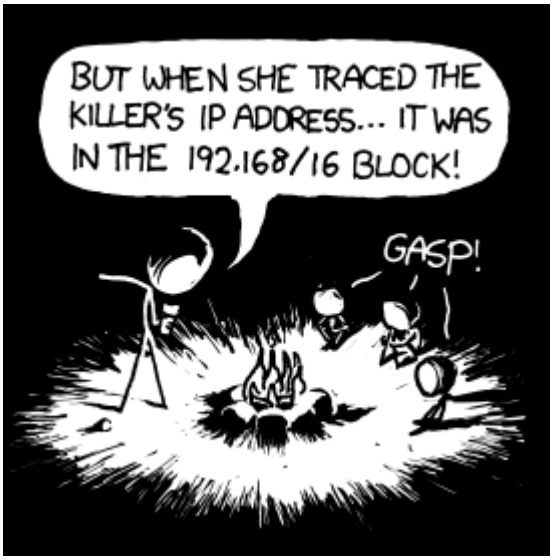
CS162 Discussion #12: Networking

Announcements

- Phase 4 design doc due Tuesday, 11/23 (`submit proj4-initial-design`)
  - Drop off your hardcopy at 346 Soda!
  - Design reviews will be after break, Monday 11/29 – Tuesday 11/30
- Final exam will be Thursday, 12/16 @ 8:00am – 11:00am in 10 Evans
- Happy Thanksgiving!



Up until now everything we've discussed has been focused on a single machine. However, we want our machines to be able to communicate with other machines, too – and this is where networking comes in. A *network* is basically some kind of connection that allows machines to communicate with each other. These machines send packets (units of transfer made up of sequences of bits) to each other through a shared medium (i.e., wires, air, etc.) following a predetermined protocol. There are two ways we can transmit data:

- *Broadcast*. Packets sent out by a sender go to everyone on the network.
- *Point-to-point.* Machines are directly connected only to each other.

Point-to-point connections are nice because only the receiver that a packet is meant for will receive it. In reality, though, there are so many machines that might want to communicate with each other that it would be very difficult to give them each their own direct connection. Instead, we can use a broadcast configuration and transform it into point-to-point connections by using routers and switches. A *router* transfers data between two networks; a *switch* turns a broadcast into a point-to-point network. We can use a hierarchical structure to break hosts down into smaller networks called *subnets* – each router reroutes packets to different subnets (or other routers) until it reaches the correct destination. Within a subnet, switches transfer data directly to their destinations.

Note: Subnets are specified by IP address, in a form such as 128.32.131.0/24. What this "/24" means is that the top 24 bits of the address must be fixed, and the rest are variable. (This is called a mask.) Therefore, since the address has 32 bits, there are 8 bits worth of possible addresses in this subnet, and they all have to start with 124.32.131.

To set up a connection, we need to establish a protocol. A *protocol* is an agreement between the sender and receiver on how the data is going to be transmitted. Here are a few:

- *Internet Protocol (IP).* Defines a method for addressing between source and destination machines, using IP addresses. This protocol delivers messages unreliably.
- *User Datagram Protocol (UDP).* Built on top of IP, unreliably transfers unordered datagrams from a source port to a destination port.
- *Transmission Control Protocol (TCP).* Built on top of IP, reliably transfers an ordered stream of bytes from a source port to a destination port.

IP allows packets to travel between machines with distinct IP addresses; UDP and TCP allows us to send messages to particular processes by using ports. Since each address can have a number of ports, each with its own connection, we define each unique connection by a 5-tuple of source address, source port, destination address, destination port, and protocol.

Ack, my packet just dropped out the window!

Packets being sent across a network can always be garbled or dropped before reaching their destination. How do we make sure that they all get to the receiver in the right order? The first step to achieving this behavior is by assigning each packet a sequence number, in increasing order. This way, we know what order the packets were meant to be in and can reorder them if necessary when we receive them. The next thing we have to do is make sure that if any packets were dropped, they get resent. The only way for the sender to know whether or not it should resend a packet is for the receiver to notify that it has gotten the message. To do this, we have the receiver send an acknowledgement packet (ack) saying the packet was successfully received. Using these two techniques, we can establish a simple way to send messages reliably: The sender will send its message, and continue resending for any packets that have timed out and it hasn't received an ack for yet.

However, since messages can be arbitrarily long, if the sender decides to send the entire message at once before resending anything, the receiver might end up with a lot of queued up packets that it can't do anything with because some packet at the beginning got dropped. At the same time, if we always wait for an ack to arrive before sending the next packet, the message will take a long time to transmit. Therefore, we go with something in the middle by using a *sliding window protocol*.

With a sliding window protocol, the sender and receiver each have a window of some established size. The sender sends all the packets it hasn't yet sent in this window, and waits for acks to come back. Every time it receives an ack, the sender's window slides forward so it can send another untransmitted packet. Meanwhile, the receiver takes in all packets corresponding to the unreceived packets in its window, dropping any packets that have already been received or are outside the range of the window. Every time a packet at the beginning of the window is received, the window slides forward until the first sequence number in the window is one that hasn't been received yet. In addition, the receiver acks all packets that it has either already received or have been stored for the first time; any packets that are beyond the window range are dropped without ack. This way, multiple packets can be sent at a time without having the possibility of building up too many packets on the receiving side blocked by dropped or unordered packets.