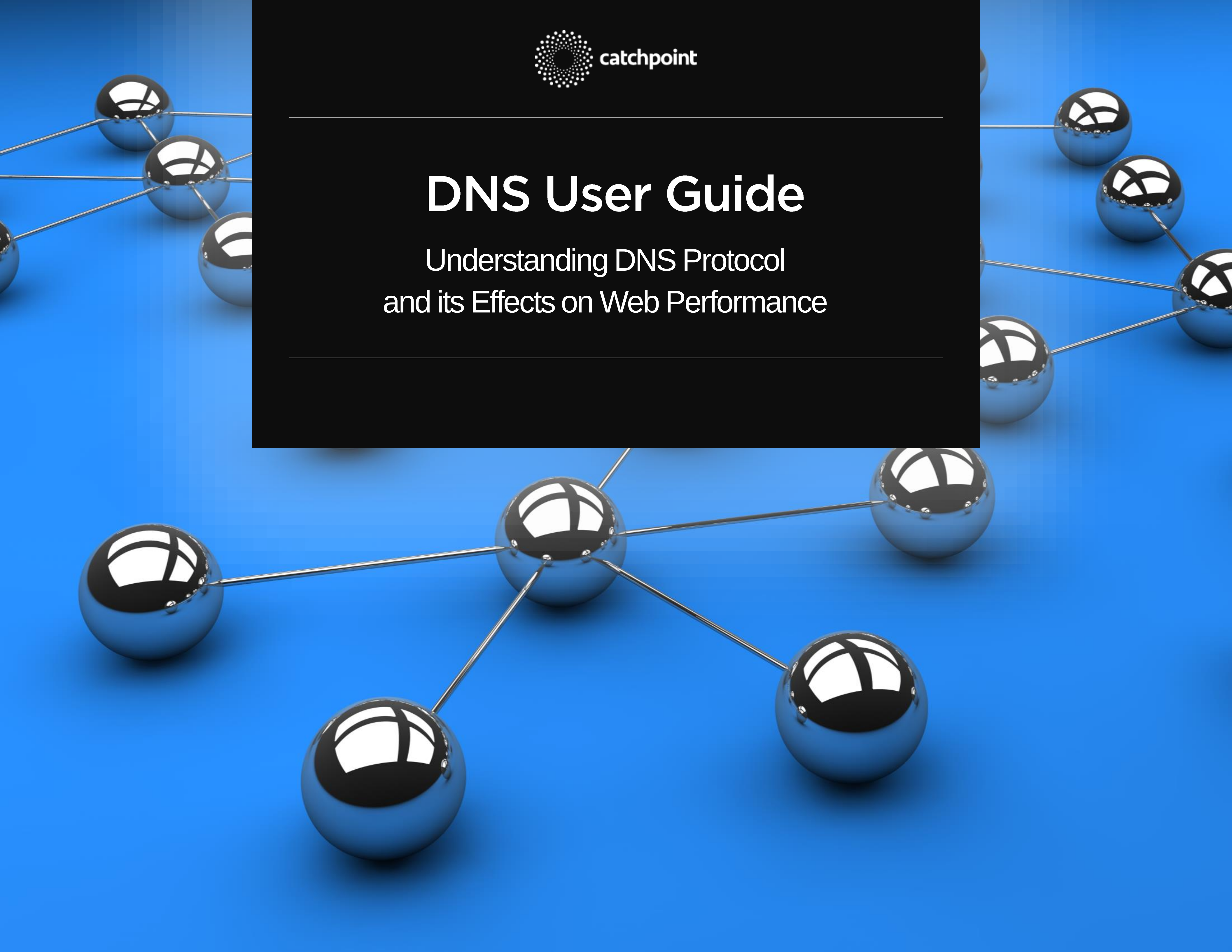
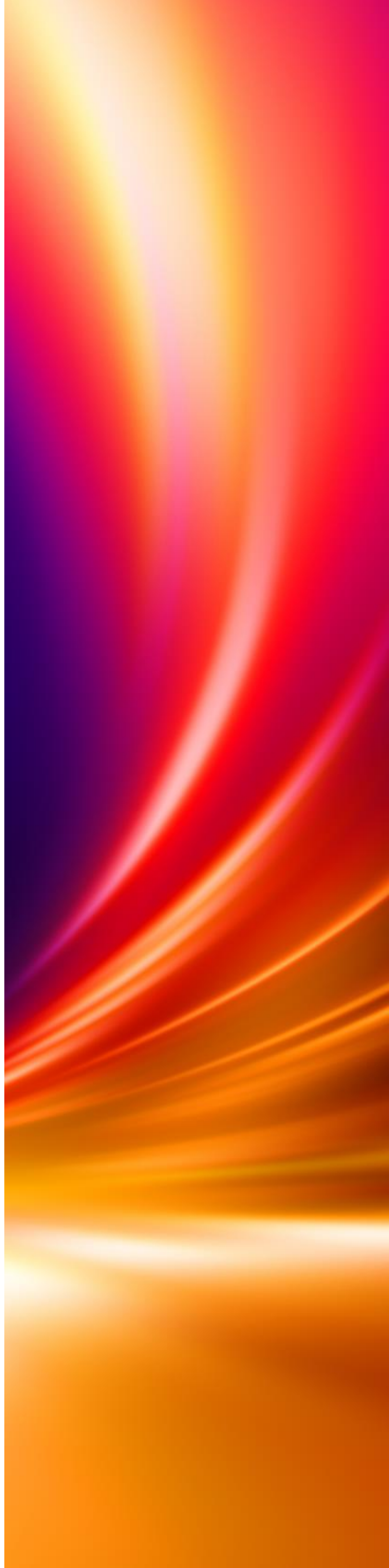


catchpoint

DNS User Guide

Understanding DNS Protocol
and its Effects on Web Performance

Table of Contents



catchpoint®

14-DAY FREE TRIAL

Actively monitor your site's performance today by requesting a free trial

REQUEST FREE TRIAL

Honeywell

"This proved the easiest business case that I have ever made to the executive team. Catchpoint paid for itself before we even finished our [proof of concept]. That simple POC environment was a game changer for us."

Paul Fries
Honeywell Monitoring Leader

Honeywell · tripadvisor · kate spade NEW YORK · Google · COMCAST · Ask · priceline.com · wayfair

Introduction: What is DNS?

At Catchpoint, we believe that a fast DNS (Domain Name System) is just as important as fast content. DNS is what translates your familiar domain name (www.google.com) into an IP address your browser can use (173.194.33.174). This system is fundamental to the performance of your webpage, yet most people don't fully understand how it works.

Therefore, in order to help you better understand the availability and performance of your site, this eBook is designed to shed light on the sometimes complex world of DNS, starting with the basics.

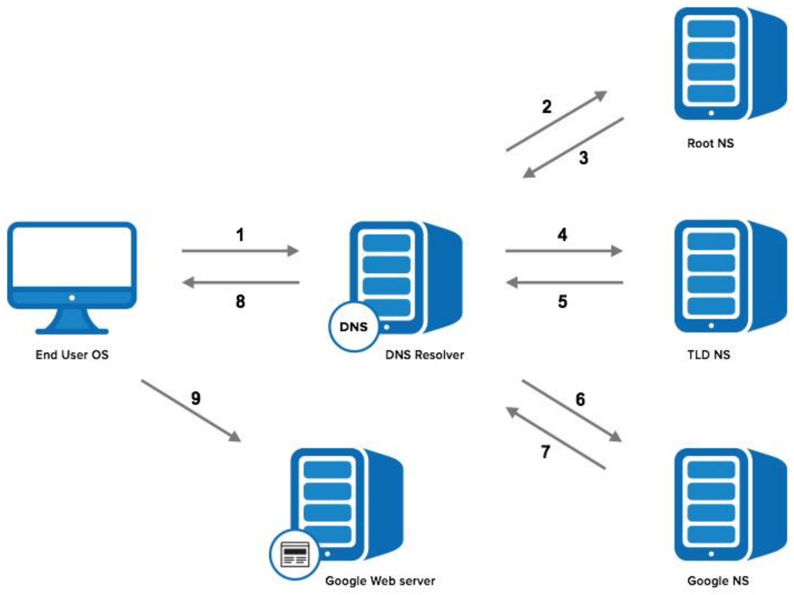
How a Domain Name is Translated to an IP Address

Before the page and any resource on the page is loaded, the DNS must be resolved so the browser can establish a TCP connection to make the HTTP request.

In addition, for every external resource referenced by a URL, the DNS resolution must complete the same steps (per unique domain) before the request is made over HTTP. The DNS Resolution process starts when the user types a URL address on the browser and hits Enter. At this point, the browser asks the operating system for a specific page, in this case google.com.

Step 1: OS Recursive Query to DNS Resolver

Since the operating system doesn't know where "www.google.com" is, it queries a DNS resolver. The query the OS sends to the DNS Resolver has a special flag that tells it is a "recursive query." This means that the resolver must complete the recursion and the response must be either an IP address or an error.



For most users, their DNS resolver is provided by their Internet Service Provider (ISP), or they are using an open source alternative such as Google DNS (8.8.8.8) or OpenDNS (208.67.222.222). This can be viewed or changed in your network or router settings. At this point, the resolver goes through a process called recursion to convert the domain name into an IP address.

Step 2: DNS Resolver Iterative Query to the Root Server

The resolver starts by querying one of the root DNS servers[1] for the IP of "www.google.com." This query does not have the recursive flag and therefore is an "iterative query," meaning its response must be an address, the location of an authoritative name server, or an error.

There are 13 root server clusters named A-M with servers in over 380 locations.[2] The root is represented in the hidden trailing "." at the end of the domain name. Typing this extra "." is not necessary as your browser automatically adds it.

Step 3: Root Server Response

These root servers hold the locations of all of the top level domains (TLDs) such as .com, .de, .io, and newer generic TLDs such as .camera. They are managed by 12 different organizations that report to the Internet Assigned Numbers Authority (IANA), such as Verisign, who controls the A and J clusters. All of the servers are copies of one master server run by IANA.
The root doesn't not have the IP info for "www.google.com," but it knows that .com might know, so it returns the location of the .com servers. The root responds with a list of the 13 locations of the .com gTLD servers, listed as NS or "name server" records.

Step 4: DNS Resolver Iterative Query to the TLD Server

Next the resolver queries one of the .com name servers for the location of google.com. Like the Root Servers, each of the TLDs have 4-13 clustered name servers existing in many locations.

There are two types of TLDs: country codes (ccTLDs) run by government organizations, and generic (gTLDs). Every gTLD has a different commercial entity responsible for running these servers. In this case, we will be using the gTLD servers controlled by Verisign, who run the .com, .net, .edu, and .gov among gTLDs.

Step 5: TLD Server Response

Each TLD server holds a list of all of the authoritative name servers for each domain in the TLD. For example, each of the 13 .com gTLD servers has a list with all of the name servers for every single .com domain.

The .com gTLD server does not have the IP addresses for google.com, but it knows the location of google.com's name servers. The .com gTLD server responds with a list of all of google.com's name servers, each represented as NS, or “name server” record.

Step 6: DNS Resolver Iterative Query to the Google.com NS

Finally, the DNS resolver queries one of Google's authoritative name server for the IP of “www.google.com.”

Step 7: Google.com NS Response

This time the queried Name Server knows the IPs and responds with an A or AAAA address record (depending on the query type) for IPv4 and IPv6, respectively.

Step 8: DNS Resolver Response to OS

At this point the resolver has finished the recursion process and is able to respond to the end user’s operating system with an IP address.

Step 9: Browser Starts TCP Handshake

At this point the operating system, now in possession of www.google.com's IP address, provides the IP to the Application (Browser), which initiates the TCP connection to start loading the page. For more information of this process, you can visit the Catchpoint blog to learn more about the anatomy of HTTP.[3]

As mentioned earlier, this is a worst case scenario. In most cases, if the user has recently accessed URLs of the same domain, or other users relying on the same DNS resolver have done such requests, there will be no DNS resolution required, or it will be limited to the query on the local DNS resolver. We will cover this in later articles.

In this DNS non-cached case, four servers were involved, hence a lot could have gone wrong. The end user has no idea what is happening behind the scenes; they are simply are waiting for the page to load and all of these DNS queries have to happen before the browser can request the webpage.

This is why we stress the importance of fast DNS. You can have a fast and well-built site, but if your DNS is slow, your webpage will still have poor response time.

Dissecting DNS Communications

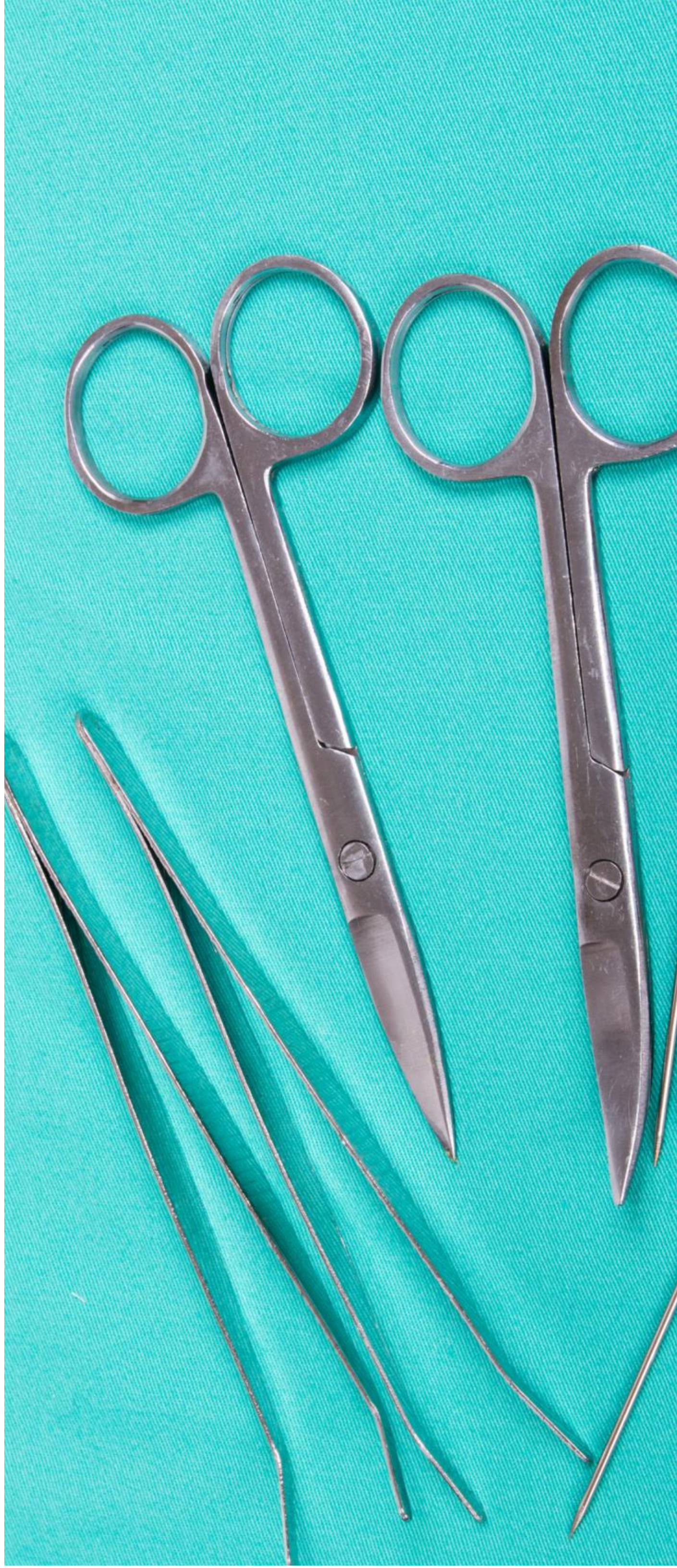
Now that you have an understanding of the recursion process, we need to delve into how the DNS protocol works before we go further on how DNS impacts performance.

In the TCP/IP Protocol suite,[4] DNS is an application layer protocol. DNS protocol relies on User Datagram Protocol (UDP) by default, but can also work over Transmission Control Protocol (TCP) as a fallback when firewalls block UDP.

Both UDP and TCP are transport layer protocols. UDP is lightweight protocol which does not require a handshake to establish connection, or confirmation of delivery, thereby reducing the number of packets required, time elapsed, and round trips. The biggest con of the UDP is that there are no assurances the packet was received by the other party, hence the application has to handle cases where no response is received.

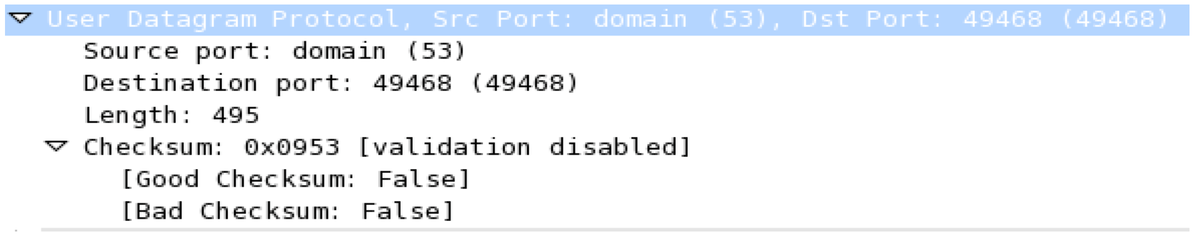
On the other side, TCP requires establishing a connection via a three-step handshake[5] and has delivery error checking, but requires more roundtrips and therefore more time.

Since we are covering TCP/IP protocols, we will go down to the bit-level in certain areas. But don’t get scared – you do not have to learn how to deal with “bit flags” to understand DNS. At Catchpoint we use – and heavily recommend using – a packet capture program such as Wireshark to make packets human readable and debugging easier.

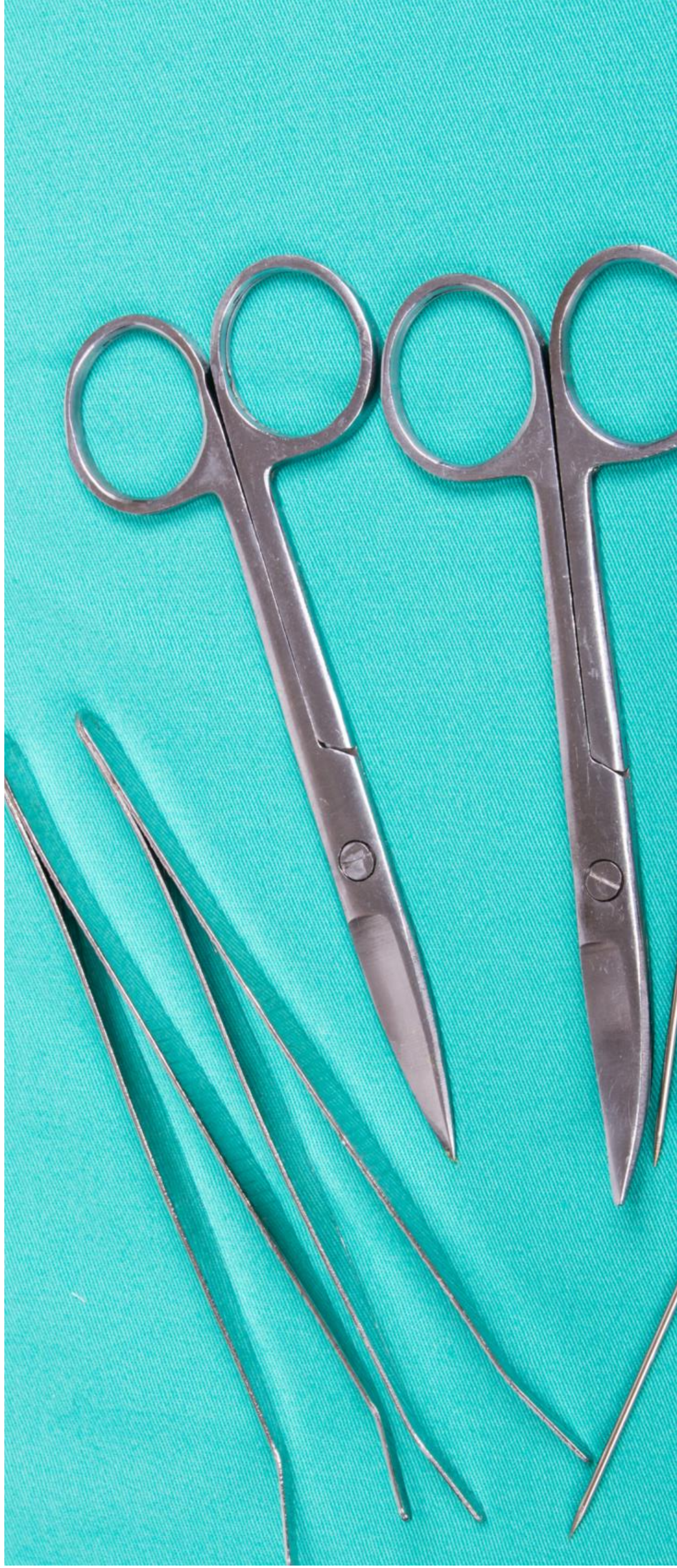


Protocol

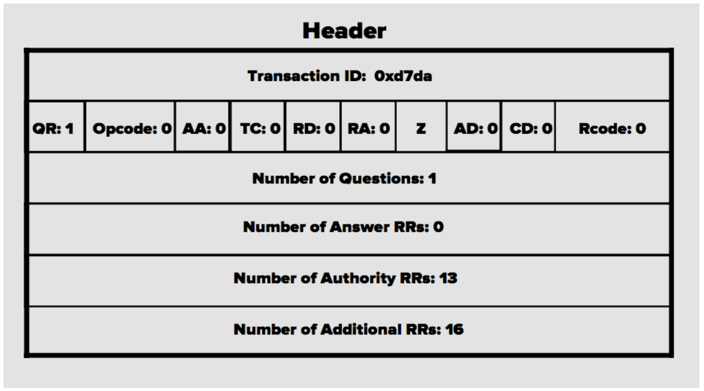
DNS protocol is composed of three types of messages queries, responses, and updates. We will not be covering "updates" in this book since it does not impact end-user experience. The DNS message can have five sections: the DNS Header, Question, Answer Resource Records, the Authority Resource Records, and the Additional Resource Records.



```
▽ User Datagram Protocol, Src Port: domain (53), Dst Port: 49468 (49468)
      Source port: domain (53)
      Destination port: 49468 (49468)
      Length: 495
   ▽ Checksum: 0x0953 [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
```



Header



Header

Transaction ID: 0xd7da									
QR: 1	Opcode: 0	AA: 0	TC: 0	RD: 0	RA: 0	Z	AD: 0	CD: 0	Rcode: 0
Number of Questions: 1									
Number of Answer RRs: 0									
Number of Authority RRs: 13									
Number of Additional RRs: 16									

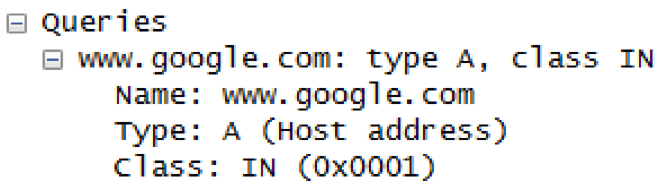
The header of the packet contains identifying information, along with hints (summaries) about what the rest of the message contains. The header is made up of six fields, sixteen bits each, for a total of twelve bytes. The first sixteen bits are for the Transaction ID, used to match the response to the query, and is created by the client on the query message and returned by the server in the response.

The next field is for flags. This is probably the most important part of the DNS packet, as these flags distinguish response from query, and iterative from recursive query. They are listed, in order:

Bit 1: QR, query/response flag. When 0, message is a query. When 1, message is response.
Bits 2-5: Opcode, operation code. Tells receiving machine the intent of the message. Generally 0 meaning normal query, however there are other valid options such as 1 for reverse query and 2 for server status.
Bit 6: AA, authoritative answer. Set only when the responding machine is the authoritative name server of the queried domain.
Bit 7: TC, truncated. Set if packet is larger than the UDP maximum size of 512 bytes.
Bit 8: RD, recursion desired. If 0, the query is an iterative query. If 1, the query is recursive. See our recursion chapter for more information.
Bit 9: RA, recursion available. Set on response if the server supports recursion.
Bit 10: Z. Reserved for future use, must be set to 0 on all queries and responses.
Bit 11: AD, authentic data. Used in DNSSEC. Considered part of Z in older machines.
Bit 12: CD, checking disabled. Used in DNSSEC. Considered part of Z in older machines.
Bits 13-16: Rcode, return code. It will generally be 0 for no error, if the name does not exist.

The remaining four header fields are number of questions, answer resource records, authority resource records, and additional resource records. These numbers vary depending on whether it is a query or response, and what kind of response. In general, however, there will always be at least one question.

Question



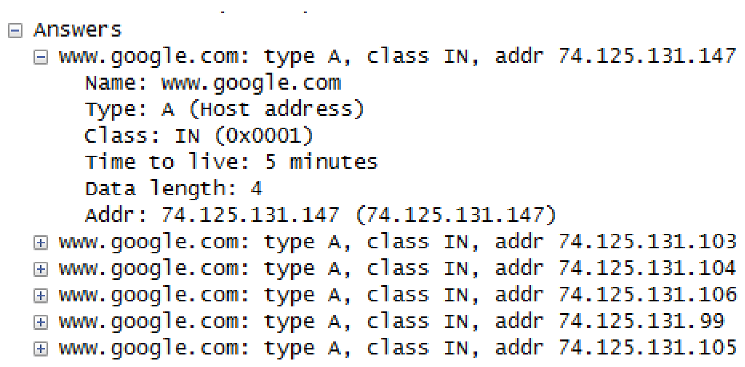
```
⊟ Queries
  ⊟ www.google.com: type A, class IN
      Name: www.google.com
      Type: A (Host address)
      Class: IN (0x0001)
```

The question is present in both the query and the response, and should be identical. Some tools, like Wireshark, call it “Query” (see image above).

In general there will only be one question, or query, per packet. The question is made up of three parts: the query name, which would be a host name such as www.google.com, a question type, and a question class, which is almost always 1 or IN for internet. The question type is the type of resource record. We listed several major types here, but there are more out there.[6]

A, IPv4 address: to which the domain name maps. Every website domain must have at least one A record, otherwise your end users won’t be able to access your website.
AAAA, Quad-A, IPv6 address record: The IPv6 address is the type to which the domain name maps. As IPv4 addresses are not available any longer, there is a big movement to support IPv6 – however not every ISP and website supports it yet.
MX, Mail eXchange record: Specifies which mail server accepts email messages on behalf of the recipient of the domain.
NS, Name Server record: Maps a domain to its authoritative name server. Every domain must have more than one NS record.

Answer Resource Records



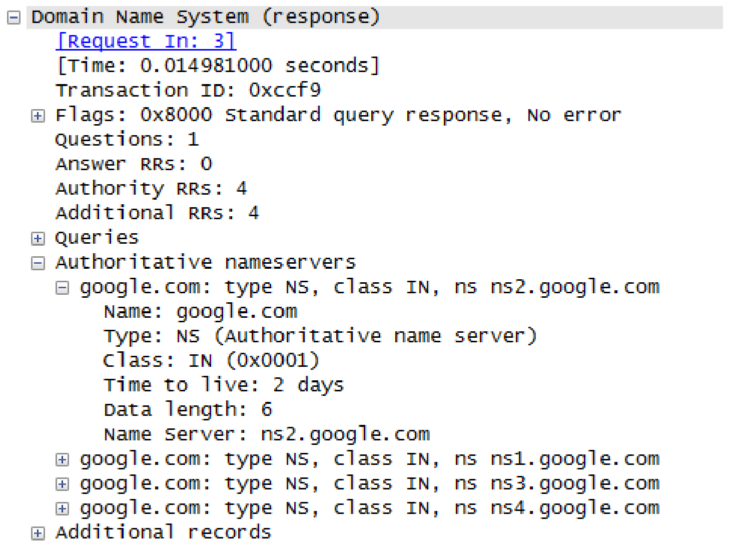
```
⊟ Answers
  ⊟ www.google.com: type A, class IN, addr 74.125.131.147
      Name: www.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 5 minutes
      Data length: 4
      Addr: 74.125.131.147 (74.125.131.147)
  ⊞ www.google.com: type A, class IN, addr 74.125.131.103
  ⊞ www.google.com: type A, class IN, addr 74.125.131.104
  ⊞ www.google.com: type A, class IN, addr 74.125.131.106
  ⊞ www.google.com: type A, class IN, addr 74.125.131.99
  ⊞ www.google.com: type A, class IN, addr 74.125.131.105
```

The answer consists of the resource records that answer the question. The “Answer” section is present on the response from recursive resolver to an end user’s computer, or in the response from the authoritative name server of the domain to the recursive resolver. It depends on the type of question, but for DNS resolution of web domains it will be A, AAAA, or CNAME. CNAME stands for Canonical Name, it means that the domain in the query is an alias for another domain.

The first three fields in the resource record are identical to the question format, and are RR name, RR type, and RR class. However, RRs contain three additional fields, a two-byte time-to-live value, a one byte data length field, and a data field. The data field contents vary depending on the type of record, but here are the major ones:

A: One data field storing a four-byte IPv4 address
AAAA: One data field storing a sixteen-byte IPv4 address
MX: Two data fields, one storing a priority value and one storing an IP address
NS: One data field storing the domain name of the authoritative name server
CNAME: One data field storing the domain name to which the question domain is alias

Authority Resource Records

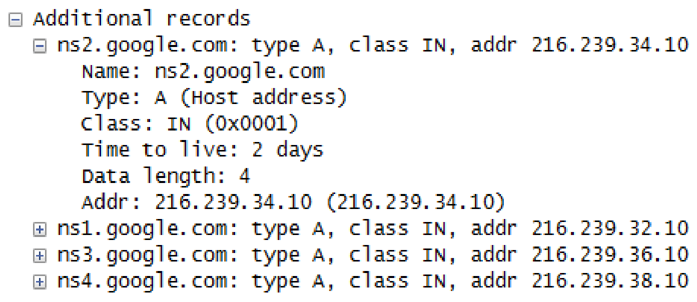


```
⊟ Domain Name System (response)
    [Request In: 3]
    [Time: 0.014981000 seconds]
    Transaction ID: 0xccf9
  ⊞ Flags: 0x8000 Standard query response, No error
    Questions: 1
    Answer RRs: 0
    Authority RRs: 4
    Additional RRs: 4
  ⊞ Queries
  ⊟ Authoritative nameservers
    ⊟ google.com: type NS, class IN, ns ns2.google.com
        Name: google.com
        Type: NS (Authoritative name server)
        Class: IN (0x0001)
        Time to live: 2 days
        Data length: 6
        Name Server: ns2.google.com
    ⊞ google.com: type NS, class IN, ns ns1.google.com
    ⊞ google.com: type NS, class IN, ns ns3.google.com
    ⊞ google.com: type NS, class IN, ns ns4.google.com
  ⊞ Additional records
```

When a name server like TLDs does not have the answer to the query (as is not authoritative), it will not send answer records. Instead it will populate the authority section with all of the name servers that it knows as authoritative to the domain or part of the domain tree (like .com), if it has them.

These NS records are different from A records, as they have a domain name in both the RR name and RR data fields. Unlike the answer section, the authority section can only have NS records. Note that NS records can be sent in other sections.

Additional Records



```
⊟ Additional records
  ⊟ ns2.google.com: type A, class IN, addr 216.239.34.10
      Name: ns2.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 2 days
      Data length: 4
      Addr: 216.239.34.10 (216.239.34.10)
  ⊞ ns1.google.com: type A, class IN, addr 216.239.32.10
  ⊞ ns3.google.com: type A, class IN, addr 216.239.36.10
  ⊞ ns4.google.com: type A, class IN, addr 216.239.38.10
```

Additional or "glue" records help avoid additional recursion by providing the IP addresses (A or AAAA records) of the NS records sent in the authoritative or answer section, so that those domains do not need to be resolved. These RRs are usually the A and/or AAAA records for the NS records in the authority section, although they can be any record.

Cache is King

There's no denying that the process for how DNS gets resolved (covered in the first chapter) is incredibly long and complicated.

Think about how many websites you visit in a given day, then consider how many of those you go to multiple times. Now imagine that every time you did, the ISP's DNS server at the other end had to repeat the entire recursion process from scratch and query all the name servers in the recursion chain.

To put that into context, think about your cell phone. When you want to make a call to a friend with whom you speak regularly, you simply go to recent calls and tap on their name.

But what if, instead of having that information readily available, you had to call 411 to get their phone number, then type it in manually. Seems pretty tedious, right?

The fact is that there are a lot of steps — and therefore a lot of time — required to change a domain name into an IP address.

Fortunately, the DNS architects had thought about how to speed up DNS and implemented caching. Caching allows any DNS server or client to locally store the records and re-use them in the future – eliminating the need for new queries.

The Domain Name System implements a time-to-live (TTL) on every DNS record. TTL specifies the number of seconds the record can be cached by a DNS client or server. When the record is stored in cache, whatever TTL value came with it gets stored as well.

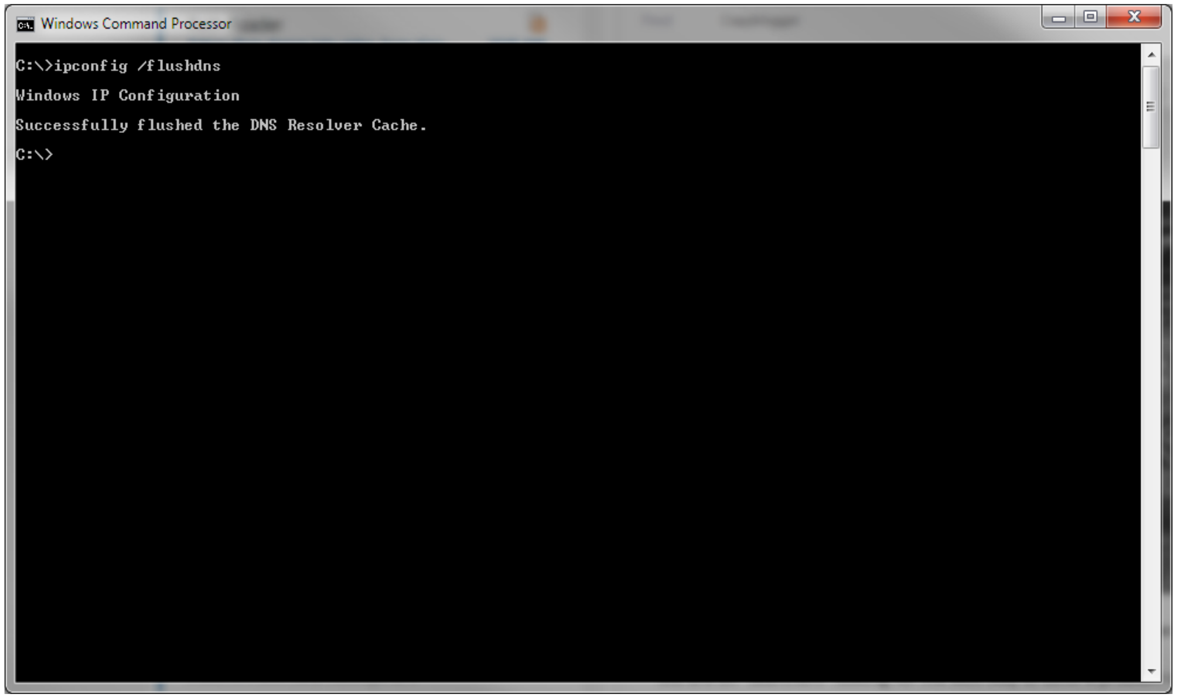
The server continues to update the TTL of the record stored in the cache, counting down every second. When it hits zero, the record is deleted or purged from the cache. At that point, if a query for that record is received, the DNS server has to start the resolution process.

To understand caching, let's look at the same example from the first chapter, resolving www.google.com. When you type www.google.com on the browser, the browser asks the Operating System for the IP address. The OS has what is known as a "stub resolver" or "DNS client," a simple resolver handling all the DNS lookups for the OS. The resolver will send queries (with recursive flag on) to a specified recursive resolver (name server) and stores the records in its cache based on their TTL.

When the "Stub Resolver" gets the request from the application it first looks in its cache, and if it has the record it gives the information to the application. If it does not have it, it sends a DNS query (with a recursive flag) to the recursive resolver (DNS server of your ISP).

When the "Recursive Resolver" gets the query it first looks in its cache to see what information it has for www.google.com. If it has the A records, it sends the records to the "Stub Resolver." If it does not have the A records, but has the NS records for the authoritative name servers, it will then query those name servers (bypassing the root and .com gTLD servers).

If it does not have the authoritative names servers, it will query the gTLD servers (the most likely are in cache since their TTL is very high, and they are used for any .com domain). The "Recursive Resolver" will query the root servers for the gTLDs only if they are not in the cache, which is quite rare (usually after a full purge).



```
C:\>ipconfig /flushdns
Windows IP Configuration
Successfully flushed the DNS Resolver Cache.
C:\>
```

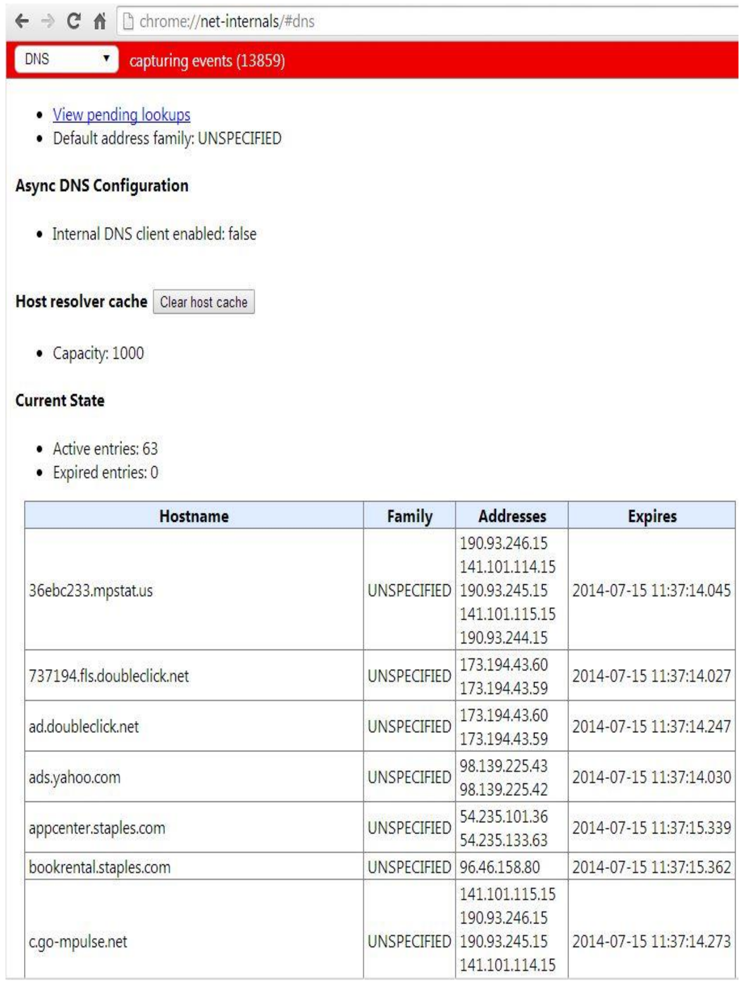
While you can't control the cache of the ISP's DNS server, you can manage the cache in your computer's "stub resolver." Troubleshooting DNS issues often involves flushing, or clearing, the DNS cache. To do this in Windows, simply open a command prompt and type **ipconfig /flushdns** and hit Enter. On iOS, you can do by opening a terminal window and typing **lookupd -flushcache**.

To prevent the propagation of expired DNS records, the DNS servers will pass the adjusted TTL to a query and not the original TTL value of the record. For example, let's assume that the TTL for an A record of www.google.com is four hours and it is stored in cache by the "Recursive Resolver" at 8 a.m. When a new user, on the same resolver, queries for the same domain at 9 a.m., the resolver will send an A record with a TTL of three hours.

So far we have covered how DNS caches on the OS and DNS Servers, however there is one last layer of cache: the application. Any application can choose to cache the DNS data, however they cannot follow the DNS specification. Applications rely on a OS function called "getaddrinfo()" to resolve a domain (all OS have the same function name). This function returns back the list of IP addresses for the domain – but it does not return DNS records, hence there is no TTL information that the application can use.

As a result, different applications cache the data for a specific period of time. IE10+ will store up to 256 domains in its cache for a fixed time of 30 minutes.[7] While 256 domains might seem like a lot, it is not – a lot of pages in the internet have more than 50 domains referenced thanks to third party tags and retargeting. Chrome, on the other hand, will cache the DNS information for one minute,[8] and stores up to 1,000 records.

You can view and clear the DNS cache of Chrome by visiting chrome://net-internals/#dns.



- View pending lookups
- Default address family: UNSPECIFIED

Async DNS Configuration

- Internal DNS client enabled: false

Host resolver cache Clear host cache

- Capacity: 1000

Current State

- Active entries: 63
- Expired entries: 0

Hostname	Family	Addresses	Expires
36ebc233.mpstat.us	UNSPECIFIED	190.93.246.15 141.101.114.15 190.93.245.15 141.101.115.15 190.93.244.15	2014-07-15 11:37:14.045
737194.fls.doubleclick.net	UNSPECIFIED	173.194.43.60 173.194.43.59	2014-07-15 11:37:14.027
ad.doubleclick.net	UNSPECIFIED	173.194.43.60 173.194.43.59	2014-07-15 11:37:14.247
ads.yahoo.com	UNSPECIFIED	98.139.225.43 98.139.225.42	2014-07-15 11:37:14.030
appcenter.staples.com	UNSPECIFIED	54.235.101.36 54.235.133.63	2014-07-15 11:37:15.339
bookrental.staples.com	UNSPECIFIED	96.46.158.80	2014-07-15 11:37:15.362
c.go-mpulse.net	UNSPECIFIED	141.101.115.15 190.93.246.15 190.93.245.15 141.101.114.15	2014-07-15 11:37:14.273

NS Cache Trap

One major trap that people fall into with DNS cache is the authoritative name server records. As we mentioned before, the authoritative name servers are specified in the query response as NS records. NS records have a TTL, but do not provide the IP addresses of the name servers. The IP information is in the additional records of the response and are A or AAAA records.

Therefore, a Recursive Resolver relies on both NS and A records to reach the name server. Ideally, the TTL on both types of records should be the same, but every once in a while someone will misconfigure their DNS zones and they pass in DNS query responses for the domain new A or AAAA records for the authoritative name servers with lower or higher TTL than what was specified in the TLDs. These new records override the old records, causing a discrepancy.

When both records are in cache, the “Recursive Resolver” will query one of the IPs of the name servers. If the “Recursive Resolver” has only the NS records in the cache, no A or AAAA records, it will have to resolve the domain off the authoritative name server, so ns1.google.com, to get the IP address for it.

This is not good, as it adds time to resolving the domain in question. And if it has the A or AAAA records of the name server but not the NS records, it will be forced to do a DNS lookup for www.google.com.

Setting TTL: A Balancing Act

So what's better, a longer or shorter TTL? When appropriate, use a longer TTL, as it leads to longer caching on resolvers and OSs – meaning better performance for end users – and it also reduces traffic to your name servers as they will be queried less often.

However, it also reduces your ability to make DNS changes, leaving you more vulnerable to DNS poisoning attacks and unable to set up offsite error pages when your datacenter is not accessible.

On the other hand, a shorter TTL limits caching and will add time to downloading the page and/or resources, while raising the stress on your name servers. Yet they let you make quicker changes to your DNS configuration.

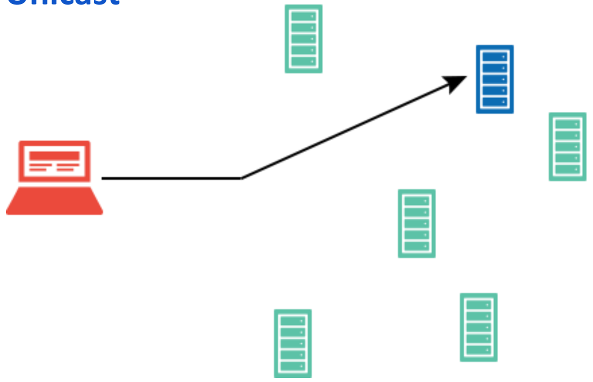
DNS resolution is a multi-step process that involves a lot of servers over the internet. Caching mechanism built in the protocol speeds up the process by storing information for periods of time and re-using it for future DNS queries.

While DNS servers and/or clients do follow the DNS specs on TTL, application like browser do not follow the spec – hence their cache is stored for an arbitrary amount of time.

Unicast vs. Anycast

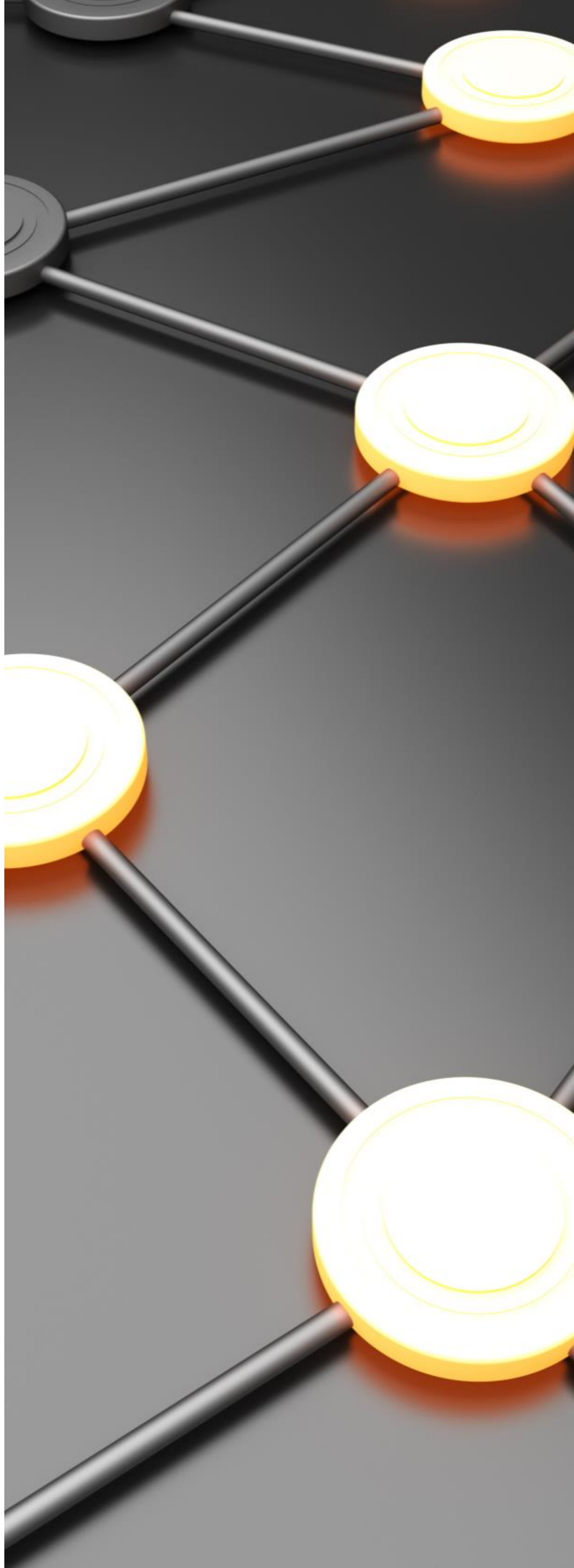
Let's now look at Anycast and its most prevalent counterpart, Unicast, to see how it speeds up DNS resolution times.

Unicast

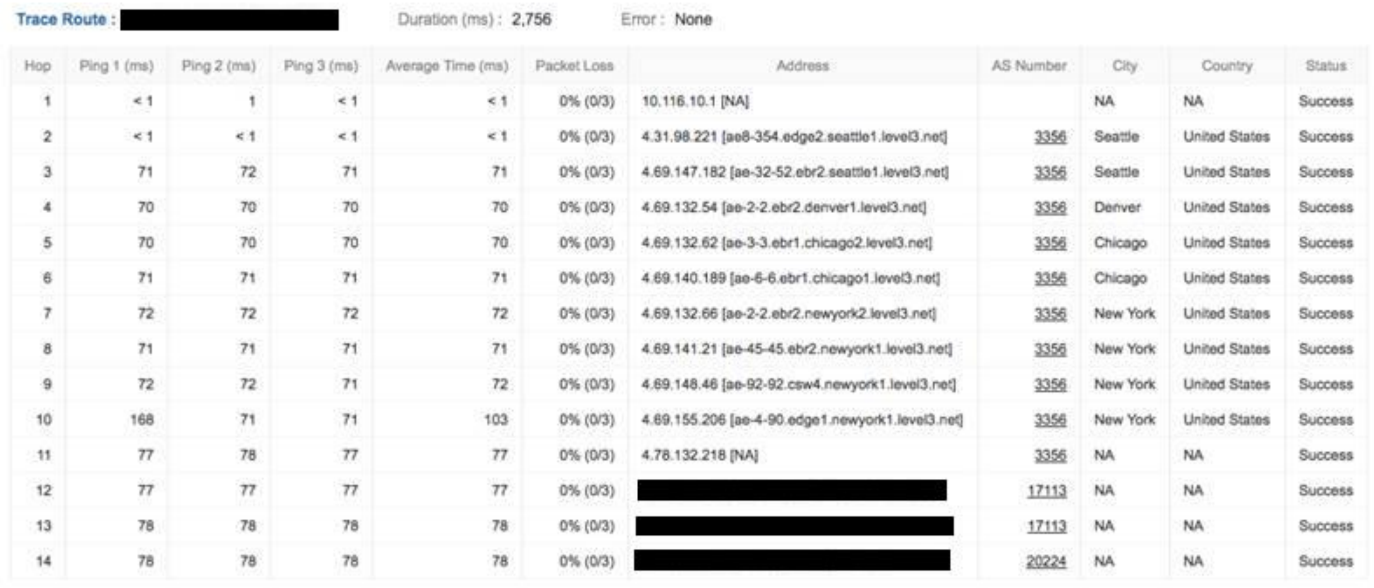


Almost all of the Internet and the technologies built on top of it rely on a routing scheme called Unicast. With Unicast, every server (or cluster of servers behind a load balancer) has a unique IP address, which any device in the world can rely on to communicate with the server. If the client device is topologically close to the server, the latency will be small and the performance of the application will therefore be perceived as fast. But if the distance between the two is large, there will be significant slowness due to network latency.

When using Unicast for the domain name servers, you can have multiple physical servers around the globe with different IPs and rely on the end user's resolver logic to reach the closest name server. The resolver DNS servers can implement special logic that measures and stores the average round trip time (RTT) of the name severs they query, and then favor those servers that have the fastest RTT. However, there is still no guarantee that the user's DNS resolution will be fast. Some queries will still go to nameservers that are not close to the resolver, and not all resolvers have this feature enabled or support it.

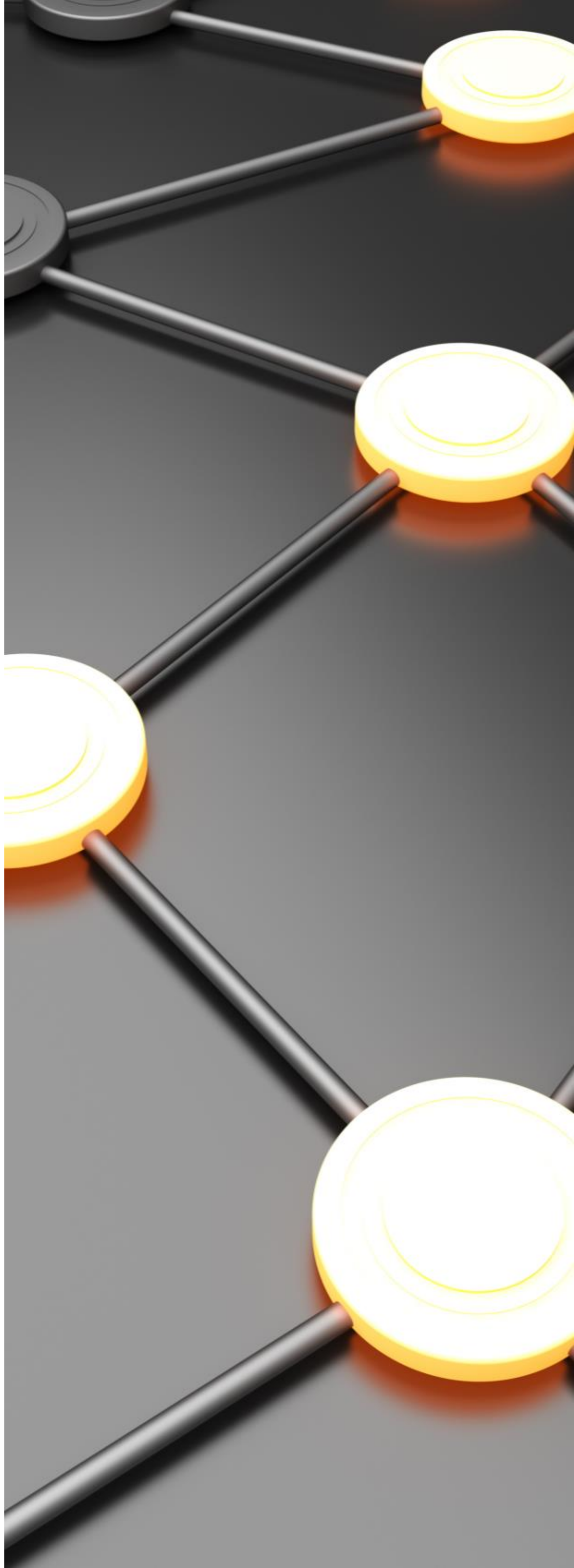


In this example of a traceroute from our Seattle Level 3 location to a Unicast IP address, we see that it has to travel all the way to its home server in New York, adding extra latency:

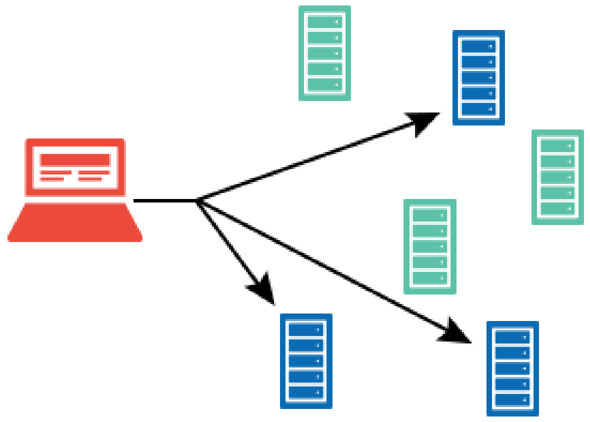


Trace Route : [redacted] Duration (ms) : 2,756 Error : None

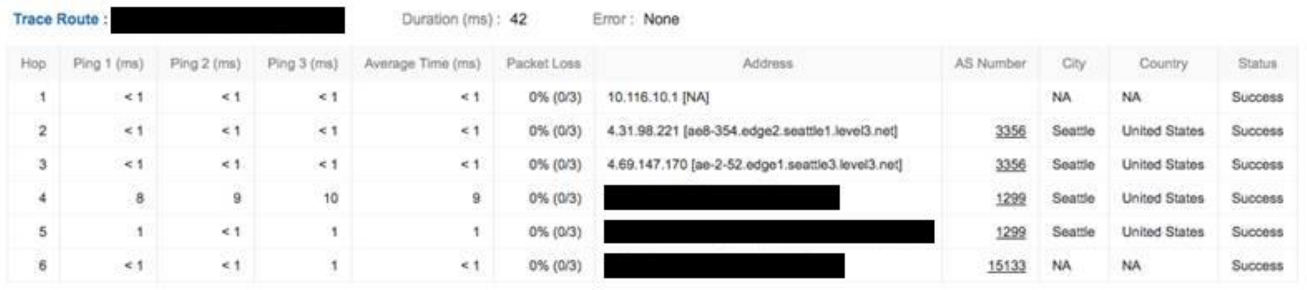
Hop	Ping 1 (ms)	Ping 2 (ms)	Ping 3 (ms)	Average Time (ms)	Packet Loss	Address	AS Number	City	Country	Status
1	< 1	1	< 1	< 1	0% (0/3)	10.116.10.1 [NA]		NA	NA	Success
2	< 1	< 1	< 1	< 1	0% (0/3)	4.31.98.221 [ae8-354.edge2.seattle1.level3.net]	3356	Seattle	United States	Success
3	71	72	71	71	0% (0/3)	4.69.147.182 [ae-32-52.ebr2.seattle1.level3.net]	3356	Seattle	United States	Success
4	70	70	70	70	0% (0/3)	4.69.132.54 [ae-2-2.ebr2.denver1.level3.net]	3356	Denver	United States	Success
5	70	70	70	70	0% (0/3)	4.69.132.62 [ae-3-3.ebr1.chicago2.level3.net]	3356	Chicago	United States	Success
6	71	71	71	71	0% (0/3)	4.69.140.189 [ae-6-6.ebr1.chicago1.level3.net]	3356	Chicago	United States	Success
7	72	72	72	72	0% (0/3)	4.69.132.66 [ae-2-2.ebr2.newyork2.level3.net]	3356	New York	United States	Success
8	71	71	71	71	0% (0/3)	4.69.141.21 [ae-45-45.ebr2.newyork1.level3.net]	3356	New York	United States	Success
9	72	72	71	72	0% (0/3)	4.69.148.46 [ae-92-92.csw4.newyork1.level3.net]	3356	New York	United States	Success
10	168	71	71	103	0% (0/3)	4.69.155.206 [ae-4-90.edge1.newyork1.level3.net]	3356	New York	United States	Success
11	77	78	77	77	0% (0/3)	4.78.132.218 [NA]	3356	NA	NA	Success
12	77	77	77	77	0% (0/3)	[redacted]	17113	NA	NA	Success
13	78	78	78	78	0% (0/3)	[redacted]	17113	NA	NA	Success
14	78	78	78	78	0% (0/3)	[redacted]	20224	NA	NA	Success



Anycast

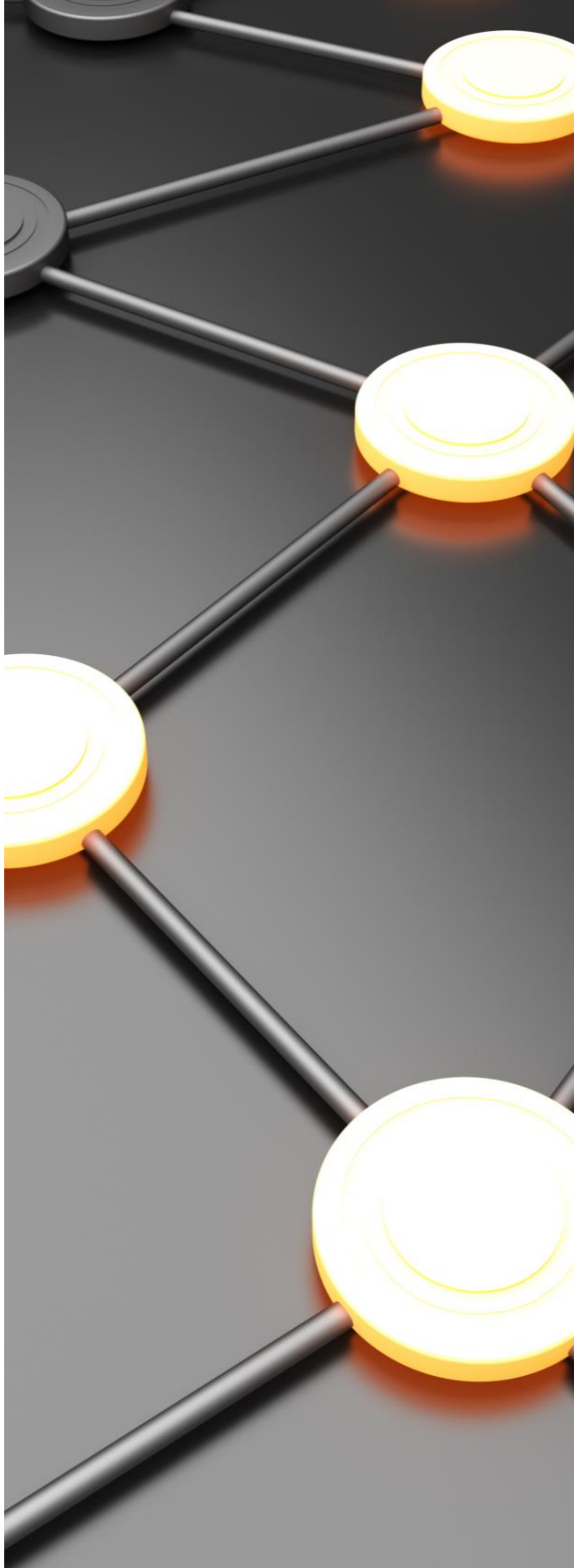


Anycast is a routing scheme alternative to Unicast. Using Border Gateway Protocol (BGP), a network operator can announce the same IP address range from various locations dispersed throughout the world. This causes routers to direct traffic targeting the IP range to the nearest location that announced it. When we run our traceroute from Seattle, this time to an Anycast IP address, the number of hops, RTT, and latency are all lower:



Trace Route : [redacted] Duration (ms) : 42 Error : None

Hop	Ping 1 (ms)	Ping 2 (ms)	Ping 3 (ms)	Average Time (ms)	Packet Loss	Address	AS Number	City	Country	Status
1	< 1	< 1	< 1	< 1	0% (0/3)	10.116.10.1 [NA]		NA	NA	Success
2	< 1	< 1	< 1	< 1	0% (0/3)	4.31.98.221 [ae8-354.edge2.seattle1.level3.net]	3356	Seattle	United States	Success
3	< 1	< 1	< 1	< 1	0% (0/3)	4.69.147.170 [ae-2-52.edge1.seattle3.level3.net]	3356	Seattle	United States	Success
4	8	9	10	9	0% (0/3)	[redacted]	1299	Seattle	United States	Success
5	1	< 1	1	1	0% (0/3)	[redacted]	1299	Seattle	United States	Success
6	< 1	< 1	1	< 1	0% (0/3)	[redacted]	15133	NA	NA	Success

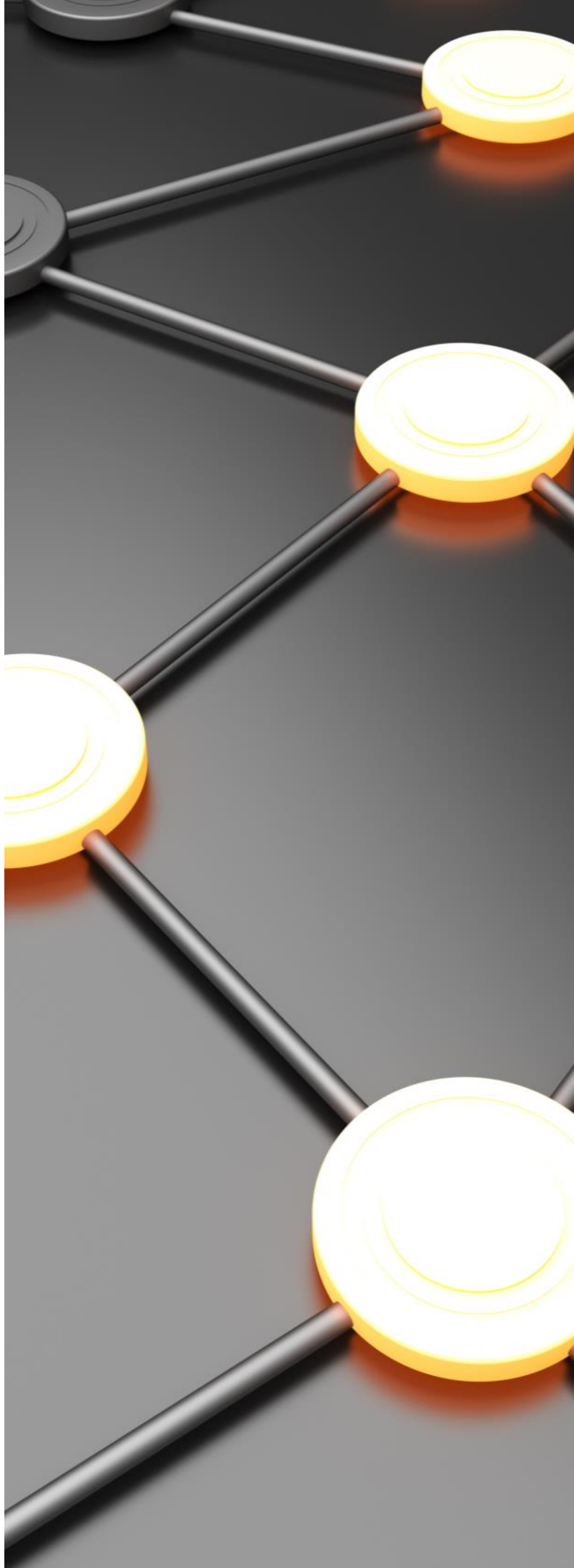


There is one catch with Anycast, and that is the definition of “nearest.” Anycast will route to the closest server as defined topologically in the network by smallest number of hops. Usually there is a direct correlation between what is close topologically and what is close geographically – but this does not mean it is always the case or that the latency will be the lowest.

The mechanisms behind the route scheme do not take latency or geography into consideration. This means that if you are in Atlanta and the closest Anycast server in terms of hops is in Dallas, but the closest geographically is in Ashburn, you will still be routed to Dallas even though the RTT will be greater.

Besides increasing performance, Anycast has several other benefits:

- Acts as a layer 3 load balancer between multiple points of presence with automatic failover.
- Increases reliability by adding redundancy via the placement of multiple points of presence dispersed geographically
- Builds resilience against DDoS attacks, as it is harder to launch such attacks against multiple network locations that are dispersed
- Limits impacts of attacks to specific locations, as opposed to the entire system

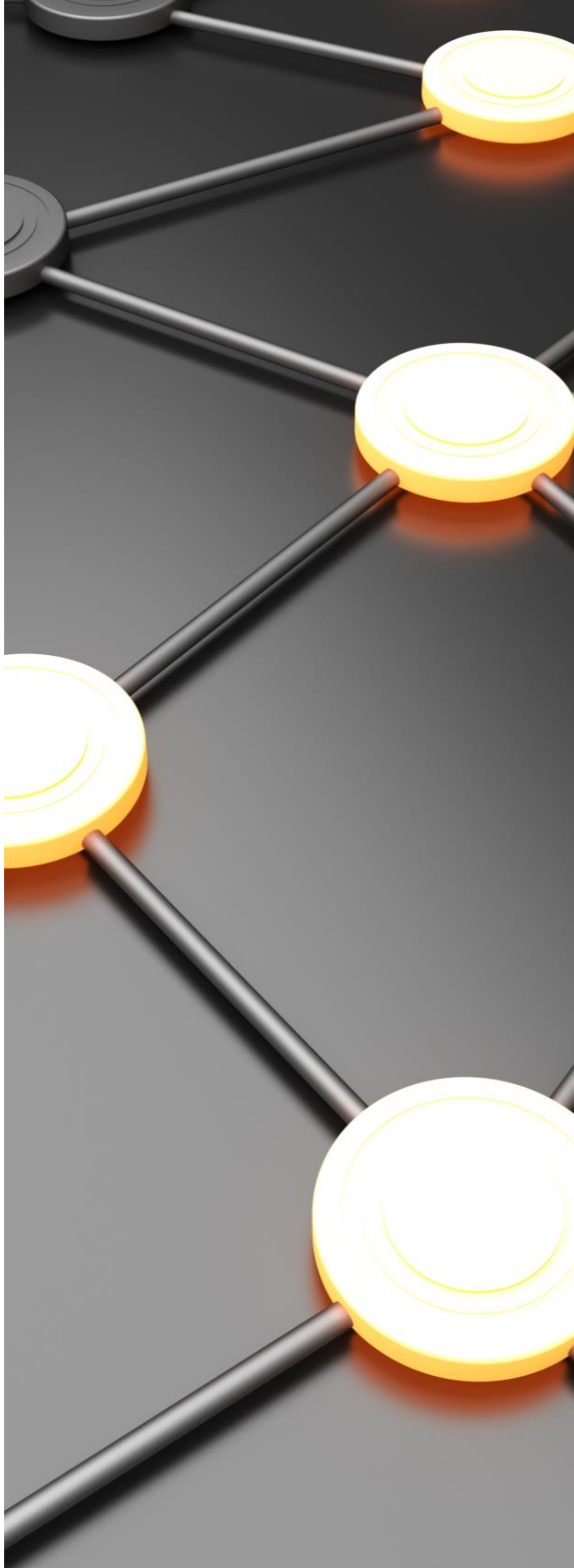


Anycast Uses

Anycast is primarily used for applications relying on the UDP protocol, like DNS. Anycast DNS has already been deployed by 12 of the 13 DNS root servers, and most major DNS vendors such as Cloudflare, Dyn, Edgecast, Verisign, and others.

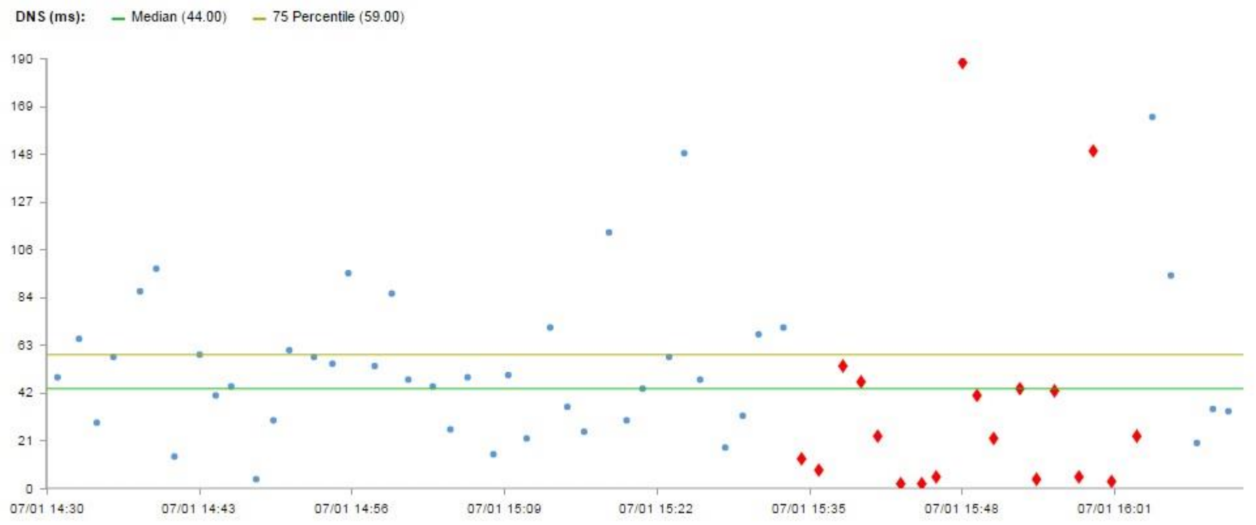
Anycast for TCP protocol is more difficult since the connections must maintain state and can have log keep alives. However, more and more CDNs are providing Anycast solutions for HTTP.

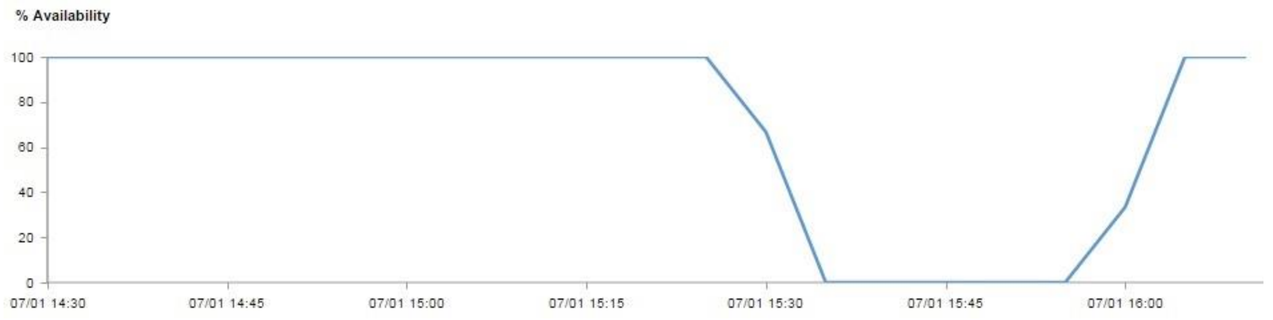
Although Anycast is more difficult to set up and troubleshoot in comparison to Unicast, Anycast DNS and HTTP services can help turn a fast site into a blazing fast site, thereby improving your end users' experience and satisfying one of the most crucial aspects of your site's marketing initiative.



DNS Failures

By now, it should be clear that a failure or delay on the part of your DNS server will cause your site to experience slow load times, or in a worst case scenario, render it completely unavailable.





Most websites rely on third parties to handle their DNS needs – either through domain register DNS servers or managed DNS providers – in order to avoid the costs and hassle of building and managing their own DNS servers. Putting it in the hands of third party experts is a smart strategy, as it generally brings down costs and improves reliability. Yet like all third party providers, their performance is ultimately out of your hands, and sometimes they will fail even if all the necessary precautions are taken.

As with all of your third party vendors, you should have Service Level Agreements (SLAs) with your DNS provider in order to protect you from the financial ramifications of a failure that you had nothing to do with.

Sadly, backup strategies for DNS are very expensive, because the only ways to fully handle a failure is to have redundant service (or your own infrastructure) working side by side with your primary ones. Obviously both of these options are cost-prohibitive for most companies, which puts even more importance on your SLA, because you have to be able to recoup revenue that might be lost in any outage.

The challenge comes from the fact that managed DNS providers often have hundreds of servers in multiple points of presence (POPs) throughout the world, which means that a “micro-outage” could be localized to a small geographic area and go undetected, yet still have a sizable impact on your site’s performance.

Therefore, it’s essential to have an effective synthetic monitoring strategy that will detect and alert you of a problem with either failures or slowness in your DNS server(s). Meanwhile, Real User Measurement will give clearer insight into how your DNS performance is affecting your end user experience, though it’s important to note that RUM will not provide insight into your DNS availability – only synthetic can do that.

References

1 IANA Root Servers

2 Root-Servers.org

3 Anatomy of an HTTP Transaction

4 Internet Protocol Suite

5 Understanding RTT Impact on TCP Retransmissions

6 DNS Record Types

7 Braindump: DNS

8 Host Resolution in Chromium

Synthetic Monitoring

Catchpoint Synthetic Monitoring is an intuitive tool that collects the right data in order for you to efficiently and accurately monitor the performance of your website and applications. Our infrastructure is built with the largest global node coverage in the industry, including over 310 Backbone nodes in 62 countries, all of which are located in Tier 1 ISPs and datacenters. We offer testing monitors that expand beyond web, like mobile, HTML code, transaction, API, DNS, TCP, and more.

Once the data is collected, you can choose from a variety of charting options to dissect and analyze according to the metrics that mean the most to your business.

Catchpoint Synthetic Monitoring offers clear, comprehensive visibility into your performance so that you can troubleshoot more efficiently and deliver exceptional user experiences.

Catchpoint Glimpse

Real User Measurement

As Catchpoint's Real User Measurement (RUM) tool, Glimpse delivers a clear view of actual user experience data so you can accurately assess how your performance – including that of your DNS – is affecting your business's bottom line.

Glimpse offers a variety of powerful features, including the Data Explorer. This allows you to examine a finite time period and analyze data on a more granular level. Data Explorer can perform multi-dimensional analyses on raw data using 20 different data attributes and three dimensions. In addition to averages, several statistical values are available within this feature, like standard deviation, 75th percentile, 95th percentile, and more.

You also have the ability to investigate the correlations between page speed and key business indicators using the Engagement Reporting feature.

Glimpse takes RUM further by collecting actual user experience data and combining it with an innovative analytics engine to make performance-based business decisions easier than ever.

catchpoint®

150 W 30th Street, 3rd Floor
New York, NY 10001

(877) 240 4450
www.catchpoint.com

About Catchpoint

Catchpoint radically transforms the way businesses manage, monitor, and test the performance of online applications. Truly understand and improve user experience with clear visibility into complex, distributed online systems.

Founded in 2008 by four DoubleClick / Google executives with a passion for speed, reliability and overall better online experiences, Catchpoint has now become the most innovative provider of web performance testing and monitoring solutions. We are a team with expertise in designing, building, operating, scaling and monitoring highly transactional Internet services used by thousands of companies and impacting the experience of millions of users. Catchpoint is funded by top-tier venture capital firm, Battery Ventures, which has invested in category leaders such as Akamai, Omniture (Adobe Systems), Optimizely, Tealium, BazaarVoice, Marketo and many more.

catchpoint®

THE POWER TO SEE DEEPER

SMARTER, FASTER PERFORMANCE ANALYTICS

Sign up for a Free Trial today: **catchpoint.com/freetrial**