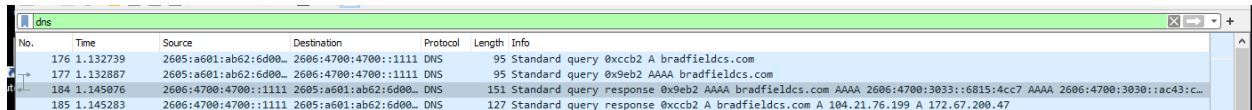


Network Packet Capture

1) Find the DNS query for *bradfieldcs.com*. How many DNS Answers were provided, and what were the TTL values of each answer in seconds?

To start let's capture the DNS requests in Wireshark from our ethernet connection. Wireshark has an easy filter for DNS, so after flushing our DNS cache, we see the request at the top here.



No.	Time	Source	Destination	Protocol	Length	Info
176	1.132739	2605:a601:ab62:6d00::	2606:4700:4700::1111	DNS	95	Standard query 0xcbb2 A bradfieldcs.com
177	1.132887	2605:a601:ab62:6d00::	2606:4700:4700::1111	DNS	95	Standard query 0x9eb2 AAAA bradfieldcs.com
184	1.145076	2606:4700:4700::1111	2605:a601:ab62:6d00::	DNS	151	Standard query response 0x9eb2 AAAA bradfieldcs.com AAAA 2606:4700:3033::6815:4cc7 AAAA 2606:4700:3030::ac43:c...
185	1.145283	2606:4700:4700::1111	2605:a601:ab62:6d00::	DNS	127	Standard query response 0xcbb2 A bradfieldcs.com A 104.21.76.199 A 172.67.200.47

It looks like we are getting two responses back—at a quick glance it looks like it is giving us both IPv4 and IPv6 options, which is pretty nice. Looking up these IPs real quick it looks like they servers are hosted by Cloudflare in a data center in California. That all checks out.

Digging down into our specific answers, it looks like we **have two options for IPv4 and IPv6, respectively**:

IPv4

```
Class: IN (0x0001)
▼ Answers
  ▼ bradfieldcs.com: type A, class IN, addr 104.21.76.199
    Name: bradfieldcs.com
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 300 (5 minutes)
    Data length: 4
    Address: 104.21.76.199
  ▼ bradfieldcs.com: type A, class IN, addr 172.67.200.47
    Name: bradfieldcs.com
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 300 (5 minutes)
    Data length: 4
    Address: 172.67.200.47
```

IPv6

```

Class: IN (0x0001)
▼ Answers
  ▼ bradfieldcs.com: type AAAA, class IN, addr 2606:4700:3033::6815:4cc7
    Name: bradfieldcs.com
    Type: AAAA (IPv6 Address) (28)
    Class: IN (0x0001)
    Time to live: 300 (5 minutes)
    Data length: 16
    AAAA Address: 2606:4700:3033::6815:4cc7
  ▼ bradfieldcs.com: type AAAA, class IN, addr 2606:4700:3030::ac43:c82f
    Name: bradfieldcs.com
    Type: AAAA (IPv6 Address) (28)
    Class: IN (0x0001)
    Time to live: 300 (5 minutes)
    Data length: 16
    AAAA Address: 2606:4700:3030::ac43:c82f
[Request In: 177]

```

TTL looks to be 300 seconds for each of these answers.

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

Now that we have the resolved IP address from the DNS query above, we can use that to make our filtering a little easier. Let's look at all TCP requests where our resolved IPv6 address is either the host or destination of the TCP request.

No.	Time	Source	Destination	Protocol	Length	Info
186	1.145453	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	86	65981 → 443 [SYN] Seq=65981 Win=0 Len=0 MSS=1420 SACK_PERM=1
187	1.145914	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	86	65982 → 443 [SYN] Seq=65982 Win=0 Len=0 MSS=1420 SACK_PERM=1
189	1.154111	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	86	443 → 65982 [ACK] Seq=65982 Ack=65535 Len=0 MSS=1420 SACK_PERM=1
190	1.154142	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	74	65982 → 443 [ACK] Seq=65982 Ack=65535 Len=0 MSS=1420 SACK_PERM=1
191	1.154258	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TLSv1.3	591	Client Hello
192	1.154358	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	86	443 → 65981 [SYN, ACK] Seq=65981 Ack=65535 Len=0 MSS=1420 SACK_PERM=1
193	1.154372	2606:4700:3033::6815:4cc7	2606:4700:3033::6815:4cc7	TCP	74	65981 → 443 [ACK] Seq=65981 Ack=65535 Len=0 MSS=1420 SACK_PERM=1

Here is the handshake. Drilling down into the SYN / ACK response, it looks like the calculated window size is **65,535 bytes**.

```

1000 ... = Header Length: 32 bytes (8)
> Flags: 0x012 (SYN, ACK)
Window: 65535
[Calculated window size: 65535]
Checksum: 0xd027 [unverified]

```

3) During the TLS handshake, which "cipher suite" did the Bradfield server select? What was the (a) symmetric encryption algorithm, and (b) data authenticity mechanism associated with that suite?

Using a similar filter to above, we can narrow down TLS requests to and from the bradfieldcs.com server.

No.	Time	Source	Destination	Protocol	Length	Info
191	1.154258	2605:a681:ab62:6d00:3d19:d718:61c2:1539	2606:4700:3033:16815:4cc7	TLSv1.3	591	Client Hello
194	1.154503	2605:a681:ab62:6d00:3d19:d718:61c2:1539	2606:4700:3033:16815:4cc7	TLSv1.3	591	Client Hello
204	1.167113	2606:4700:3033:16815:4cc7	2605:a681:ab62:6d00:3d19:d718:61c2:1539	TLSv1.3	1434	Server Hello, Change Cipher Spec
205	1.167307	2606:4700:3033:16815:4cc7	2605:a681:ab62:6d00:3d19:d718:61c2:1539	TLSv1.3	826	Application Data
207	1.167690	2606:4700:3033:16815:4cc7	2605:a681:ab62:6d00:3d19:d718:61c2:1539	TLSv1.3	1434	Server Hello, Change Cipher Spec

Digging down into the server response, it looks like the cipher suite is **TLS_AES_128_GCM_SHA256**

```

  ▾ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 62f7bc77dc71420d9945975398bf95a2c423f98b9bccad36a6b37d6d1d4791a2
    Session ID Length: 32
    Session ID: c3bb985be24ead68dd7ea3267a1d4219e409ae1e81e39ef27e0c4d396f60d878
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Compression Method: null (0)
    Extensions Length: 46
  ▾ Extension: key_share (len=36)
    Type: key_share (51)
    Length: 36
    > Key Share extension

```

The symmetric encryption algorithm in this cipher suite is AES128 and the data authenticity mechanism is SHA256.

Dynamo

1) What is “eventual consistency”? What benefits are gained by tolerating it? How and when does Dynamo handle conflicts that can arise as a result?

Using their own words from the Dynamo paper: all data updates will reach all replicas.... *Eventually*. If a user adds an item to their shopping cart, even if there is partial system failure, that action will propagate to the rest of the system. This could result in other parts of the system querying “old” information, but eventually the source of truth will settle with a cart that reflects that user action of adding an item.

The main benefit that Amazon is trying to achieve here with this tradeoff is high availability—a customer being unable to interact with the system is the most detrimental outcome they are trying to avoid. If a customer gets an error trying to add an item to a shopping cart, they will probably go to another website and purchase from there instead. By tolerating temporary mismatches in data, this becomes far more possible. There are also benefits around cost-effectiveness and performance, but for Amazon availability is likely the primary concern.

Dynamo treats modifications as immutable transactions, allowing for multiple versions to be present in the system at a time. Using vector clocks and context objects, this process of conflict resolution can sometimes be handled in the background server side. However, like working in a big monorepo in git, you’re going to need to resolve some merge conflicts sometimes. For Amazon pushing this complexity down to the client is worth it—for example shopping carts can just merge two conflicting versions of a cart and have the user reconcile at checkout. This new reconciled version is then fed back into the system.

2) 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?

To *guarantee* a consistent read after write, the number of read and write nodes (R , W) would need to exceed the number of hosts the data is replicated on (N), and R would be greater than W to ensure the client received the new value (in addition to old versions provided by the nodes that had not received that write yet). This ultimately results in a system where the latency is determined by the slowest of these operations (quorum-like), which is not ideal when trying to achieve a complete performant, low latency system. However, depending on your application needs, you can work around this shortcoming. For instance, if you have a read-heavy application, you can set R to be 1 and W to N , which would allow for performant reads at the cost of slower writes.

Tuning the values of R , W , and N is central to achieving your applications desired level of performance, availability, and consistency. Low values of W and R increase the likeliness of inconsistency.

3) What is consistent hashing? What information is needed in order to locate the node corresponding to a particular key, and where does Dynamo store that information? Finally, what are the tradeoffs between a) consistent hashing and b) assigning a given key to the node $\text{hash}(\text{key}) \% N$?

Taking the basic definition of consistent hashing from the paper “Distributed Caching Protocols for Relieving Hotspots on the World Wide Web”, a consistent hash function is one which changes minimally as the “range” of the function changes. In the specific context of distributed systems, this means that

the output of the hash function does not wildly vary when nodes are added and removed from the system. Consistent hashing also tries to address the problem of clients having different “views” of the world. Previously, solutions around multicasting tried to handle the problem of coordination of distributed file systems, but the multicasting message volume became too great as the size of the caches increased. Consistent hashing provided a solution to this distributed coordination without needing to have a single source of truth agreed upon, modeling nodes as a position on a ring.

Dynamo stores the list of nodes responsible for a particular key in a preference list, which is designed so that all nodes in the system determine which nodes store a particular key. Once you have the position of the node on the ring, you can locate it by walking the ring.

Consistent hashing is slower than traditional hashing, as you need to walk the ring to find a particular node instead of having direct access. The use of binary search makes this operation $O(1)$ vs $O(\log N)$. However traditional hashing won't distribute load consistently (prone to hotspots), and the results of the hashing function are sensitive to changes in the system. Dynamo has implemented additional features on top of the basic ideas of consistent hashing like “virtual nodes” to help even more with load distribution and allowing them to leverage performance variance between nodes.