Section 12: TCP and Distributed Systems

CS162

April 19, 2019

Contents

1 Warmup

a) (True/False) IPv4 can support up to 2^{64} different hosts.

False, it has 32 bits per IP.

b) (True/False) Port numbers are in the IP packet.

False, they are in the transport layer. (TCP/UDP).

c) (True/False) UDP has a built in abstraction for sending packets in an in order fashion.

False, this is a part of the TCP protocol. In UDP there is no notion of a sequence number.

d) (True/False) TCP is built in order to provide a reliable and ordered byte stream abstraction to networking.

True.

e) (True/False) TCP attempts to solve the congestion control problem by adjusting the sending window when packets are dropped.

True.

f) In TCP, how do we achieve logically ordered packets despite the out of order delivery of the physical reality? What field of the TCP packet is used for this?

The seqno field.

g) Describe how a client opens a TCP connection with the server. Elaborate on how the sequence number is initially chosen.

3 way handshake. Client sends a random sequence number (x) in a syn packet. Server sees this and sends another random seqeuence number back (y) in addition to acknowledging the sequence number that it received from the client (sends back x+1) in a syn-ack packet. Client acknowledges this sequence number in an ack packet by sending back y+1.

It is important to randomize the sequence number so an off path attacker cannot guess it and send spurious packets to you.

h) Describe the semantics of the acknowledgement field and also the window field in a TCP ack.

The acknowledgement field says that the receiver has received all bytes up until that number (x). The window field says how many additional bytes past x the receiver is ready to receive.

2 Vocabulary

- **TCP** - Transmission Control Protocol (TCP) is a common L4 (transport layer) protocol that guarantees reliable in-order delivery. In-order delivery is accomplished through the use of sequence numbers attached to every data packet, and reliable delivery is accomplished through the use of ACKs (acknowledgements).
- **Flow Control** - Flow control is the process of managing the rate of data transmission such that a fast sender doesn't overwhelm a slow receiver. In TCP, flow control is accomplished through the use of a sliding window, where the receiver tells the sender how much space it has left in its receive buffer so that the sender doesn't send too much.
- **RPC** - Remote procedure calls (RPCs) are simply cross-machine procedure calls. These are usually implemented through the use of stubs on the client that abstract away the details of the call. From the client, calling an RPC is no different from calling any other procedure. The stub handles the details behind marshalling the arguments to send over the network, and interpreting the response of the server.
- **General's Paradox** - The idea that there is no way to guarantee that two entities do something simultaneously if they can only send messages to each other over an unreliable network. There is no way to be sure that the last message gets through, so one entity can never be sure that the other entity will act at a specific time.

3 Problems

3.1 Designing the Internet

In class we learned about two fundamental concepts on internet architecture: layering, fate sharing, and the end to end principle.

a) List the 5 layers specified in the TCP/IP model. Layering adds modularity to the internet and allows innovation to happen at all layers largely in parallel. What is the function of each layer?

1) Physical Layer: the physical layer is responsible for the delivery of raw bits from one endpoint to another. It includes ethernet, fiber, and other mediums of data transmission.
2) Datalink Layer: this layer adds the packet abstraction and is responsible for the local delivery of packets between devices within the same network (usually a LAN but can also include WANs). Devices here include switches, bridges, and network interface cards (NICs).
3) Network Layer: this layer is the skinny waist of the internet and routing algorithms here only speak IP. The network layer is responsible for global delivery of packets across one or more networks.
4) Transport Layer: this layer provides host to host or end to end communication. Because processes operate in terms of streams of data rather than individual packets, the transport layer handles the multiplexing of packets into streams, end to end reliable delivery, and introduces the concept of flows. This layer lives in the operating system, and TCP and UDP are examples of popular transport layer protocols.
5) Application Layer: the application layer provides network support for applications. The protocols that live here include: HTTP, FTP, DNS, SSH, TLS, etc.

b) When the internet was very young, there was an intense debate between packet switching and circuit switching. Define the two concepts, and discuss the pros and cons of each.

In packet switching, a data stream is broken down into chunks of data called packets that are transported individually. Circuit switching, which is how the telephone network is operated, doesn't do this but rather has the concept of reservations. The user requests for some bandwidth (B bps) between two end hosts, and a reservation is made on every link between the two end hosts for B. This creates a circuit of reservations, and the user is guaranteed that every second the user has complete control of the circuit for however long it takes to send B bits.

The pro of circuit switching is that it can guarantee Quality of Service. For applications like video conferencing, it is useful to know that the service will be guaranteed to be B bps. A pro of packet switching is statistical multiplexing, which says that the sum of the peaks is greater than the peak of the sums. Packet switching allows for higher bandwidth usage since an application reserving B bps may not actually use B bps. So in instead of wasting it as in circuit switching, packet switching just puts as many packets onto the the link as possible without guarantee that any one flow will receive a certain bandwidth.

c) The fate sharing principle dictates where data should be stored in the internet. In the words of David Clark:

> The fate-sharing model suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost.

The idea begin fate sharing is that in a distributed system, state should be colocated with the entities that rely on that state. That way the only way to suffer a critical loss of state is if the entity that cares about it also fails, in which case it doesn't matter. What does the fate sharing principle say about the argument of packet switching verses circuit switching?

It argues in favor of packet switching, since circuit switching requires keeping the flow state inside the network in the routers and switches. Packet switching, on the other hand, stores flow state in the end hosts within the transport layer protocols such as TCP. One reason keeping flow state in the routers is bad is that hosts now must rely on every in-path router to maintain the connection.

d) The end to end principle is one of the most famed design principles in all of engineering. It argues that functionality should **only** be placed in the network if certain conditions are met. Otherwise, they should be implemented in the end hosts. These conditions are:

- Only If Sufficient: Don't implement a function at the lower levels of the system unless it can be completely implemented at this level.
- Only If Necessary: Don't implement anything in the network that can be implemented correctly by the hosts.
- Only If Useful: If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement.

Take for example the concept of reliability: making all efforts to ensure that a packet sent is not lost or corrupted and is indeed received by the other end. Using each of the three criteria, argue if reliability should be implemented in the network.

i) Only If Sufficient

NO. It is not sufficient to implement reliability in the network. The argument here is that a network element can misbehave (i.e. forwards a packet and then forget about it, thus not making sure if the packet was received on the other side). Thus the end hosts still need to implement reliability, so it is not sufficient to just have it in the network.

ii) Only If Necessary

NO. Reliability can be implemented fully in the end hosts, so it is not necessary to have to implement it in the network.

iii) Only If Useful

Sometimes. Under circumstances like extreme lossy links, it may be beneficial to implement it in the network. Lets say a packet crosses 5 links and each link has a 50% chance of losing the packet. Each link takes 1 ms to cross and there is an magic oracle tells the sender the packet was lost. The probability that a packet will successfully cross all 5 links in one go is $(1/2)^5 = 3.125\%$. This means the end hosts need to try 32 times before it expects to see the packet make it through, taking up to # of tries $\times$ max # of links per try $= 32 \times 5 = 160$ ms. Likewise at each hop, if the router itself is responsible for making sure the packet made it to the next router, each router would know if the packet was dropped on the link to the next router. Thus each router only has to send the packet until it reaches the next router, which will be twice on average. So to send this packet, it will take on average # of tries per link # number of links $= 2 \times 5 = 10$ ms. This is a huge boost in performance, which makes it useful to implement reliability in the network under some cases.

e) An important concept of router design is the separation of the data plane and the control plane. The control plane is like the brains of routing: it makes decisions of where to forward the packets. In constructs a routing table, which is a mapping of packet metadata (usually just the destination IP but sometimes may include source IP, flow ID, etc) to an outgoing port. The data plane takes the routing table, and is responsible for the actual action of looking up a packet and finding its chosen outbound port.

Traditionally, both the control plane and the data plane reside on the routers themselves and each router works in a distributed fashion to calculate their individual routing tables. Recently, there has been a movement to detach the control plane from the routers and instead have one or a set of centralized controllers that act as the control plane for all routers. What may be the pros and cons of doing this?

The general idea of removing the control plane from the router is called software defined networking, or SDN. The con of a decentralized control plane is its immense complexity and as a result a rigid inflexibility of network configuration, slow to near impossible updating of infrastructure, and difficult troubleshooting of network issues. SDN attempts to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). By centralizing the control plane, SDN controllers have a full view of the network topology, and thus can easily make routing decisions based on network conditions and custom operator programmed logic (thus software defined). Some major concerns of SDN however are security, scalability, and elasticity.

3.2 Distributed Systems

a) Consider a distributed key-value store using a directory-based architecture.

i) What are some advantages and disadvantages to using a recursive query system?

Advatanges: Faster, easier to maintain consistency.
Disadvantages: Scalability bottleneck at the directory/master server.

ii) What are some advantages and disadvantages to using an iterative query system?

Advatanges: More scalable.
Disadvantages: Slower, harder to maintain consistency.

b) **Quorum consensus:** Consider a fault-tolerant distributed key-value store where each piece of data is replicated N times. If we optimistically return from a put() call as soon as we have received acknowledgements from W replicas, how many replicas must we wait for a response from in a get() query in order to guarantee consistency?

We must wait for at least $R > N - W$ responses. If we have any fewer than this number, there is a possibility that none of our responses contain the latest value for the key we are requesting.

c) In a distributed key-value store, we need some way of hashing our keys in order to roughly evenly distribute them across our servers. A simple way to do this is to assign key K to server i such that $i = \text{hash}(K) \mod N$, where N is the number of servers we have. However, this scheme runs into an issue when N changes — for example, when expanding our cluster or when machines go down. We would have to re-shuffle all the objects in our system to new servers, flooding all of our servers with a massive amount of requests and causing disastrous slowdown. Propose a hashing scheme (just an idea is fine) that minimizes this problem.

We can treat the possible hash space as a circle, where every possible hash maps to some point on the circle. We then roughly evenly distribute our servers across this circle, and have each hash be stored on the next closest server on the circle. Then, when we add or remove servers, we need only move a portion of the objects on one server adjacent to the server we just added or removed. This technique is commonly known as **consistent hashing**.