ECE 333: Introduction to Communication Networks
Fall 2001

Lecture 26: Transport layer I

Transport Layer

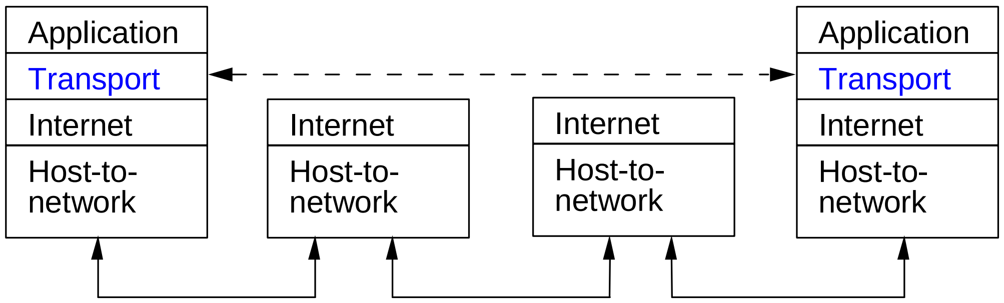
In the remaining lectures, we will discuss several issues that are commonly addressed at the transport layer in a network. We will consider several general transport layer issues, and, as a specific example, we will use the transport layer in the Internet. The Internet has two primary transport layer protocols – TCP and UDP – both of these will be discussed.

In the Internet architecture the transport layer sits between the application layer and the network layer. For this reason, it is the most visible layer to application designers. The service of the transport layer is to provide a virtual end-to-end "message-pipe" for applications. In other words, two applications on different hosts can communicate messages to each other as if they are directly connected – the details of the underlying network are hidden.

Transport Layer

The transport layer is an end-to-end layer – this means that nodes within the subnet do not participate in transport layer protocols – only the end hosts.

As with other layers, transport layer protocols send data as a sequence of packets. In TCP/IP these packets are called ***segments***.[1]



[1] Actually only TCP packets are called segments and UDP packets are referred to as datagrams – but we will use segments to refer to either of these.

Transport layer issues

Two main issues addressed at the transport layer are

- Multiplexing
- Improving the service offered by network layer

Multiplexing

The network layer provides communication between two hosts. The transport layer provides communication between two ***processes*** running on different hosts. A process is an instance of a program that is running on a host. There may be multiple processes communicating between two hosts – for example, there could be a FTP session and a Telnet session between the same two hosts. The transport layer provides a way to multiplex/demultiplex communication between various processes.

To provide multiplexing, the transport layer adds an address to each segment indicating the source and destination processes. Note these addresses need only be unique locally on a given host. In TCP/IP these transport layer addresses are called ***port-numbers***. In other protocols, different names are used transport layer addresses; for example, in ATM networks these are called AAL-SAP's (ATM Adaptation Layer –Service Access Points).

In the Internet port numbers are 16-bits long. The port numbers between 0 and 1023 are reserved and called **well-known ports.** These numbers are assigned to well-known applications such as FTP, Telnet, and HTTP. For example HTTP is assigned well-known port number 80.

Multiplexing

Most network applications are based on a ***client-server architecture*** – where one process, "the client," requests services from a second process, "the server." For example, a web browser implements the client side of HTTP and a web server implements the server side. The client initiates communication and the server responds.

A client sending a request to a web server would use the destination port 80, and in IP would indicate the transport layer protocol is TCP (this is the transport layer protocol used by HTTP). The sending port number is picked from some other port number that is not being used on the client. In this way if several different HTTP sessions are established between the two hosts, the sending port number distinguishes them. Thus, the 5-tuple: (sending port, sending IP address, destination port, destination IP address, transport layer protocol) uniquely identifies a process-to-process connection in the Internet.

The use of well-known ports is just one way for finding the destination address. Another example is to use a special process called a ***process or directory server***, which responds to requests for services with the correct port number.

Improving the Network layer service

When the underlying network layer does not offer adequate service for an application, it is the job of the transport layer to improve this service. For example, suppose the network layer does not guarantee that every packet will arrive. Then the transport layer could provide a way to recover from lost packets and ensure that each packet is delivered correctly.

As with the other layers, the service provided by the transport layer can have different characteristics. For example, it may be reliable or unreliable, connection-oriented or connectionless, etc. It could also try to guarantee a certain ***Quality of Service (QoS)***, i.e. a given data rate or delay.

The service provided by the underlying network layer will determine:

1. How much work has to be performed at the transport layer to provide a certain service.
2. What types of service can even be provided.

For example, consider designing a transport layer to offer a service that guarantees the delay of a message. If the underlying network layer already offers this, than little work would be needed at the transport layer. However, if the network layer did not provide any delay guarantees, then it may be impossible to provide this service at the transport layer.

Transport layer services in the Internet

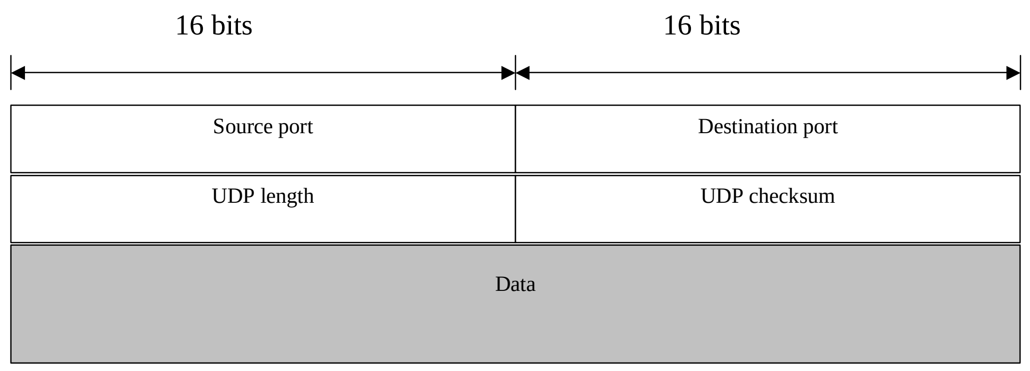
The network layer in the Internet provides an unreliable, connectionless datagram service. IP tries to deliver each datagram to the destination, but makes no guarantees, that the data will arrive correctly, in order or even at all. This is sometimes described as a “best effort” service.

In the Internet there are two transport layer protocols, each offering a different service. **UDP (User Datagram Protocol)** provides an unreliable, connectionless transport layer protocol. The second transport layer protocol, **TCP (Transport Control Protocol)**, provides a reliable, connection-oriented transport service. Some common applications and the corresponding transport layer protocol used are shown below:

Application protocol	Typical transport protocol
SMTP(e-mail)	TCP
DNS	UDP
HTTP (web browsing)	TCP
Internet Telephony	UDP
FTP	TCP
Telnet	TCP

UDP

The packet format for UDP is shown below. Basically all UDP provides is a multiplexing capability on top of IP. The length field gives the length of the entire packet including the header. The checksum is calculated on the entire packet (and also on part of the IP header). Calculating the checksum is optional, and if an error is detected the packet may be discarded or may be passed on to the application with an error flag.



TCP

TCP converts the unreliable, best effort service of IP into a reliable service, i.e., it ensures that each segment is delivered correctly, only once, and in order.

Converting an unreliable connection into a reliable connection is basically the same problem we have considered at the data link layer, and essentially the same solution is used. That is TCP numbers each segment and uses an ARQ protocol to recover lost segments. Some versions of TCP implement Go Back N and other versions implement Selective Repeat.

However, there are a few important differences between the transport layer and the data link layer. At the data link layer, we viewed an ARQ protocol as being operated between two nodes connected by a point-to-point link. At the transport layer, this protocol is implemented between two hosts connected over network. In the later case, packets can arrive out-of-order, and packets may also be stored in buffers within the network and then arrive at much later times. Also the round-trip time will change with different connections and connection establishment is more complicated.

Connection establishment

In TCP, a connection establishment phase is required, to ensure that the receiving process is available and to synchronize sequence numbers, etc. Connection establishment is a basic problem that arises in many other places as well. We look at the generic problem of connection establishment first.

Consider the following simple connection establishment protocol

1. The client transmits a special packet requesting a connection.
2. The server responds to a connection request with an accept packet.
3. After receiving the accept packet the client and server begin sending data using some ARQ protocol with initial sequence numbers of zero.
4. If the client does not receive a reply to a connection request after a certain time, it times-out and retransmits the request.

This simple scenario is complicated by the fact that the network can lose, store and re-order packets.

Example: Suppose the client wants to send one data segment. First it requests a connection, then sends the data and closes the connection. Suppose the client times out after doing each of these steps and then retransmits. This results in the following sequence.

Client Transmits:

CONNECT CONNECT DATA(0) DATA(O) CLOSE CLOSE

Since data can be reordered in the network, the server could receive:

CONNECT DATA(0) CLOSE CONNECT DATA(0) CLOSE

In this case the server may think that the client opened two sessions and sent two different data packets.

Similar problems can arise if a client crashes and then re-opens a connection, while a packet from a previous connection is still in the network.

We will consider some possible solutions to these problems next.