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.

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

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

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

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

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?

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

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

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?

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.

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?

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

ii) Only If Necessary

iii) Only If Useful

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?

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?

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

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?

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.