UNIT 4 PART 1

The Internet Transport Protocol (UDP): Introduction to UDP, Remote Procedure Call, The Real-Time Transport Protocol.

The Internet Transport Protocols: UDP

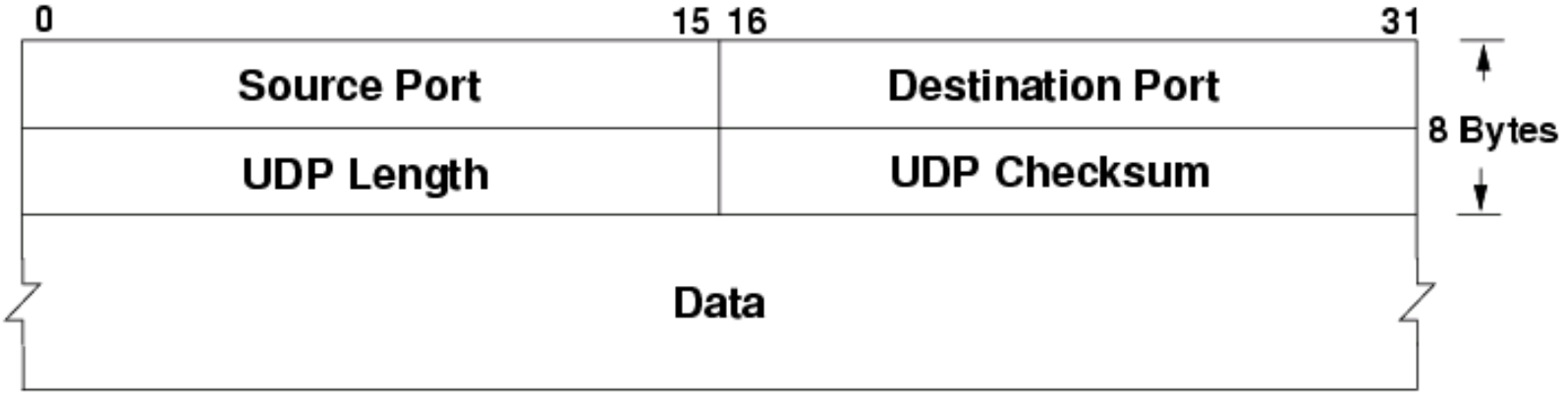
- The Internet has two main protocols in the transport layer, a **connectionless protocol** and a **connection-oriented protocol**.
- The **connectionless protocol** is **UDP**.
- The **connection-oriented protocol** is **TCP**.
- Because UDP is basically just IP with a short header added.

Introduction to UDP

- The Internet protocol suite supports a connectionless transport protocol, UDP (User Datagram Protocol).
- UDP provides a way for applications to send encapsulated IP datagrams and send them without having to establish a connection.

Introduction to UDP

- UDP transmits segments consisting of an 8-byte header followed by the payload. The header is shown in below figure.
- The two ports serve to identify the end points within the source and destination machines.
- When a UDP packet arrives, its payload is handed to the process attached to the destination port.



Introduction to UDP

- The UDP datagram contains a source port number and destination port number.
- **Source port** number identifies the port of the sending application process.
- **Destination port** number identifies the receiving process on the destination host machine
- The source port is primarily needed when a reply must be sent back to the source. By copying the source port field from the incoming segment into the destination port field of the outgoing segment, the process sending the reply can specify which process on the sending machine is to get it.

Introduction to UDP

- The **UDP length** field includes the 8-byte header and the data.
- The **UDP checksum** is optional and stored as 0 if not computed (a true computed 0 is stored as all 1s).
- Turning it off is foolish unless the quality of the data does not matter (e.g., digitized speech).
- UDP checksum covers the **UDP header** and the **UDP data**.
- UDP checksum is end-to-end checksum. It is calculated by the sender, and then verified by receiver. It is designed to catch any modification of the UDP header or data anywhere between sender and receiver.

Remote Procedure Call

- When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2.
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing is visible to the programmer.
- This technique is known as **RPC (Remote Procedure Call).**
- The calling procedure is known as the **client** and the called procedure is known as the **server**.

Remote Procedure Call

- The idea behind RPC is to make a remote procedure call look as much as possible like a local one.
- To call a remote procedure, the client program must be bound with a small library procedure, called the **client stub**, that represents the server procedure in the client's address space.
- Similarly, the server is bound with a procedure called the **server stub**.
- These procedures hide the fact that the procedure call from the client to the server is not local.
- The actual steps in making an RPC are shown in below figure.

Remote Procedure Call

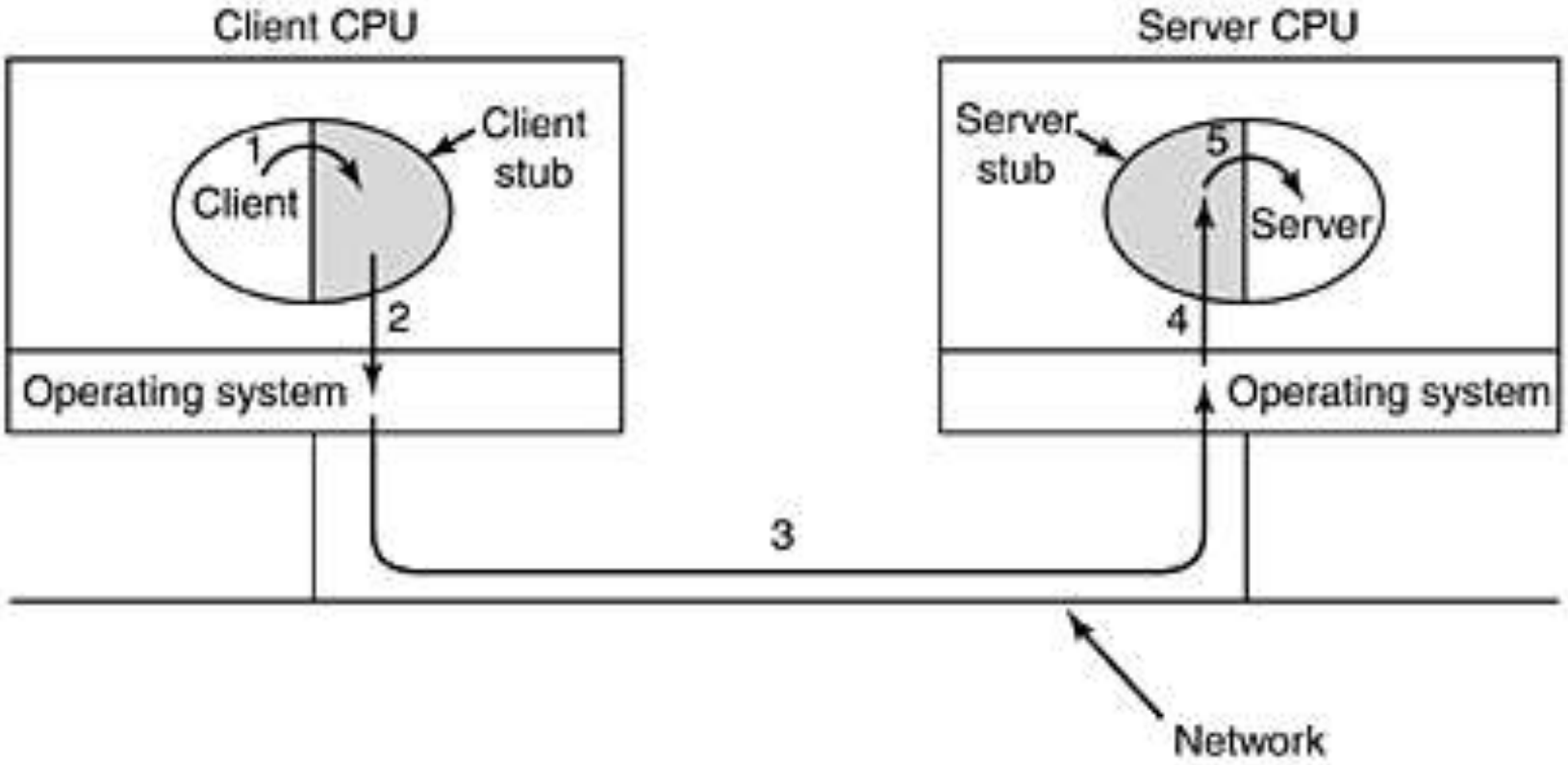


Figure: Steps in making a remote procedure call. The stubs are shaded

Remote Procedure Call

- **Step 1** is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.
- **Step 2** is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called **marshaling**.
- **Step 3** is the kernel sending the message from the client machine to the server machine.
- **Step 4** is the kernel passing the incoming packet to the server stub.
- Finally, **step 5** is the server stub calling the server procedure with the unmarshaled parameters. The reply traces the same path in the other direction.

Remote Procedure Call

- Disadvantages of RPC:
  - A First problem is the use of pointer parameters.
  - A second problem is that in weakly-typed languages, like C, it is perfectly legal to write a procedure that computes the inner product of two vectors (arrays), without specifying how large either one is.
  - A third problem is that it is not always possible to deduce the types of the parameters, not even from a formal specification or the code itself.
  - A fourth problem relates to the use of global variables.

The Real-Time Transport Protocol

- Client-server RPC is one area in which UDP is widely used.
- **Real-time Transport Protocol** run over user datagram protocol.
- **Real-time Transport Protocol** used in multimedia applications such as Internet radio, Internet telephony, music-on-demand, videoconferencing, video-on-demand. Audio, video and text are the content of the multimedia.
- Multimedia application also contains other types of data streams. All these data is stored into the RTP library in user space along with the application. This library then multiplexes the streams and encodes them in RTP packets, which it then stuffs into a socket.

The Real-Time Transport Protocol

- Below figure(b) shows the RTP and packet nesting.
- Socket means communication end points.
- At the other sides of socket, UDP packets are generated and it is embedded in the IP packets. RTP uses user space and linked with the application program.
- So that it look like an application protocol.
- RTP is a generic and application independent protocol.

The Real-Time Transport Protocol

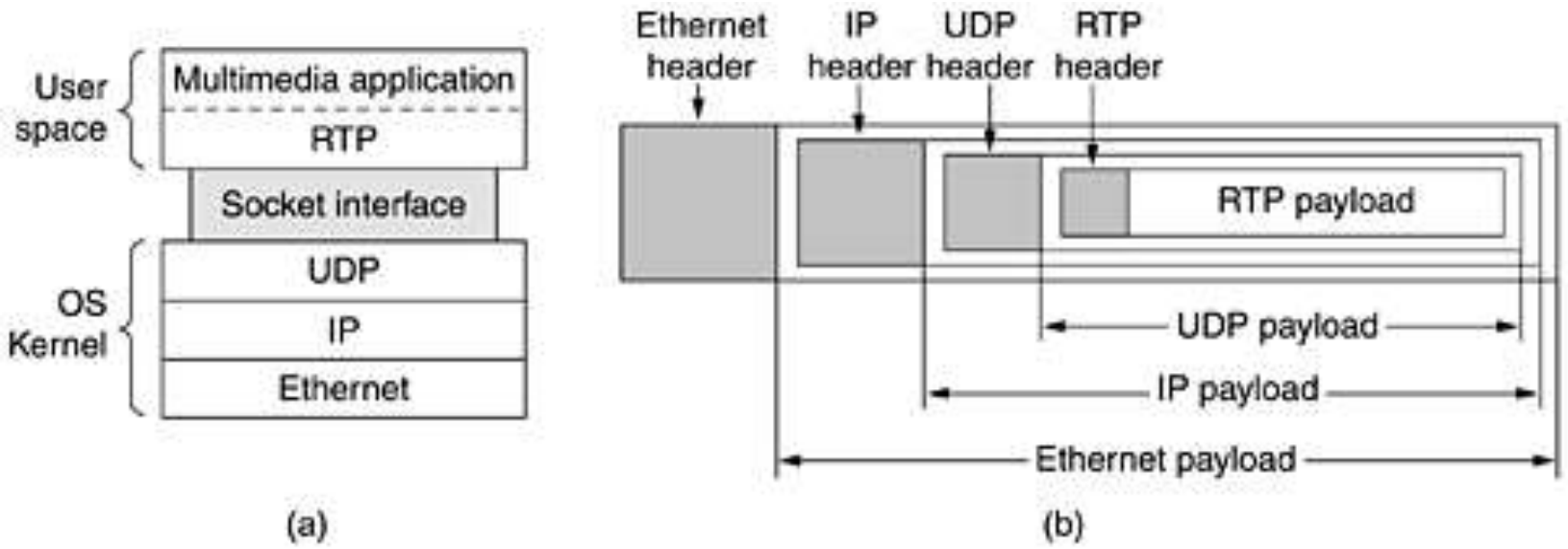


Figure: (a) The position of RTP in the protocol stack. (b) Packet nesting

The Real-Time Transport Protocol

- The basic function of RTP is to multiplex several real-time data streams onto a single stream of UDP packets.
- The UDP stream can be sent to a single destination (unicasting) or to multiple destinations (multicasting).
- Because RTP just uses normal UDP, its packets are not treated specially by the routers unless some normal IP quality-of-service features are enabled.
- In particular, there are no special guarantees about delivery, jitter, etc.

The Real-Time Transport Protocol

- Each packet sent in an RTP stream is given a number one higher than its predecessor.
- This numbering allows the destination to determine if any packets are missing.
- If a packet is missing, the best action for the destination to take is to approximate the missing value by interpolation.
- Retransmission is not a practical option since the retransmitted packet would probably arrive too late to be useful.
- As a consequence, RTP has no flow control, no error control, no acknowledgements, and no mechanism to request retransmissions.

The Real-Time Transport Protocol

- The RTP header is illustrated in below figure.
- It consists of three 32-bit words and potentially some extensions.
- The first word contains the **Version** field, which is already at 2.
- The **P** bit indicates that the packet has been padded to a multiple of 4 bytes. The last padding byte tells how many bytes were added.
- The **X** bit indicates that an extension header is present.

The Real-Time Transport Protocol

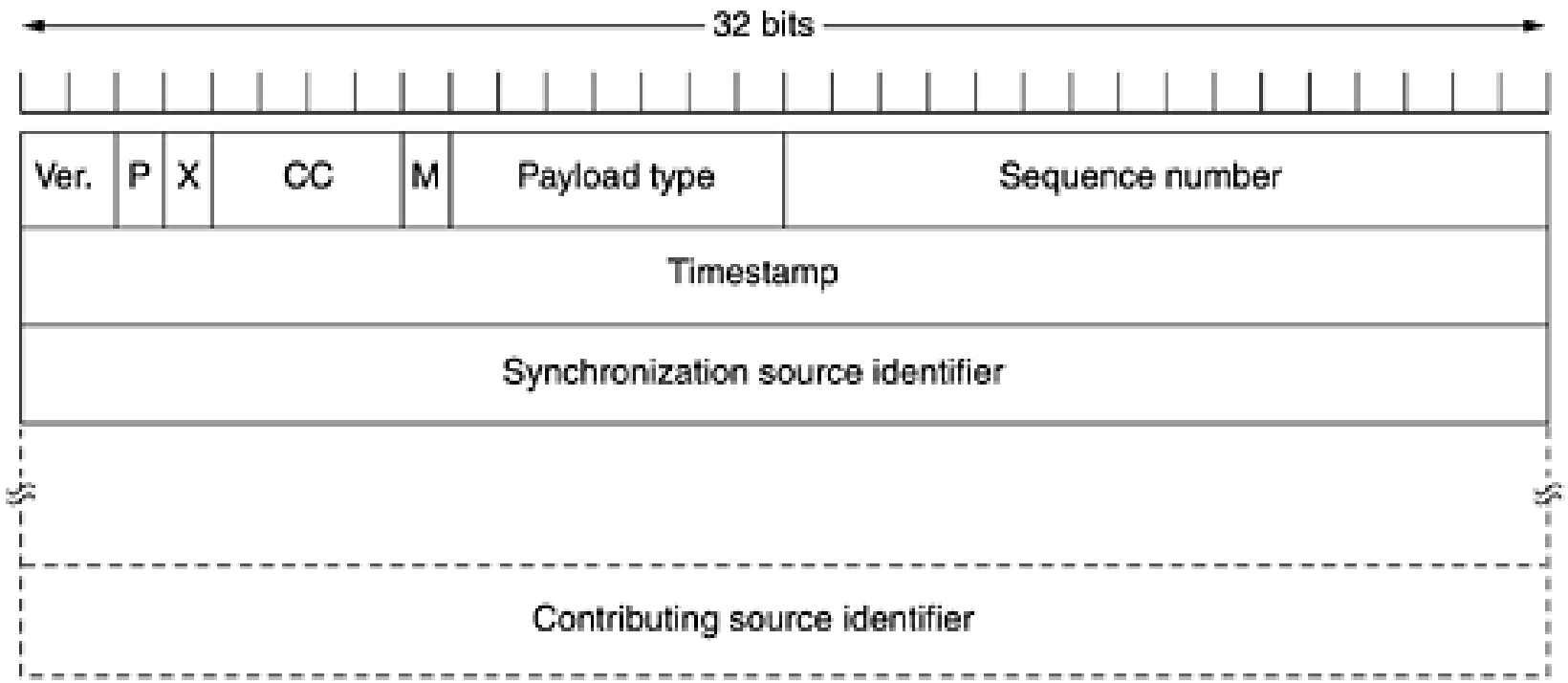


Figure: The RTP header

The Real-Time Transport Protocol

- The **CC** field tells how many contributing sources are present, from 0 to 15.
- The **M** bit is an application-specific marker bit. It can be used to mark the start of a video frame, the start of a word in an audio channel, or something else that the application understands.
- The **Payload** type field tells which encoding algorithm has been used (e.g., uncompressed 8-bit audio, MP3, etc.). Since every packet carries this field, the encoding can change during transmission.
- The **Sequence** number is just a counter that is incremented on each RTP packet sent. It is used to detect lost packets.

The Real-Time Transport Protocol

- The **timestamp** is produced by the stream's source to note when the first sample in the packet was made. This value can help reduce jitter at the receiver by decoupling the playback from the packet arrival time.
- The **Synchronization source identifier** tells which stream the packet belongs to. It is the method used to multiplex and demultiplex multiple data streams onto a single stream of UDP packets.
- The **Contributing source identifiers**, if any, are used when mixers are present in the studio. In that case, the mixer is the synchronizing source, and the streams being mixed are listed here.

UNIT 4 PART 2

The Internet Transport Protocol (TCP): Introduction to TCP, The TCP Service Model, The TCP Protocol, The TCP Segment Header, TCP Connection Establishment, TCP Connection Release, Modeling TCP Connection Management, TCP Transmission Policy, TCP Congestion Control, TCP Timer Management.

The Internet Transport Protocol (TCP)

- UDP is a simple protocol and it has some niche uses, such as client-server interactions and multimedia, but for most Internet applications, reliable, sequenced delivery is needed.
- UDP cannot provide this, so another protocol is required.
- It is called TCP and is the main workhorse of the Internet.

Introduction to TCP

- TCP (Transmission Control Protocol) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork.
- An internetwork differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters.
- TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

Introduction to TCP

- Each machine supporting TCP has a TCP transport entity, either a library procedure, a user process, or part of the kernel.
- In all cases, it manages TCP streams and interfaces to the IP layer.
- A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64 KB and sends each piece as a separate IP datagram.
- When datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.

Introduction to TCP

- For simplicity, we will sometimes use just ''TCP'' to mean the TCP transport entity (a piece of software) or the TCP protocol (a set of rules). From the context it will be clear which is meant. For example, in ''The user gives TCP the data,'' the TCP transport entity is clearly intended.

Introduction to TCP

- The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be.
- Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence.
- In short, TCP must furnish the reliability that most users want and that IP does not provide.

The TCP Service Model

- TCP service is obtained by both the sender and receiver creating end points, called sockets.
- Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port.**
- A port is the TCP name for a TSAP.
- For TCP service to be obtained, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine.

The TCP Service Model

- A socket may be used for multiple connections at the same time.
- In other words, two or more connections may terminate at the same socket.
- Connections are identified by the socket identifiers at both ends, that is, (socket1, socket2).
- No virtual circuit numbers or other identifiers are used.

The TCP Service Model

- Port numbers below 1024 are called well-known ports and are reserved for standard services.
- For example, any process wishing to establish a connection to a host to transfer a file using FTP can connect to the destination host's port 21 to contact its FTP daemon.

The TCP Service Model

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial file transfer protocol
79	Finger	Lookup information about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

The TCP Service Model

- All TCP connections are full duplex and point-to-point.
  - Full duplex means that traffic can go in both directions at the same time.
  - Point-to-point means that each connection has exactly two end points. TCP does not support multicasting or broadcasting.

The TCP Service Model

- A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end.
- For example, if the sending process does four 512-byte writes to a TCP stream, these data may be delivered to the receiving process as four 512-byte chunks, two 1024-byte chunks, one 2048-byte chunk (see below figure), or some other way. There is no way for the receiver to detect the unit(s) in which the data were written.

The TCP Service Model

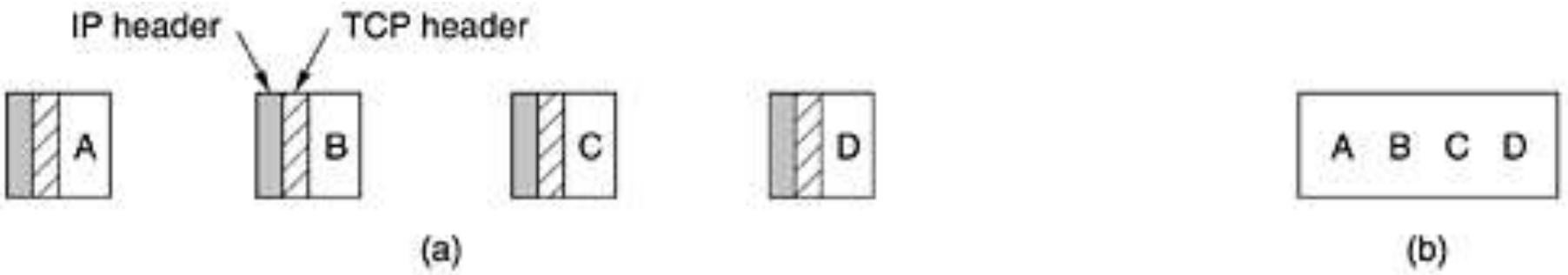


Figure: (a) Four 512-byte segments sent as separate IP datagrams.
(b) The 2048 bytes of data delivered to the application in a single READ call.

The TCP Service Model

- When an application passes data to TCP, TCP may send it immediately or buffer it (in order to collect a larger amount to send at once), at its discretion. Sometimes, the application really wants the data to be sent immediately.
- To force data out, applications can use the PUSH flag, which tells TCP not to delay the transmission.

The TCP Service Model

- One last feature of the TCP service that is worth mentioning here is **urgent data**.
- When an interactive user hits the DEL or CTRL-C key to break off a remote computation that has already begun, the sending application puts some control information in the data stream and gives it to TCP along with the URGENT flag.
- This event causes TCP to stop accumulating data and transmit everything it has for that connection immediately.

OR

The TCP Service Model

- The **TCP (Transmission Control Protocol) Service Model** provides reliable, ordered, and error-checked delivery of data over a network.
- It operates at the **Transport Layer** of the OSI and TCP/IP models and is a cornerstone of internet communication, particularly for applications that require accuracy and dependability, like web browsing, file transfers, and email.
- Here are some key features of the TCP Service Model:

1. Connection-Oriented Communication

- TCP establishes a connection between the sender and receiver before data is transmitted.
- This connection is created through a **three-way handshake** (SYN, SYN-ACK, ACK) that synchronizes the sender and receiver and establishes initial sequence numbers for tracking data packets.

2. Reliable Data Transfer

- TCP ensures that data is delivered accurately and in the correct order.
- It uses **acknowledgments (ACKs)** to confirm the receipt of data segments, and if any segment is lost or corrupted, TCP will retransmit it.
- **Error-checking** is done through checksums to verify the integrity of data.

3. Flow Control

- TCP controls the rate of data transmission to avoid overwhelming the receiver, using a **sliding window mechanism**.
- The receiver specifies a **window size** to indicate how much data it can handle at once. This allows TCP to adjust the sending rate to match the receiver’s capacity.

4. Congestion Control

- TCP includes mechanisms to manage network congestion. It adjusts the data transmission rate based on network conditions.
- **Congestion control algorithms** like **Slow Start**, **Congestion Avoidance**, **Fast Retransmit**, and **Fast Recovery** help optimize data flow, reducing the likelihood of congestion and maintaining smooth network performance.

5. Ordered Data Delivery

- TCP numbers each byte of data and ensures that data arrives at the receiver in the correct order.
- If packets arrive out of sequence, TCP will reorder them before delivering the data to the application layer.

6. Full-Duplex Communication

- TCP supports **bi-directional communication**, allowing data to be sent and received simultaneously between the sender and receiver.

Summary

- The TCP Service Model is essential for applications where data integrity, order, and reliability are critical.
- While TCP provides these robust features, it comes with added overhead and complexity, which can make it less efficient for real-time or lightweight applications (such as video calls or online gaming).
- In such cases, **UDP (User Datagram Protocol)**, which is connectionless and does not guarantee reliability, may be preferred for faster data transfer.

The TCP Protocol

- A key feature of TCP is that every byte on a TCP connection has its own 32-bit sequence number.
- Separate 32-bit sequence numbers are used for acknowledgements and for the window mechanism.

The TCP Protocol

1. Segments

- The sending and receiving TCP entities exchange data in the form of segments.
- A TCP segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes.

The TCP Protocol

2. Segment Size

- The TCP software decides how big segments should be.
- It can accumulate data from several writes into one segment or can split data from one write over multiple segments.
- Two limits restrict the segment size.
  - First, each segment, including the TCP header, must fit in the 65,515-byte IP payload.
  - Second, each network has a **maximum transfer unit**, or **MTU**, and each segment must fit in the MTU. In practice, the MTU is generally 1500 bytes (the Ethernet payload size) and thus defines the upper bound on segment size.

The TCP Protocol

3. Fragmentation

- If a segment is too large, then it should be taken into small segments.
- Using fragmentation by a router, each new segment gets a new IP header.
- Therefore, the fragmentation by router will increase the overhead.

The TCP Protocol

4. Timer

- The basic protocol used by TCP entities is the sliding window protocol.
- When a sender transmits a segment, it also starts a timer.
- When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive.
- If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

The TCP Protocol

5. Possible Problems

- As the segments can be fragmented, a part of the transmitted segment only may reach the destination with the remaining part lost.
- Segments can arrive out of order. For example, bytes 3072–4095 can arrive but cannot be acknowledged because bytes 2048–3071 have not turned up yet.
- Segments can also be delayed so long in transit that the sender times out and retransmits them.
- The retransmissions may include different byte ranges than the original transmission, requiring a careful administration to keep track of which bytes have been correctly received so far. However, since each byte in the stream has its own unique offset, it can be done.
- There is possibility of congestion or broken network along the path. TCP Should be able to solve these problems in an efficient manner.

The TCP Segment Header

- Below figure shows the layout of a TCP segment.
- Every segment begins with a fixed-format, 20-byte header.
- The fixed header may be followed by header options.
- After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header.
- Segments without any data are legal and are commonly used for acknowledgements and control messages.

The TCP Segment Header

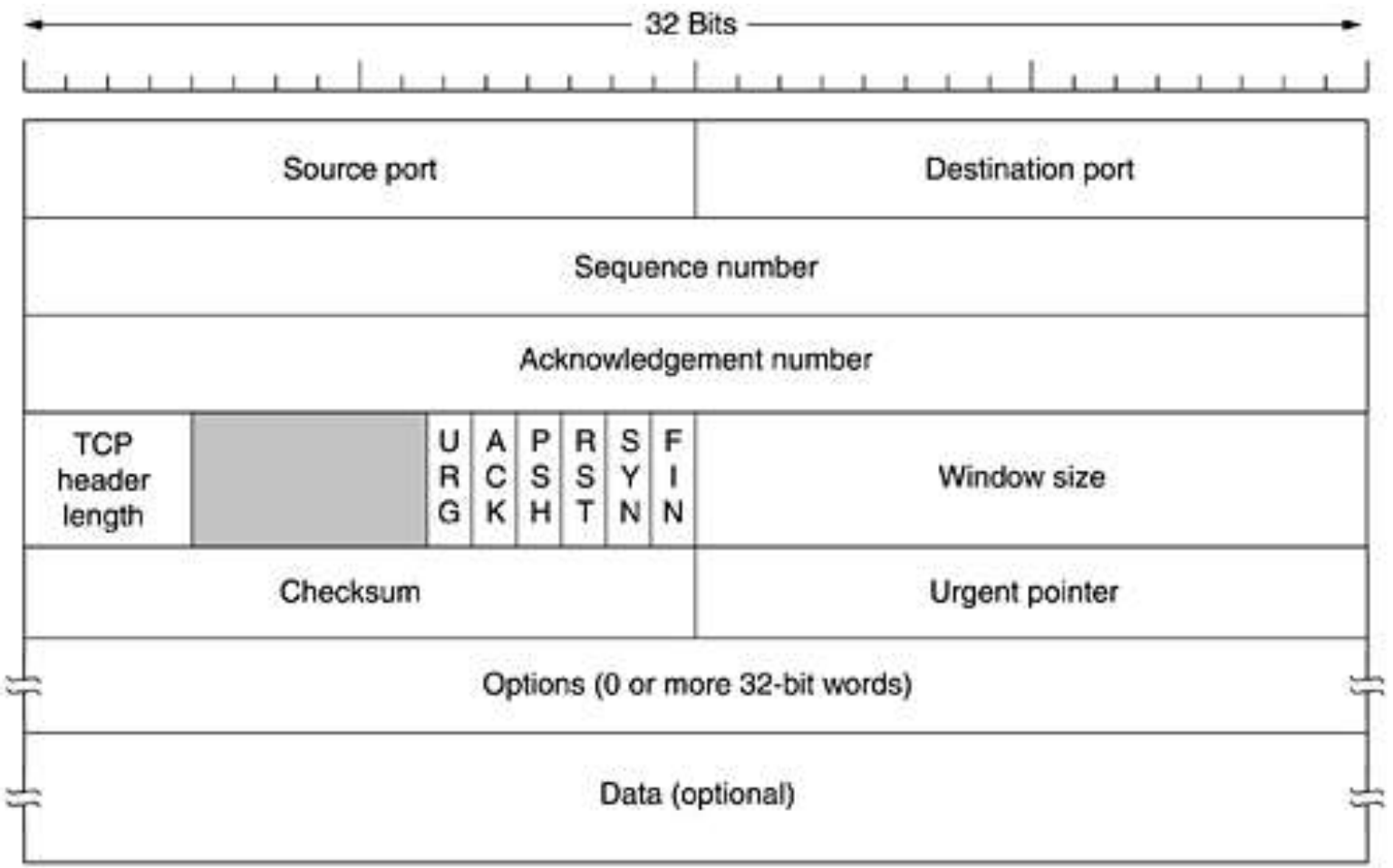


Figure: The TCP header

The TCP Segment Header

- The **Source port** and **Destination port** fields identify the local end points of the connection. A port plus its host's IP address forms a 48-bit unique end point. The source and destination end points together identify the connection.
- The **Sequence number** and **Acknowledgement number** fields perform their usual functions. Both are 32 bits long because every byte of data is numbered in a TCP stream.

The TCP Segment Header

- The **TCP header length** tells how many 32-bit words are contained in the TCP header. This information is needed because the **Options** field is of variable length, so the header is, too.
- Next comes a **6-bit field that is not used**.
- Now come **six 1-bit flags**.
  - **URG** is set to 1 if the Urgent pointer is in use. The **Urgent pointer** is used to indicate a byte offset from the current sequence number at which urgent data are to be found.

The TCP Segment Header

- The **ACK** bit is set to 1 to indicate that the Acknowledgement number is valid. If ACK is 0, the segment does not contain an acknowledgement so the Acknowledgement number field is ignored.
- The **PSH** bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency).
- The **RST** bit is used to reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if you get a segment with the RST bit on, you have a problem on your hands.

The TCP Segment Header

- The **SYN** bit is used to establish connections. The connection request has SYN = 1 and ACK = 0 to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has SYN = 1 and ACK = 1.
- The **FIN** bit is used to release a connection. It specifies that the sender has no more data to transmit.

The TCP Segment Header

- The **Window size** field tells how many bytes may be sent starting at the byte acknowledged. A **Window size** field of 0 is legal and says that the bytes up to and including **Acknowledgement number** - 1 have been received, but that the receiver is currently badly in need of a rest and would like no more data for the moment, thank you. The receiver can later grant permission to send by transmitting a segment with the same **Acknowledgement number** and a nonzero **Window size** field.

The TCP Segment Header

- A **Checksum** is also provided for extra reliability. It checksums the header, the data, and the conceptual pseudoheader shown in below figure.
- When performing this computation, the TCP **Checksum** field is set to zero and the data field is padded out with an additional zero byte if its length is an odd number.
- The checksum algorithm is simply to add up all the 16-bit words in one's complement and then to take the one's complement of the sum.
- As a consequence, when the receiver performs the calculation on the entire segment, including the **Checksum** field, the result should be 0.

The TCP Segment Header

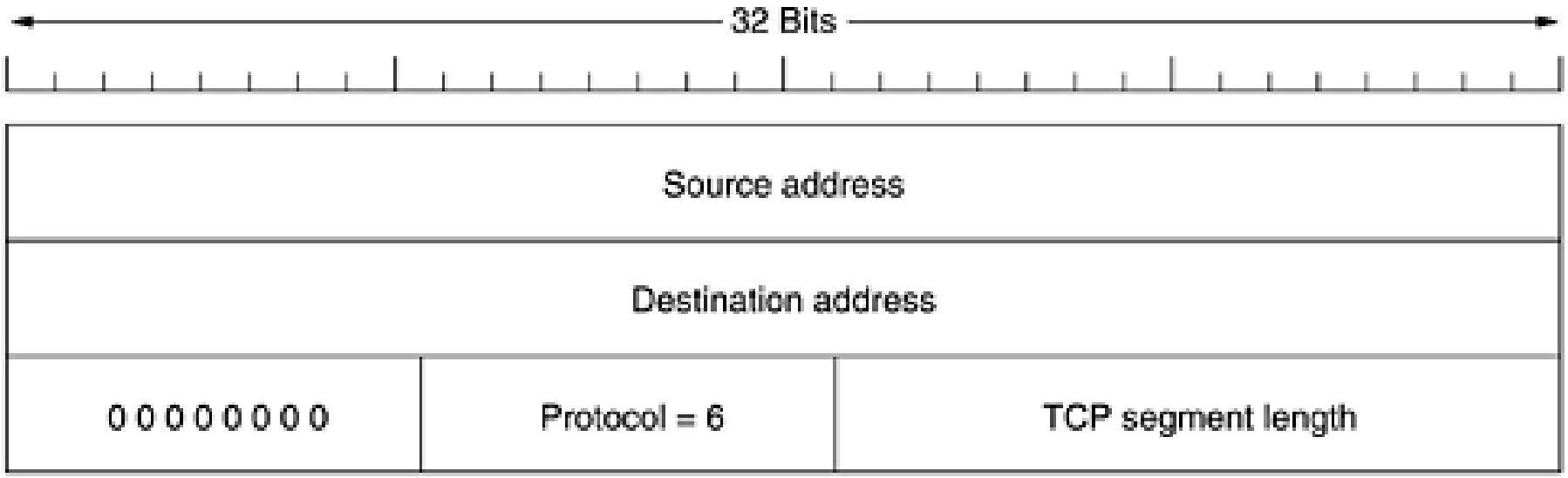


Figure: The pseudoheader included in the TCP checksum

The TCP Segment Header

- The pseudoheader contains the 32-bit IP addresses of the source and destination machines, the protocol number for TCP (6), and the byte count for the TCP segment (including the header).
- Including the pseudoheader in the TCP checksum computation helps detect misdelivered packets, but including it also violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not to the TCP layer.
- UDP uses the same pseudoheader for its checksum.

The TCP Segment Header

- The **Options** field provides a way to add extra facilities not covered by the regular header.
- The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept.
- Using large segments is more efficient than using small ones because the 20-byte header can then be amortized over more data, but small hosts may not be able to handle big segments.
- During connection setup, each side can announce its maximum and see its partner's.
- If a host does not use this option, it defaults to a 536-byte payload.
- All Internet hosts are required to accept TCP segments of 536 + 20 = 556 bytes.
- The maximum segment size in the two directions need not be the same.

TCP Connection Establishment

- Connection establishment in a TCP session is initialized through a **three-way handshake**.
- To establish a connection, one side (**server**) passively waits for an incoming connection by executing the **LISTEN** and **ACCEPT** primitives, either specifying a specific source.
- The other side (**client**) executes a **CONNECT** primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password).
- The **CONNECT** primitive sends a TCP segment with the **SYN** bit on and **ACK** bit off and waits for a response.

TCP Connection Establishment

- When this segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a **LISTEN** on the port given in the **Destination port** field. If not, it sends a reply with the **RST** bit on to reject the connection.
- If some process is listening to the port, that process is given the incoming TCP segment. It can then either accept or reject the connection. If it accepts, an acknowledgement segment is sent back.

TCP Connection Establishment

- The sequence of TCP segments sent in the normal case is shown in figure(a).
- Note that a SYN segment consumes 1 byte of sequence space so that it can be acknowledged unambiguously.

TCP Connection Establishment

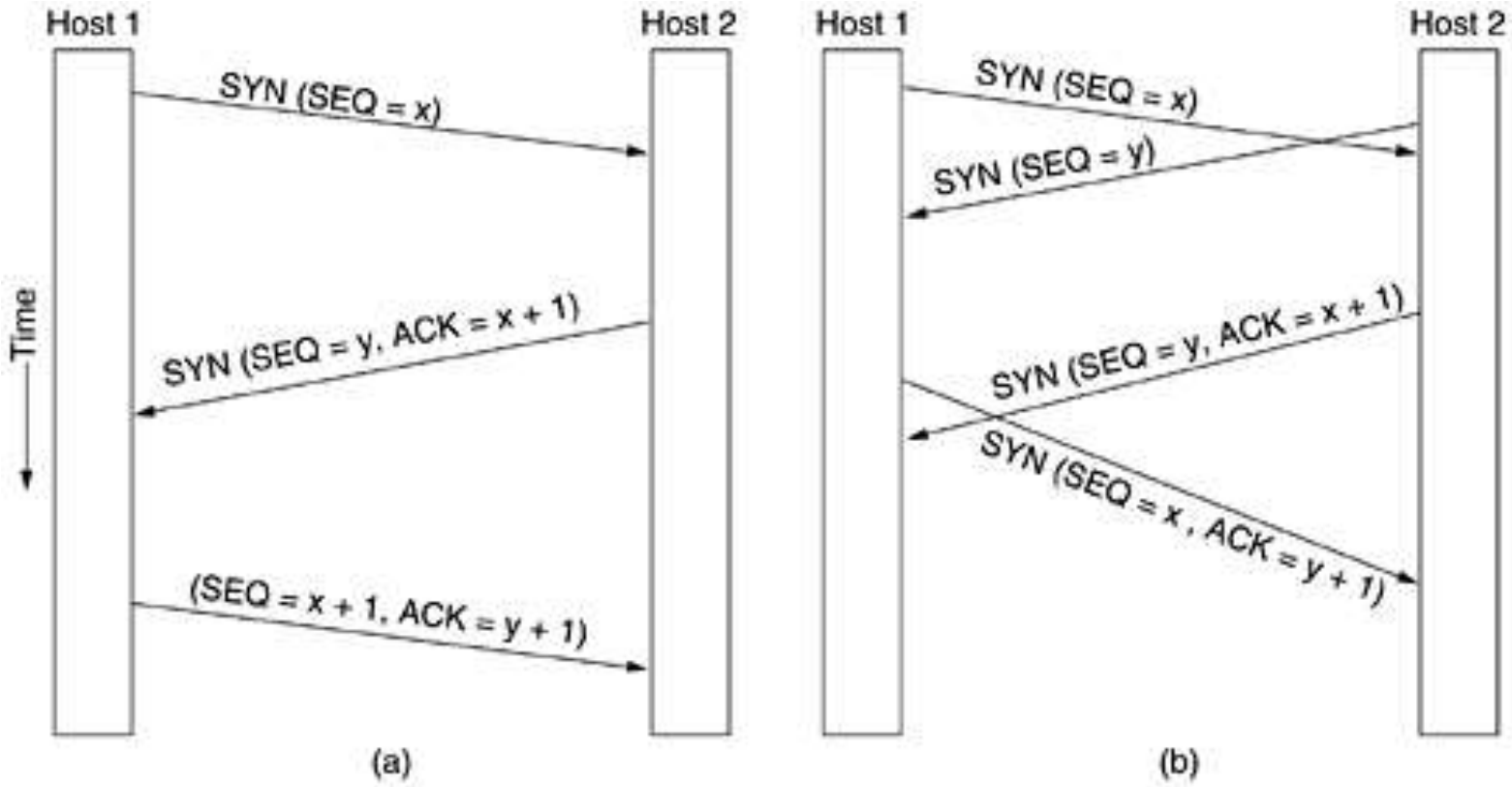


Figure: (a) TCP connection establishment in the normal case (b) Call collision

TCP Connection Establishment

- In the event that two hosts simultaneously attempt to establish a connection between the same two sockets, the sequence of events is as illustrated in figure(b).
- The result of these events is that just one connection is established, not two because connections are identified by their end points.
- If the first setup results in a connection identified by (x, y) and the second one does too, only one table entry is made, namely, for (x, y).

TCP Connection Establishment

- The initial sequence number on a connection is not 0.
- A clock-based scheme is used, with a clock tick every 4 μsec.
- For additional safety, when a host crashes, it may not reboot for the maximum packet lifetime to make sure that no packets from previous connections are still roaming around the Internet somewhere.

TCP Connection Release

- Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections.
- Each simplex connection is released independently of its sibling.

TCP Connection Release

- To release a connection, either party can send a TCP segment with the **FIN** bit set, which means that it has no more data to transmit.
- When the **FIN** is acknowledged, that direction is shut down for new data.
- Data may continue to flow indefinitely in the other direction, however.
- When both directions have been shut down, the connection is released.
- Normally, four TCP segments are needed to release a connection, one **FIN** and one **ACK** for each direction.
- However, it is possible for the first **ACK** and the second **FIN** to be contained in the same segment, reducing the total count to three.

TCP Connection Release

- Just as with telephone calls in which both people say goodbye and hang up the phone simultaneously, both ends of a TCP connection may send **FIN** segments at the same time.
- These are each acknowledged in the usual way, and the connection is shut down.
- There is, in fact, no essential difference between the two hosts releasing sequentially or simultaneously.

TCP Connection Release

- To avoid the two-army problem, timers are used.
- If a response to a **FIN** is not forthcoming within two maximum packet lifetimes, the sender of the **FIN** releases the connection.
- The other side will eventually notice that nobody seems to be listening to it any more and will time out as well.

TCP Connection Release

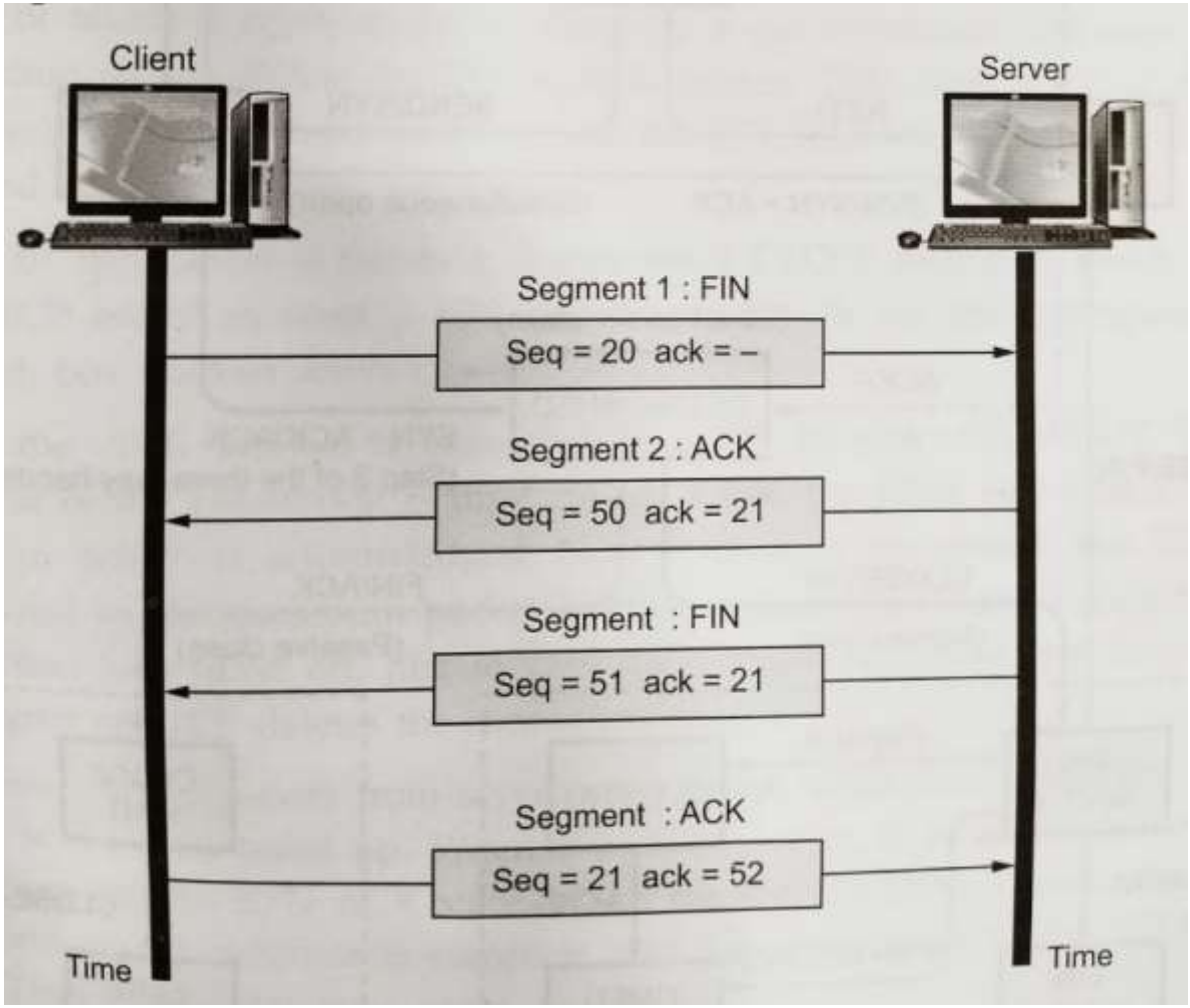


Figure: Four steps connection termination

TCP Connection Management Modeling

- The steps required to **establish and release connections** can be represented in a **finite state machine with the 11 states** listed in below figure.
- In each state, certain events are legal.
- When a legal event happens, some action may be taken.
- If some other event happens, an error is reported.

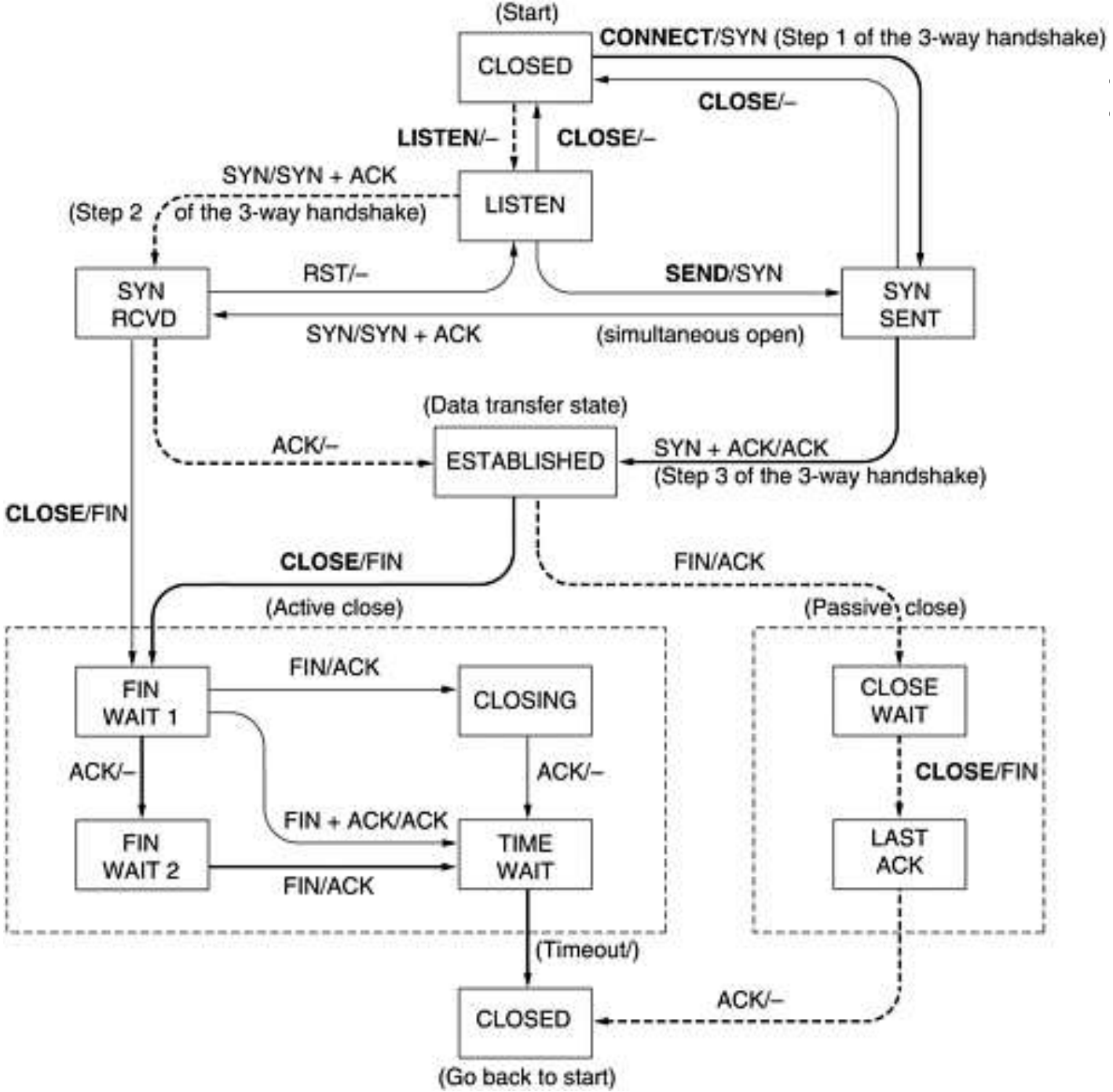
TCP Connection Management Modeling

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Figure: The states used in the TCP connection management finite state machine

TCP Connection Management Modeling

- Each connection starts in the **CLOSED** state.
- It leaves that state when it does either a passive open (**LISTEN**), or an active open (**CONNECT**).
- If the other side does the opposite one, a connection is established and the state becomes **ESTABLISHED**.
- Connection release can be initiated by either side.
- When it is complete, the state returns to **CLOSED**.


(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way handshake)
CLOSE/–
LISTEN/–
CLOSE/–
SYN/SYN + ACK
(Step 2 of the 3-way handshake)
LISTEN
RST/–
SEND/SYN
SYN RCVD
SYN SENT
SYN/SYN + ACK
(simultaneous open)
(Data transfer state)
ACK/–
ESTABLISHED
SYN + ACK/ACK
(Step 3 of the 3-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
(Active close)
(Passive close)
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/–
ACK/–
CLOSE/FIN
FIN + ACK/ACK
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
(Timeout/)
CLOSED
ACK/–
(Go back to start)

TCP Connection Management Modeling

- The finite state machine itself is shown in below figure.
- The common case of a client actively connecting to a passive server is shown with heavy lines—**solid** for the **client**, **dotted** for the **server**.
- The lightface lines are unusual event sequences. Each line in below figure is marked by an event/action pair.
- The event can either be a user-initiated system call (**CONNECT, LISTEN, SEND,** or **CLOSE**), a segment arrival (**SYN, FIN, ACK**, or **RST**), or, in one case, a timeout of twice the maximum packet lifetime.
- The action is the sending of a control segment (**SYN, FIN, or RST**) or nothing, indicated by —.
- Comments are shown in parentheses.

TCP Connection Management Modeling

- One can best understand the diagram by first following the path of a client (the heavy solid line), then later following the path of a server (the heavy dashed line).
- When an application program on the client machine issues a **CONNECT** request, the local TCP entity creates a connection record, marks it as being in the **SYN SENT** state, and sends a **SYN** segment.

TCP Connection Management Modeling

- Note that many connections may be open (or being opened) at the same time on behalf of multiple applications, so the state is per connection and recorded in the connection record.
- When the **SYN+ACK** arrives, TCP sends the final **ACK** of the three-way handshake and switches into the **ESTABLISHED** state.
- Data can now be sent and received.

TCP Connection Management Modeling

- When an application is finished, it executes a **CLOSE** primitive, which causes the local TCP entity to send a **FIN** segment and wait for the corresponding **ACK** (dashed box marked active close).
- When the **ACK** arrives, a transition is made to state **FIN WAIT 2** and one direction of the connection is now closed.
- When the other side closes, too, a **FIN** comes in, which is acknowledged.
- Now both sides are closed, but TCP waits a time equal to the maximum packet lifetime to guarantee that all packets from the connection have died off, just in case the acknowledgement was lost.
- When the timer goes off, TCP deletes the connection record.

TCP Connection Management Modeling

- Now let us examine connection management from the server's viewpoint.
- The server does a **LISTEN** and settles down to see who turns up.
- When a **SYN** comes in, it is acknowledged and the server goes to the **SYN RCVD** state.
- When the server's **SYN** is itself acknowledged, the three-way handshake is complete and the server goes to the **ESTABLISHED** state.
- Data transfer can now occur.

TCP Connection Management Modeling

- When the client is done, it does a **CLOSE**, which causes a FIN to arrive at the server (dashed box marked passive close).
- The server is then signaled. When it, too, does a **CLOSE**, a **FIN** is sent to the client.
- When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

TCP Transmission Policy

- As mentioned earlier, window management in TCP is not directly tied to acknowledgements as it is in most data link protocols.
- For example, suppose the receiver has a 4096-byte buffer, as shown in below figure.

TCP Transmission Policy

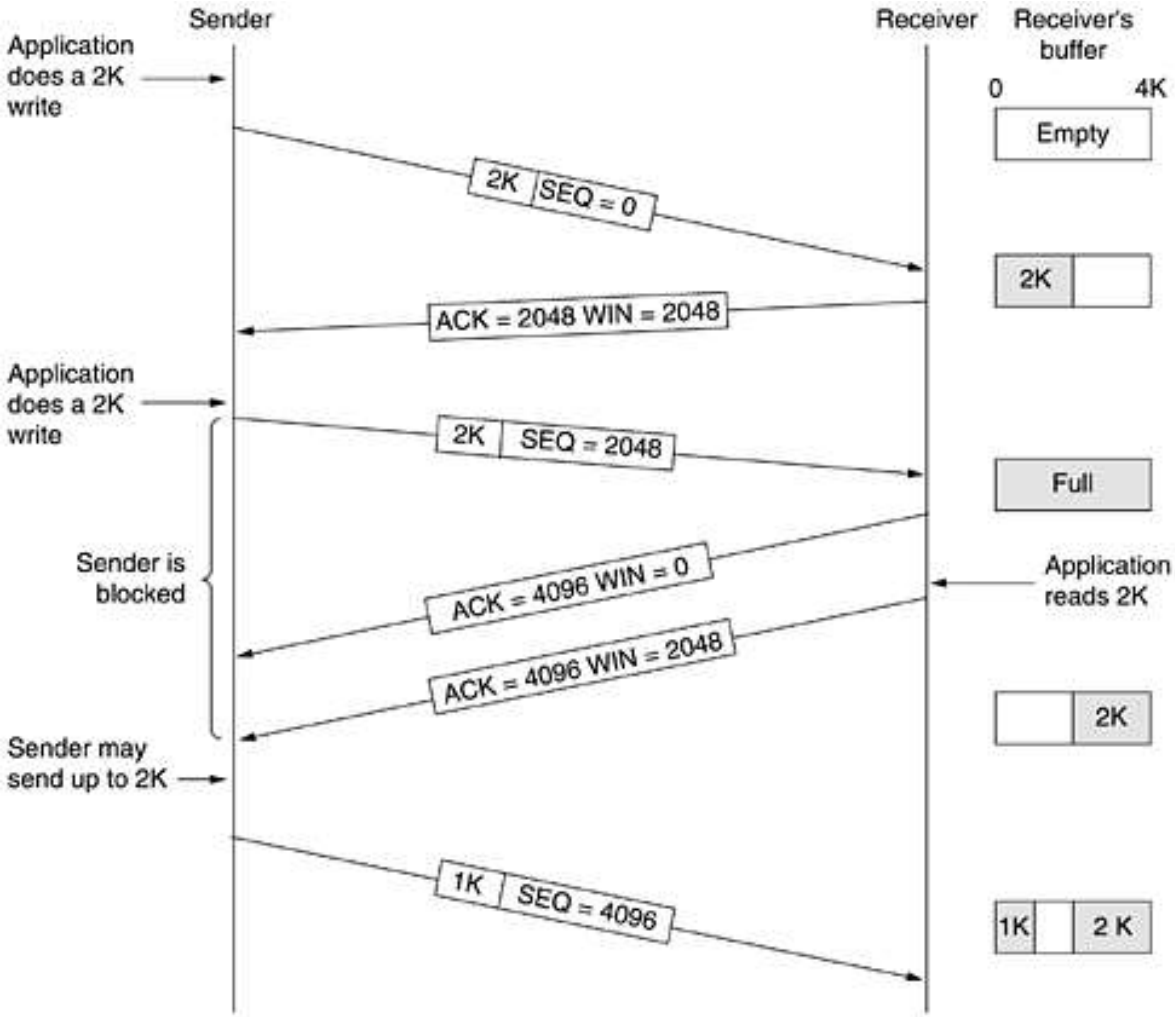


Figure: Window management in TCP

TCP Transmission Policy

- If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
- However, since it now has only 2048 bytes of buffer space (until the application removes some data from the buffer), it will advertise a window of 2048 starting at the next byte expected.

TCP Transmission Policy

- Now the sender transmits another 2048 bytes, which are acknowledged, but the advertised window is 0.
- The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a larger window.

TCP Transmission Policy

- When the window is 0, the sender may not normally send segments, with two exceptions.
  - First, urgent data may be sent, for example, to allow the user to kill the process running on the remote machine.
  - Second, the sender may send a 1-byte segment to make the receiver reannounce the next byte expected and window size. The TCP standard explicitly provides this option to prevent deadlock if a window announcement ever gets lost.

TCP Transmission Policy

- Problem that can degrade TCP performance is the **silly window syndrome**.
- This problem occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads data 1 byte at a time.
- To see the problem, look at below figure.

TCP Transmission Policy

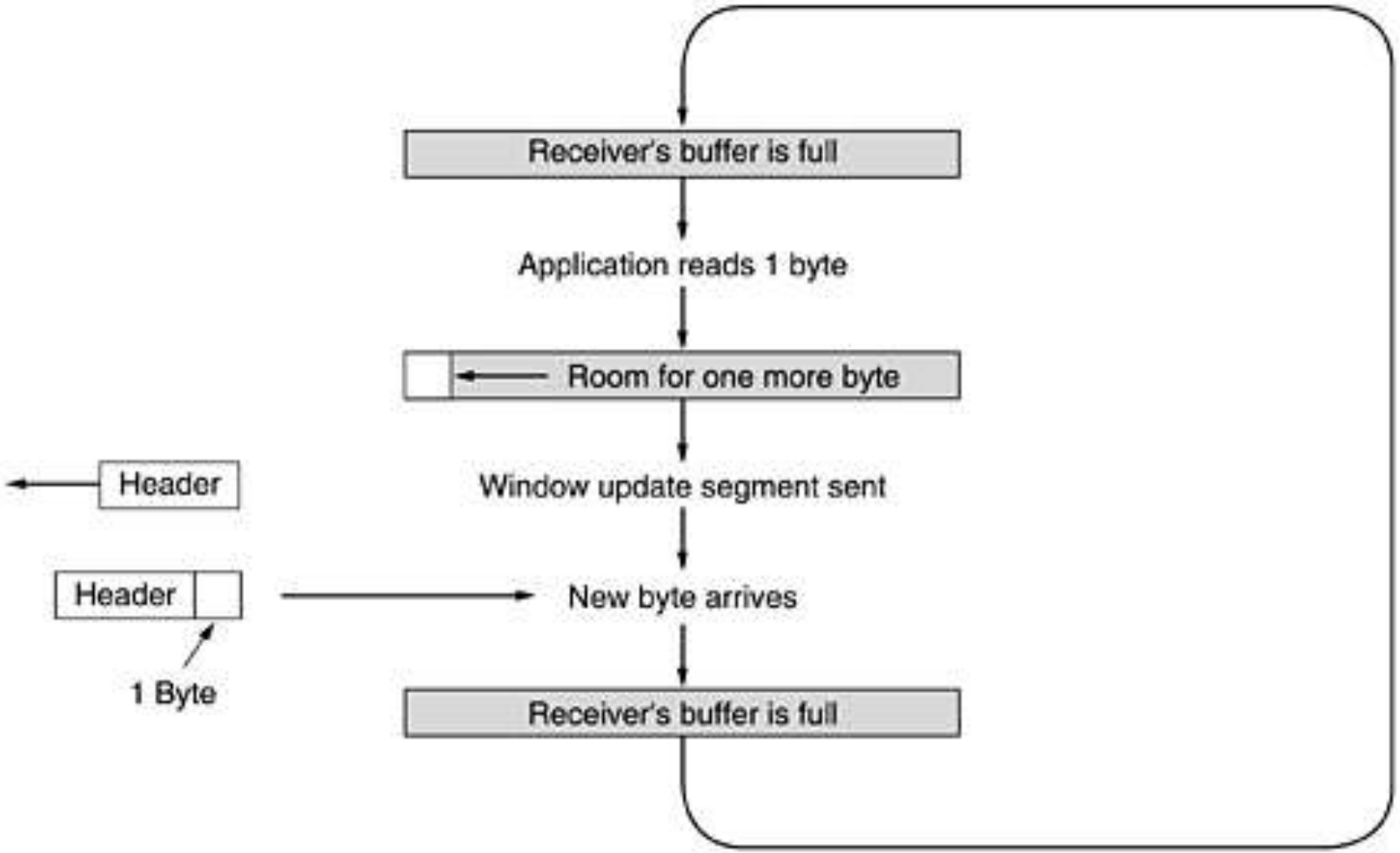


Figure: Silly window syndrome

TCP Transmission Policy

- Initially, the TCP buffer on the receiving side is full and the sender knows this (i.e., has a window of size 0).
- Then the interactive application reads one character from the TCP stream.
- This action makes the receiving TCP happy, so it sends a window update to the sender saying that it is all right to send 1 byte.
- The sender obliges and sends 1 byte.
- The buffer is now full, so the receiver acknowledges the 1-byte segment but sets the window to 0.
- This behavior can go on forever.

TCP Transmission Policy

- **Solution for this problem** : **Nagle's Algorithm (Sender side)**
- when data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged.
- Then send all the buffered characters in one TCP segment and start buffering again until they are all acknowledged.
- If the user is typing quickly and the network is slow, a substantial number of characters may go in each segment, greatly reducing the bandwidth used.
- The algorithm additionally allows a new packet to be sent if enough data have trickled in to fill half the window or a maximum segment.

TCP Transmission Policy

- **Clark's solution (Receiver Side)**
- Clark's solution is to prevent the receiver from sending a window update for 1 byte.
- Instead it is forced to wait until it has a decent amount of space available and advertise that instead.
- Specifically, the receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established or until its buffer is half empty, whichever is smaller.
- Furthermore, the sender can also help by not sending tiny segments.
- Instead, it should try to wait until it has accumulated enough space in the window to send a full segment or at least one containing half of the receiver's buffer size (which it must estimate from the pattern of window updates it has received in the past).

TCP Transmission Policy

- **Nagle's algorithm** and **Clark's solution** to the silly window syndrome are complementary.
- **Nagle** was trying to solve the problem caused by the sending application delivering data to TCP a byte at a time.
- **Clark** was trying to solve the problem of the receiving application sucking the data up from TCP a byte at a time.
- Both solutions are valid and can work together.
- The goal is for the sender not to send small segments and the receiver not to ask for them.

TCP Congestion Control

- When the load offered to any network is more than it can handle, congestion builds up.
- The Internet is no exception.
- Although the network layer also tries to manage congestion, most of the heavy lifting is done by TCP because the real solution to congestion is to slow down the data rate.

TCP Congestion Control

- The first step in managing congestion is detecting it.
- In the old days, detecting congestion was difficult.
- A timeout caused by a lost packet could have been caused by either (1) noise on a transmission line or (2) packet discard at a congested router.
- Telling the difference was difficult.

TCP Congestion Control

- Before discussing how TCP reacts to congestion, let us first describe what it does to try to prevent congestion from occurring in the first place.
- When a connection is established, a suitable window size has to be chosen. The receiver can specify a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end, but they may still occur due to internal congestion within the network.

TCP Congestion Control

- In the below figure, we see this problem illustrated hydraulically.
- In figure(a), we see a thick pipe leading to a small-capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost.
- In figure(b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network. If too much water comes in too fast, it will back up and some will be lost (in this case by overflowing the funnel).

TCP Congestion Control

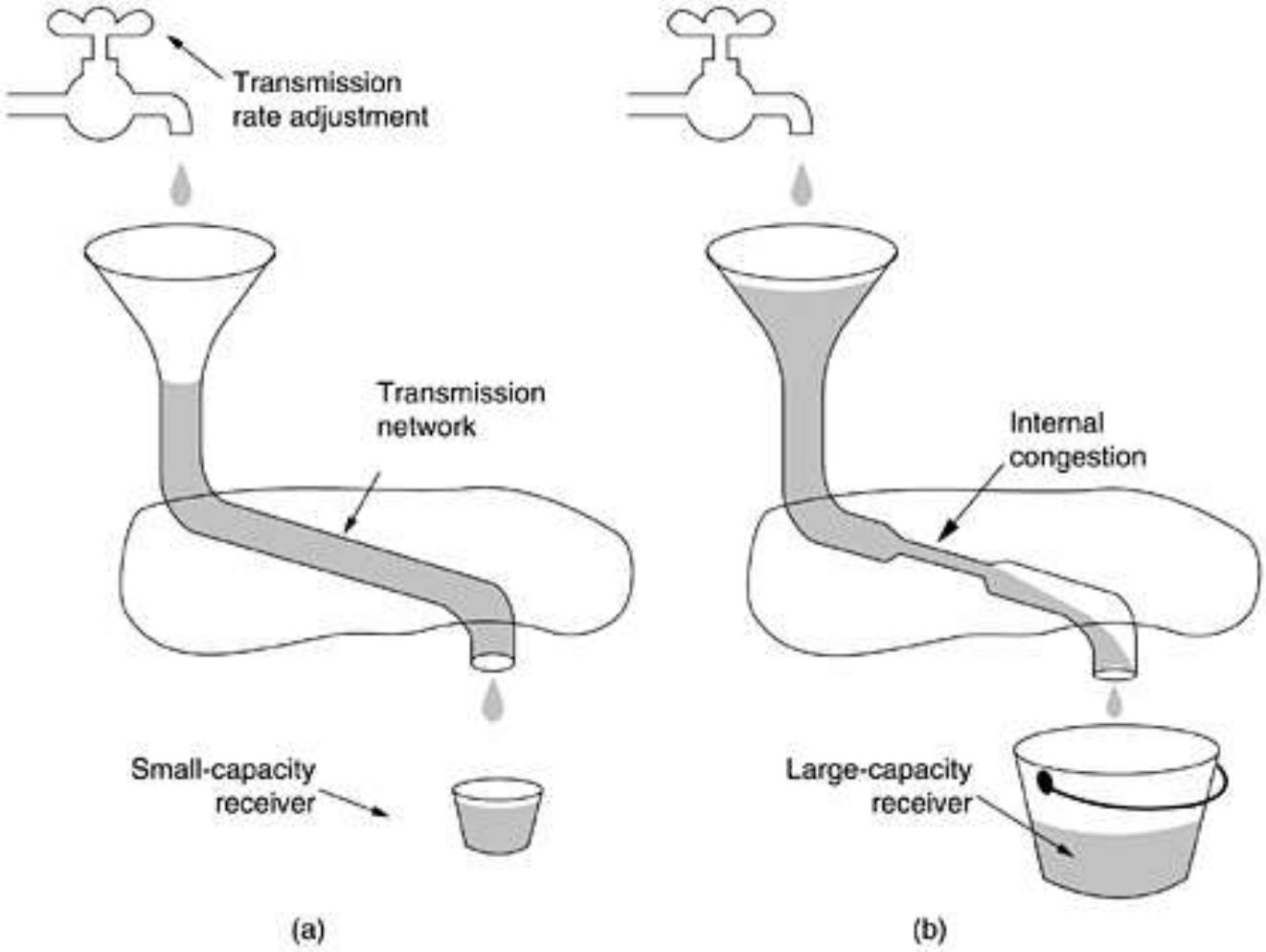


Figure:(a) A fast network feeding a low-capacity receiver (b) A slow network feeding a high-capacity receiver

TCP Congestion Control

- When a connection is established, the sender initializes the congestion window to the size of the maximum segment in use on the connection.
- It then sends one maximum segment. If this segment is acknowledged before the timer goes off, it adds one segment's worth of bytes to the congestion window to make it two maximum size segments and sends two segments. As each of these segments is acknowledged, the congestion window is increased by one maximum segment size.
- When the congestion window is n segments, if all n are acknowledged on time, the congestion window is increased by the byte count corresponding to n segments. In effect, each burst acknowledged doubles the congestion window.

TCP Congestion Control

- The congestion window keeps growing exponentially until either a timeout occurs or the receiver's window is reached.
- The idea is that if bursts of size, say, 1024, 2048, and 4096 bytes work fine but a burst of 8192 bytes gives a timeout, the congestion window should be set to 4096 to avoid congestion.
- As long as the congestion window remains at 4096, no bursts longer than that will be sent, no matter how much window space the receiver grants.
- This algorithm is called **slow start,** but it is not slow at all. It is exponential. All TCP implementations are required to support it.

TCP Congestion Control

- Now let us look at the Internet congestion control algorithm.
- It uses a third parameter, the threshold, initially 64 KB, in addition to the receiver and congestion windows.
- When a timeout occurs, the threshold is set to half of the current congestion window, and the congestion window is reset to one maximum segment.
- Slow start is then used to determine what the network can handle, except that exponential growth stops when the threshold is hit.
- From that point on, successful transmissions grow the congestion window linearly (by one maximum segment for each burst) instead of one per segment.
- In effect, this algorithm is guessing that it is probably acceptable to cut the congestion window in half, and then it gradually works its way up from there.

TCP Congestion Control

- As an illustration of how the congestion algorithm works, see below figure.
- The maximum segment size here is 1024 bytes. Initially, the congestion window was 64 KB, but a timeout occurred, so the threshold is set to 32 KB and the congestion window to 1 KB for transmission 0 (zero) here.
- The congestion window then grows exponentially until it hits the threshold (32 KB). Starting then, it grows linearly.

TCP Congestion Control

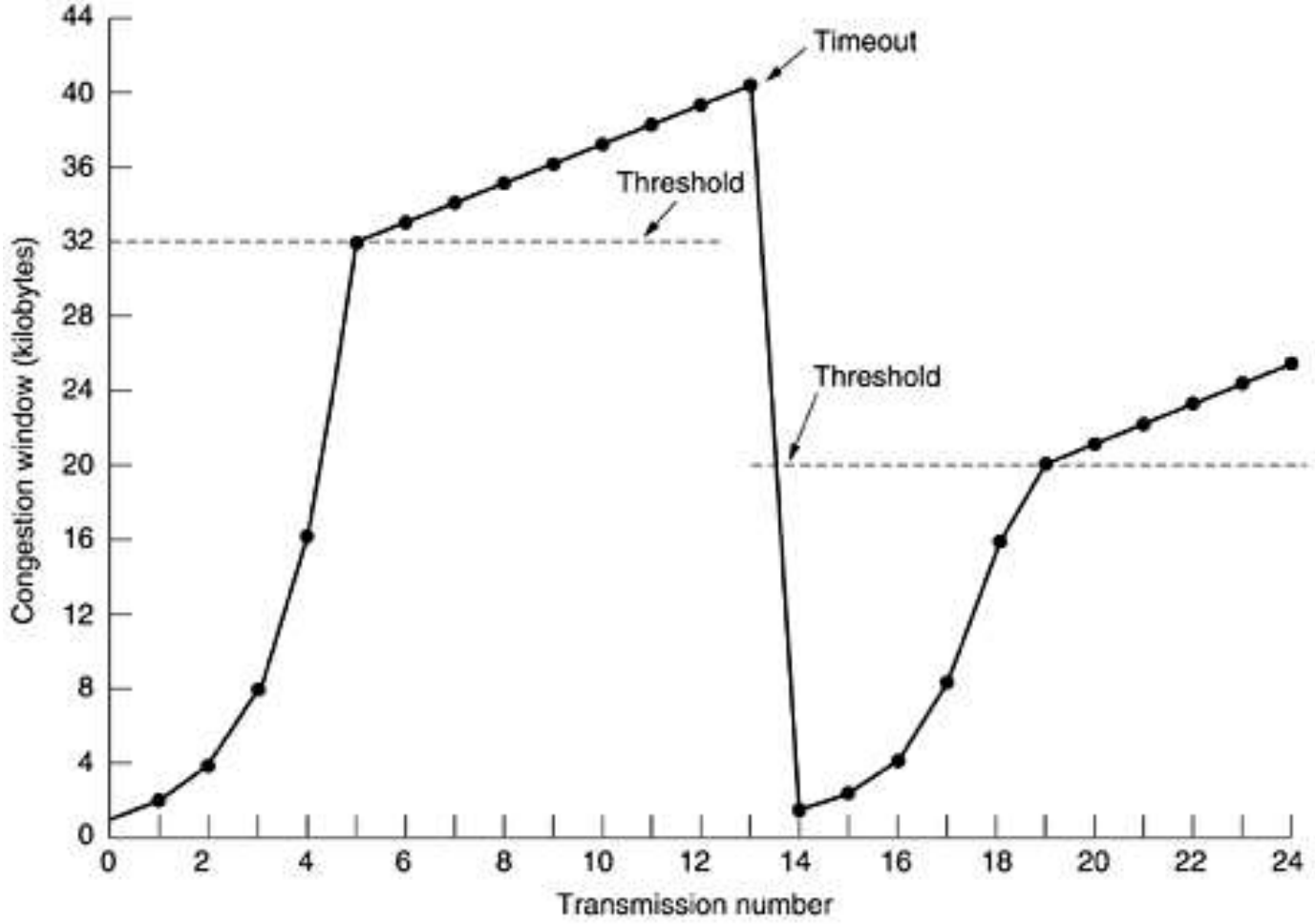


Figure: An example of the Internet congestion algorithm

TCP Congestion Control

- Transmission 13 is unlucky (it should have known) and a timeout occurs.
- The threshold is set to half the current window (by now 40 KB, so half is 20 KB), and slow start is initiated all over again.
- When the acknowledgements from transmission 14 start coming in, the first four each double the congestion window, but after that, growth becomes linear again.

TCP Congestion Control

- If no more timeouts occur, the congestion window will continue to grow up to the size of the receiver's window.
- At that point, it will stop growing and remain constant as long as there are no more timeouts and the receiver's window does not change size.
- As an aside, if an ICMP SOURCE QUENCH packet comes in and is passed to TCP, this event is treated the same way as a timeout.

TCP Timer Management

- TCP uses multiple timers (at least conceptually) to do its work.
- The most important of these is the **retransmission timer**.
- When a segment is sent, a retransmission timer is started.
- If the segment is acknowledged before the timer expires, the timer is stopped.
- If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer started again.

TCP Timer Management

- The algorithm that constantly adjusts the time-out interval, based on continuous measurements of n/w performance was proposed by JACOBSON and works as follows:
  - for each connection, TCP maintains a variable RTT, that is the best current estimate of the round trip time to the destination inn question.
  - When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.
  - If the acknowledgement gets back before the timer expires, TCP measures how long the measurements took say M It then updates RTT according to the formula:

TCP Timer Management

$$\mathbf{RTT} = \alpha\mathbf{RTT} + (1-\alpha)\,\mathbf{M}$$

Where α = *a smoothing factor that determines how much weight is given to the old value. Typically,* $\alpha = 7/8$

Retransmission timeout is calculated as

$$\mathbf{D} = \alpha\,\mathbf{D} + (1-\alpha)\,|\mathbf{RTT}-\mathbf{M}|$$

Where D = *another smoothed variable, Mean Deviation*

RTT = *expected acknowledgement value*

M = *observed acknowledgement value*

$$\mathbf{Timeout} = \mathbf{RTT}+(4*\mathbf{D})$$

TCP Timer Management

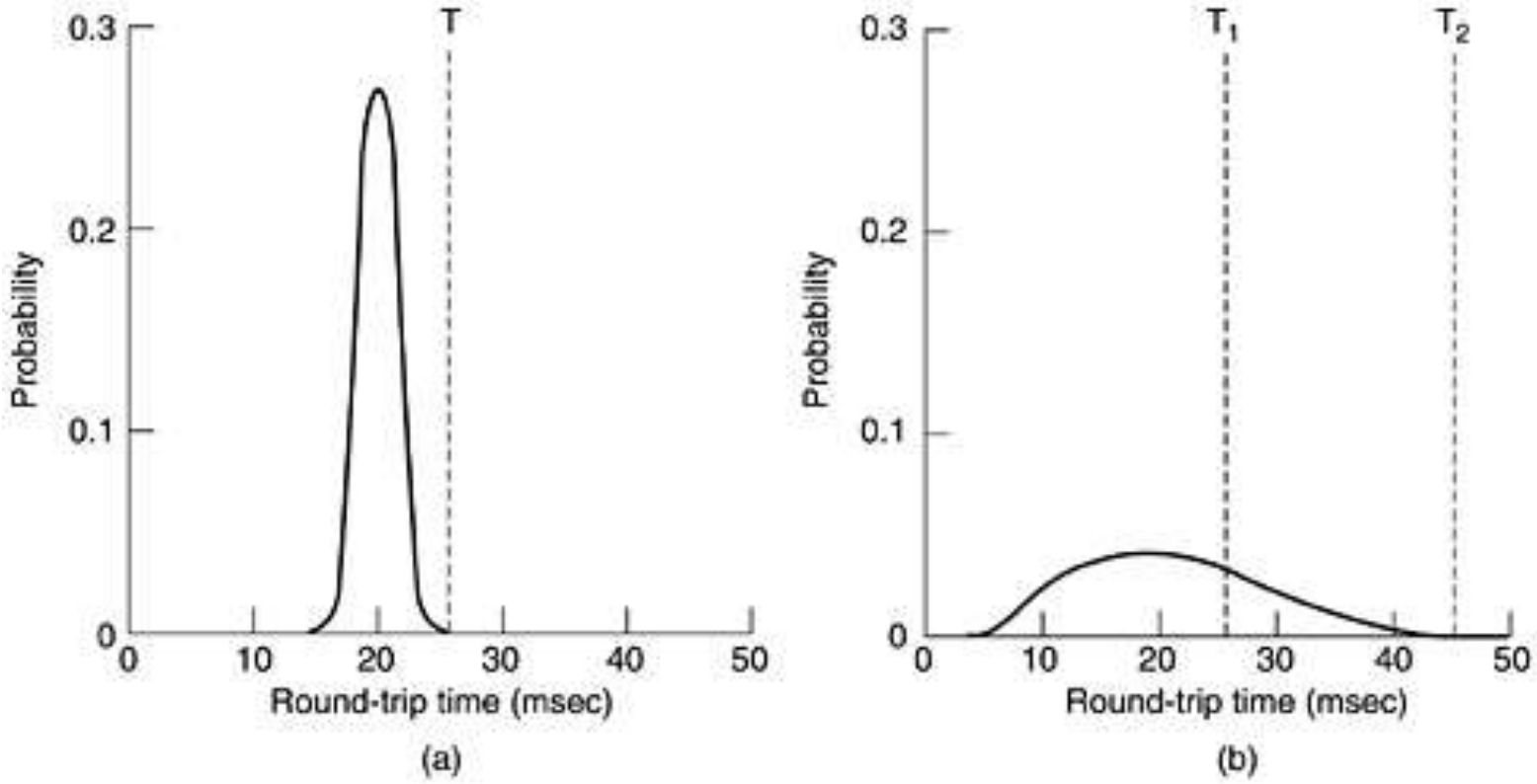


Figure: (a) Probability density of acknowledgement arrival times in the data link layer (b) Probability density of acknowledgement arrival times for TCP.

TCP Timer Management

- One problem that occurs with the dynamic estimation of RTT is what to do when a segment times out and is sent again.
- When the acknowledgement comes in, it is unclear whether the acknowledgement refers to the first transmission or a later one.
- Guessing wrong can seriously contaminate the estimate of RTT.

TCP Timer Management

- Phil Karn discovered this problem the hard way.
- He is an amateur radio enthusiast interested in transmitting TCP/IP packets by ham radio, a notoriously unreliable medium (on a good day, half the packets get through).
- He made a simple proposal: do not update RTT on any segments that have been retransmitted. Instead, the timeout is doubled on each failure until the segments get through the first time.
- This fix is called **Karn's algorithm.** Most TCP implementations use it.

TCP Timer Management

- A second timer is the **persistence timer.** It is designed to prevent the following deadlock.
- The receiver sends an acknowledgement with a window size of 0, telling the sender to wait. Later, the receiver updates the window, but the packet with the update is lost.
- Now both the sender and the receiver are waiting for each other to do something.
- When the persistence timer goes off, the sender transmits a probe to the receiver.
- The response to the probe gives the window size. If it is still zero, the persistence timer is set again and the cycle repeats. If it is nonzero, data can now be sent.

TCP Timer Management

- A third timer that some implementations use is the **keepalive timer.**
- When a connection has been idle for a long time, the keepalive timer may go off to cause one side to check whether the other side is still there.
- If it fails to respond, the connection is terminated.
- This feature is controversial because it adds overhead and may terminate an otherwise healthy connection due to a transient network partition.

TCP Timer Management

- The last timer used on each TCP connection is the one used in the **TIMED WAIT** state while closing.
- It runs for twice the maximum packet lifetime to make sure that when a connection is closed, all packets created by it have died off.

UNIT 4 PART 3

The Domain Name System(DNS): The DNS Name Space, Resource Records, Name Servers.

The Domain Name System (DNS)

- The Domain Name System (DNS) is a critical component of the internet that translates human-friendly domain names (like example.com) into IP addresses (like 192.0.2.1) that computers use to identify each other on the network.
- It functions like a phone book for the internet, enabling users to access websites and services without needing to memorize complex numerical addresses.

The Domain Name System (DNS)

- The essence of DNS is the invention of a hierarchical, domain-based naming scheme and a distributed database system for implementing this naming scheme.
- It is primarily used for mapping host names and e-mail destinations to IP addresses but can also be used for other purposes.

The Domain Name System (DNS)

- Very briefly, the way DNS is used is as follows.
- To map a name onto an IP address, an application program calls a library procedure called the **resolver**, passing it the name as a parameter.
- The resolver sends a UDP packet to a local DNS server, which then looks up the name and returns the IP address to the resolver, which then returns it to the caller.
- Armed with the IP address, the program can then establish a TCP connection with the destination or send it UDP packets.

The DNS Name Space

- The Internet is divided into over 200 top-level domains, where each domain covers many hosts.
- Each domain is partitioned into subdomains, and these are further partitioned, and so on.
- All these domains can be represented by a tree, as shown in below figure.
- The leaves of the tree represent domains that have no subdomains (but do contain machines, of course).
- A leaf domain may contain a single host, or it may represent a company and contain thousands of hosts.

The DNS Name Space

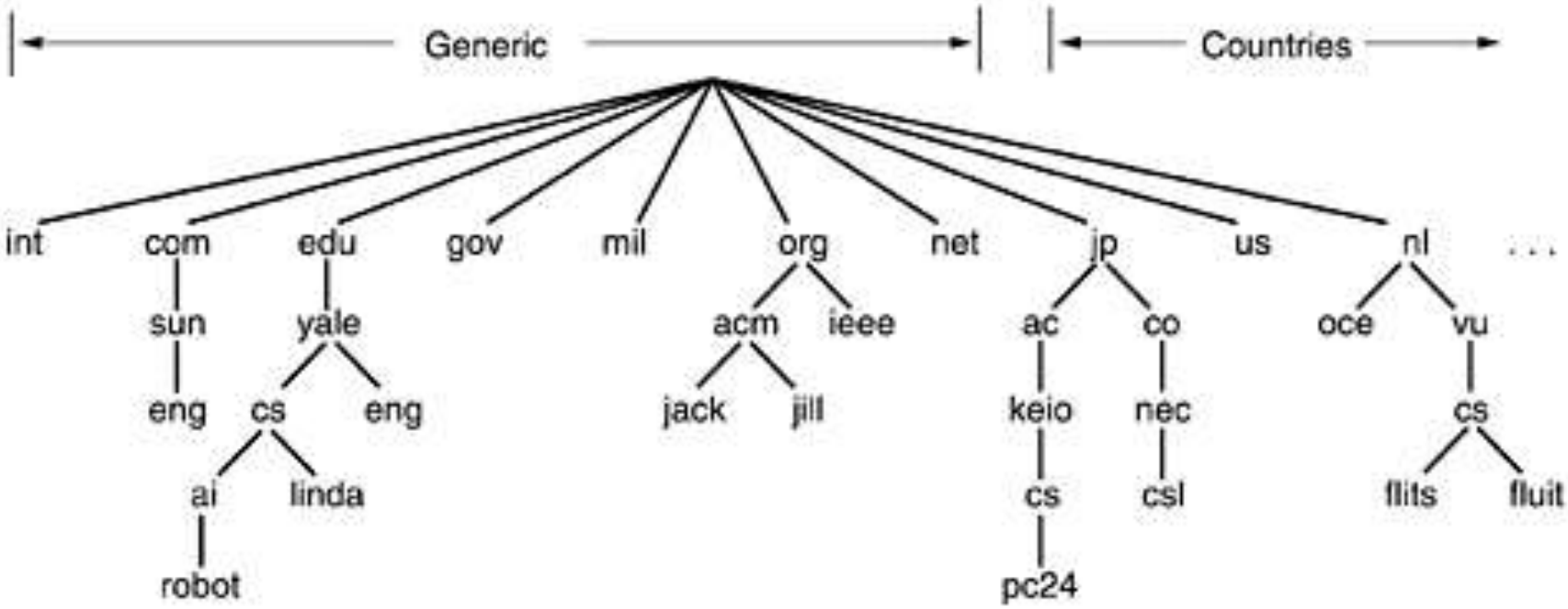


Figure: A portion of the Internet domain name space

The DNS Name Space

- The top-level domains come in two flavors: generic and countries.
- The original generic domains were com (commercial), edu (educational institutions), gov (the U.S. Federal Government), int (certain international organizations), mil (the U.S. armed forces), net (network providers), and org (nonprofit organizations).
- The country domains include one entry for every country.
- In November 2000, ICANN approved four new, general-purpose, top-level domains, namely, biz (businesses), info (information), name (people's names), and pro (professions, such as doctors and lawyers).
- In addition, three more specialized top-level domains were introduced at the request of certain industries.
- These are aero (aerospace industry), coop (co-operatives), and museum (museums).
- Other top-level domains will be added in the future.

The DNS Name Space

- In general, getting a second-level domain, such as name-of-company.com, is easy.
- It merely requires going to a registrar for the corresponding top-level domain (com in this case) to check if the desired name is available and not somebody else's trademark.
- If there are no problems, the requester pays a small annual fee and gets the name.
- By now, virtually every common (English) word has been taken in the com domain.
- Try household articles, animals, plants, body parts, etc. Nearly all are taken.

The DNS Name Space

- Each domain is named by the path upward from it to the (unnamed) root.
- The components are separated by periods (pronounced ''dot'').
- Thus, the engineering department at Sun Microsystems might be eng.sun.com., rather than a UNIX-style name such as /com/sun/eng.
- Notice that this hierarchical naming means that eng.sun.com. does not conflict with a potential use of eng in eng.yale.edu., which might be used by the Yale English department.

The DNS Name Space

- Domain names can be either absolute or relative.
- An absolute domain name always ends with a period (e.g., eng.sun.com.), whereas a relative one does not.
- Relative names have to be interpreted in some context to uniquely determine their true meaning.
- In both cases, a named domain refers to a specific node in the tree and all the nodes under it.
- Domain names are case insensitive, so edu, Edu, and EDU mean the same thing.
- Component names can be up to 63 characters long, and full path names must not exceed 255 characters.

The DNS Name Space

- In principle, domains can be inserted into the tree in two different ways.
- For example, cs.yale.edu could equally well be listed under the us country domain as cs.yale.ct.us.
- In practice, however, most organizations in the United States are under a generic domain, and most outside the United States are under the domain of their country.
- There is no rule against registering under two top-level domains, but few organizations except multinationals do it (e.g., sony.com and sony.nl).

The DNS Name Space

- Each domain controls how it allocates the domains under it.
- For example, Japan has domains ac.jp and co.jp that mirror edu and com.
- The Netherlands does not make this distinction and puts all organizations directly under nl.
- Thus, all three of the following are university computer science departments:

1. cs.yale.edu (Yale University, in the United States)
2. cs.vu.nl (Vrije Universiteit, in The Netherlands)
3. cs.keio.ac.jp (Keio University, in Japan)

The DNS Name Space

- To create a new domain, permission is required of the domain in which it will be included.
- For example, if a VLSI group is started at Yale and wants to be known as vlsi.cs.yale.edu, it has to get permission from whoever manages cs.yale.edu.
- Similarly, if a new university is chartered, say, the University of Northern South Dakota, it must ask the manager of the edu domain to assign it unsd.edu.
- In this way, name conflicts are avoided and each domain can keep track of all its subdomains.
- Once a new domain has been created and registered, it can create subdomains, such as cs.unsd.edu, without getting permission from anybody higher up the tree.

The DNS Name Space

- Naming follows organizational boundaries, not physical networks.
- For example, if the computer science and electrical engineering departments are located in the same building and share the same LAN, they can nevertheless have distinct domains.
- Similarly, even if computer science is split over Babbage Hall and Turing Hall, the hosts in both buildings will normally belong to the same domain.

Resource Records

- Every domain, whether it is a single host or a top-level domain, can have a set of resource records associated with it.
- For a single host, the most common resource record is just its IP address, but many other kinds of resource records also exist.
- When a resolver gives a domain name to DNS, what it gets back are the resource records associated with that name.
- Thus, the primary function of DNS is to map domain names onto resource records.

Resource Records

- A resource record is a five-tuple.
- Although they are encoded in binary for efficiency, in most expositions, resource records are presented as ASCII text, one line per resource record.
- The format we will use is as follows:
  - Domain_name
  - Time_to_live
  - Class
  - Type
  - Value

Resource Records

- The **Domain_name** tells the domain to which this record applies.
- Normally, many records exist for each domain and each copy of the database holds information about multiple domains.
- This field is thus the primary search key used to satisfy queries.
- The order of the records in the database is not significant.

Resource Records

- The **Time_to_live** field gives an indication of how stable the record is.
- Information that is highly stable is assigned a large value, such as 86400 (the number of seconds in 1 day).
- Information that is highly volatile is assigned a small value, such as 60 (1 minute).

Resource Records

- The third field of every resource record is the **Class**.
- For Internet information, it is always IN.
- For non-Internet information, other codes can be used, but in practice, these are rarely seen.

Resource Records

- The **Type** field tells what kind of record this is. The most important types are listed in below figure.

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

Figure: The principal DNS resource record types for IPv4

Resource Records

- An **SOA** record provides the name of the primary source of information about the name server's zone (described below), the e-mail address of its administrator, a unique serial number, and various flags and timeouts.

Resource Records

- The most important record type is the **A** (Address) record.
- It holds a 32-bit IP address for some host.
- Every Internet host must have at least one IP address so that other machines can communicate with it.
- Some hosts have two or more network connections, in which case they will have one type A resource record per network connection (and thus per IP address).
- DNS can be configured to cycle through these, returning the first record on the first request, the second record on the second request, and so on.

Resource Records

- The next most important record type is the **MX** record.
- It specifies the name of the host prepared to accept e-mail for the specified domain.
- It is used because not every machine is prepared to accept e-mail.
- If someone wants to send e-mail to, for example, bill@microsoft.com, the sending host needs to find a mail server at microsoft.com that is willing to accept e-mail.
- The MX record can provide this information.

Resource Records

- The **NS** records specify name servers.
- For example, every DNS database normally has an NS record for each of the top-level domains, so, for example, e-mail can be sent to distant parts of the naming tree.

Resource Records

- **CNAME** records allow aliases to be created. For example, a person familiar with Internet naming in general and wanting to send a message to someone whose login name is paul in the computer science department at M.I.T. might guess that paul@cs.mit.edu will work.
- Actually, this address will not work, because the domain for M.I.T.'s computer science department is lcs.mit.edu.
- However, as a service to people who do not know this, M.I.T. could create a CNAME entry to point people and programs in the right direction.
- An entry like this one might do the job:
  - **cs.mit.edu 86400 IN CNAME lcs.mit.edu**

Resource Records

- Like CNAME, **PTR** points to another name.
- However, unlike CNAME, which is really just a macro definition, PTR is a regular DNS datatype whose interpretation depends on the context in which it is found.
- In practice, it is nearly always used to associate a name with an IP address to allow lookups of the IP address and return the name of the corresponding machine.
- These are called **reverse lookups.**

Resource Records

- **HINFO** records allow people to find out what kind of machine and operating system a domain corresponds to.
- Finally, **TXT** records allow domains to identify themselves in arbitrary ways.
- Both of these record types are for user convenience. Neither is required, so programs cannot count on getting them (and probably cannot deal with them if they do get them).

Resource Records

- Finally, we have the **Value** field.
- This field can be a number, a domain name, or an ASCII string.
- The semantics depend on the record type.
- A short description of the Value fields for each of the principal record types is given in above table.

Name Servers

- In theory at least, a single name server could contain the entire DNS database and respond to all queries about it.
- In practice, this server would be so overloaded as to be useless.
- Furthermore, if it ever went down, the entire Internet would be crippled.

Name Servers

- To avoid the problems associated with having only a single source of information, the DNS name space is divided into nonoverlapping zones.
- One possible way to divide the name space of (Figure: DNS) shown in below figure.
- Each zone contains some part of the tree and also contains name servers holding the information about that zone.
- Normally, a zone will have one primary name server, which gets its information from a file on its disk, and one or more secondary name servers, which get their information from the primary name server.
- To improve reliability, some servers for a zone can be located outside the zone.

Name Servers

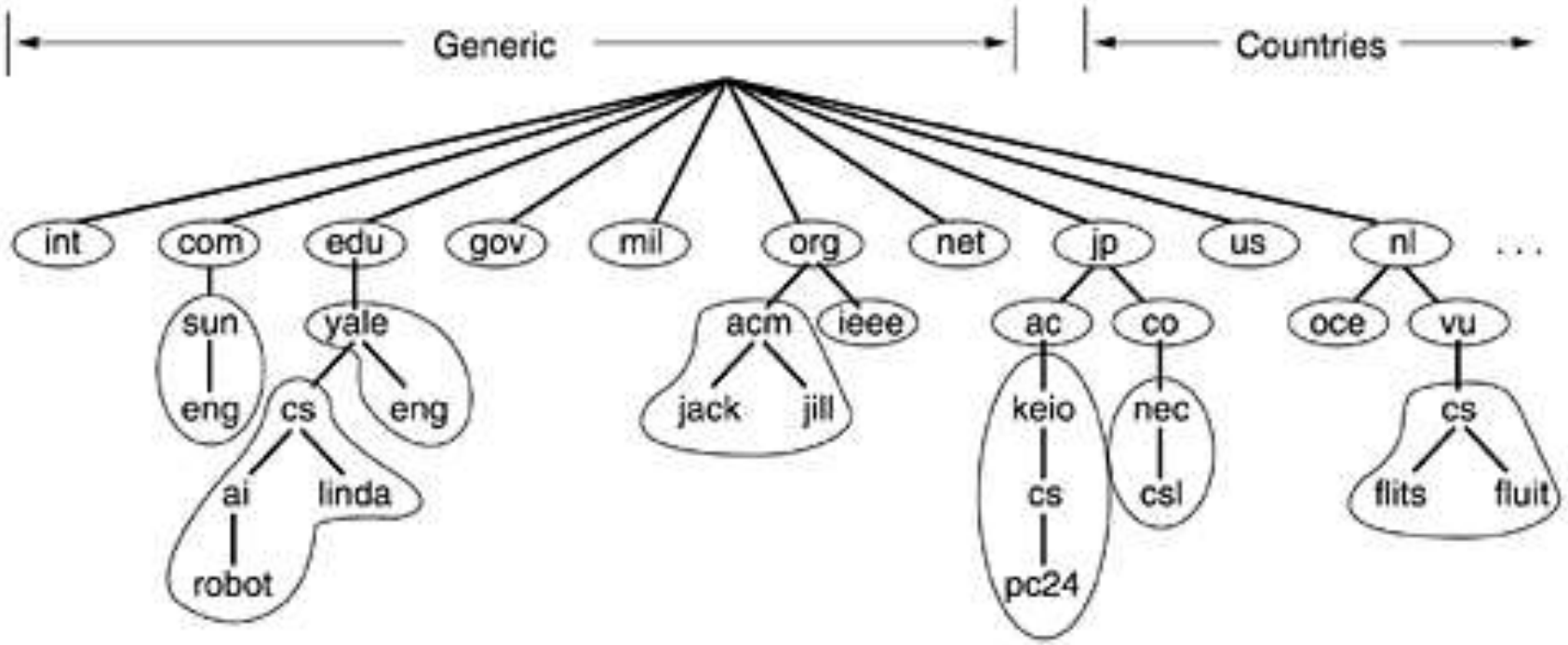


Figure: Part of the DNS name space showing the division into zones.

Name Servers

- Where the zone boundaries are placed within a zone is up to that zone's administrator.
- This decision is made in large part based on how many name servers are desired, and where.
- For example, in above figure, Yale has a server for yale.edu that handles eng.yale.edu but not cs.yale.edu, which is a separate zone with its own name servers.
- Such a decision might be made when a department such as English does not wish to run its own name server, but a department such as computer science does.
- Consequently, cs.yale.edu is a separate zone but eng.yale.edu is not.

Name Servers

- When a resolver has a query about a domain name, it passes the query to one of the local name servers.
- If the domain being sought falls under the jurisdiction of the name server, such as ai.cs.yale.edu falling under cs.yale.edu, it returns the authoritative resource records.
- An authoritative record is one that comes from the authority that manages the record and is thus always correct.
- Authoritative records are in contrast to cached records, which may be out of date.

Name Servers

- If, however, the domain is remote and no information about the requested domain is available locally, the name server sends a query message to the top-level name server for the domain requested.
- To make this process clearer, consider the example of below figure.

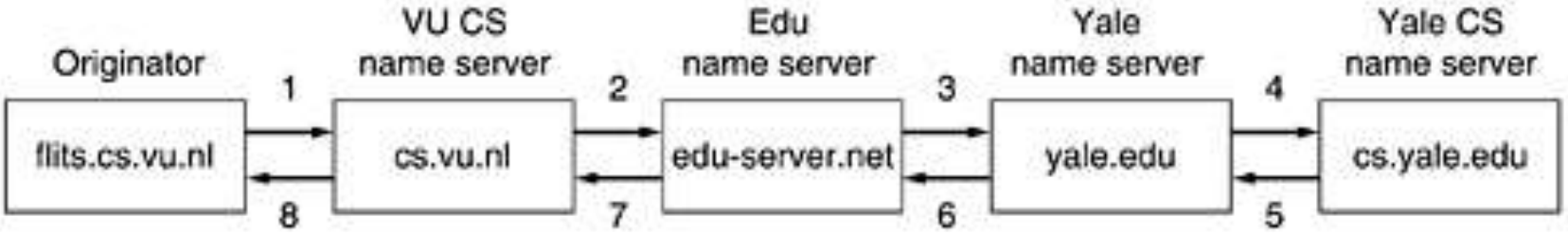


Figure: How a resolver looks up a remote name in eight steps.

Name Servers

- Here, a resolver on flits.cs.vu.nl wants to know the IP address of the host linda.cs.yale.edu.
- In step 1, it sends a query to the local name server, cs.vu.nl. This query contains the domain name sought, the type (A) and the class (IN).
- Let us suppose the local name server has never had a query for this domain before and knows nothing about it.
- It may ask a few other nearby name servers, but if none of them know, it sends a UDP packet to the server for edu given in its database (see above figure), edu-server.net.

Name Servers

- It is unlikely that this server knows the address of linda.cs.yale.edu, and probably does not know cs.yale.edu either, but it must know all of its own children, so it forwards the request to the name server for yale.edu (step 3).
- In turn, this one forwards the request to cs.yale.edu (step 4), which must have the authoritative resource records.
- Since each request is from a client to a server, the resource record requested works its way back in steps 5 through 8.