Solution for Homework 1

May 4, 1999

Summary

The solution for each problem consists of the problem text in helvetica, a description of the grading scheme, a list of common mistakes, and a sample answer that was given full credit, even if it's a little bit wrong, in `courier`. A 1.0 represents full credit.

In addition, I've scripted up some cumulative distribution functions (the little graphs) for each of the problems and the overall grade. A cumulative distribution is like an integrated histogram. You can count the number of students who got a better or worse grade than you by finding your score on the x axis, and seeing what percentile that corresponds to on the y axis. This seemed more informative than just an average score or a median.

The cdf for the entire homework set is below.

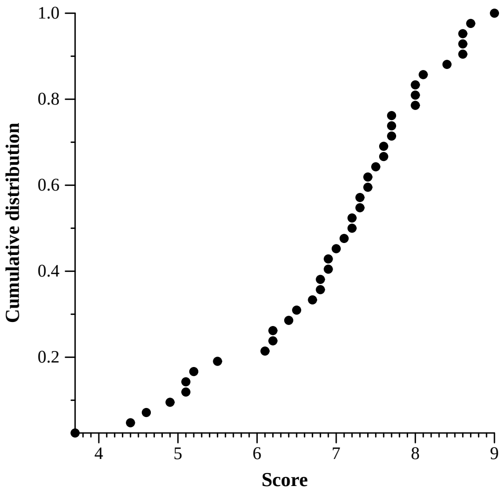


Figure 1: CDF over the entire homework

The median score is where the y-value is 0.5. You can see the lowest score at the far left, and the highest score on the far right. If you have trouble reading these, let me know, I've used them for a while now.

1

The book claims that the initial sequence numbers (ISN) of a TCP connection are chosen "randomly" by each end point. This is not quite true. Find out what TCP actually does to choose ISNs and briefly describe the algorithm. You might look this up in the TCP specification itself, as described in "Request for Comments 793" (RFC-793). To find an on-line copy, try your favorite search engine or go to http://www.ietf.org/, and follow links.

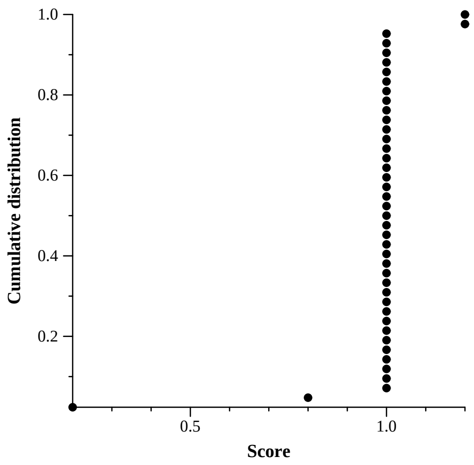
Most everyone got this right.

1.1 Grading Scheme

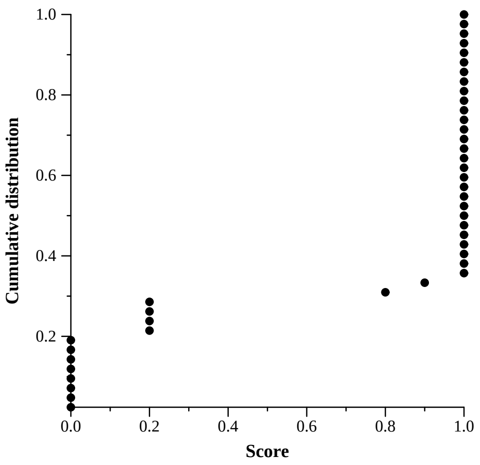
Full credit to those answers including "4 microseconds" and "4.5 hours". A little extra for those who discovered and concisely described that the BSD and derived implementations incremented every 8 microseconds, or that BSD also increments by an extra random component each second to avoid predictability.

2

Hugh Hopeful is the President of JTBD (Just to Be Different) Network Company. After consulting marketing, Hugh decides to offer a new framing protocol where the user can specify their own



Problem 1



Problem 2

flags to delineate the start and end of frames. One popular framing protocol is HDLC, which uses 01111110 as the start/stop signal; to prevent confusion, HDLC stuffs an extra 0 after every 5 bits in the user data that are all 1's.

Suppose the user provides an 8 bit flag of the form "aSbc" where a, b, and c are arbitrary bits, and S is an arbitrary 5 bit string. Hugh invents the following bit stuffing rule – whenever you find the string S in the user data, an extra bit equal to the complement of b is stuffed after it. Notice that Hugh's rule gives exactly the HDLC bit stuffing rule. Can Alyssa P. Hacker find a flag that Hugh's stuffing rule will fail on? Explain.

There were two ways to make a framing protocol fail. Simplest is to make the frame marker show up somehow inside user data, despite any transform. An alternate solution is to have the end frame marker seem like user data, so that the frame can't end.

Both are ultimately the same answer. Key is to generate an ambiguous bitstream on output: one that might be the end of a frame, or might be encoded user data.

It was unnecessary to come up with a general form for all such frames that would fail. Unnecessary mostly because it took me way too long to figure out why your general form worked, and to try enough examples to convince myself it was ok. It seemed clear that this was a proof-by-example sort of question.

2.1 Grading Scheme

Full credit to those that showed a frame marker turn into user data or user data turn into a frame marker.

Partial credit to those who made mistakes along the way, but who seemed to have the right idea.

2.2 Sample Correct Answer

```
2.  It's not gonna work.  Take the fol-
lowing flag:  00100000 and the followng
two byte data stream 00100000 00000000.
According to the rule, we'd stuff to cre-
ate 001000100 00000000.  But it's pretty
easy to see that the stuffed bit creates
a new eight bit pattern that matches the
above flag.
```

3

Suppose a single sender and receiver connected to the same Ethernet are communicating using a sliding window protocol. Explain why we might see a collision on nearly every packet.

Key to answering this question was the simultaneous transmission of data packets from the sender and acknowledgement packets from the receiver. Ideally, the answer would note that data packet $i+1$ would collide with the acknowledgement of packet i.

3.1 Common mistakes:

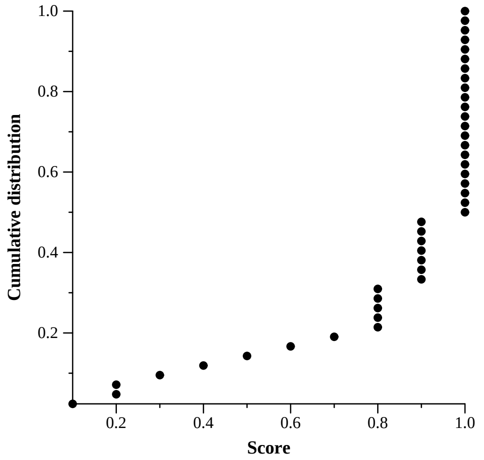
Stating that ethernet used a 1-persistent protocol really wasn't important. Proposing solutions, especially those that aren't used, didn't help either.

An answer of the form, "senders send packets, receivers send acknowledgements for every packet, therefore we might see a collision on nearly every packet" is incomplete.

Stating that the sender sends packets and the receiver sends acknowledgements wasn't enough. The receiver could wait until after the sender was done, paticularly because the sender has to wait between sending consecutive packets.

Note that the sender can't send packets that collide with its own previously sent packets.

Ultimately the correct answer is that if the sliding window protocol can generate an acknowledgement for the previous packet at about the same time the sender decides the link is idle, both will collide.



Problem 3

3.2 Grading Scheme

Answers containing the phrase "packet i+1 collides with ack i" generally got full credit.

Partial credit was given if the answer wasn't clear or contained an incorrect statement that implied it wasn't well thought out.

3.3 Sample Correct Answer

```
3.  After sending a frame the sender
must wait a bit before sending the next
one.  Suppose the receiver sees that the
line is idle so it creates an ack packet.
Since the receiver wasn't the last one to
transmit it may start right away.  If the
time for the receive to decide to send
and prepare the ack is close to the time
the sender must wait then they could eas-
ily collide when they both tried to start
to send.
```

4

Ethernets use Manchester encoding. Why does this allow for collisions to be detected soon after they occur, without waiting for the CRC to be computed at the end of the packet?

The trick with collision detection is to realize first that it occurs on the sender side. That is, when a sender sees the link is being used by another sender, a collision is detected. The only thing that remains for the sender to do before waiting is to continue sending the packet to make sure the other sender knows a collision has occurred, by sending enough bytes.

Manchester encoding has two relevant properties. First, the clock signal is embedded within the data signal. That is, the receiver can synchronize with the sender on every bit. A destruction of the clock signal, or detecting out of phase clocks on the wire could be used to indicate a collision.

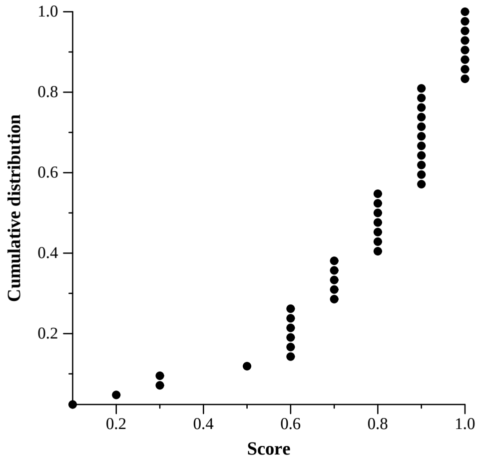
Second, as a consequence to the first, it frequently makes transitions between zero and one.

4.1 Common mistakes:

Details of what ethernet does after detecting a collision (exponential backoff, sending additional bytes, use of CRC, etc.) were unnecessary. A description of Manchester encoding, while useful for providing context, was also unnecessary. (I have a copy of the book too.)

The terms 'abnormal voltages', 'invalid bits' and 'illegal transitions' were useless without definition. Usually this was part of an answer of the form: Collision causes an x on the wire, and that x is easy to detect.

Signals like these don't necessarily add in straightforward ways. In particular, when sender 1 and sender 2 are both, well, sending, a receiver close to sender 1 will see a signal dominated by sender 1's signal (because of resistance of the wire; think resistors in series). It won't be the case that the signals will add: if a 'high' is encoded as 3V, two highs aren't 6V, and a high and a low merged isn't 3V, or even necessarily 1.5V.



Problem 4

4.2 Grading Scheme

Nearly full credit to answers that described some sort of munging of the clock signal, or the inability to recover the clock signal from the wire, or something that really leveraged frequent transitions.

Partial credit to clear answers that described a scheme that specifically leveraged manchester encoding.

Less credit to schemes that would work without manchester encoding.

4.3 Sample Correct Answer

```
4.  Manchester encoding synchronizes
the sender and receiver by merging the
clock with the message Rs signal.  If two
packets collide, their signals will most
likely not be exactly aligned.  One will
be out of phase with the other, which
will appear as noise on the wire.  No
CRC needs to be computed because it is
already apparent that the data is cor-
rupt.
```

5

Suppose we are only concerned with reducing the latency to transmit a packet over an unloaded store and forward network (for example, to download a page from a web server). Suppose each stage in the network has both a fixed overhead (o) to handle a packet (prepare it to be forwarded), and a bandwidth (b) to transmit the packet once the overhead has been paid. Explain why, as a function of (o), (b), and the packet size (s), it is sometimes more efficient to fragment the packet before sending it.

The key realization was that sending a packet as small fragments would have the same effect as pipelining. With one big fragment, the entire packet has to be received by the next hop router before it can be forwarded along. If the overhead o is zero, sending one byte per fragment would allow maximum parallelism and throughput.

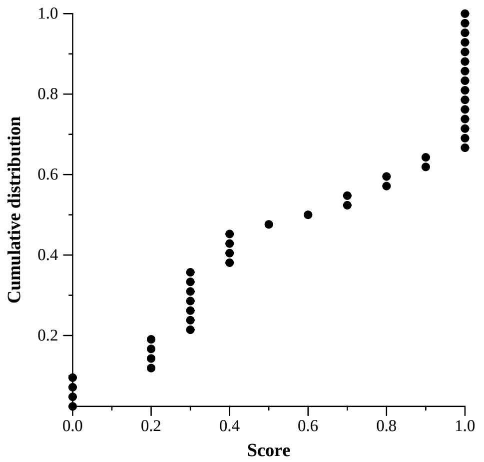
5.1 Common mistakes:

Some just used the $t = o + s/b$ equation to realize small s implies smaller t. Unfortunately, smaller s implies more fragments to send a packet, and thus, more times to pay the cost of o.

Others made the realization that the first fragment would get through faster, and that for some applications, this would be good enough. A complete answer described why the entire packet (all fragments, including the last one) would get through.

Some suggested that if a fragment is lost, less is wasted than if an entire packet is lost. Unfortunately, one of the reasons fragmentation is harmful from the Mogul paper was that losing fragments in the network necessitates resending the entire packet (and fragmenting it again).

MTU was not an issue.



Problem 5

5.2 Grading Scheme

Full credit for any answer that realized the pipelining, and a little extra for those that came up with an equation that described when fragmentation would improve latency.

Partial credit for those who noticed that concurrency would allow the first fragment of a packet to come through the link more quickly than a not-fragmented packet.

5.3 Sample Correct Answer

```
5) Given the parameters the latency at
node i is Oi + (Bi * S). The reason that
it is sometimes more efficient to frag-
ment a packet is essentially the same
reasoning as RISC instruction pipelining.
When several small things are done in
parallel, any given step n only costs the
maximum time for any single stage.  For
example, fragmenting a packet over a path
consisting of many 1Mbps and 10Mbps links
will likely be faster because in one step
you can move many packets at a "constant"
cost of the slower 1Mbps link.  This is
especially true if the slower link has
low fixed overhead cost and the fast link
has a high overhead cost.
```

6

If fragmentation is considered harmful, why don't all networks support arbitrarily large packets?

There were four reasons that made sense as reasons to avoid large packets. Most important were:

- a large packet could tie up the link for a long time, preventing more important, but smaller, packets from meeting real-time deadlines.
- an error in tranmitting a large packet is more costly, and more likely than in smaller packets.
- forwarding delay, or the result from number 5.

- limited buffer space in the router to hold the packet for checksum/CRC before forwarding. The reason limited buffer space was an issue (to check the packet) was critical to credit using this answer.

It took some special words to argue the validity of this last one, since many small packets can also cause a buffer to overrun. The only way to get credit for this answer alone was to note that an arbitrarily large packet might never fit into a not-quite so arbitrarily large buffer. And that an arbitrarily large packet needs to be stored and checked.

6.1 Common mistakes:

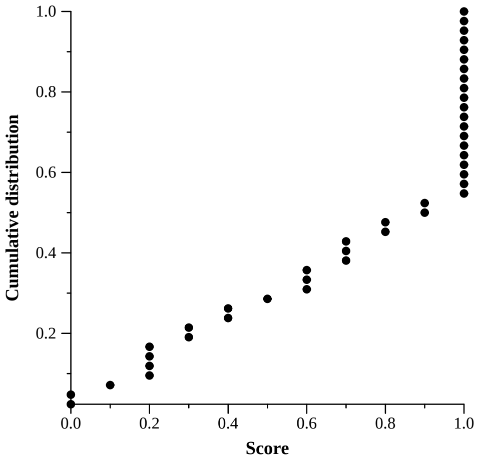
Some argued that overhead is increased by having large packets. Unfortunately, since many overheads are per-packet, having a large packet size would reduce the total overhead. For example, the cost of a CRC of a 30KB packet should be no more than the cost of 30 CRC's on 1KB packets.

Others argued that larger packets meant more congestion. This wasn't so good, since to a sender, it doesn't matter whether it sends 5 1000 byte packets or one 50000 byte packet. If a sender wants to send too much and cause congestion, it doesn't matter whether large packets are supported or not.

6.2 Grading Scheme

Full credit to any single, well described answer. A little extra for those who clearly described more than one.

Partial credit for those that weren't clear, but did approach one of the reasons described above.



Problem 6

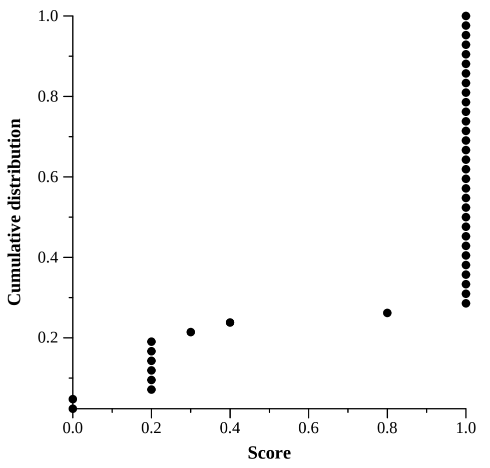
6.3 Sample Correct Answer

```
6) Using large packet size has its disadvantages:
```

- When you use large packet sizes the transmitter has to wait for the entire packet to be transmitted before it can proceed to process the next packet in the queue, thus it will increase the latency of the next packet in the queue.
- It is possible that because of the above a small sized higher priority frame has to wait for a large lower priority frame.
- The protocol layer will have to wait for the large packet size to be received before it can start processing it thus the link will be sitting idle while a large sized packet arrive.
- In certain cases it is not possible to chose an arbitrarily large packet size because of the proability that an error will be introduced in a frame due to a noisy media, in that case a large chunk of data will have to be retransmitted.

7

A common mistake in implementing the alternating bit protocol is to have the receiver only acknowledge packets whose sequence number is expected. Explain how this strategy will cause the protocol to fail.



Problem 7

7.1 Grading Scheme

Lots of full credit for describing the situation when an ack gets lost.

0.2 points who described the vulnerability of the single bit protocol to reordered packets. This is a problem that applies to the single bit protocol without the mistake.

7.2 Sample Correct Answer

```
7.  If the receiver in the alternating
bit protocol receives a packet with an
unexpected bit, it may be a retransmis-
sion from the sender due to the fact that
the sender never received an ACK for the
first transmission of the same packet.
Because the network is unreliable, ev-
ery packet must be acknowledged by the
receiver.  If the receiver does not ac-
knowledge the unexpected packet, the
sender may continue to repeatedly send
the same packet.
```

8

TCP uses a 16 bit checksum per segment, typically 576 bytes for Internet connections. As we noted in class, most link level protocols do their own bit error detection and correction. Is this really necessary – what is the maximum link layer bit error rate you would be willing rely on TCP to catch? Defend your answer.

Since most everyone answered this admittedly subjective question qualitatively, giving quantitative scores to this question didn't make much sense. We threw it out: pretend that everyone got full credit.

The important thing to get from this question is that checksums are weak. Two opposite bit flips ($0 \rightarrow 1$ and $1 \rightarrow 0$) the same column will be undetected. Hence, if bit errors occur randomly, with a k bit checksum, if 1/r packets are corrupted, roughly $\frac{1}{2kr}$ of the corrupted packets will be undetected. In practice, the bit flips don't occur randomly, but occur in bursts, which makes undetected errors more likely.

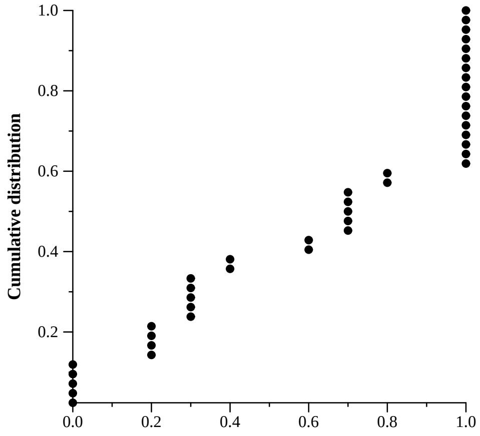
9

The Internet architecture supports "probe" packets - on receipt of a probe packet, the destination immediately transmits a response to the source. Further, routers will reply immediately back to the source when a packet's hop count reaches 0. Explain how a source can use the elapsed round-trip time for these probe packets to infer the bandwidth and delay of every link between the source and the destination. (First show how it can be done for the first hop, and then show how it can be done recursively.) Assume the rate and delay are the same in both directions.

Key to answering this question was to realize that it asked for both delay and bandwidth. In order to solve for two different variables, you need two different equations, materialized by having two different packet sizes.

Two strategies were generally given full credit.

- Send consecutive packets of different sizes and measure the time between acknowledgements.
- Send packets of different sizes and measure the round trip time.



Problem 9

9.1 Grading Scheme

Generally any answer that used varied packet size, and presented an equation (in words is ok) parameterized by measured values was given full credit.

9.2 Sample Correct Answer

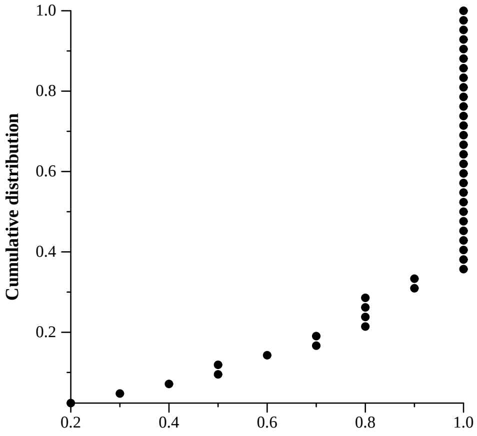
```
9.  This can be done to the first node
by sending out a probe packet that has
a hop count of 1.  The router will im-
mediately send that packet back.  The
the round trip time of the probe packet
is the transit time to the first node.
The transit times to all other nodes can
be calculated by sending a probe packet
with a hop count that will cause it to be
sent back by that node.  The transit time
from node n-1 to n is 1/2 (roundtrip(n)
- roundtrip(n-1)).  You could then find
the band width of each hop in a similar
manner by sending two large packets to
each node and finding the gap in the acks
that came back.  Given the Transit time
message size and bandwidth you could then
calculate the delay using this formula.
D = T - M / R where T is the transit
time, M is the packet size, and D is the
delay.
```

10

For short connections (such as for web requests!), the overhead of TCP's three-way handshake can dominate the actual data transfer. Design a TCP-lite protocol that can send data directly without waiting for the handshake (even on the first connection between two hosts), or show why it can't be done.

Most everyone came up with a solution that included sending data in the same packet used by the syn connection request, and sending at least the first packet of reply in the syn/ack response. Some added an early connection close to the mix.

Since the question asked you to design a protocol, the above was not really enough. Some discussion, either about what happens if a tcp-lite packet is dropped, or about how to preserve exactly once, reliable semantics of tcp, was necessary to support an answer.



Problem 10

10.1 Sample Correct Answer

```
A TCP-lite protocol could be used that
did not need the handshake unless an
error occurred (e.g.  a packet was
dropped.)  Data could be sent in the
```

first packet (the document request), and the response could also contain data (the document). If the sender does not receive the document (request or response is lost), then after a timeout the original TCP protocol could be used. Therefore for the common case of small documents that can be transmitted without error, TCP-lite will be more efficient, and when an erorr does occur it is not less efficient than TCP because TCP would have also suffered a lost packet.