1. Consider two hosts, A and B, connected by a single link with a capacity of $R$ bps. Suppose the two hosts are separated by $m$ meters, the propagation speed along the link is $s$ meters/sec, and host A needs to send a packet of size $L$ bits to host B.
   a. Express the propagation delay (one-way delay for a bit)
   \[ \frac{m}{s} \text{ seconds} \]
   b. Express the transmission delay (time to send the packet)
   \[ \frac{L}{R} \text{ seconds} \]
   c. Express the end-to-end delay between when the packet is sent and it is completely received (assume no overhead or queuing delay)
   \[ \left( \frac{m}{s} + \frac{L}{R} \right) \text{ seconds} \]

2. Many companies transfer very large amounts of data (i.e., many terabytes) across the country by physically sending disk drives via UPS instead of transmitting it over a network. Why might this technique work well for these applications (e.g., sending a month of customer purchasing data at Walmart to a remote data center), but not be widely used in data networking?

   **This technique will work well for applications that are bandwidth heavy but do not have interactive latency demands.** A modern disk drive can hold a TB in a small package. 100 such drives could easily be packed into a few boxes. UPS overnight would provide delivery in 24 hours. 100TB in 24 hours corresponds to an average bandwidth of \(~10\text{Gbps/sec}\) – far more than one could expect to get out of a typical Internet connection. On the other hand the first bit will take as long as the last bit to arrive (the UPS channel has very long bandwidth delay product). For many data networking applications (e.g., Web surfing, interactive games, voice, etc) you don’t need that level of bandwidth, but you need interactive response time (certainly a turn-around time much less than 48 hours is required for most such applications).
3. Individual protocol layers generally encode their layer-specific state in protocol headers, prepended to the front of the data they are encapsulating. However, a common exception is the error detection code value (e.g., the CRC) which is commonly appended after the data. For example, an Ethernet frame is composed of a header – which provides length base sentinel, addressing and other meta-data, followed by the actual data that it is trying to send, then followed by a CRC calculated over the header and data. Why do you think these error detection fields are not directly incorporated into the protocol header?

It is an implementation optimization, because the error detection code must be calculated over all the data in the packet. Putting it at the end leads to a more efficient implementation since the sender can be calculating the CRC as the packet data streams by and then append it at the end (otherwise the packet would need to be buffered during the CRC calculation). It also makes it easy for the receiver since the data can be streamed through a CRC calculation as it is received and then after all the packet data has been processed the next bytes (the packet CRC) can be compared with the code that has been calculated.

4. If all the links in the Internet were to provide a reliable delivery service would a reliable transport layer service be completely redundant? Why or why not?

No, even if all the individual links were completely reliable, this would not guarantee that the end-to-end communication between hosts was reliable. For example, individual routers or switches might drop packets for forwarding from one link to another, the sending host’s network interface card might drop the packet, or the receiving host’s network interface card might lose the packet. The reliability at the link-layer, between point-to-point nodes on the Internet only guarantees local reliability. To provide end-to-end reliability between two arbitrary hosts requires a higher layer service.

5. Consider the following byte stream: 0xff 0x00 0x14 0x15 0x00 0x67 0x88 0x92 Describe what the output stream looks like under the following conditions:
   a) Byte-level sentinel framing (where STX = 0x93, ETX = 0x92 and DLE=0x14) 0x93 0xff 0x00 0x14 0x15 0x00 0x67 0x88 0x14 0x92 0x92
   b) Consistent overhead byte stuffing (reminder, the first byte will indicate how many bytes to the first zero or 0xff if there is no zero in the first 254 bytes). 0x02 0xff 0x03 0x14 0x15 0x04 0x67 0x88 0x92 (will also accept as correct answer with a final 0x00… although not actually necessary in COBS)