[SW95]. Also, the third volume of Comer and Stevens' TCP/IP series of books describes how to write client/server applications on top of TCP and UDP, using the Posix socket interface [CS00], the Windows socket interface [CS97], and the BSD socket interface [CS96]. SCTP, a reliable transport protocol that stakes out a different point in the design space than TCP, is described in a helpful overview [OY02] and specified in [Ste07].

Several papers evaluate the performance of different transport protocols at a very detailed level. For example, the article by Clark et al. [CJRS89] measures the processing overheads of TCP, a paper by Mosberger et al. [MPBO96] explores the limitations of protocol processing overheads, and Thekkath and Levy [TL93] and Schroeder and Burrows [SB89] examine RPC's performance in great detail.

The original TCP timeout calculation was described in the TCP specification (see above), while the Karn/Partridge algorithm was described in [KP91] and the Jacobson/Karels algorithm was proposed in [Jac88]. The TCP extensions are defined by Jacobson et al. [JBB92], while O'Malley and Peterson [OP91] argue that extending TCP in this way is not the right approach to solving the problem.

Several distributed operating systems have defined their own RPC protocols. Notable examples include the V system, described by Cheriton and Zwaenepoel [CZ85]; Sprite, described by Ousterhout et al. [OCD+88]; and Amoeba, described by Mullender [Mul90].

RTP is described in RFC 3550 [SCFJ03], and there are numerous other RFCs (such as RFC 3551 [SC03]) that describe the profiles of various applications that use RTP. McCanne and Jacobson [MJ95] describe vic, one of the early video applications to use RTP.

Finally, the following live reference provides lots of information related to transport and other protocols:

- http://www.iana.org/protocols/: Information about all the options, constants, port numbers, etc., that have been defined for various Internet protocols. You can find here, among other things, the lists of TCP header flags, TCP options, and well-known port numbers for protocols that run over TCP and UDP.

## EXERCISES

1. If a UDP datagram is sent from host A, port P to host B, port Q, but at host B there is no process listening to port Q, then B is to

send back an ICMP Port Unreachable message to A. Like all ICMP messages, this is addressed to A as a whole, not to port P on A.

**(a)** Give an example of when an application might want to receive such ICMP messages.

**(b)** Find out what an application has to do, on the operating system of your choice, to receive such messages.

**(c)** Why might it not be a good idea to send such messages directly back to the originating port P on A?

**2.** Consider a simple UDP-based protocol for requesting files (based somewhat loosely on the Trivial File Transport Protocol, or TFTP). The client sends an initial file request, and the server answers (if the file can be sent) with the first data packet. Client and server then continue with a stop-and-wait transmission mechanism.

**(a)** Describe a scenario by which a client might request one file but get another; you may allow the client application to exit abruptly and be restarted with the same port.

**(b)** Propose a change in the protocol that will make this situation much less likely.

**3.** Design a simple UDP-based protocol for retrieving files from a server. No authentication is to be provided. Stop-and-wait transmission of the data may be used. Your protocol should address the following issues:

**(a)** Duplication of the first packet should not duplicate the "connection."

**(b)** Loss of the final ACK should not necessarily leave the server in doubt as to whether the transfer succeeded.

**(c)** A late-arriving packet from a past connection shouldn't be interpretable as part of a current connection.

**4.** This chapter explains three sequences of state transitions during TCP connection teardown. There is a fourth possible sequence, which traverses an additional arc (not shown in Figure 5.7) from FIN_WAIT_1 to TIME_WAIT and labelled FIN + ACK/ACK. Explain the circumstances that result in this fourth teardown sequence.

**5.** When closing a TCP connection, why is the two-segment-lifetime timeout not necessary on the transition from LAST_ACK to CLOSED?

6. A sender on a TCP connection that receives a 0 advertised window periodically probes the receiver to discover when the window becomes nonzero. Why would the receiver need an extra timer if it were responsible for reporting that its advertised window had become nonzero (i.e., if the sender did not probe)?

7. Read the man page (or Windows equivalent) for the Unix/Windows utility netstat. Use netstat to see the state of the local TCP connections. Find out how long closing connections spend in TIME_WAIT.

8. The sequence number field in the TCP header is 32 bits long, which is big enough to cover over 4 billion bytes of data. Even if this many bytes were never transferred over a single connection, why might the sequence number still wrap around from $2^{32} - 1$ to 0?

9. You are hired to design a reliable byte-stream protocol that uses a sliding window (like TCP). This protocol will run over a 1-Gbps network. The RTT of the network is 100 ms, and the maximum segment lifetime is 30 seconds.
   (a) How many bits would you include in the AdvertisedWindow and SequenceNum fields of your protocol header?
   (b) How would you determine the numbers given above, and which values might be less certain?

10. You are hired to design a reliable byte-stream protocol that uses a sliding window (like TCP). This protocol will run over a 1-Gbps network. The RTT of the network is 140 ms, and the maximum segment lifetime is 60 seconds. How many bits would you include in the AdvertisedWindow and SequenceNum fields of your protocol header?

11. Suppose a host wants to establish the reliability of a link by sending packets and measuring the percentage that is received; routers, for example, do this. Explain the difficulty doing this over a TCP connection.

12. Suppose TCP operates over a 1-Gbps link.
   (a) Assuming TCP could utilize the full bandwidth continuously, how long would it take the sequence numbers to wrap around completely?

**(b)** Suppose an added 32-bit timestamp field increments 1000 times during the wraparound time you found above. How long would it take for the timestamp to wrap around?

✓ **13.** Suppose TCP operates over a 40-Gbps STS-768 link.

**(a)** Assuming TCP could utilize the full bandwidth continuously, how long would it take the sequence numbers to wrap around completely?

**(b)** Suppose an added 32-bit timestamp field which increments 1000 times during the wraparound time you found above. How long would it take for the timestamp to wrap around?

**14.** If host A receives two SYN packets from the same port from remote host B, the second may be either a retransmission of the original or, if B has crashed and rebooted, an entirely new connection request.

**(a)** Describe the difference as seen by host A between these two cases.

**(b)** Give an algorithmic description of what the TCP layer needs to do upon receiving a SYN packet. Consider the duplicate/new cases above and the possibility that nothing is listening to the destination port.

**15.** Suppose $x$ and $y$ are two TCP sequence numbers. Write a function to determine whether $x$ comes before $y$ (in the notation of *Request for Comments* 793, "$x =< y$") or after $y$; your solution should work even when sequence numbers wrap around.

**16.** Suppose an idle TCP connection exists between sockets A and B. A third party has eavesdropped and knows the current sequence number at both ends.

**(a)** Suppose the third party sends A a forged packet ostensibly from B and with 100 bytes of new data. What happens? (Hint: Look up in *Request for Comments* 793 what TCP does when it receives an ACK that is not an "acceptable ACK.")

**(b)** Suppose the third party sends each end such a forged 100-byte data packet ostensibly from the other end. What happens now? What would happen if A later sent 200 bytes of data to B?

**17.** Suppose party A connects to the Internet via a wireless network using DHCP to assign IP addresses. A opens several Telnet

connections (using TCP) and is then disconnected from the wireless network. Party B then connects and is assigned the same IP address that A had had. Assuming B were able to guess to what host(s) A had been connected, describe a sequence of probes that could enable B to obtain sufficient state information to continue with A's connections.

**18.** Diagnostic programs are commonly available that record the first 100 bytes, say, of every TCP connection to a certain ⟨host, port⟩. Outline what must be done with each received TCP packet, P, in order to determine if it contains data that belongs to the first 100 bytes of a connection to host HOST, port PORT. Assume the IP header is P.IPHEAD, the TCP header is P.TCPHEAD, and header fields are as named in Figures 3.16 and 5.4. (Hint: To get initial sequence numbers (ISNs) you will have to examine every packet with the SYN bit set. Ignore the fact that sequence numbers will eventually be reused.)

**19.** If a packet arrives at host A with B's source address, it could just as easily have been forged by any third host C. If, however, A accepts a TCP connection from B, then during the three-way handshake A sent $ISN_A$ to B's address and received an acknowledgment of it. If C is not located so as to be able to eavesdrop on $ISN_A$, then it might seem that C could not have forged B's response.

However, the algorithm for choosing $ISN_A$ does give other unrelated hosts a fair chance of guessing it. Specifically, A selects $ISN_A$ based on a clock value at the time of connection. *Request for Comments* 793 specifies that this clock value be incremented every 4 μs; common Berkeley implementations once simplified this to incrementing by 250,000 (or 256,000) once per second.

**(a)** Given this simplified increment-once-per-second implementation, explain how an arbitrary host C could masquerade as B in at least the opening of a TCP connection. You may assume that B does not respond to SYN + ACK packets A is tricked into sending to it.

**(b)** Assuming real RTTs can be estimated to within 40 ms, about how many tries would you expect it to take to implement the strategy of part (a) with the unsimplified "increment every 4 μs" TCP implementation?

20. The Nagle algorithm, built into most TCP implementations, requires the sender to hold a partial segment's worth of data (even if PUSHed) until either a full segment accumulates or the most recent outstanding ACK arrives.
    (a) Suppose the letters abcdefghi are sent, one per second, over a TCP connection with an RTT of 4.1 seconds. Draw a timeline indicating when each packet is sent and what it contains.
    (b) If the above were typed over a full-duplex Telnet connection, what would the user see?
    (c) Suppose that mouse position changes are being sent over the connection. Assuming that multiple position changes are sent each RTT, how would a user perceive the mouse motion with and without the Nagle algorithm?

21. Suppose a client C repeatedly connects via TCP to a given port on a server S, and that each time it is C that initiates the close.
    (a) How many TCP connections a second can C make here before it ties up all its available ports in TIME_WAIT state? Assume client ephemeral ports are in the range of 1024 to 5119, and that TIME_WAIT lasts 60 seconds.
    (b) Berkeley-derived TCP implementations typically allow a socket in TIME_WAIT state to be reopened before TIME_WAIT expires, if the highest sequence number used by the old incarnation of the connection is less than the ISN used by the new incarnation. This solves the problem of old data being accepted as new; however, TIME_WAIT also serves the purpose of handling late final FINs. What would such an implementation have to do to address this and still achieve strict compliance with the TCP requirement that a FIN sent anytime before or during a connection's TIME_WAIT receive the same response?

22. Explain why TIME_WAIT is a somewhat more serious problem if the server initiates the close than if the client does. Describe a situation in which this might reasonably happen.

23. What is the justification for the exponential increase in timeout value proposed by Karn and Partridge? Why, specifically, might a linear (or slower) increase be less desirable?

☆ **24.** The Jacobson/Karels algorithm sets TimeOut to be 4 mean deviations above the mean. Assume that individual packet round-trip times follow a statistical normal distribution, for which 4 mean deviations are $\pi$ standard deviations. Using statistical tables, for example, what is the probability that a packet will take more than TimeOut time to arrive?

**25.** Suppose a TCP connection, with window size 1, loses every other packet. Those that do arrive have RTT $= 1$ second. What happens? What happens to TimeOut? Do this for two cases:

**(a)** After a packet is eventually received, we pick up where we left off, resuming with EstimatedRTT initialized to its pre-timeout value, and TimeOut double that.

**(b)** After a packet is eventually received, we resume with TimeOut initialized to the last exponentially backed-off value used for the timeout interval.

In the following four exercises, the calculations involved are straightforward with a spreadsheet.

**26.** Suppose, in TCP's adaptive retransmission mechanism, that EstimatedRTT is 4.0 seconds at some point and subsequent measured RTT's all are 1.0 second. How long does it take before the TimeOut value, as calculated by the Jacobson/Karels algorithm, falls below 4.0 seconds? Assume a plausible initial value of Deviation; how sensitive is your answer to this choice? Use $\delta = 1/8$.

✓ **27.** Suppose, in TCP's adaptive retransmission mechanism, that EstimatedRTT is 90 at some point and subsequent measured RTTs are all 200. How long does it take before the TimeOut value, as calculated by the Jacobson/Karels algorithm, falls below 300? Assume initial Deviation value of 25; use $\delta = 1/8$.

**28.** Suppose TCP's measured RTT is 1.0 second except that every $N$th RTT is 4.0 seconds. What is the largest $N$, approximately, that doesn't result in timeouts in the steady state (i.e., for which the Jacobson/Karels TimeOut remains greater than 4.0 seconds)? Use $\delta = 1/8$.

**29.** Suppose that TCP is measuring RTTs of 1.0 second, with a mean deviation of 0.1 second. Suddenly the RTT jumps to 5.0 seconds,

with no deviation. Compare the behaviors of the original and Jacobson/Karels algorithms for computing TimeOut. Specifically, how many timeouts are encountered with each algorithm? What is the largest TimeOut calculated? Use $\delta = 1/8$.

**30.** Suppose that, when a TCP segment is sent more than once, we take SampleRTT to be the time between the original transmission and the ACK, as in Figure 5.10(a). Show that if a connection with a 1-packet window loses every other packet (i.e., each packet is transmitted twice), then EstimatedRTT increases to infinity. Assume TimeOut = EstimatedRTT; both algorithms presented in the text always set TimeOut even larger. (Hint: EstimatedRTT = EstimatedRTT + $\beta \times$ (SampleRTT − EstimatedRTT).)

**31.** Suppose that, when a TCP segment is sent more than once, we take SampleRTT to be the time between the most recent transmission and the ACK, as in Figure 5.10(b). Assume, for definiteness, that TimeOut = $2 \times$ EstimatedRTT. Sketch a scenario in which no packets are lost but EstimatedRTT converges to a third of the true RTT, and give a diagram illustrating the final steady state. (Hint: Begin with a sudden jump in the true RTT to just over the established TimeOut.)

**32.** Consult *Request for Comments* 793 to find out how TCP is supposed to respond if a FIN or an RST arrives with a sequence number other than NextByteExpected. Consider both when the sequence number is within the receive window and when it is not.

**33.** One of the purposes of TIME_WAIT is to handle the case of a data packet from a first incarnation of a connection arriving very late and being accepted as data for the second incarnation.

**(a)** Explain why, for this to happen (in the absence of TIME_WAIT), the hosts involved would have to exchange several packets in sequence *after* the delayed packet was sent but before it was delivered.

**(b)** Propose a network scenario that might account for such a late delivery.

**34.** Propose an extension to TCP by which one end of a connection can hand off its end to a third host; that is, if A were connected to B, and A handed off its connection to C, then afterwards C would be connected to B and A would not. Specify the new states and

transitions needed in the TCP state-transition diagram and any new packet types involved. You may assume all parties will understand this new option. What state should A go into immediately after the handoff?

**35.** TCP's simultaneous open feature is seldom used.

**(a)** Propose a change to TCP in which this is disallowed. Indicate what changes would be made in the state diagram (and if necessary in the undiagrammed event responses).

**(b)** Could TCP reasonably disallow simultaneous close?

**(c)** Propose a change to TCP in which simultaneous SYNs exchanged by two hosts lead to two separate connections. Indicate what state diagram changes this entails and what header changes become necessary. Note that this now means that more than one connection can exist over a given pair of ⟨host, port⟩s. (You might also look up the first "Discussion" item on page 87 of *Request for Comments* 1122.)

**36.** TCP is a very symmetric protocol, but the client/server model is not. Consider an asymmetric TCP-like protocol in which only the server side is assigned a port number visible to the application layers. Client-side sockets would simply be abstractions that can be connected to server ports.

**(a)** Propose header data and connection semantics to support this. What will you use to replace the client port number?

**(b)** What form does TIME_WAIT now take? How would this be seen through the programming interface? Assume that a client socket could now be reconnected arbitrarily many times to a given server port, resources permitting.

**(c)** Look up the rsh/rlogin protocol. How would the above break this?

**37.** The following exercise is concerned with the TCP state FIN_WAIT_2 (see Figure 5.7).

**(a)** Describe how a client might leave a suitable server in state FIN_WAIT_2 indefinitely. What feature of the server's protocol is necessary here for this scenario?

**(b)** Try this with some appropriate existing server. Either write a stub client or use an existing Telnet client capable of connecting to an arbitrary port. Use the netstat utility to verify that the server is in FIN_WAIT_2 state.

**38.** *Request for Comments* 1122 states (of TCP):

> A host MAY implement a "half-duplex" TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection. If such a host issues a CLOSE call while received data is still pending in TCP, or if new data is received after CLOSE is called, its TCP SHOULD send an RST to show that data was lost.

Sketch a scenario involving the above in which data sent by *(not to!)* the closing host is lost. You may assume that the remote host, upon receiving an RST, discards all received data still unread in buffers.

**39.** When TCP sends a $\langle \text{SYN}, \text{SequenceNum} = x \rangle$ or $\langle \text{FIN}, \text{SequenceNum} = x \rangle$, the consequent ACK has $\text{Acknowledgment} = x + 1$; that is, SYNs and FINs each take up one unit in sequence number space. Is this necessary? If so, give an example of an ambiguity that would arise if the corresponding Acknowledgment were $x$ instead of $x + 1$; if not, explain why.

**40.** Find out the generic format for TCP header options from *Request for Comments* 793.

**(a)** Outline a strategy that would expand the space available for options beyond the current limit of 44 bytes.

**(b)** Suggest an extension to TCP allowing the sender of an option a way of specifying what the receiver should do if the option is not understood. List several such receiver actions that might be useful, and try to give an example application of each.

**41.** The TCP header does not have a boot ID field. Why isn't there a problem with one end of a TCP connection crashing and rebooting, then sending a message with an ID it had previously used?

**42.** Suppose we were to implement remote file system mounting using an unreliable RPC protocol that offers zero-or-more semantics. If a message reply is received, this improves to at-least-once semantics. We define read(n) to return the specified $n$th block, rather than the next block in sequence; this way, reading once is the same as reading twice and at-least-once semantics is thus the same as exactly once.

**(a)** For what other file system operations is there no difference between at-least-once and exactly once semantics? Consider open, create, write, seek, opendir, readdir, mkdir, delete (*aka* unlink), and rmdir.

**(b)** For the remaining operations, which can have their semantics altered to achieve equivalence of at-least-once and exactly once? What file system operations are irreconcilable with at-least-once semantics?

**(c)** Suppose the semantics of the rmdir system call are now that the given directory is removed if it exists, and nothing is done otherwise. How could you write a program to delete directories that distinguishes between these two cases?

**43.** The RPC-based NFS remote file system is sometimes considered to have slower than expected write performance. In NFS, a server's RPC reply to a client write request means that the data is physically written to the server's disk, not just placed in a queue.

**(a)** Explain the bottleneck we might expect, even with infinite bandwidth, if the client sends all its write requests through a single logical channel, and explain why using a pool of channels could help. Hint: You will need to know a little about disk controllers.

**(b)** Suppose the server's reply means only that the data has been placed in the disk queue. Explain how this could lead to data loss that wouldn't occur with a local disk. Note that a system crash immediately after data was enqueued doesn't count, because that would cause data loss on a local disk as well.

**(c)** An alternative would be for the server to respond immediately to acknowledge the write request and to send its own separate request later to confirm the physical write. Propose different RPC semantics to achieve the same effect, but with a single logical request and reply.

**44.** Consider a client and server using an RPC mechanism that includes a channel abstraction and boot IDs.

**(a)** Give a scenario involving server reboot in which an RPC request is sent twice by the client and is executed twice by the server, with only a single ACK.

**(b)** How might the client become aware this had happened? Would the client be sure it had happened?

45. Suppose an RPC request is of the form: "Increment the value of field X of disk block N by 10%." Specify a mechanism to be used by the executing server to guarantee that an arriving request is executed exactly once, even if the server crashes while in the middle of the operation. Assume that individual disk block writes are either complete or else the block is unchanged. You may also assume that some designated "undo log" blocks are available. Your mechanism should include how the RPC server is to behave at restart.

46. Consider a SunRPC client sending a request to a server.
    (a) Under what circumstances can the client be sure its request has executed exactly once?
    (b) Suppose we wished to add at-most-once semantics to SunRPC. What changes would have to be made? Explain why adding one or more fields to the existing headers would not be sufficient.

47. Suppose TCP were to be used as the underlying transport in an RPC protocol; each TCP connection is to carry a sequential stream of requests and replies. What analog, if any, would TCP have for:
    (a) Channel ID
    (b) Message ID
    (c) Boot ID
    (d) A message type for requests
    (e) A message type for replies
    (f) A message type for acknowledgments
    (g) A message type for are-you-alive? messages
    Which of these would the overlying RPC protocol have to provide? Would some analog of implicit acknowledgments exist?

48. Write a test program that uses the socket interface to send messages between a pair of Unix workstations connected by some LAN (e.g., Ethernet, 802.11). Use this test program to perform the following experiments:
    (a) Measure the round-trip latency of TCP and UDP for different message sizes (e.g., 1 byte, 100 bytes, 200 bytes, . . . , 1000 bytes).

(b) Measure the throughput of TCP and UDP for 1-KB, 2-KB, 3-KB, . . ., 32-KB messages. Plot the measured throughput as a function of message size.

(c) Measure the throughput of TCP by sending 1 MB of data from one host to another. Do this in a loop that sends a message of some size—for example, 1024 iterations of a loop that sends 1-KB messages. Repeat the experiment with different message sizes and plot the results.

**49.** Try to find situations where an RTP application might reasonably do the following:

- Send multiple packets at essentially the same time that need different timestamps.
- Send packets at different times that need the same timestamp.

Argue, in consequence, that RTP timestamps must, in at least some cases, be provided (at least indirectly) by the application. (Hint: Think of cases where the sending rate and playback rate might not match.)

**50.** Having the RTP timestamp clock count time in units of one frame time or one voice sample time would be the minimum resolution to ensure accurate playback, but the time unit is usually considerably smaller; what is the purpose of this?

**51.** Suppose we want returning RTCP reports from receivers to amount to no more than 5% of the outgoing primary RTP stream. If each report is 84 bytes, the RTP traffic is 320 kbps, and there are 1000 recipients, how often do individual receivers get to report? What if there are 10,000 recipients?

**52.** RFC 3550 specifies that the time interval between receiver RTCP reports include a randomization factor to avoid having all the receivers sending at the same time. If all the receivers sent in the same 5% subinterval of their reply time interval, the arriving upstream RTCP traffic would rival the downstream RTP traffic.

(a) Video receivers might reasonably wait to send their reports until the higher-priority task of processing and displaying one frame is completed; this might mean their RTCP transmissions were synchronized on frame boundaries. Is this likely to be a serious concern?

**(b)** With 10 receivers, what is the probability of their all sending in one particular 5% subinterval?

**(c)** With 10 receivers, what is the probability half will send in one particular 5% subinterval? Multiply this by 20 for an estimate of the probability half will all send in the same arbitrary 5% subinterval. (Hint: How many ways can we choose 5 receivers out of 10?)

**53.** What might a server actually do with the packet-loss-rate data and jitter data in receiver reports?

**54.** Propose a mechanism for deciding when to report an RTP packet as lost. How does your mechanism compare with the TCP adaptive retransmission mechanisms of Section 5.2.6?