

Transport Services and Protocols

Transport Services and Protocols

- The **transport layer** is responsible for providing **logical communication** between **application processes** running on **different hosts**.
- There are two actions the transport layer needs to achieve:
 1. **Send action**: The transport layer needs to **divide application messages into segments**, and pass them to the **network layer**.
 2. **Receive action**: The transport layer needs to **reassemble segments** from the **network layer** into **messages**, and pass them to the **application layer**.
- There are **two transport protocols** available: the **User Datagram Protocol (UDP)**, and the **Transmission Control Protocol (TCP)**.

The Application Layer, The Transport Layer, and The Network Layer

- The **application layer** is the **process** that is running on the **network host**.
- The **transport layer** is responsible for the **logical communication** between **processes**.
- The **network layer** is responsible for the **logical communication** between **network hosts**.
- When a **message is sent** from an application over the network, it follows the following order: application layer → transport layer → network layer.
- When a **message is received** from the network, and sent to an application, it follows the following order: network layer → transport layer → application layer.

Transport Layer Actions

- When a message is **sent from an application**, the **transport layer** does the following:
 1. Receives the application-layer message.
 2. Determines segment header field values.
 3. Creates the segment.
 4. Passes the segment to the internet protocol.
- When a message is **received from the network**, the **transport layer** does the following:
 1. Receives the segment from the internet protocol.
 2. Checks the header values.
 3. Extracts the application-layer message.
 4. Demultiplexes message up to the application via a socket.

User Datagram Protocol and Transmission Control Protocol

- The **User Datagram Protocol (UDP)** is an **unreliable, unordered** delivery protocol. This protocol has **minimal overhead** but is **not reliable**.
- The **Transmission Control Protocol (TCP)** is a **reliable, in-order delivery** protocol that supports **congestion control**, **flow control**, and **connection setup**. This protocol has **significant overhead**, but is **very reliable**.
- Both **UDP** and **TCP** do not provide **delay guarantees** and **bandwidth guarantees**.

Multiplexing

Multiplexing

- **Multiplexing** is a method used by **networks** to **consolidate multiple signals** into a **single composite signal** that is then **transported over a common medium**.
- When **sending data**, **multiplexing** is used to **transmit segments** from **several sockets** via **transport headers**.
- When **receiving data**, the **header info** is used to **demultiplex** the data, sending it to the **correct socket**.
- **Demultiplexing** works by receiving **IP datagrams** containing a **source and destination IP address and port as headers**. Each datagram contains a **single segment**.
 - The IP addresses and port numbers are used to route the segment to the correct socket.
 - This is why you must specify a port (and sometimes an address) number when creating a socket.

Connectionless Multiplexing

- When creating a **server socket** you must specify a **port**, that the **socket will be bound to**.
- When creating a **datagram (UDP)** you must specify the **destination host and port**.
- When a **host** receives a **datagram** with the **port destination** that matches the **port the server socket is bound to**, it will **route the datagram to said socket**.
 - The host address does not matter when a datagram is received by a host, it will simply attempt to route it to the destination port. This means that several hosts can send datagrams to the same port, and they will all be available at the same socket.

Connection-Oriented Multiplexing

- When creating a **connection oriented socket (TCP)**, you create a **4-tuple** containing the **source address, source port, destination address, and destination port**.
- The **4-tuple** is used to **route inbound packets** to the **correct socket**.
- **Servers** may support **several simultaneous TCP sockets**, typically one for **each client**. Each socket will have a **unique 4-tuple**.

Summary

- Multiplexing and demultiplexing are based on segment, datagram header values.
- UDP uses the port for demultiplexing.
- TCP uses the 4-tuple for demultiplexing.
- Multiplexing happens at all layers.

User Datagram Protocol

User Datagram Protocol

- The **User Datagram Protocol (UDP)** is a **connectionless (no handshake) communication protocol** that uses the **internet protocol**.
- UDP has **minimal overhead**, at the cost of having **no connection state, no congestion control, and no packet delivery verification**.
- UDP is used for **loss-tolerant, rate sensitive** applications.
- It is possible to have **reliable UDP communication** by implementing it at the **application layer**.

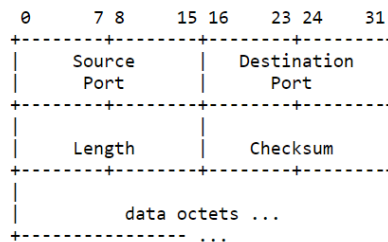
User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



User Datagram Header Format

- **UDP datagrams** have the following headers:
 1. **Destination port** — The port the datagram is being sent to.
 2. **Length** — The length in octets of the datagram.
 3. **Checksum** — The checksum is used to ensure the packet received was not corrupted during transmission.
 4. **Pseudo** — A header containing the source address, destination address, the protocol, and the UDP length. This information is used to give protection against misrouted datagrams.
- The **checksum** is computed by taking the **one's complement sum** of the **entire content** of the **datagram**, treating the bits of the datagram as a **series of 16-bit integers**.
 - The checksum is not perfect, several different datagrams can result in the same checksum.

Reliable Data Transfer