IP MULTICASTING

IP Multicasting enables an IP packet to be sent to more than one (remotely located) receiver simultaneously. This is necessary and useful in implementing replicated databases, distributed systems, and web-casting. The IP protocol supports multicasting via the use of class D addresses, each of which is meant to identify a group of hosts. A multicast router, connected to a LAN, distinguishes a Multicast Group address by the fact that its leading bits are 1110. The remaining 28 bits are interpreted as a Multicast Group Id. Some groups may be permanent, in which case no prior setup procedure is involved, and, the hosts belonging to the group are clearly identifiable. E.g. the group address 224.0.0.1 (i.e. 11100000.0.0.1) includes all the systems connected to the LAN which has a Multicast Router on it, while 224.0.0.2 includes all the routers. A temporary group address represents a collection of IP addresses (non-Multicast) identifying the hosts that wish to receive IP packets addresses to the group address. All hosts on a LAN are periodically asked by the LAN’s Multicast Router to update it about the Multicast group addresses (one or more) in which they wish to be added, removed, or remain included. This group information is exchanged using IGMP (Internet Group Management Protocol) and enables the association of hosts with group addresses to be managed dynamically. The routing of IP packets addressed to Multicast groups is handled by special Multicast Routers, which use an extension of the OSPF protocol in combination with tunneling techniques, to route such packets to remote clusters of hosts, without congesting the intermediate networks. Like any other IP packets, such packets may be lost, delayed, duplicated or delivered out of order. Note that a Multicast address can only be specified as a Destination address within an IP header; the source must be a (single) host IP address.

(Review sec 5.2.8 on Multicast routing –self study)

Tunnel

Origin-ating Host

MR

MR

MR

MR

a

a

a

a

IP

SNA

IP

IP

b

b

b

b

b

MR

TRANSPORT LAYER: UDP< TCP CONCEPTS

The main purpose of the transport layer is to provide a reliable, efficient and cost-effective data transport service to its users, usually processes in the application layer, so that data can be sent from a source to a destination machine. The transport layer provides connection-oriented (CO) and connection-less (CL) services on top of the underlying network layer whose services it uses. However, the transport layer code runs entirely on the user’s machines, while network layer code runs mostly on routers belonging to a WAN, over which the end-users have little control. This fact necessitates the existence of the transport layer which makes it possible for the transport service to be reliable than the underlying network service, and allows the implementation of the transport service primitives as calls to library procedures (on the host machine) independent of the network service primitives. In this sense the transport layer forms the major boundary between the *transport provider* layers (i.e. 1 to 4 in the OSI model) and the tra*nsport user* layers (i.e. 5 to 7).

Fig 6-1 depicts this interfacing between the application, transport and network layers. The software within the transport layer that implements these primitives is referred to as the *Transport Entity*. A commonly-used set of transport layer primitives is provided by the Berkeley Socket API for Network programming (both CL and CO services).

See section 6.1.3 for a brief description of socket API (self study) refer updates table for socket and related primitives (CN-1) -\*-.

In order to support these primitives for CO and CL service, the Transport Layer provides two protocols.

1. UDP (User Datagram Protocol, or, Unreliable Delivery Protocol) which is CL.
2. TCP (Transmission Control Protocol) which is CO.

UDP: This CL transport protocol provides a simple way for application processes to exchange IP packets containing user-defined data, in such a way that the sending and receiving processes can be clearly identified even when multiple sending (or receiving) processes are using the same IP address (i.e. running on the same machine). UDP does not require the sending and receiving processes to establish a connection, while the packet is being exchanged, but does provide them confirmation of success. The UDP protocol exchanges data in the form of segments, each of which has an 8 byte header followed by payload data. The header identifies the sending and receiving processes by source and destination port numbers, which identify the specific processes, on the sending and receiving machines, to which the Transport Layer must deliver the segment or send back the reply to. A UDP segment is embedded within a standard IP packet, the transmission of which is handled by the network layer. UDP is particularly useful in client-server implementations which involve exchange of short requests and replies. DNS, which maps host names to IP addresses, uses UDP.

TCP: The TCP protocol provides a reliable end-to-end byte stream over an unreliable internetwork, which may consist of different topologies, bandwidth and delay characteristics, and packet sizes. TCP is mainly used by Internet applications for which reliable, sequenced delivery is needed. The Transport-Entity that supports TCP accepts user data streams from processes running on the sending host, breaks them into IP packets which may be theoretically as large as 64KB, but are limited in practice to the MTU of the underlying Datalink layer. (MTU=1500 for Ethernet).

On the receiving host, a Transport-Entity supporting TCP, reconstructs the original data streams (from the packets received) and hands them over to the appropriate local processes. The TCP protocol and the socket API for using TCP, make the entire process of breaking up and reconstructing the data stream, invisible to the sending and receiving processes. They simply read from and write into a *full duplex*, *point-to-point* TCP connection, which resembles a *file descriptor*.

The Transport Entities handling TCP exchange data in the form of segments. A TCP segment has a 20 byte header followed by variable number of payload bytes. The header contains source and destination port numbers as in UDP. See fig 6-29-\*-.

Communicating Transport Entities determine the payload size; may accumulate data from several “writes” into a single segment, or, break a single write into multiple segments. A segment is transported within an IP packet or fragment. The Transport Entities involved at two ends of the connection handle segment retransmission and reassembly.

SOCKETS

Sending and Receiving processes obtain UDP and TCP services, on their respective hosts, by creating communication end-points called sockets. A socket is identified by a socket number (similar to a file descriptor). A user process can explicitly associate a socket with an IP address (32 bit) and a port number (16 bit), or, the OS may do so implicitly, whenever required.

PORT NUMBERS

The addition of port numbers, to which processes can attach using the socket API BIND primitive, is a key feature that UDP and TCP add to IP. This feature allows a UDP or TCP based server process to receive requests from clients at an advertised port number. Port Numbers in the range 0→1023 are reserved for well-known service (see fig 6-27).

Ports 1024→49151 may be used for user-defined services after registering with IANA (Internet Assigned Numbers Authority). Ports 49152→65535 are dynamic or ephemeral ports which are granted by the OS temporarily to a client process (TCP or UDP based) when it makes a request to send a data packet (using SENDTO in UDP), or, it attempts to establish an end-to-end byte-stream (using CONNECT in TCP).

For a detailed understanding of the socket API refer to pages 415-425 (chapter 22, Comer –vol1)read it! IN class go over this handout CN-2)-\*-

Explain section 6.1.4 updated client and server distribute updated client code.

LAB study section 6.1.4 andentre code for TCP-based implementation of a simple FTP program, provided in the book. Compile and test server and client code

Client side app.dev, using sockets. -\*-

NETWORK BYTE ORDER

In order to create an internet that is independent of any particular vendor’s machine architecture, or, network hardware, networking software must define a standard representation for data. Whenever software on one computer sends a binary integer to another, the physical layer hardware moves the sequence of bits from the first machine to the second without changing the order. However, all architectures do not store 32-bit integers in the same way. On Little-Endian machines, the lowest memory address contains the low-order byte of the integer, while on Big-Endian ones, the lowest memory address holds the high-order byte of the integer. Thus, direct copying of bytes, from one machine to another, may change the value of the number. Standardization of byte-order for integer is especially important because IP packets carry binary numbers that specify destination addresses, packet lengths etc. which must be understood by both the senders and receivers. The TCP/IP protocol defines a Network Standard Byte Order that All machines must use for binary fields in IP packets.

Each machine must convert binary data from the machine-specific byte-order to network standard byte-order before sending a packet and back to host-specific byte-order when receiving a packet.

Although this conversion is not required for user-defined payload-data, application programmers often adhere to this convention when sending integer values representing application-specific data.

The Network Standard Byte Order specifies that integers are sent with the most significant byte first, i.e. Big-Endian style.

Memory Address:

A

A + 1

A + 2

A + 3

Data Bytes:

B2

B1

B4

B3

Beginning of Packet

End of Packet

Considering the successive bytes in a packet, as it travels from one machine to another, a binary integer in that packet has its most significant byte nearest the beginning of the packet and its least significant byte nearest the end of the packet.

Network Byte Order Conversion Routines:

See handout CN-2 refer man pages for ntohs, ntohl, htons, htonl) -\*-

FIREWALLS

An AS, or an area, or even a single physical network may secure itself by using a Firewall, which uses some combination of hardware and software in the form of a specialized router or gateway. In general, a firewall serves two main goals:

1. To limit access to the network (and therefore the intranet or websites it supports) from the rest of the Internet (In-bound Access Limits).
2. To limit services or sites accessed from your LAN or Intranet (Out-bound Access Limits).

Three types of firewalls are commonly used:

1. Packet Filter: Examines each packet (in-bound, out-bound). The problems with this approach are the complexity involved in configuration and the slow speed.
2. Application Gateway: Security mechanisms are applied to specific applications e.g. FTP, TELNET, HTTP, SMTP.
3. Proxy Server: Hides true network address – the actual IP address of the node or intranet connection point cannot be determined externally. Also used to provide limited access to Internet from within an organization’s Intranet.

A commonly used configuration for a firewall consists of an application gateway, which can handle multiple applications, sandwiched between two routers which serve to filter incoming and outgoing packets, as depicted below:

Application Gateway

Incoming Packet Filter

Outgoing Packet Filter

Connection To Internet

H

H

H

Secured Network

FIREWALL

ENCRYTION TECHNIQUES

Traditional cryptography, which is also called Private Key Cryptography, uses only one secret key for both encryption & decryption. The key, which is often a n (e.g. 64, 128) bit number, is an input to a publicly known encryption/decryption algorithm. The key is known only to the sender and receiver of the encrypted message. The secret key is usually exchanged by using a separate method e.g. the *Diffie-Hellman* exchange, a protocol that allows strangers to establish a shared secret key. Examples of Private Key Cryptography include DES (Data Encryption Standard) and IDEA (International Data Encryption Algorithm). UNIX provides a very simple encryption utility, *crypt*, for educational purposes.

Active Intruder (can *modify* message)

Passive Intruder (can *read* message)

ENCRYPTION ALGORITHM

DECRYPTION ALGORITHM

Plain Text, P

Plain Text, P = DK (C)

Cipher Text C = EK (P)

Decryption Key, K

Encryption Key, K

DK (EK (P)) ≡ P

Modern Cryptography, which is also called Public Key Cryptography, uses two keys:

1. A public key used by any sender for encrypting messages to be sent to the receiver. This key is published by the receiver.
2. A private key which is only known to the receiver and is used by the receiver to decrypt received messages.

Even though the public and private keys are distinct, they are related. But, it is computationally difficult to derive the private key from the public one. The most widely used industry standard for modern cryptography is the RSA Algorithm developed at MIT and licensed by RSA Data Security. This standard is used for data encryption by browsers such as Netscape Navigator and Internet Explorer.