INTERNETWORKING TUTORIAL

Juan-Pablo Cáceres
Network Sound and Data
Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
What is a NETWORK?
1.2 Requirements

This section attempts to distill these different perspectives into a high-level introduction to the major considerations that drive network design, and in doing so, identifies the challenges addressed throughout the rest of this book.

1.2.1 Connectivity

Starting with the obvious, a network must provide connectivity among a set of computers. Sometimes it is enough to build a limited network that connects only a few select machines. In fact, for reasons of privacy and security, many private (corporate) networks have the explicit goal of limiting the set of machines that are connected. In contrast, other networks (of which the Internet is the prime example) are designed to grow in a way that allows them the potential to connect all the computers in the world. A system that is designed to support growth to an arbitrarily large size is said to scale. Using the Internet as a model, this book addresses the challenge of scalability.

Links, Nodes, and Clouds

Network connectivity occurs at many different levels. At the lowest level, a network can consist of two or more computers directly connected by some physical medium, such as a coaxial cable or an optical fiber. We call such a physical medium a link, and we often refer to the computers it connects as nodes. (Sometimes a node is a more specialized piece of hardware rather than a computer, but we overlook that distinction for the purposes of this discussion.) As illustrated in Figure 1.2, physical links are sometimes limited to a pair of nodes (such a link is said to be point-to-point), while in other cases, more than two nodes may share a single physical link (such a link is said to be multiple-access). Whether a given link supports point-to-point or multiple-access connectivity depends on how the node is attached to the link. It is also the case that multiple-access links are often limited in size, in terms of both the geographical distance they can cover and the number of nodes they can connect. The exception is a satellite link, which can cover a wide geographic area.

Figure 1.2 Direct links: (a) point-to-point; (b) multiple-access.
If computer networks were limited to situations in which all nodes are directly connected to each other over a common physical medium, then either networks would be very limited in the number of computers they could connect, or the number of wires coming out of the back of each node would quickly become both unmanageable and very expensive. Fortunately, connectivity between two nodes does not necessarily imply a direct physical connection between them—indirect connectivity may be achieved among a set of cooperating nodes. Consider the following two examples of how a collection of computers can be indirectly connected.

Figure 1.3 shows a set of nodes, each of which is attached to one or more point-to-point links. Those nodes that are attached to at least two links run software that forwards data received on one link out on another. If organized in a systematic way, these forwarding nodes form a switched network. There are numerous types of switched networks, of which the two most common are circuit switched and packet switched. The former is most notably employed by the telephone system, while the latter is used for the overwhelming majority of computer networks and will be the focus of this book.

The important feature of packet-switched networks is that the nodes in such a network send discrete blocks of data to each other. Think of these blocks of data as corresponding to some piece of application data such as a file, a piece of email, or an image. We call each block of data either a packet or a message, and for now we use these terms interchangeably; we discuss the reason they are not always the same in Section 1.2.2. Packet-switched networks typically use a strategy called store-and-forward. As the name suggests, each node in a store-and-forward network first receives a complete packet and then switches it to an output link. This process can repeat itself as many times as necessary until the packet reaches its destination.
Internetwork

The cloud in Figure 1.3 distinguishes between the nodes on the inside that implement the network (they are commonly called switches, and their sole function is to store and forward packets) and the nodes on the outside of the cloud that use the network (they are commonly called hosts, and they support users and run application programs). Also note that the cloud in Figure 1.3 is one of the most important icons of computer networking. In general, we use a cloud to denote any type of network, whether it is a single point-to-point link, a multiple-access link, or a switched network. Thus, whenever you see a cloud used in a figure, you can think of it as a placeholder for any of the networking technologies covered in this book.

A second way in which a set of computers can be indirectly connected is shown in Figure 1.4. In this situation, a set of independent networks (clouds) are interconnected to form an internetwork, or internet for short. We adopt the Internet's convention of referring to a generic internetwork of networks as a lowercase internet, and the clouds router

address

Figure 1.4 Interconnection of networks.
Typical Campus Network Infrastructure

WAN

Router

Backbone

Router

Router

Router

Subnet

Subnet

Subnet
Global Network Infrastructure

Campus Network

WAN

Campus Network

Campus Network

Campus Network
Simple Internetworking

One way to think of IP is that it runs on all the nodes (both hosts and routers) in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single logical internetwork. For example, Figure 4.2 shows how hosts H1 and H8 are logically connected by the internet in Figure 4.1, including the protocol graph running on each node. Note that higher-level protocols, such as TCP and UDP, typically run on top of IP on the hosts.

Most of the rest of this chapter is about various aspects of IP. While it is certainly possible to build an internetwork that does not use IP—for example, Novell created an internetworking protocol called IPX, which was in turn based on the XNS internet designed by Xerox—IP is the most interesting case to study simply because of the size of the Internet. Said another way, it is only the IP Internet that has really faced the issue of scale. Thus it provides the best case study of a scalable internetworking protocol.
Demo: traceroute
Given a collection of nodes indirectly connected by a nesting of networks, it is possible for any pair of hosts to send messages to each other across a sequence of links and nodes. Of course, we want to do more than support just one pair of communicating hosts—we want to provide all pairs of hosts with the ability to exchange messages. The question then is, How do all the hosts that want to communicate share the network, especially if they want to use it at the same time? And, as if that problem isn't hard enough, how do several hosts share the same link when they all want to use it at the same time?

To understand how hosts share a network, we need to introduce a fundamental concept, multiplexing, which means that a system resource is shared among multiple users. At an intuitive level, multiplexing can be explained by analogy to a timesharing computer system, where a single physical CPU is shared (multiplexed) among multiple jobs, each of which believes it has its own private processor. Similarly, data being sent by multiple users can be multiplexed over the physical links that make up a network.

To see how this might work, consider the simple network illustrated in Figure 1.5, where the three hosts on the left side of the network (L1–L3) are sending data to the three hosts on the right (R1–R3) by sharing a switched network that contains only one physical link. (For simplicity, assume that host L1 is communicating with host R1, and so on.) In this situation, three flows of data—corresponding to the three pairs of hosts—are multiplexed onto a single physical link by switch 1 and then demultiplexed back into separate flows by switch 2. Note that we are being intentionally vague about exactly what a “flow of data” corresponds to. For the purposes of this discussion, assume that each host on the left has a large supply of data that it wants to send to its counterpart on the right.

There are several different methods for multiplexing multiple flows onto one physical link. One common method is synchronous time-division multiplexing (STDM). The idea of STDM is to divide time into equal-sized quanta and, in a round-robin manner, allocate time to each user in turn.
Switch Multiplexing Packets

The decision as to which packet to send next on a shared link can be made in a number of different ways. For example, in a network consisting of switches interconnected by links such as the one in Figure 1.5, the decision would be made by the switch that transmits packets onto the shared link. (As we will see later, not all packet-switched networks actually involve switches, and they may use other mechanisms to determine whose packet goes onto the link next.) Each switch in a packet-switched network makes this decision independently, on a packet-by-packet basis. One of the issues that faces a network designer is how to make this decision in a fair manner. For example, a switch could be designed to service packets on a first-in-first-out (FIFO) basis. Another approach would be to service the different flows in a round-robin manner, just as in STDM. This might be done to ensure that certain flows receive a particular share of the link’s bandwidth, or that they never have their packets delayed in the switch for more than a certain length of time. A network that allows flows to request such treatment is said to support quality of service (QoS).

Also, notice in Figure 1.6 that since the switch has to multiplex three incoming packet streams onto one outgoing link, it is possible that the switch will receive packets faster than the shared link can accommodate. In this case, the switch is forced to buffer these packets in its memory. Should a switch receive packets faster than it can send them for an extended period of time, then the switch will eventually run out of buffer space, and some packets will have to be dropped. When a switch is operating in this state, it is said to be congested.
Identifying Common Communication Patterns

Designing abstract channels involves first understanding the communication needs of a representative collection of applications, then extracting their common communication requirements, and finally incorporating the functionality that meets these requirements in the network.

One of the earliest applications supported on any network is a file access program like FTP (File Transfer Protocol) or NFS (Network File System). Although many details vary—for example, whether whole files are transferred across the network or only single blocks of the file are read/written at a given time—the communication...
Open Systems Interconnection (OSI) Architecture

The OSI was one of the first organizations to formally define a common way to connect computers. Their architecture, called the **Open Systems Interconnection (OSI) architecture** and illustrated in Figure 1.13, defines a partitioning of network functionality into

- **Application**
- **Presentation**
- **Session**
- **Transport**
- **Network**
- **Data link**
- **Physical**

**End host**

One or more nodes within the network

**Intermediate switches and routers**
While the seven-layer OSI model can, with some imagination, be applied to the Internet, a four-layer model is often used instead. At the lowest level are a wide variety of network protocols, denoted NET\textsubscript{1}, NET\textsubscript{2}, and so on. In practice, these protocols are implemented by a combination of hardware (e.g., a network adaptor) and software (e.g., a network device driver). For example, you might find Ethernet or Fiber Distributed Data Interface (FDDI) protocols at this layer. (These protocols in turn may actually involve several sublayers, but the Internet architecture does not presume anything about them.) The second layer consists of a single protocol—the Internet Protocol (IP). This is the protocol that supports the interconnection of multiple networking technologies into a single, logical internetwork. The third layer contains two main protocols—the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP and UDP provide alternative logical channels to application hardware/software.
## Datagram Delivery and Packer Format (IPv4)

<table>
<thead>
<tr>
<th>Field</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Specifies the IP version. The current version is 4, and it is sometimes called IPv4.</td>
</tr>
<tr>
<td>HLen</td>
<td>Specifies the length of the header in 32-bit words. When there are no options, which is most of the time, the header is 5 words (20 bytes) long.</td>
</tr>
<tr>
<td>TOS</td>
<td>8-bit (type of service) field has had a number of different definitions over the years, but its basic function is to allow packets to be treated differently based on application needs. For example, the TOS value might determine whether or not a packet should be placed in a special queue that receives low delay. We discuss the use of this field (and a new name for it) in more detail in Section 6.5.3.</td>
</tr>
<tr>
<td>Ident</td>
<td>SourceAddr</td>
</tr>
<tr>
<td>Flags</td>
<td>DestinationAddr</td>
</tr>
<tr>
<td>Offset</td>
<td>Options (variable)</td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
</tr>
<tr>
<td>Protocol</td>
<td>Checksum</td>
</tr>
<tr>
<td>Checksum</td>
<td>Pad (variable)</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.4 IP datagrams traversing the sequence of physical networks graphed in Figure 4.1.

is 1500 bytes for the two Ethernets, 4500 bytes for the FDDI network, and 532 bytes for the point-to-point network, then a 1420-byte datagram (20-byte IP header plus 1400 bytes of data) sent from H1 makes it across the first Ethernet and the FDDI network without fragmentation but must be fragmented into three datagrams at router R2. These three fragments are then forwarded by router R3 across the second Ethernet to the destination host. This situation is illustrated in Figure 4.4. This figure also serves to reinforce two important points:

1. Each fragment is itself a self-contained IP datagram that is transmitted over a sequence of physical networks, independent of the other fragments.
2. Each IP datagram is reencapsulated for each physical network over which it travels.

The fragmentation process can be understood in detail by looking at the header fields of each datagram, as is done in Figure 4.5. The unfragmented packet, shown at the top, has 1400 bytes of data and a 20-byte IP header. When the packet arrives at router R2, which has an MTU of 532 bytes, it has to be fragmented. A 532-byte MTU leaves 512 bytes for data after the 20-byte IP header, so the first fragment contains 512 bytes of data. The router sets the M bit in the Flags field (see Figure 4.3), meaning that there are more fragments to follow, and it sets the Offset to 0, since this fragment contains the first part of the original datagram. The data carried in the second fragment starts with the 513th byte of the original data, so the Offset field in this header is set to 64, which is 512 \(÷\) 8. Why the division by 8? Because the designers of IP decided that fragmentation should always happen on 8-byte boundaries, which means that the Offset field counts 8-byte chunks, not bytes. (We leave it as an exercise for you to figure out why this design decision was made.) The third fragment contains the last...
IP Global Addresses (32 bits)

Network Host

(a)  

Network Host

(b)  

Network Host

(c)  

Classes | Network Sizes

<table>
<thead>
<tr>
<th>Classes</th>
<th>Network Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>7 bits for network, 24 bits for host</td>
</tr>
<tr>
<td>Class B</td>
<td>14 bits for network, 16 bits for host</td>
</tr>
<tr>
<td>Class C</td>
<td>21 bits for network, 8 bits for host</td>
</tr>
</tbody>
</table>

On the face of it, this addressing scheme has a lot of flexibility, allowing networks of vastly different sizes to be accommodated fairly efficiently. The original idea was that the Internet would consist of a small number of wide area networks (these would be class A networks), a modest number of site- (campus-) sized networks (these would be class B networks), and a large number of LANs (these would be class C networks).

However, as we shall see in Section 4.3, additional flexibility has been needed, and some innovative ways to provide it are now in use. Because one of these techniques actually removes the distinction between address classes, the addressing scheme just described is now known as “classful” addressing to distinguish it from the newer “classless” approach.

Before we look at how IP addresses get used, it is helpful to look at some practical matters, such as how you write them down. By convention, IP addresses are written in dotted decimal notation.
The Address Resolution Protocol, ARP, allows a host to find the physical address of a target host on the same physical network, given only the target's IP address.

Figure 5.1 The ARP protocol. To determine PB, B's physical address, from IB, its IP address, (a) host A broadcasts an ARP request containing IB to all machines on the net, and (b) host B responds with an ARP reply that contains the pair (IB, PB).

5.6 The Address Resolution Cache It may seem silly that for A to send a packet to B it first sends a broadcast that reaches B. Or it may seem even sillier that A broadcasts the question, "how can I reach you?" instead of just broadcasting the packet it wants to deliver. But there is an important reason for the exchange. Broadcasting is far too expensive to be used every time one machine needs to transmit a packet to another because every machine on the network must receive and process the broadcast packet.
Localhost (loopback)

127.0.0.1
End-to-End Protocols

Figure 1.14 Internet protocol graph.

Figure 1.15 Alternative view of the Internet architecture.

While the seven-layer OSI model can, with some imagination, be applied to the Internet, a four-layer model is often used instead. At the lowest level are a wide variety of network protocols, denoted NET\(^1\), NET\(^2\), and so on. In practice, these protocols are implemented by a combination of hardware (e.g., a network adaptor) and software (e.g., a network device driver). For example, you might find Ethernet or Fiber Distributed Data Interface (FDDI) protocols at this layer. (These protocols in turn may actually involve several sublayers, but the Internet architecture does not presume anything about them.) The second layer consists of a single protocol—the Internet Protocol (IP). This is the protocol that supports the interconnection of multiple networking technologies into a single, logical internetwork. The third layer contains two main protocols—the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP and UDP provide alternative logical channels to application...
The simplest possible transport protocol is one that extends the host-to-host delivery service of the underlying network into a process-to-process communication service. There are likely to be many processes running on any given host, so the protocol needs to add a level of demultiplexing, thereby allowing multiple application processes on each host to share the network. Aside from this requirement, the transport protocol adds no other functionality to the best-effort service provided by the underlying network. The Internet's User Datagram Protocol (UDP) is an example of such a transport protocol.

The only interesting issue in such a protocol is the form of the address used to identify the target process. Although it is possible for processes to directly identify each other with an OS-assigned process id (pid), such an approach is only practical in a closed distributed system in which a single OS runs on all hosts and assigns each process a unique id. A more common approach, and the one used by UDP, is for processes to indirectly identify each other using an abstract locator, often called a port or mailbox. The basic idea is for a source process to send a message to a port and for the destination process to receive the message from a port.

The header for an end-to-end protocol that implements this demultiplexing function typically contains an identifier (port) for both the sender (source) and the receiver (destination) of the message. For example, the UDP header is given in Figure 5.1. Notice that the UDP port field is only 16 bits long. This means that there are up to 64K possible ports, clearly not enough to identify all the processes on all the hosts in the Internet. Fortunately, ports are not interpreted across the entire Internet, but only on a single host. That is, a process is really identified by a port on some particular host—a ⟨port, host⟩ pair. In fact, this pair constitutes the demultiplexing key for the UDP protocol.

The next issue is how a process learns the port for the process to which it wants to send a message. Typically, a client process initiates a message exchange with a server.

---

**User Datagram Protocol (UDP)**

**Unreliable Datagrams (like postal mail)**
- no acknowledgment
- ports to distinguish between applications

![UDP Header Diagram](image-url)
UDP (Comer’s definition)

The User Datagram Protocol (UDP) provides an unreliable connectionless delivery service using IP to transport messages between machines. It uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer.
For the protocols we have examined, encapsulation means that UDP prepends a header to the data that a user sends and passes it to IP. The IP layer prepends a header to what it receives from UDP. Finally, the network interface layer embeds the datagram in a frame before sending it from one machine to another. The format of the frame depends on the underlying network technology. Usually, network frames include an additional header.

On input, a packet arrives at the lowest layer of network software and begins its ascent through successively higher layers. Each layer removes one header before passing the message on, so that by the time the highest level passes data to the receiving process, all headers have been removed. Thus, the outermost header corresponds to the lowest layer of protocol, while the innermost header corresponds to the highest protocol layer. When considering how headers are inserted and removed, it is important to keep in mind the layering principle. In particular, observe that the layering principle applies to UDP, so the UDP datagram received from IP on the destination machine is identical to the datagram that UDP passed to IP on the source machine. Also, the data that UDP delivers to a user process on the receiving machine will be exactly the data that a user process passed to UDP on the sending machine.

The division of duties among various protocol layers is rigid and clear; the IP layer is responsible only for transferring data between a pair of hosts on an internet, while the UDP layer is responsible only for differentiating among multiple sources or destinations within one host. Thus, only the IP header identifies the source and destination hosts; only the UDP layer identifies the source or destination ports within a host.

**Encapsulation**
Ports and Demultiplexing

Figure 5.2 UDP message queue.

Packets arrive between the correct two endpoints. For example, if the destination IP address was modified while the packet was in transit, causing the packet to be misdelivered, this fact would be detected by the UDP checksum.

5.2 Reliable Byte Stream (TCP)

In contrast to a simple demultiplexing protocol like UDP, a more sophisticated transport protocol is one that offers a reliable, connection-oriented, byte-stream service. Such a service has proven useful to a wide assortment of applications because it frees the application from having to worry about missing or reordered data. The Internet's Transmission Control Protocol (TCP) is probably the most widely used protocol of this type; it is also the most carefully tuned. It is for these two reasons that this section studies TCP in detail, although we identify and discuss alternative design choices at the end of the section.

In terms of the properties of transport protocols given in the problem statement at the start of this chapter, TCP guarantees the reliable, in-order delivery of a stream of bytes. It is a full-duplex protocol, meaning that each TCP connection supports a
Reserved Ports

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Keyword</th>
<th>UNIX Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>ECHO</td>
<td>echo</td>
<td>Echo</td>
</tr>
<tr>
<td>9</td>
<td>DISCARD</td>
<td>discard</td>
<td>Discard</td>
</tr>
<tr>
<td>11</td>
<td>USERS</td>
<td>systat</td>
<td>Active Users</td>
</tr>
<tr>
<td>13</td>
<td>DAYTIME</td>
<td>daytime</td>
<td>Daytime</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>netstat</td>
<td>Network status program</td>
</tr>
<tr>
<td>17</td>
<td>QUOTE</td>
<td>qotd</td>
<td>Quote of the Day</td>
</tr>
<tr>
<td>19</td>
<td>CHARGEN</td>
<td>chargen</td>
<td>Character Generator</td>
</tr>
<tr>
<td>37</td>
<td>TIME</td>
<td>time</td>
<td>Time</td>
</tr>
<tr>
<td>42</td>
<td>NAMESERVER</td>
<td>name</td>
<td>Host Name Server</td>
</tr>
<tr>
<td>43</td>
<td>NICNAME</td>
<td>whois</td>
<td>Who Is</td>
</tr>
<tr>
<td>53</td>
<td>DOMAIN</td>
<td>nameserver</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>67</td>
<td>BOOTPS</td>
<td>bootps</td>
<td>BOOTP or DHCP Server</td>
</tr>
<tr>
<td>68</td>
<td>BOOTPC</td>
<td>bootpc</td>
<td>BOOTP or DHCP Client</td>
</tr>
<tr>
<td>69</td>
<td>TFTP</td>
<td>tftp</td>
<td>Trivial File Transfer</td>
</tr>
<tr>
<td>88</td>
<td>KERBEROS</td>
<td>kerberos</td>
<td>Kerberos Security Service</td>
</tr>
<tr>
<td>111</td>
<td>SUNRPC</td>
<td>sunrpc</td>
<td>Sun Remote Procedure Call</td>
</tr>
<tr>
<td>123</td>
<td>NTP</td>
<td>ntp</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>161</td>
<td>-</td>
<td>snmp</td>
<td>Simple Network Management Proto</td>
</tr>
<tr>
<td>162</td>
<td>-</td>
<td>snmp-trap</td>
<td>SNMP traps</td>
</tr>
<tr>
<td>512</td>
<td>-</td>
<td>biff</td>
<td>UNIX comsat</td>
</tr>
<tr>
<td>513</td>
<td>-</td>
<td>who</td>
<td>UNIX rwho daemon</td>
</tr>
<tr>
<td>514</td>
<td>-</td>
<td>syslog</td>
<td>System log</td>
</tr>
<tr>
<td>525</td>
<td>-</td>
<td>timed</td>
<td>Time daemon</td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

Reliable

Byte-stream oriented (as opposed to Datagram oriented)

Virtual Circuit Connection

Buffered Transfer

Unstructured Stream

Full Duplex Connection
Transmission Control Protocol (TCP)

Reliable

Byte-stream oriented (as opposed to Datagram oriented)

Figure 5.3 How TCP manages a byte stream.

5.2.2 Segment Format

TCP is a byte-oriented protocol, which means that the sender writes bytes into a TCP connection and the receiver reads bytes out of the TCP connection. Although "byte stream" describes the service TCP offers to application processes, TCP does not, itself, transmit individual bytes over the Internet. Instead, TCP on the source host buffers enough bytes from the sending process to fill a reasonably sized packet and then sends this packet to its peer on the destination host. TCP on the destination host then empties the contents of the packet into a receive buffer, and the receiving process reads from this buffer at its leisure. This situation is illustrated in Figure 5.3, which, for simplicity, shows data flowing in only one direction. Remember that, in general, a single TCP connection supports byte streams flowing in both directions.

The packets exchanged between TCP peers in Figure 5.3 are called segments, since each one carries a segment of the byte stream. Each TCP segment contains the header schematically depicted in Figure 5.4. The relevance of most of these fields will become apparent throughout this section. For now, we simply introduce them.

The **SrcPort** and **DstPort** fields identify the source and destination ports, respectively, just as in UDP. These two fields, plus the source and destination IP addresses, combine to uniquely identify each TCP connection. That is, TCP's demux key is given by the 4-tuple $\langle \text{SrcPort}, \text{SrcIPAddr}, \text{DstPort}, \text{DstIPAddr} \rangle$.

Note that because TCP connections come and go, it is possible for a connection between a particular pair of ports to be established, used to send and receive data, and closed, and then at a later time for the same pair of ports to be involved in a second connection. Thus, the connection ID is not a permanent identifier of a connection.
5.2 Reliable Byte Stream (TCP)

The TCP header format is shown in Figure 5.4. It includes fields for source and destination ports, sequence numbers, acknowledgment numbers, and window sizes.

Options (variable)

Data

Checksum

SrcPort | DstPort

SequenceNum

Acknowledgment

HdrLen 0 Flags

AdvertisedWindow

Checksum

UrgPtr

Options (variable)

Data (SequenceNum)

Sender

Receiver

Acknowledgment + AdvertisedWindow

Connection. We sometimes refer to this situation as two different incarnations of the same connection.

The Acknowledgment, SequenceNum, and AdvertisedWindow fields are all involved in TCP's sliding window algorithm. Because TCP is a byte-oriented protocol, each byte of data has a sequence number; the SequenceNum field contains the sequence number for the first byte of data carried in that segment. The Acknowledgment and AdvertisedWindow fields carry information about the flow of data going in the other direction. To simplify our discussion, we ignore the fact that data can flow in both directions, and we concentrate on data that has a particular SequenceNum flowing in one direction and Acknowledgment and AdvertisedWindow values flowing in the opposite direction, as illustrated in Figure 5.5. The use of these three fields is described more fully in Section 5.2.4.

The 6-bit Flags field is used to relay control information between TCP peers. The possible flags include SYN, FIN, RESET, PUSH, URG, and ACK.

The SYN and FIN flags...
Connection Establishment and Termination
Three-way Handshake

Active participant
(client)

SYN, SequenceNum = x

SYN + ACK, SequenceNum = y',
Acknowledgment = x + 1

ACK, Acknowledgment = y + 1

Passive participant
(server)
Under the Hood

Conceptually, a sliding window protocol always remembers which packets have been acknowledged and keeps a separate timer for each unacknowledged packet. If a packet is lost, the timer expires and the sender retransmits that packet. When the sender slides its window, it moves past all acknowledged packets.

At the receiving end, the protocol software keeps an analogous window, accepting and acknowledging packets as they arrive. Thus, the window partitions the sequence of packets into three sets: those packets to the left of the window have been successfully transmitted, received, and acknowledged; those packets to the right have not yet been transmitted; and those packets that lie in the window are being transmitted.

Events at Sender Site

- Send Packet 1
- Send Packet 2
- Send Packet 3
- Receive ACK 1
- Receive ACK 2
- Receive ACK 3

Network Messages

- Receive Packet 1
- Send ACK 1
- Receive Packet 2
- Send ACK 2
- Receive Packet 3
- Send ACK 3

Events at Receiver Site
TCP vs UDP for Audio and Messages
Music Through Messages
Open Sound Control (OSC)
What’s OSC

Networking protocol for real-time musical control information

Introduced by CNMAT (UC Berkeley) in 1997

Transport-independent (UDP, TCP, WiFi, serial connections, and within applications)
OSC Messages

Address:
  URL-style

Arguments:
  strings, floats, ints, binary numbers, "blobs", etc.

/npm2010/JPC/freq  2220.02

↑ address
↑ argument
# Argument Types

<table>
<thead>
<tr>
<th>i</th>
<th>int32</th>
<th>c</th>
<th>ASCII character</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>float32</td>
<td>r</td>
<td>RGBA color</td>
</tr>
<tr>
<td>s</td>
<td>OSC-string</td>
<td>m</td>
<td>MIDI Message</td>
</tr>
<tr>
<td>b</td>
<td>blob (binary data)</td>
<td>T</td>
<td>TRUE</td>
</tr>
<tr>
<td>h</td>
<td>int64</td>
<td>F</td>
<td>FALSE</td>
</tr>
<tr>
<td>t</td>
<td>Time Tag</td>
<td>N</td>
<td>nil</td>
</tr>
<tr>
<td>d</td>
<td>float64</td>
<td>I</td>
<td>infinitum</td>
</tr>
<tr>
<td>s</td>
<td>symbol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Address Space

Every address space is application-specific

Symbolic names of features, parameters...
Arbitrary arrangement into tree structure

OSC standard proscribes nothing

+ Utterly flexible
– No automatic “plug and play”
Time

“Bundle” - group of messages

Transmitted together
Must take effect atomically

Bundles have time-tags saying when messages should take effect
Demo Pd Patch
Credits

Some networking images taken from:
- Peterson, “Computer Networks”, 3rd edition

OSC slides Inspired from:
- Wright, “Brief Overview of OSC and its Application Areas”, OSC Conference 2004