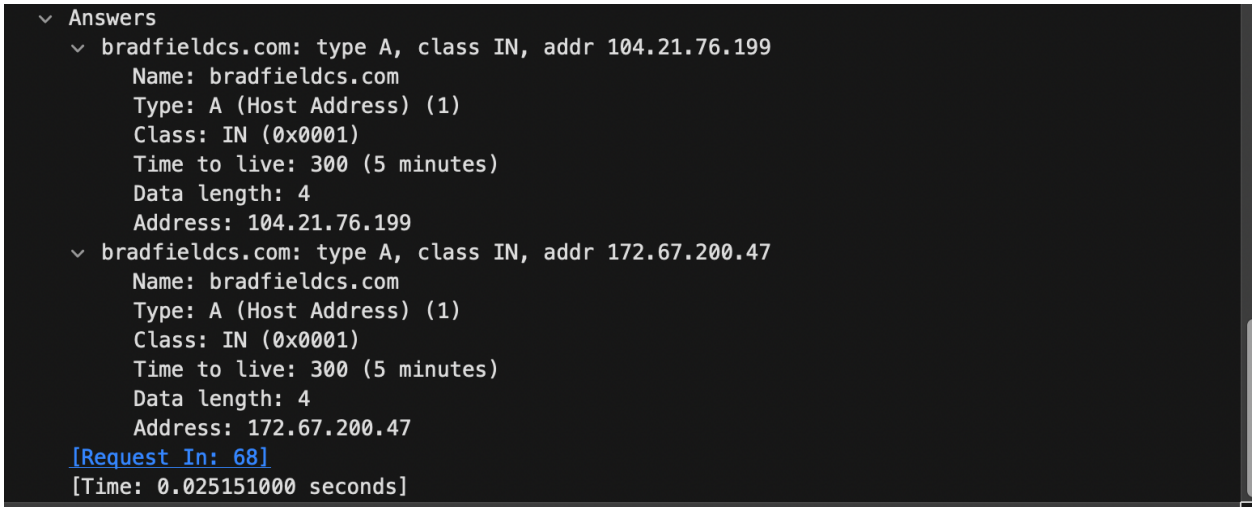
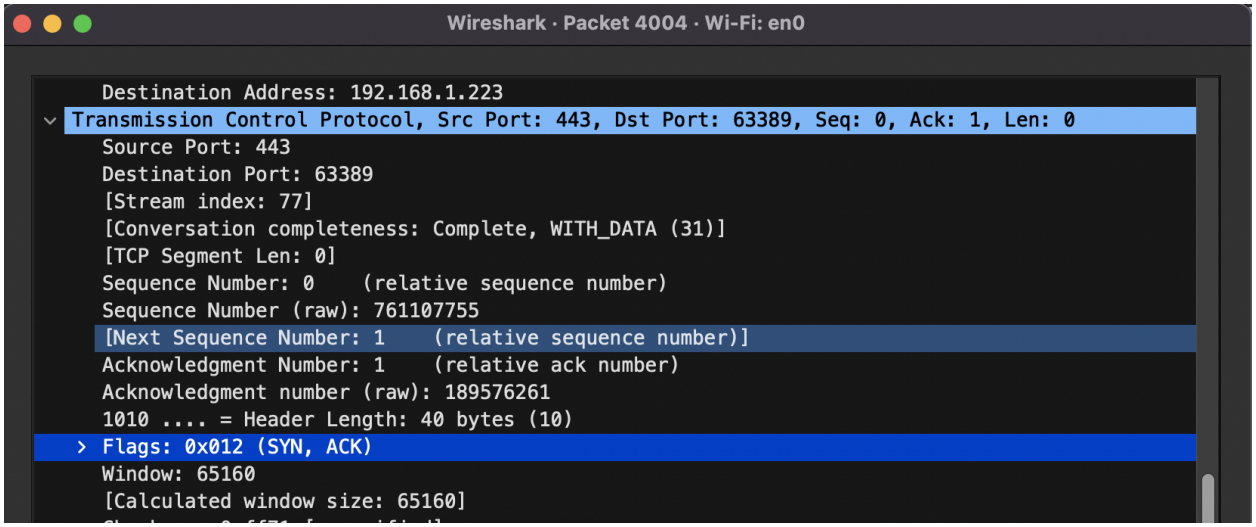
Find the DNS query for bradfieldcs.com (if you're unable to find the query, you may need to clear your DNS cache). How many DNS answers were provided, and what were the "time to live" values of each answer in seconds?

There were two answers provided in the DNS query for bradfieldcs.com.
104.21.76.199 and 172.67.200.47
Both had a time to live of 300 seconds.



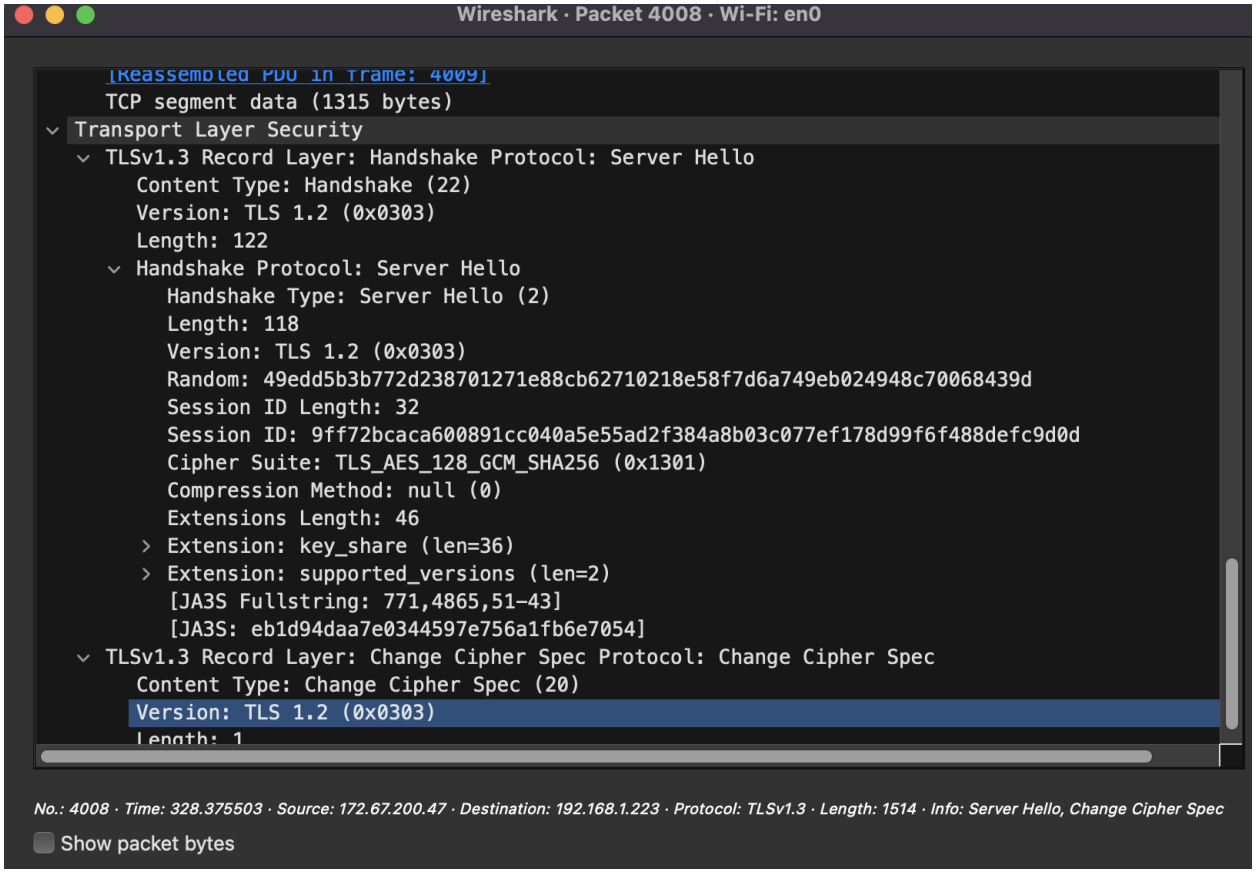
Find the SYN/ACK segment sent from port 443 of the Bradfield server. What receive window size did the server advertise?

The server advertised a window size of 65160 bytes.



During the TLS handshake, which ”cipher suite” did the Bradfield server select?

The server choose the TLS_AES_128_GCM_SHA256 cipher suite.



Dynamo

What's "eventual consistency"? What benefits are gained by tolerating it? How and when does Dynamo handle conflicts that can arise as a result?

"Eventual consistency" is the practice of allowing discrepancies to arise in replica databases during write operations. These discrepancies occur because, during a write operation, of the total N nodes storing the data that needs to be updated, only W-1 nodes need to respond to the coordinator node that they have completed the transaction before the coordinator sends a success response to the client application. Note that the coordinator also updates its data, making it a total of W nodes that have confirmed their writes. Also W <= N.

The remaining nodes may not have updated the data because of a network partition. So the database may have two or more separate values for the data, thus no longer being consistent.

Later, when a read operation is performed on that data, the discrepancies are revealed and the coordinator node can perform a syntactic reconciliation if it can determine a causal order between all the variations. If that is not possible, then the coordinator sends all versions of the data to the client application and lets them decide what to do.

The benefit of eventual consistency is that it makes write operations highly available and fast. Instead of waiting for all the replicas to finish their writes and report to the coordinator, the coordinator only waits for the first W-1 responses. Also, as long as at least W nodes are up, the write will succeed.

Suppose we want to increase the level of consistency. What conditions on the Dynamo cluster configuration are needed to guarantee that if a client writes a value for a given key, a subsequent read of the same key by the same client will include the value that was just written? Please include justification for your answer. You can assume that "hinted handoff" and "sloppy quorum" are disabled, and that each key's preference list has exactly N items.

To balance the tradeoffs between consistency, availability, and performance, three parameters can be adjusted. N, R, and W.

N is the number of nodes that store the value of each key. R is the minimum number of nodes that have to respond with the value for a given key before the coordinator node aggregates the responses and sends it to the client. W is the minimum number of nodes that have to respond reporting a successful write before the coordinator sends a success response to the client. (W includes the coordinator too)

To ensure that any writes are included in subsequent reads, R + W > N must be true. This ensures at least one of the nodes that are being read will have seen the new value. For example, if N was 5 and W was 2, then R would be 4. A write operation would only succeed after changing the values of at least 2 nodes. At worst, the remaining 3 nodes would not be updated. The subsequent read, which needs the values from 4 nodes will include at least one of the nodes with the updated value.

When using ”consistent hashing”, how does the coordinator determine the node(s) responsible for a particular key? What state or data structures are involved, and how does the state change when a node is added or removed from the cluster?

The coordinator finds the node responsible for a particular key by hashing the key, then finding the next node with a position value larger that the hash of the key. if no node is found before the end of the hash space, it loops back up from 0.

Nodes keep track of the previous N nodes before them since they are in charge of storing the keys in that range. If a new node is added in that range, then the node no longer needs to store some of its keys. The node offers to send the new node the keys it no longer needs to store, and sends them if the new node confirms it has not received them yet. Before a node is removed from the cluster, it sends the corresponding keys to the new node that is responsible for them.

Merkle trees are used to check what data between nodes is the same, and only transfer what is different.

What are the tradeoffs between (a) consistent hashing and (b) assigning a given key to the nodes using the result of hash(key) % CLUSTER_SIZE?

Consistent hashing allows for the cluster to scale easily since new nodes are just assigned a position randomly. It also limits the amount of data transfer, since only the new node and the next N nodes need to update their keys.

However, because of the random position assignment, a few nodes may end up being responsible for a disproportionately large amount of the hash space. This problem can be alleviated by using virtual nodes, which allow many nodes to map to one real node. This reduces the chance of one node being overburdened. Also by using virtual nodes, more powerful machines can choose to take on more of the hash space than less powerful machines. This allows for the data center to be incrementally upgraded.

If you were to instead divide the hash space evenly among the number of nodes in the cluster, you could ensure that the work is being divided evenly among the nodes, unless the keys were disproportionately hashing to one area.

However, there are a couple of big downsides to this approach. Adding new nodes to the cluster would be extremely costly, as every single node in the cluster would have to rebalance which keys they were storing and transfer the ones they were no longer responsible for to the correct node. They would also have to recalculate all their Merkle trees.