Homework 1

Due: 27 September 2012, 4pm

Question 1 - Layering

a. Why are networked systems layered? What are the advantages of layering? Are there any disadvantages? [2 pt]

Some of the reasons for (and advantaged of) layering: managing complexity in implementing networks that scale, work in different environments, and incorporate a large number of devices and requirements; separating functionality in modules; allowing for independent evolution of individual layers; allowing for independent implementations of the layers.

Some of the disadvantages of layering: overhead both in terms of bytes and in performance. Encapsulating and decapsulating successively adds both of these overheads. Another disadvantage is that the information hidden behind the interfaces can sometimes be useful for upper layers. Lastly, poorly designed layers can impose uneeded functionality in upper layers: this is what motivated separating UDP from TCP in the early days of the Internet.

b. TCP provides a useful service abstraction, what is it? Why do we also have UDP, instead of just building all applications on top of TCP? [1 pt]

TCP provides the abstraction of a connection-oriented reliable infinite stream of bytes. Some people wrote 'in-order packet delivery'.

While TCP internally does deliver segments in order, it does not *expose* the packets, but rather a stream of bytes. Another common mistake was to say that TCP, by providing a connection-oriented service, reserves bandwidth. **It does not!** TCP still uses packet switching underneath. As we will see, it offers fairness instead of reservation, and if n connections share a bottleneck link, each one will get approximately $1/n$-th of the bandwidth. If a connection stops sending data but remains open, however, it will take no bandwidth in practice.

We also have UDP because providing TCP's abstraction has a cost, and some applications which do not need TCP's functionality of reliable, in-order delivery, shouldn't have to pay this cost.

c. Describe an analogy between the layered system in networking and the postal system. How would you divide the postal system into layers? What are the services each of these layers provide? When is there multiplexing and demultiplexing, encapsulation, aggregation? (Don't write more than a page!) [1 pt]

Here's one possible set of analogies (there were some good variations in the solutions): The person writing the letter conceptually communicates with the person receiving the letter. This is the highest layer, analogous to the application layer. Their communication follows letter writing etiquete (format, content, greetings, addressing, etc), which is their *protocol*. They use the services of the lower layer, the postal service. Letters are placed in envelopes with labels, analogous to encapsulation and packet headers, respectively.

The postal service offers the abstraction of delivering a letter to the recipient's address, with optional reliability (in the form of the letter returning in case the recipient can't be found, and optional delivery confirmation). The protocol involves the form of addressing, forwarding, pricing, and routing (through several post offices and intermediate points). This is analogous to the transport and networking layers. The post office does multiplexing and demultiplexing of several local addresses (homes and businesses) into one aggregating point (approximately a zip code).

Between two points in the post office topology, packages are possibly bundled together and send in batches. The postal service uses the services provided by transportation vehicles to go from one point to the next. These can be cars or planes, for example. They follow their own protocols in navigating, scheduling, and accessing their medium, following the rules of the road or of air traffic control, and are analogous to the link and physical layers.

The key here is that the higher layers use services from the lower layers, and are not concerned with details of their implementation.

Question 2 - Link Layer Design

You are designing a link layer protocol for a link with bandwidth of 1 Gbps (10^9 bits/s), over a fiber link with length of 800 km. Assume the speed of light in this medium is 200,000 km/s. (In these types of questions, beware of bits and bytes!)

a. What is the propagation delay in this link? [1 pt]

$$\frac{8 \times 10^2 km}{2 \times 10^5 km/s} = 4 \times 10^{-3} s,\ or\ 4ms$$

b. What is the transmission delay for a 2,000 byte frame in this link? [1 pt]

$$2 \times 10^3 B \times \frac{8b}{B} \times \frac{s}{10^9 b} = 16 \times 10^{-6} s\ or\ 16 \mu s$$

c. If your protocol will be a sliding window protocol, how many bits do you need to represent the sequence numbers in the protocol, with the 2,000-byte frame? (Assume that the receiver window will be as large as the sender window. Also assume that the ack is so small that its transmission delay – but *not* its propagation delay – is negligible). [2 pt]

The first step is to determine the window size: how many frames can we fit in one round-trip time? The round trip time includes: 1. the propagation time for the first bit to arrive at the other end, 2. the transmission time for all bits to arrive at the other end, and 3. the propagation time for the first bit of the ack to come back. This will be all, since we can assume the ack transmission time to be negligible.

$$1 + 2 + 3 = 8ms + 16\mu s = 8.016ms$$

Now we determine how many frames can fit in that time: $8.016ms/0.016ms = 501$ frames. You arrive at the same result if you multiply the bandwitdh (10^9 bits/s) by the delay ($8.016 \times 10^{-3} s$), and divide by the frame size (2000 bytes).

The last remaining part is to determine how many bits you need if you want to represent a window of size 501. From lecture, we saw that the sending window can't be more than half of the space of valid sequence numbers, which is the same as saying that the number of valid sequence numbers has to be *at least twice the size of the sender window*. Thus, we need a space of at least 1002 valid sequence numbers, which means we need at least 10 bits to represent the sequence numbers.

Common mistakes included not taking into account the transmission time (yielding 500, and not 501 frames), and not considering that you need at least twice as many sequence numbers as there are frames in the window. The reason for that is that the receiver has to be able to be sure that a sequence number can't be reused by the sender before it gets an acknowledgment.

d. You design your protocol to have a header + trailer of 100 Bytes, and thus a payload of 1900 bytes. If there are no losses, and the window is always full, what is the throughput seen by the network layer? [1 pt]

If the window is always full, we are sending at the full bandwidth of the link. We are paying an overhead of 100/2000, or 5%. The throughput seen by the network layer is 95% of the bandwidth, or $9.5 \times 10^8 b/s = 1.1875 \times 10^8 B/s$.

A common mistake here was to consider the transmission time. In steady state, if the window is full, the receiver will be receiving frames back-to-back, i.e., will be receiving bits at the full bandwidth. Another way to see it is that over the course of a window, which lasts $8.016ms$, the receiver will receive 501 frames of 1900 bytes: $501 * 1900 * 8 \frac{bits}{window} / 8.016 \times 10^{-3} \frac{s}{window} = 9.5 \times 10^8 b/s$.

e. What is the difference in your performance if you set the receiver window to be 1? What about if you set the receiver window to be twice the sender window? [1 pt]

If you set the receiver window to one, the performance does not change unless there is a loss, or if the receiving process at the other end can't consume frames fast enough. In case there is a loss, however, there is a degradation in performance because the receiver won't be able to retain any subsequent frames that are received after the lost frame. These will have to be retransmitted after the sender retransmits the missing frame. With a larger receiving window, only the missing frame has to be retransmitted. In the worst case, you can lose an entire window worth of frames for every loss if the receiving window size is one.

A common mistake here was to think that because the receiver window is 1, only one frame per sender window can be received. This is not necessarily true: if the receiver can drain the frames fast enough from the buffer, then it can theoretically run at full speed. This situation is just a lot more sensitive to any disruptions!

If the receiving window is larger than the sending window, however, you don't gain anything, as there can never be more outstanding frames (which would potentially require space at the receiver) than the sending window size.

Ethernet

a. Why is a non-switched 10 Mbps Ethernet limited to 2,500m? [1 pt]

This limitation comes from the need to set an upper bound on the end-to-end propagation latency in the network. The reason this is important is that this limit sets the minimum transmission time for frames. The important issue here is that you understand the interdependency among the minimum transmission time, the maximum length, the bandwidth, and the size of the minimum-length frame.

To see why, recall the situation in which there is a maximum-length Ethernet (2,500m), and two nodes, A and B, in the two extremes. A starts a transmission at time 0, the first bit of which reaches B at time d. But an instant before that (say, $d - \epsilon$, B starts another transmission, before there was a collision. When the first bit of B's transmission reaches A, at time $2d$, *A must still be transmitting*. So, the minimum transmission time has to be equal to $2d$, which, for a 10 Mbps Ethernet was set to the time to send 64 Bytes, or 512 bits, or $51.2\mu s$. The 2,500m length was set so that this time was respected.

There is a small detail missing: if you do the math, the propagation delay of an electromagnetic wave at 2/3 of the speed of light in a wire of 2,500 is $2.5 \times 10^3 m / 2 \times 10^8 m/s = 12.5 \times 10^{-6} s = 12.5 \mu s$. The round trip time would be $25 \mu s$. This seems too conservative! The extra time allowed takes into account that you have at most 4 repeaters, and each of them adds a little delay of their own, in the form of processing delay, and the designers wanted to make sure that even with some extra delay they would be safe.

b. Ethernet is also limited to 1,024 hosts. What happens when we increase the number of hosts in the same Ethernet segment? [1 pt]

As we increase the number of hosts, the probability of collision increases. This was the answer we were seeking. A lot of people wrote that the reason for 1024 is that since you have 1024 maximum different backoff intervals, beyond that you would be guaranteed to have a collision. In fact, because of the probabilistic nature of choosing slots, you would have a collision with very high probability much sooner, and most Ethernets are limited in practice to about 200 nodes *in the same collision domain*.

c. What is the difference between an Ethernet repeater, an Ethernet bridge, and an Ethernet switch? [1 pt]

An Ethernet repeater connects two Ethernet segments and repeats the electrical signals seen on one to the other. This means that the two segments are part of the same collision domain, as well as in the same broadcast domain.

An Ethernet bridge connects two segments and copies the frames that it sees on one to the other. The two sides of a bridge are *not* in the same collision domain. Further, a bridge does not forward corrupted frames, and can also refrain from forwarding a frame if it knows that the destination is not on the particular segment.

An Ethernet switch is simply a bridge that connects more than two segments.

A common misconception was that some people wrote that bridges are implemented in hardware and switches in software, and this is not necessarily true. Either can be implemented in software or hardware.

Another one was that switches know where to send frames. Just like bridges, they run a learning algorithm, they do not know in advance what to do with frames, and copy an incoming frame with an unknown destination to all other ports.

d. What is the difference (in terms of traffic in the network) if I have two Ethernet segments connected by a learning bridge, and by a non-learning bridge? [1 pt]

The difference is that the learning bridge suppresses traffic destined to a node to only the port where it knows to receive frames from that node. This greatly reduces the need to broadcast messages, and essentially converts frames between two nodes who have previously communicated into unicast messages.

A non-learning bridge, on the other hand, always has to broadcast all frames on all of the ports that are not the incoming port, as it does not know through which port to find a destination node.