TCP/IP Internet Protocol Suite

Lecturer: Bruce Maggs
Scribe: Rob Miller

October 31, 2000

Contents

1 Internet Addressing

Every network interface connected to the Internet is assigned a distinct address, its *IP address*. Most users' machines have only one network interface, and hence only one IP address. Networking equipment and high-performance servers may have multiple network interfaces, however. Machines with multiple network interfaces are called *multihomed*.

IP addresses are 32 bits long, commonly written as four decimal bytes separated by dots, as in 128.2.205.42. Note that the dots in an IP address have nothing to do with the dots in a hostname, such as `www.cs.cmu.edu`. Dots are just a convenient separator used in both notations. You may notice that hostnames ending `.cmu.edu` generally refer to IP addresses in the range 128.2.x.x, but this is simply because CMU has been assigned a large chunk of each address space. The mapping between hostnames and IP addresses is controlled by a distributed database called the Domain Name Service (DNS), which is beyond the scope of this lecture.

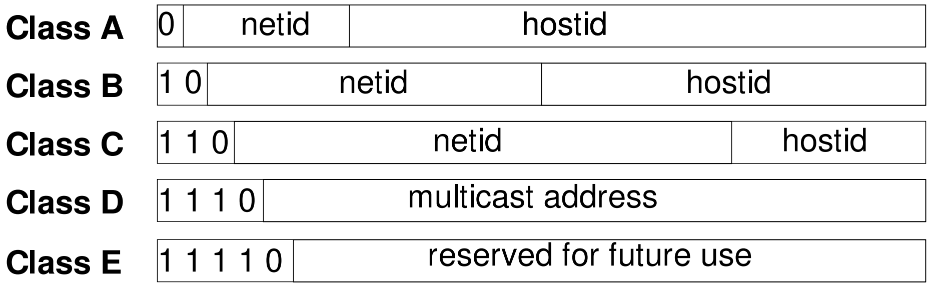


Figure 1: Internet address classes. The netid and hostid fields in Class A, B, and C addresses are anachronisms which have been largely replaced by *classless addressing*.

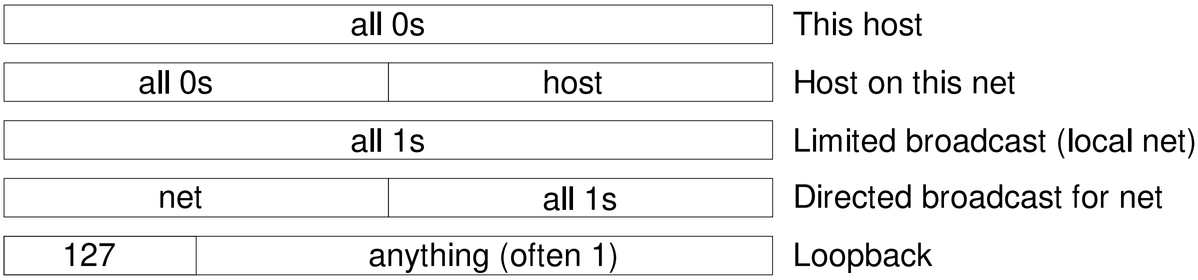


Figure 2: Special IP addresses. Notice that 0 means "this," and 1 means "every."

The space of IP addresses is divided into several classes based on their high-order bits, as shown in Figure 1. Class A, Class B, and Class C addresses are used to identify hosts. Class D is used for multicasting to multiple hosts, and Class E addresses are reserved for future use.

In the original Internet, addresses from Class A, B, or C could be further subdivided into two fields: a network ID and a host ID. The network ID was used by network routers to decide whether a host could be found on the local network, or, if not, on which other network the host could be found. Networks were assigned addresses from Class A, B, or C depending on the network's size. In a Class A address, the network ID is 1 byte and the host ID is 3 bytes, so a Class A network can have as many as 2^{24}, or about 16 million, hosts. Class B networks are midsized, allowing up to 65536 hosts on a single network, and Class C addresses allow only 256 hosts per network. CMU was assigned one of the first Class B network addresses, 128.2.x.x.

As the Internet grew, it was realized that the class system doesn't scale well. Fully half of the entire IP address space was held captive in Class A networks that were mostly unused, since no single organization had anywhere near 16 million hosts. Meanwhile, Class B and Class C addresses were being rapidly used up.

To solve this problem, the Internet moved to a scheme called *Classless Inter-Domain Routing* (CIDR). In CIDR, an IP address is still divided into a network ID and a host ID, but the dividing point can no longer be determined directly by inspecting the high-order bits of the address. Instead, network routers have an identifier for every known network, called a *CIDR block*, which specifies both the network ID and its the length. A CIDR block is usually written in the form $w.x.y.z/n$, where n is the number of bits in the network ID. For example, the CIDR block 14.36.128.0/17 is a network containing all IP addresses that match 14.36.128.0 in the first 17 bits. Under the new system, CMU's Class B network ID would be written 128.2.0.0/16, or 128.2/16. All modern routing protocols use classless addresses to route messages around the Internet.

A few IP addresses are given special local meanings, as shown in Figure 2. By convention, "0" means

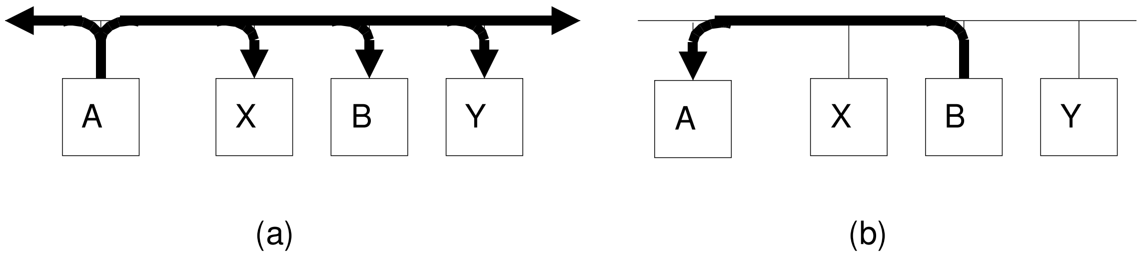


Figure 3: ARP exchange. (a) Host A broadcasts an ARP request for host B's hardware address. (b) B sends an ARP reply to A containing its hardware address, which A stores in its routing table.

"this", and "1" means "every." For example, network ID 0 and host ID 0 refers to the sending host itself, and network ID 0 and a given host ID will be received by the given host on the local network. Messages to these special IP addresses are never routed between networks. In fact, some never even reach the network. For example, 127.x.x.x ("loopback") is implemented entirely in software, and should never appear on the actual wire.

Several classes of Internet addresses are reserved for private networks. A private network is a network that is isolated from the rest of the Internet, such as a corporate network behind a firewall, or a small LAN that is unconnected to the Internet. The private IP addresses are:

- 10.x.x.x (CIDR 10/8)
- 172.16.x.x - 172.31.x.x (CIDR 172.16/12)
- 192.168.x.x (CIDR 192.168/16)

These IP ranges are quite useful for setting up temporary or experimental networks that don't need to be connected to the rest of the Internet. The 10.x.x.x range is particularly easy to remember: if you need to pick an arbitrary IP address, just pick one one starting with 10. A private IP address should never be used on the Internet, however.

2 Address Resolution

A local network within the Internet is typically a shared bus, such as Ethernet. In order to send to another host on the local network, the sender must have some way to discover the Ethernet hardware address corresponding to the target's IP address. This process is called *address resolution*, and it is handled by the Address Resolution Protocol (ARP).

2.1 Address Resolution Protocol (ARP)

The Address Resolution Protocol maps IP addresses to hardware addresses on the local network. Suppose host A has a message to send to host B, and A knows that B is on the same local network (which it can test by comparing network IDs), but A does not know B's hardware address. Host A initiates an ARP exchange (Figure 3):

1. Host A broadcasts an ARP request for I_B, requesting the hardware address for host B. A's request includes its own addresses (I_A, P_A), so that the responder knows where to send the response.

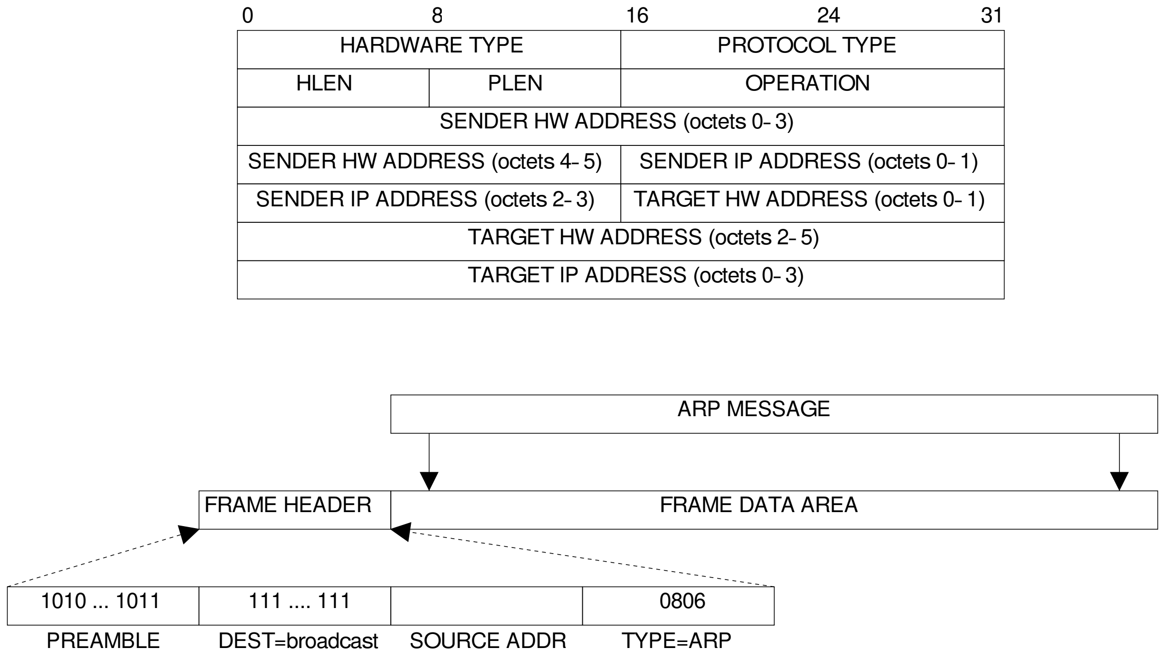


Figure 4: ARP packet format.

2. Host B responds with an ARP message containing (I_B, P_B).
3. A caches the (I_B, P_B) mapping in its routing table, so that future messages to B can be sent directly to P_B without an ARP exchange.

The format of an ARP packet is shown in Figure 4. A few notes about the fields of the ARP packet follow:

- **Hardware Type** is 1 for Ethernet.
- **Protocol Type** is 0x0800 for IP.
- **HLen** is 7, the header length in 32-bit words.
- **PLen** is 4, the number of bytes in an IP address.
- **Operation** takes one of four values:

  1 ARP request
  2 ARP response
  3 RARP request
  4 RARP response

When host A makes an ARP request, it fills in its own addresses for Sender HW Address and Sender IP Address, and sets Target IP Address to B's IP address. The Target HW Address is set to zero. When B makes its ARP response, B swaps the Sender and Target addresses and fills in the Sender HW Address with its own hardware address.

The entire ARP packet is packaged inside the data area of an Ethernet frame. This is an example of an important technique in networking: *encapsulation*, in which one protocol (in this case, ARP) is carried by another protocol (Ethernet). To indicate that the packet is for ARP, the Ethernet type field in the frame header is set to 0x0806 (in Ethernet II) . The destination hardware address is set to all 1s to broadcast the frame to all other hosts on the local net.

ARP is more general than IP-to-Ethernet address mapping. It can bind other addresses as well. ARP's main use in practice is for IP, however.

2.2 Reverse Address Resolution Protocol (RARP)

Reverse Address Resolution Protocol (RARP), closely related to ARP, performs the inverse mapping, taking a physical address and returning the associated IP address. RARP was originally designed for diskless workstations, network printers, and other networking equipment that needs to discover its own IP address at boot time. RARP has largely been replaced by DHCP, described in the next section.

RARP requires a server on each local network that knows the mapping between physical and IP addresses and can service RARP requests. Suppose host A is booting up. A can discover its own hardware address P_A by querying its network interface. In order to send IP messages, however, A must discover its IP address I_A. Here is an example of a RARP exchange:

1. A broadcasts a RARP request containing P_A
2. The RARP server responds with a message containing (I_A, P_A).

A RARP packet has the same structure as an ARP packet, shown in Figure 4.

2.3 Dynamic Host Configuration Protocol (DHCP)

For automatic network configuration, the Dynamic Host Configuration Protocol (DHCP) is now more widely used than RARP, because DHCP provides more information about the local network (such as gateways and subnet masks) and more sophisticated features (such as time-limited address bindings, called *leases*).

Like RARP, DHCP requires a server to be listening for DHCP requests. When a host A boots up, it conducts the following exchange:

1. A broadcasts a DHCP discovery message containing its own physical address P_A (among other information)
2. The DHCP server replies with the information A needs to become a full network participant, including:

- A's IP address I_A
- Subnet mask: a bitmask that defines the network ID of the local network. Hosts whose IP addresses match I_A wherever this bitmask is 1 are reachable directly on the local net.
- DNS name servers: the IP addresses of the servers that map hostnames (like `www.cs.cmu.edu`) into IP addresses.
- Default gateway: the IP address of a router on the local network that can forward messages to hosts on other networks.

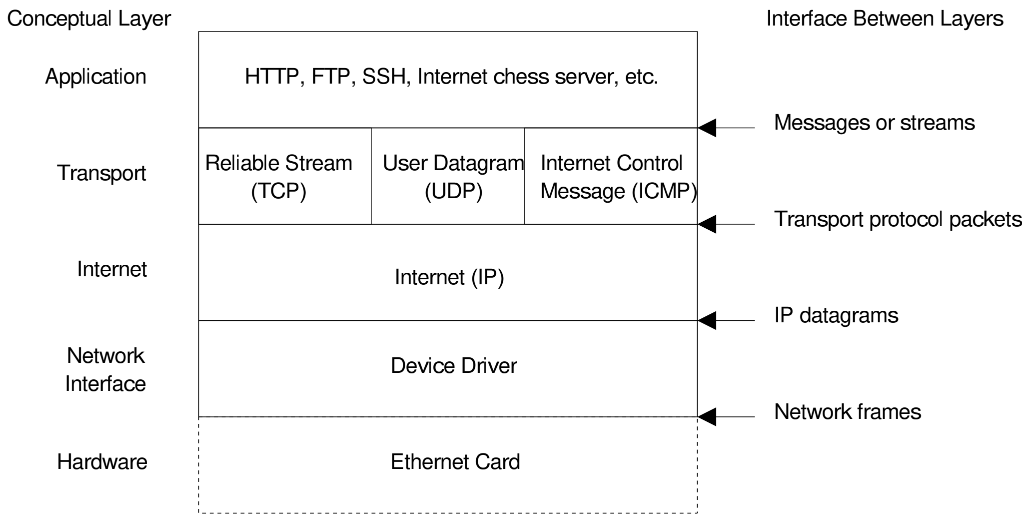


Figure 5: Network protocol layers.

Using the subnet mask and the default gateway, host A can determine how to send a message destined for some other host B. If I_B matches I_A wherever the subnet mask is 1, then B is on the local network, so A uses ARP to discover B's hardware address and sends the message directly. Otherwise, A sends the message to the default gateway (which is on the local network and therefore resolvable by ARP), and the gateway is responsible for forwarding it to its destination. The algorithms and protocols used by the gateway router to forward the packet will be the subject of the next lecture.

3 Network Protocol Layers

The network protocols used in the Internet are *layered*. Higher protocol layers present successively more abstract interfaces to the network, and every layer is implemented using the layer directly below it. The conceptual protocol layers found in the Internet are show in Figure 5.

Protocol layers are designed to be *transparent*. When data is passed down into a lower layer on host A, destined to be sent to host B, exactly the same data comes up out of the equivalent layer on host B. Figure 6 illustrates transparency.

Older textbooks describe a seven-layer protocol stack, called the Open Systems Interconnection (OSI) model, but the five-layer stack shown above has a much better fit to the actual protocols used on the Internet.

4 Internet Protocol (IP)

The Internet Protocol, or IP, provides an unreliable, connectionless delivery mechanism. By "unreliable," we mean *best effort* delivery, meaning that the network will do its best to get an IP packet to its destination, but in the face of network congestion or network errors, the entire packet may be lost. By "connectionless," we mean that IP packets are independent, and neither the sender nor the receiver needs to remember anything about previous packets or maintain any state about a connection. IP usually involves one sender and one receiver, although there are some variants (broadcast and IP multicast, which are beyond the scope of this paper).

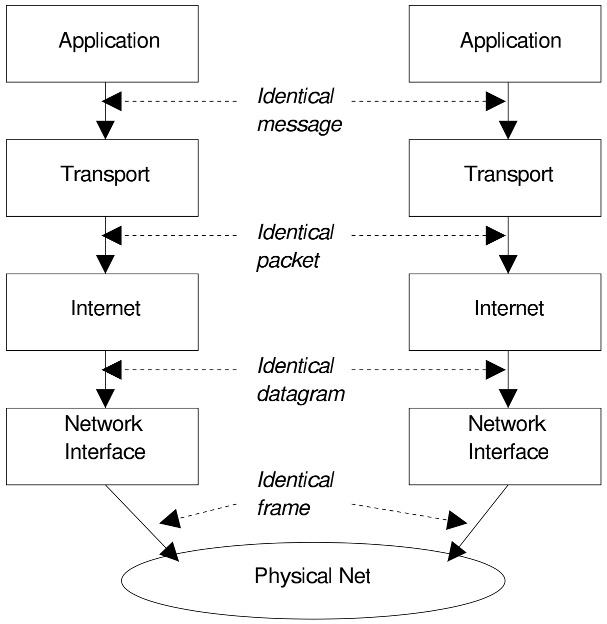


Figure 6: Network protocol layers are *transparent*. Data passed down into a layer on the sending host is transmitted across the network and passed up from the equivalent layer on the receiving host, unchanged.

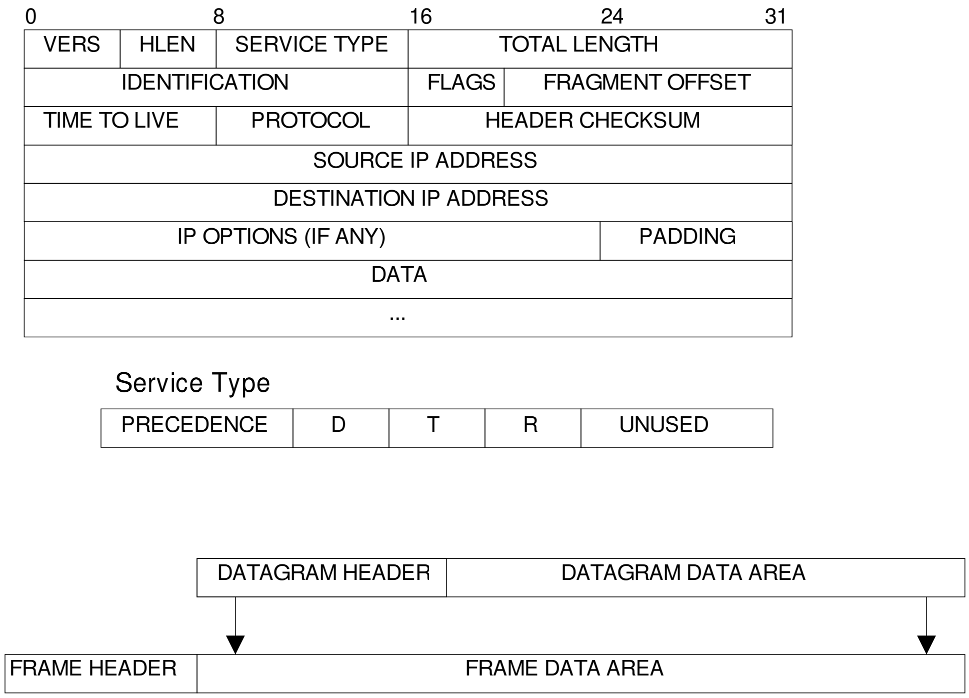


Figure 7: IP datagram format.

IP resides in the Internet protocol layer. As such, most network applications do not use IP directly, but instead use transport-layer protocols, like TCP and UDP, that are implemented on top of IP.

The format of an IP datagram is shown in Figure 7. Some notes about the fields:

- **Vers** is 4 for IP version 4, the current Internet standard. (Work is underway on IP version 6.)
- **HLen** is the number of 32-bit words in the header, usually 5.
- **Service Type** has several flags indicating the datagram's priority and service requirements. Routers generally ignore these flags.
- **Total Length** is the total number of bytes in the datagram, including the header.
- **Identification**, **Flags**, and **Fragment Offset** are used for fragmentation, discussed in the next section.
- **Time To Live** indicates how long the packet should be allowed to move around the network. Originally the units of this field were seconds, but now the time to live is measured in *hops*. Every time a router forwards a packet, its time-to-live field is decremented. When time-to-live reaches 0, the packet is dropped.
- **Protocol** specifies which protocol this packet is being used for. Sample values are 1 for ICMP, 6 for TCP, and 17 for UDP.
- **Header Checksum** is the 1's complement sum of all 16-bit fields in the header (excluding the checksum field itself). The checksum must be fast to compute because it must be recomputed by every router after it decrements the time-to-live field.
- **IP Options** (optional) are used for tracing routes and timestamping.

IP datagrams are encapsulated in Ethernet frames. In Ethernet II, the type field of the Ethernet frame header is set to 0x0800 to indicate that the frame is an IP packet.

Notice that an IP datagram has a header checksum, but no data checksum. Checks of data integrity must be provided by higher-level protocol layers, such as TCP.

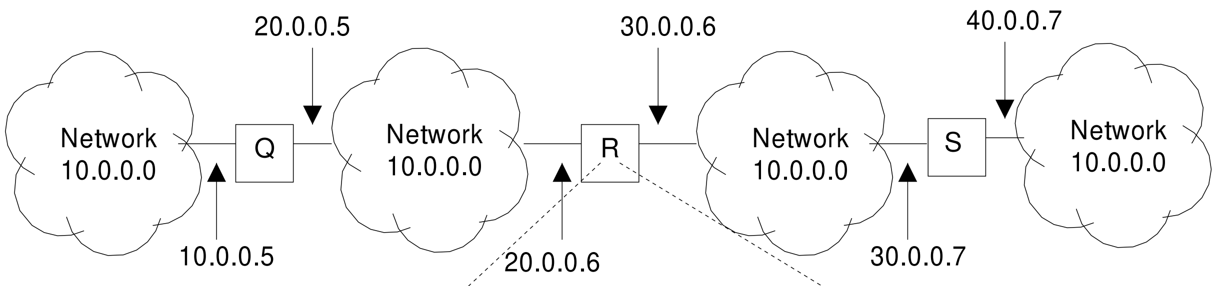
4.1 Fragmentation

Since the IP header has a 16-bit length field, an IP datagram can be up to 64 KB in length. Ethernet frames are limited to 1500 data bytes, however, and other kinds of network hardware have similar length limitations. To send an IP datagram longer than the hardware limit, the datagram must be *fragmented*.

An IP datagram is fragmented by splitting up its data among multiple IP packets. All fragment headers keep the packet's identification number, and fragment offset indicates where the fragment's data field was taken from the original packet. The last fragment is marked by turning off the *more-fragments* bit in its flags field.

The fragments are then independently routed to their destination, as if they were ordinary IP datagrams. Intermediate routers do not try to reassemble the fragments. Instead, the final receiver is responsible for waiting for and reassembling all the fragments in the correct order so that the entire payload data can be passed up from the IP protocol layer.

Fragmentation sometimes occurs while a packet is being routed to its destination, in order to be sent along a link whose hardware limit is smaller than the packet size. The sender can prevent this from happening by setting the *do-not-fragment* bit in the packet's flags field. If *do-not-fragment* is set, then the packet is simply dropped instead of being fragmented.



Routing Table for R

To Reach Hosts on Network:	Route to This Address:
20.0.0.0	DELIVER DIRECTLY
30.0.0.0	DELIVER DIRECTLY
10.0.0.0	20.0.0.5
40.0.0.0	30.0.0.7
Default gateway	20.0.0.5

Figure 8: IP datagram routing in a small internet.

Network engineers try to eliminate fragmentation in the network as much as possible, since it slows down routers and creates network congestion. The Internet has generally standardized on the Ethernet limit, 1500 bytes, as the maximum packet size (called the *maximum transfer unit*, or MTU). TCP/IP protocol stacks generally limit their outgoing packets to 1500 bytes, and network hardware manufacturers build hardware that can handle at least 1500-byte packets.

4.2 Routing IP Datagrams

IP datagrams are passed between networks by *routers*. By definition, a router is multihomed, with a different network interface (and different IP address) on every network it connects to.

When a router receives an IP datagram, it determines where to forward the datagram by consulting its *routing table*. The routing table maps a network ID (described by a CIDR block) to a path to that network. The algorithms that discover and propagate routing tables will be covered in the next lecture. For now, we assume that every router in the network has a correct routing table.

The routing table for a router in a small internet is shown in Figure 8. To route an IP datagram, the router searches for a routing table entry whose network ID matches the target IP address. If a matching entry is found, the router forwards it to the next hop on the path indicated by the entry, which is always a local interface on one of the networks connected to the router. For example, if router R has a packet destined for 40.0.0.12, R checks its routing table and finds the entry 40.0.0.0 -> 30.0.0.7, so R sends the packet over the 30.0.0.0 network to router S.

If no matching entry is found in the routing table, the router may be configured with a *default gateway*, the address of a router that can handle packets sent to unknown destinations (such as the rest of the Internet). If no default gateway is set, the router just drops the packet (optionally sending an error control message back to the sender).

A router can also be used to connect network segments, using a hack called *proxy ARP*. A proxy ARP

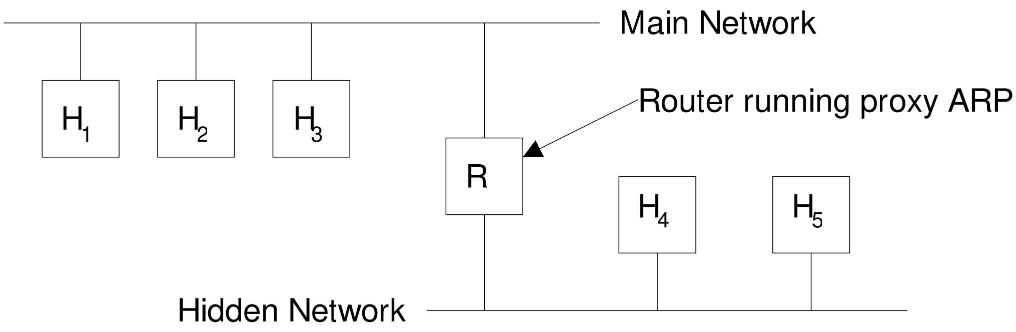


Figure 9: Proxy ARP. Router R connects the hidden network, with hosts H_4 and H_5, onto the main network by answering ARP requests for H_4 and H_5 and forwarding packets back and forth between the networks.

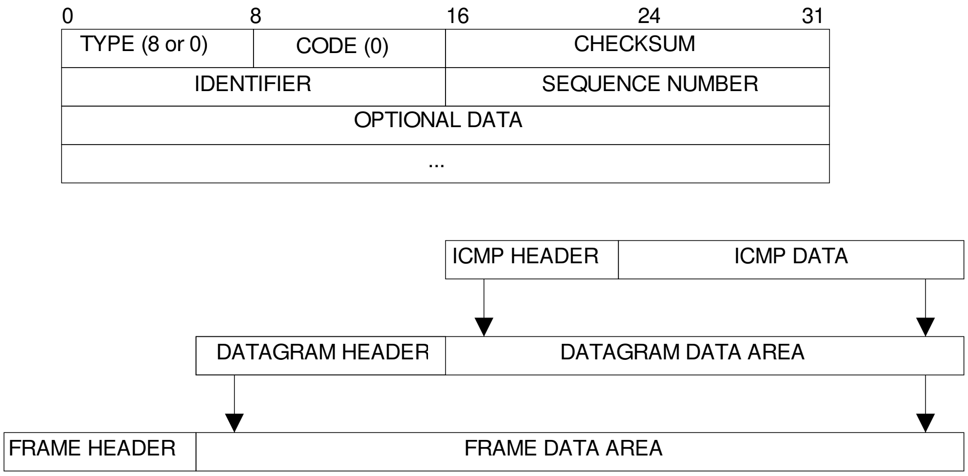


Figure 10: ICMP packet format.

scenario is shown in Figure 8. A small "hidden" network, with hosts H_4 and H_5, is connected to the main network through a router, R. Both networks assign addresses from the same IP address space, such as 128.2.0.0/16, so a message destined for H_4 or H_5 will eventually reach the main network. Router R listens for ARP requests on the main network for hosts H_4 and H_5, and answers with its own hardware address. When datagrams intended for H_4 or H_5 are subsequently sent to R, R forwards them to H_4 or H_5 on the hidden network.

Proxy ARP is just one way to connect unrouted network segments. Another option is *bridging*, which forwards Ethernet frames between network segments, instead of IP datagrams. The SCS network uses bridging to connect its network together.

5 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) carries control and error information around the Internet. ICMP is layered on top of IP datagrams. The format of an ICMP packet is shown in Figure 10. The meaning of an ICMP packet depends on the Type and Code fields. For example, Type 8/Code 0 stands

```
traceroute to berkeley.edu (128.32.25.19), 30 hops max, 40 byte packets
1 GIGROUTER.NET.CS.CMU.EDU (128.2.254.36) 1.606 ms 0.764 ms 0.736 ms
2 RTRBONE-FA6-0-0.GW.CMU.NET (128.2.255.2) 1.284 ms 0.827 ms 0.806 ms
3 killifish.psc.net (198.32.224.11) 2.517 ms 2.177 ms 1.959 ms
4 192.88.115.121 (192.88.115.121) 9.201 ms 5.532 ms 5.729 ms
5 ipls-clev.abilene.ucaid.edu (198.32.8.25) 12.821 ms 11.186 ms 12.065 ms
6 kscy-ipls.abilene.ucaid.edu (198.32.8.5) 20.64 ms 21.511 ms 20.741 ms
7 dnvr-kscy.abilene.ucaid.edu (198.32.8.13) 31.594 ms 32.41 ms 32.307 ms
8 scrm-dnvr.abilene.ucaid.edu (198.32.8.1) 62.864 ms 54.414 ms 53.342 ms
9 198.32.249.61 (198.32.249.61) 58.753 ms 56.56 ms 56.322 ms
10 BERK--SUNV.POS.calren2.net (198.32.249.13) 57.212 ms 58.205 ms 57.159 ms
11 pos1-0.inr-000-eva.Berkeley.EDU (128.32.0.89) 57.41 ms 57.732 ms 57.777 ms
12 pos5-0-0.inr-001-eva.Berkeley.EDU (128.32.0.66) 59.022 ms 58.851 ms 58.126 ms
13 128.32.0.7 (128.32.0.7) 58.56 ms 59.552 ms 59.836 ms
14 f8-0.inr-100-eva.Berkeley.EDU (128.32.235.100) 60.896 ms 61.121 ms 60.186 ms
15 chaparral.Berkeley.EDU (128.32.25.19) 59.836 ms * 61.412 ms
```

Figure 11: Output of `traceroute berkeley.edu` on an SCS workstation.

for echo request, and Type 0/Code 0 for echo reply. Other type/code combinations include time exceeded, source quench (which means "slow down!"), and destination unreachable.

ICMP is used for several purposes. The *ping* program found on many platforms sends ICMP echo request packets to a chosen destination, which responds with ICMP echo reply packets. Ping is useful for testing whether a host is reachable and estimating the network latency to the host. ICMP is also used to return errors to a sender, such as when a destination address is unreachable. ICMP messages are used in a particularly clever way by the *traceroute* utility, described in the next section.

5.1 Traceroute

The traceroute utility maps out a path through the network to a destination host. An example of traceroute's output is shown in Figure 11.

Although the design of IP included some options to enable path-tracing, like storing the router name in the datagram header, these features are not widely implemented by routers. Traceroute takes another approach, using the time-to-live field to trick routers into revealing their identity. First, traceroute sends a packet with time-to-live set to 1. The first router to receive the packet decrements the time-to-live field and detects that the field has reached 0. Most routers then return an ICMP time-exceeded error message back, which tells traceroute the identity of the first router on the path. Further packets are sent with time-to-live set to 2, 3, 4, etc., until traceroute has traced the path all the way to the destination.

Traceroute is not 100% reliable. The path through the network may change from packet to packet. Some routers do not send ICMP error messages back when the time-to-live field expires. Also, some networks do not want to be probed. Traceroute uses UDP packets by default, which can look suspiciously like a portscan (a common preliminary to a computer breakin). Irate network administrators may call you if you overuse it. Traceroute can also probe with ICMP packets, which are less rude, but routers rarely send back error messages about ICMP packets. Sending control messages about control messages can easily flood the network.

Traceroute also measures only the *outgoing* path, from the source to the destination. The return path may be considerably different. A key reason why outgoing and return paths can differ is the routing policy common used by network backbones. To reduce traffic on its own network, a backbone tries to get rid of a packet destined for another backbone's customer as early as possible. Suppose host A and host B are on different sides of the continent and connected to different backbones. Packets sent from A to B will be handed off to B's backbone as soon as possible, so the path from A to B will cross the continent mainly on B's backbone. Conversely, packets sent from B back to A will be handed off to A's backbone as soon as possible, so the return path will be mainly on A's backbone.

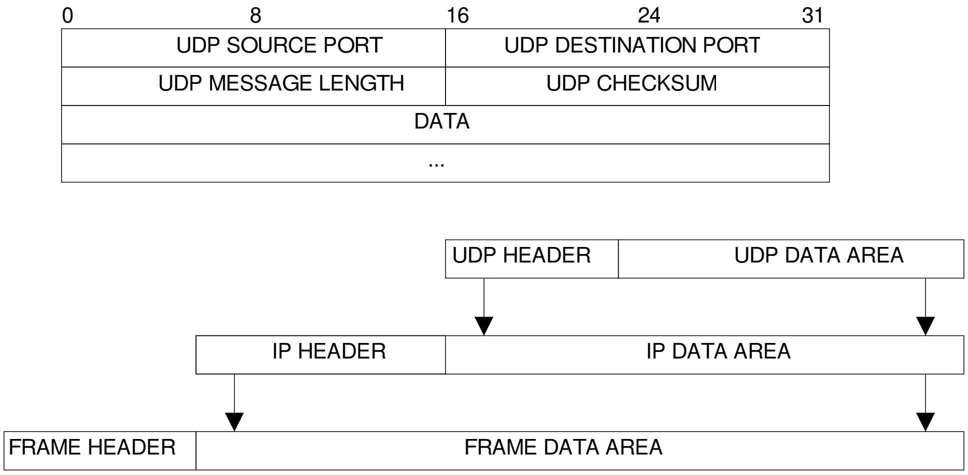


Figure 12: UDP packet format.

6 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a transport-layer protocol with the same delivery guarantees as IP: unreliable, connectionless delivery. UDP is layered on top of IP. The format of a UDP packet is shown in Figure 12.

Unlike IP, which delivers datagrams between two hosts, UDP delivers datagrams between *ports* on two hosts, where a port is a number ranging from 0 to 65535. Also, UDP provides a checksum on the data carried by the packet, so that errors can be detected in received packets. Erroneous packets are simply discarded.

Although TCP (described next) is generally preferred for most network applications, since it provides reliable delivery, UDP may be used in special cases. UDP is preferred for applications where the overhead of TCP is undesirable, where realtime requirements make retransmission impractical, or where lost packets don't matter much. Streaming multimedia is a common application of UDP. RealAudio and Windows Media both have UDP options.

7 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) provides a *reliable, stream-based* transport service layered on top of IP. TCP creates a virtual circuit through the network connecting two ports on two hosts. If individual packets arrive out of order, TCP takes care of buffering data until the late packets arrive, and reorders packets in the order they were sent. To deal with dropped packets, TCP sends acknowledgements for every packet received and resends unacknowledged packets, if necessary.

The format of a TCP packet is shown in Figure 13. Some notes about the fields:

- **Sequence Number** indicates the byte offset of the current packet's data in the stream, so that the receiver can reassemble packets in the correct order.
- **Acknowledgement Number** and **Window** are control messages sent from the receiver back to the sender. They are frequently "piggybacked" on data packets.
- **Code Bits** are control flags as shown in Figure 13.

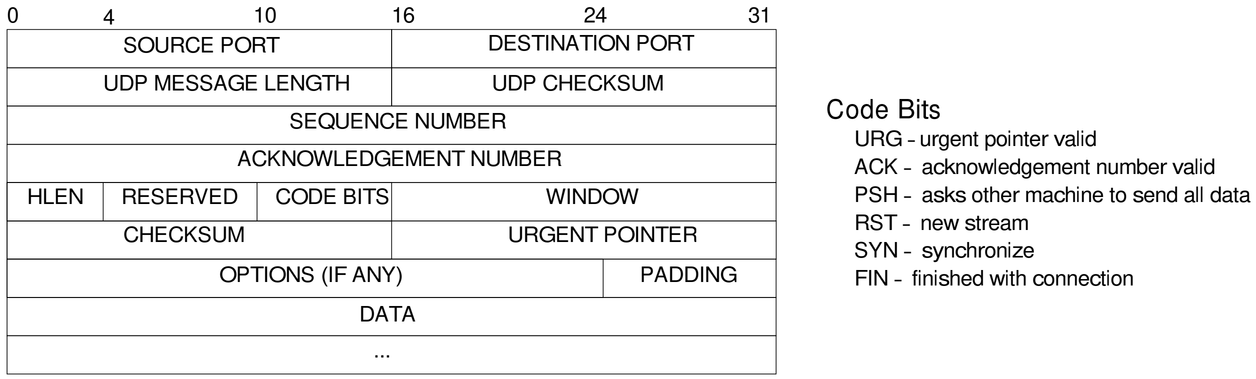


Figure 13: TCP packet format.

- **Urgent Pointer** is used to send out-of-band data.

Because of TCP's reliability guarantees, many network protocols are layered on top of TCP. For example, a web browser uses TCP to retrieve a web page from a remote server. Other examples include FTP, Telnet, and ssh.

7.1 TCP Acknowledgements

To ensure that all data is received at the destination, TCP sends *acknowledgement* (ACK) messages. Figure 14 shows an example of a simple acknowledgement exchange. An acknowledgement message is a TCP packet whose ACK field is set to the sequence number of the highest packet successfully received. An acknowledgement message can be sent alone, or it can be *piggybacked* on a TCP packet containing data.

Figure 15 shows what happens when a TCP packet is lost in transit. Since the receiver never gets the lost packet, it never sends an ACK for the lost packet. If the sender waits for a certain (random) timeout without receiving an ACK, it sends the packet again. If the second packet is lost also, the sender doubles the period it waits for an ACK before resending the packet. This *exponential backoff* mechanism is designed to reduce congestion in the network. Unfortunately, losing a few packets in a row can rapidly make a TCP connection unusable, by driving the retransmit timeout up from milliseconds to minutes. Experiments show that if the network has more than 3% packet loss, TCP connections have big trouble.

One problem with the simple mechanism shown in Figure 14 is that the sender waits for an ACK after each packet before sending the next packet. Since sending a packet and receiving an ACK takes a roundtrip through the network, this simple mechanism limits the throughput of the connection by the latency of the network. In practice, TCP keeps multiple packets in flight with a *sliding window.* The sender can have multiple unacknowledge packets outstanding, up to the size of the sliding window, which improves throughput. Sliding windows are illustrated in Figure 16.

The size of the sliding window changes dynamically with conditions on the network. If the receiver finds it is receiving packets effectively, it signals the sender to increase the window size. If packets are being dropped, the window size decreases.

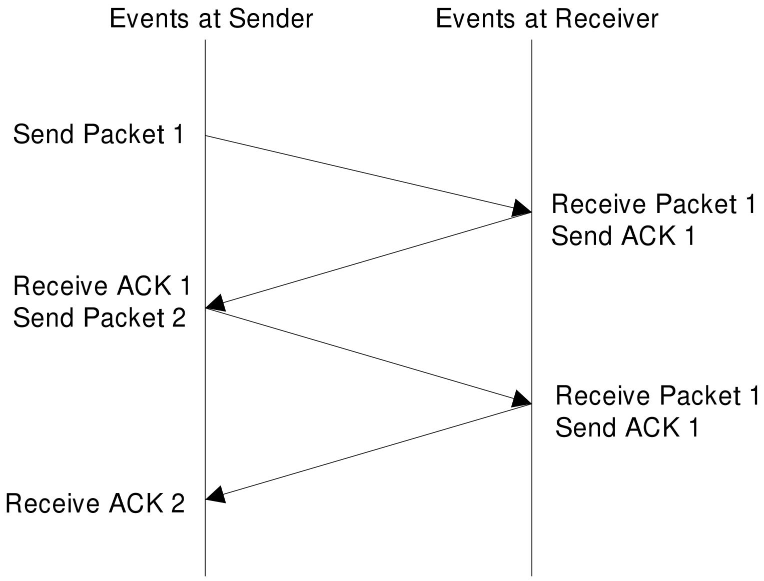


Figure 14: Simple acknowledgement handshake in TCP.

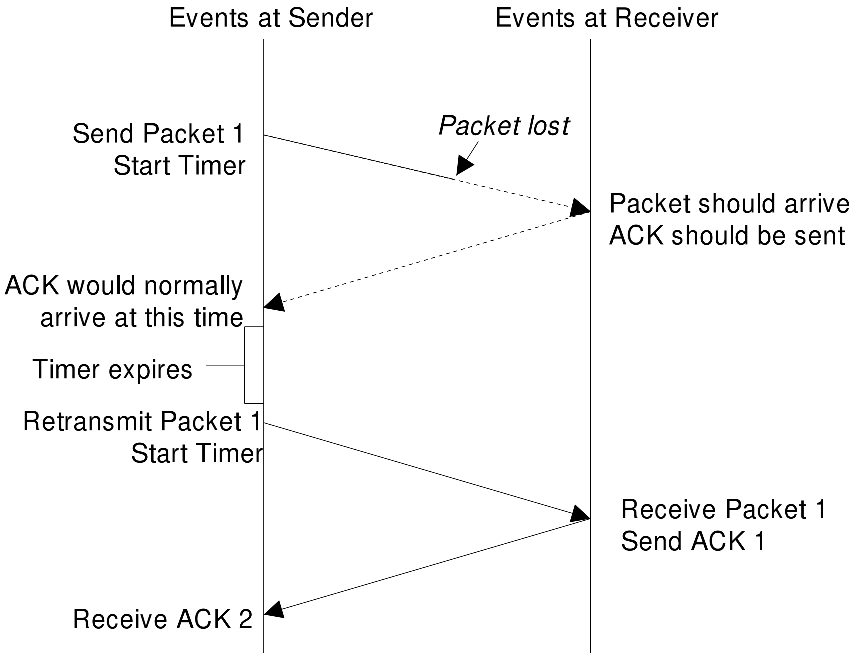


Figure 15: TCP resends packets which are not acknowledged before a timeout elapses. As more packets are lost, the timeout increases exponentially.

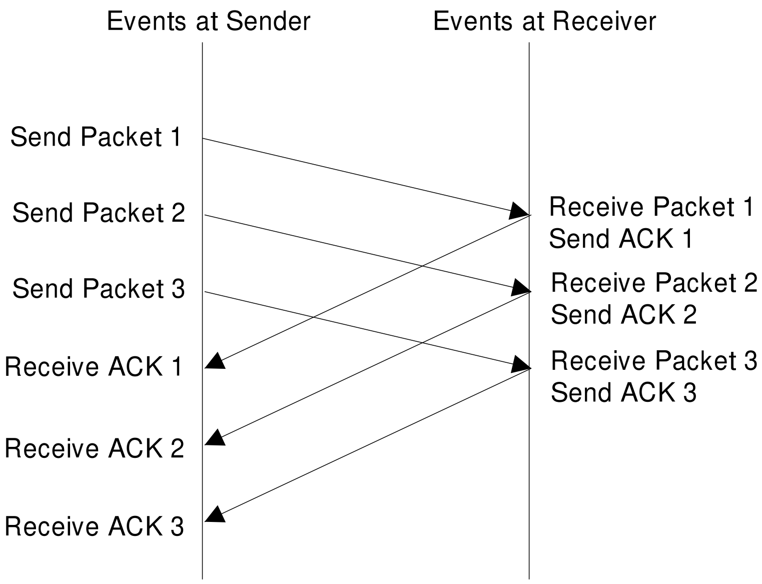


Figure 16: Sliding window acknowledgements in TCP.

7.2 TCP Handshaking

A TCP connection is established by exchanging the sequence of messages shown in Figure 17. Suppose host A is originating a connection to host B:

1. A sends a SYN message (a TCP packet with the SYN flag set). The sequence number of this message is a random number x.
2. B replies with a SYN ACK message with the acknowledgement set to $x + 1$, and sequence number set to another random number y.
3. Finally, A sends a packet acknowledging $y + 1$, usually piggybacked on a data packet.

Notice that no data is sent before each party receives an ACK indicating that the other party is ready for the connection. This prevents certain kinds of denial-of-service attacks. For example, a hostile third party C might pretend to be A and send a spoofed message to server B requesting data. If B just started sending data, without verifying that A was actually expecting it, then A could become flooded with unrequested data.

Closing a TCP connection is shown in Figure 18. Either side can close the connection, regardless of which host originated the connection, and the same messages are exchanged in either case. Suppose host A closes:

1. A sends a FIN message (a TCP packet with the FIN flag set) with the current sequence number x (the total number of bytes of data A has sent).
2. B sends a plain ACK message acknowledging $x + 1$.
3. B notifies the application using the TCP connection and waits for it to clean up the connection.
4. B sends a FIN ACK message acknowledging $x + 1$.

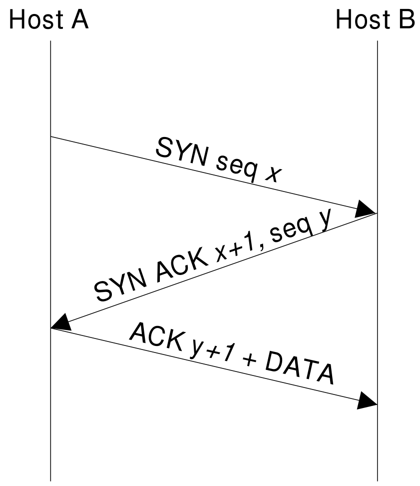


Figure 17: Opening a TCP connection.

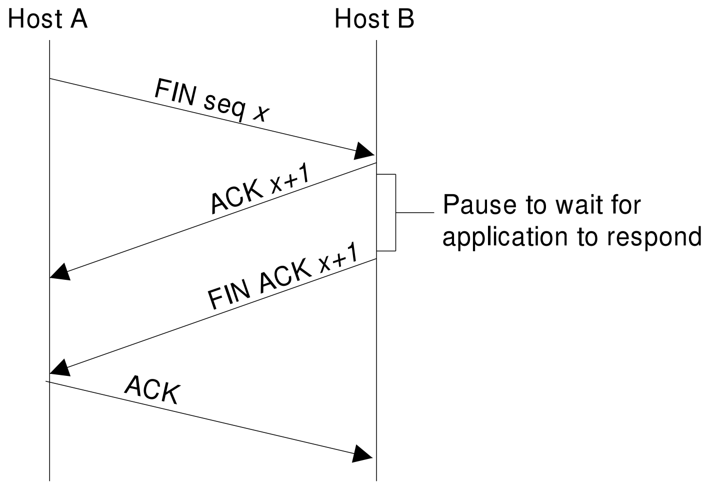


Figure 18: Closing a TCP connection.

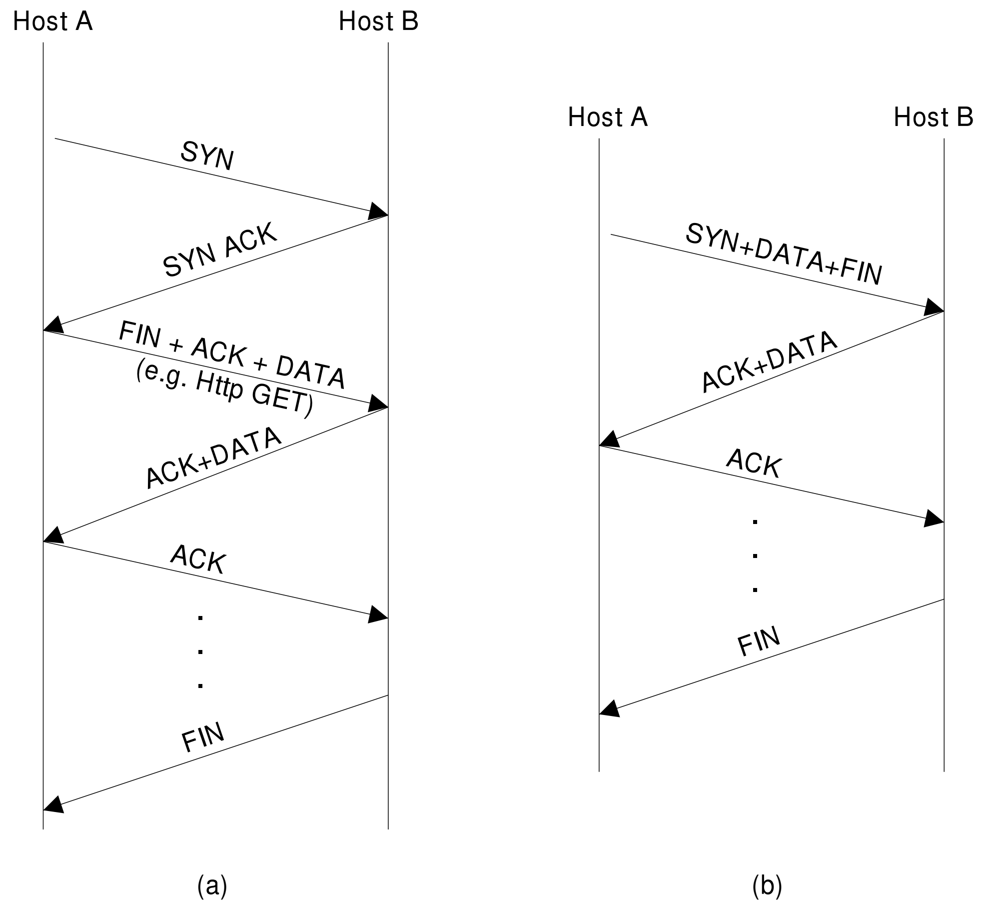


Figure 19: TCP handshaking variants: (a) half-closed connection, (b) transactional TCP.

5. Finally, A sends an ACK message.

One drawback of the standard TCP open and close protocols is the number of roundtrips required through the network: one roundtrip to open the connection, and another to close it. Roundtrips can be expensive, especially for servers that open and close many small TCP connections, like web servers. Variants of TCP have been developed that reduce this problem. Two of these variants, half-closed connections and transaction TCP, are shown in Figure 19.

In a half-closed connection, the originator piggybacks a FIN message on its first data packet, (typically an HTTP GET request that fits into a single packet). In essence, the originator is closing its outgoing channel immediately after sending the request. The server handles the request, sends back the data, and finally sends a FIN message to indicate that the connection is terminated.

In transactional TCP, the originator piggybacks SYN, data, and FIN on a single packet. The server responds immediately with data, and finally sends a FIN to terminate the connection. Unfortunately, transactional TCP is dangerous, because a server can be spoofed into sending data to a host that didn't request it.