The sender treats the segment contents as a sequence of 16 bit integers. The sender then performs addition, one complements sum, of segment contents.

Thereafter the sender puts the checksum vale into the UDP checksum field.

The receiver then simply computes the checksum on the segment contents and compares it to the checksum that is calculated by the sender. If they differ then an error has been detected.

Principles of Reliable Data Transfer

In the case of an unreliable channel between end-to-end process communication we can use a reliable data transfer protocol. The next lecture will describe the implementation of a reliable data transfer protocol in greater detail. For now, here is a visualization of the implementation.

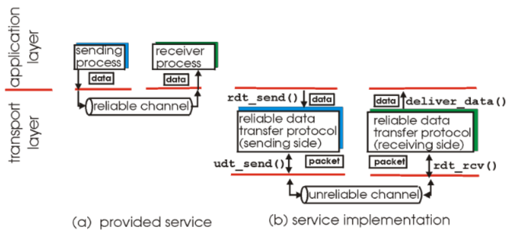

Figure 6: Reliable Data Transfer

A description of each of the methods shown in Figure 6 is shown in Figure 7.

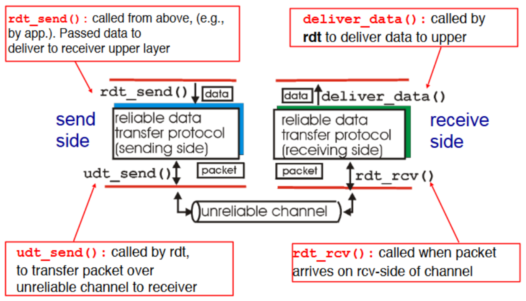
rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer
deliver_data(): called by rdt to deliver data to upper
rdt_send()
data
deliver_data()
send side
reliable data transfer protocol (sending side)
reliable data transfer protocol (receiving side)
receive side
udt_send()
packet
rdt_rcv()
unreliable channel
udt_send(): called by rdt, to transfer packet over unreliable channel to receiver
rdt_rcv(): called when packet arrives on rcv-side of channel

Figure 7: Reliable Data Transfer Method Details

We will implement a reliable data transfer protocol. In this implementation we will develop separate sender and receiver sides of the reliable data transfer protocol. Within both the sender and receiver side we will implement a finite state machine.

Within these finite state machines we will use acknolwedgements (ACKS) and negative acknowledgements (NAKS) to tell the sender the status of the packet sent. In particular, whether or not the checksum detected bit errors within the packet.