

Class Meeting: Lectures 13 and 14 DNS (+Security), Anycast



Kyle Jamieson
COS 461: Computer Networks

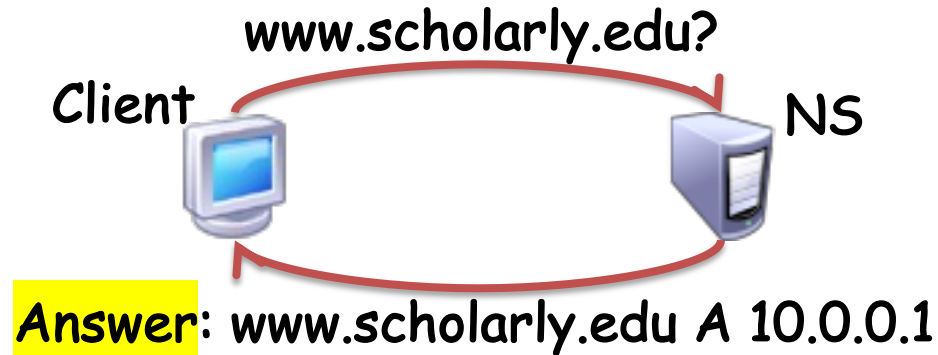
Today

1. Review of Domain Name System (DNS)
2. DNS security

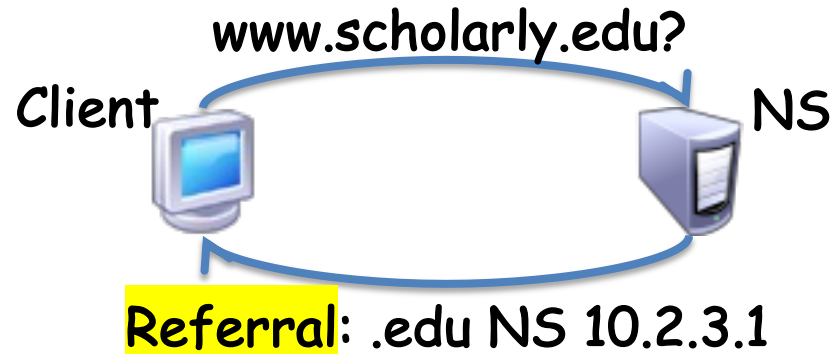
DNS in operation

- Most queries and responses are UDP datagrams
- Two types of queries:

- **Recursive:**



- **Iterative:**



A recursive DNS lookup (simplified, without local nameserver)

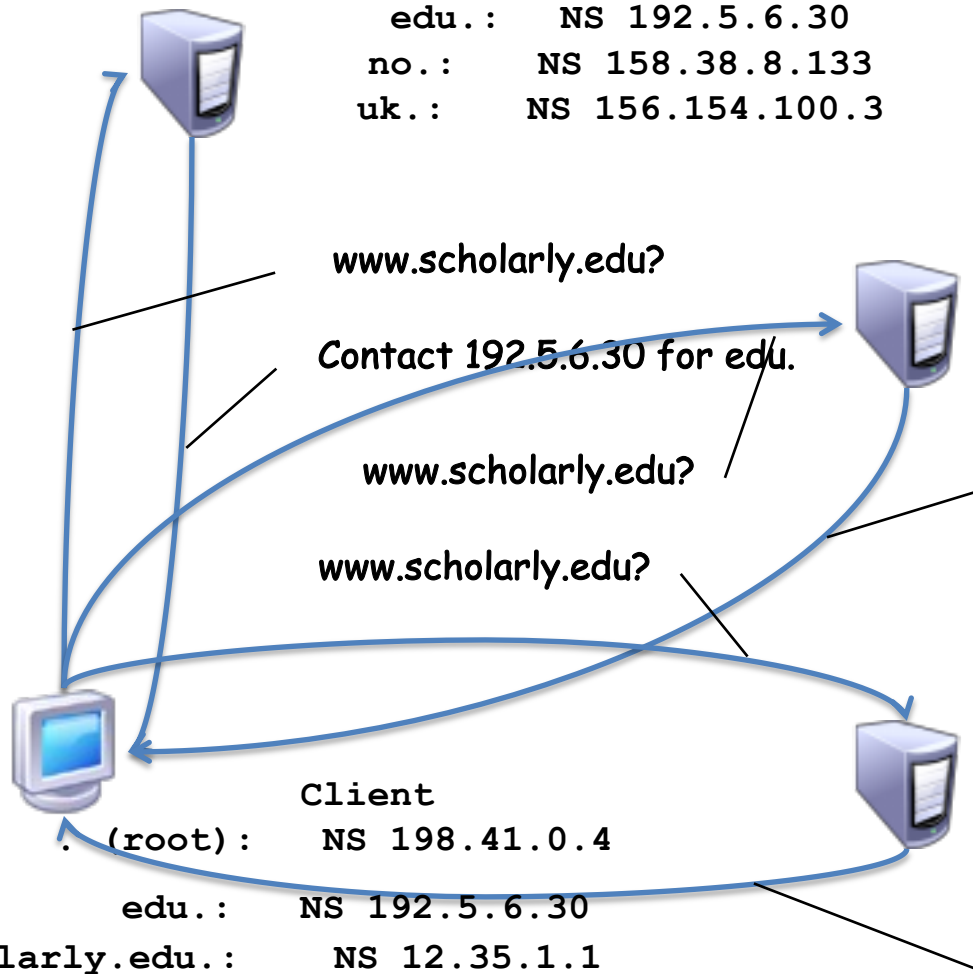
```
. (root) authority 198.41.0.4
  edu.:  NS 192.5.6.30
  no.:   NS 158.38.8.133
  uk.:   NS 156.154.100.3
```

```
edu. authority 192.5.6.30
scholarly.edu.: NS 12.35.1.1
pedantic.edu.:  NS 19.31.1.1
```

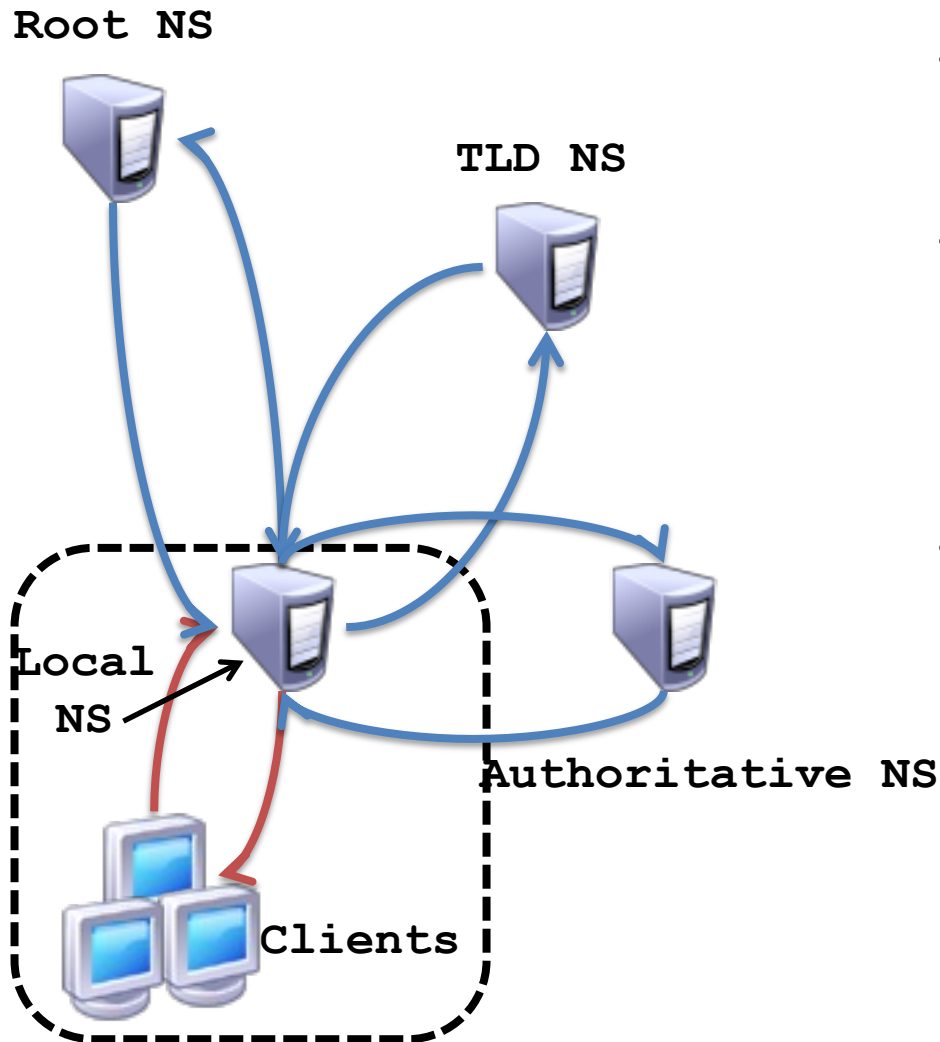
Contact 12.35.1.1 for scholarly.edu.

```
scholarly.edu. authority 12.35.1.1
www.scholarly.edu.: A 12.35.2.30
imap.scholarly.edu.: A 12.35.2.31
```

www.scholarly.edu.: A 12.35.51.30



Local Name server performs iterative query work on behalf of clients



- Client's resolver makes a **recursive** query to local NS
- Local NS processing:
 - Local NS sends **iterative** queries to other NS's
 - or, finds answer in cache
- Local NS responds with an answer to the client's request

Recursive versus iterative queries

Recursive query

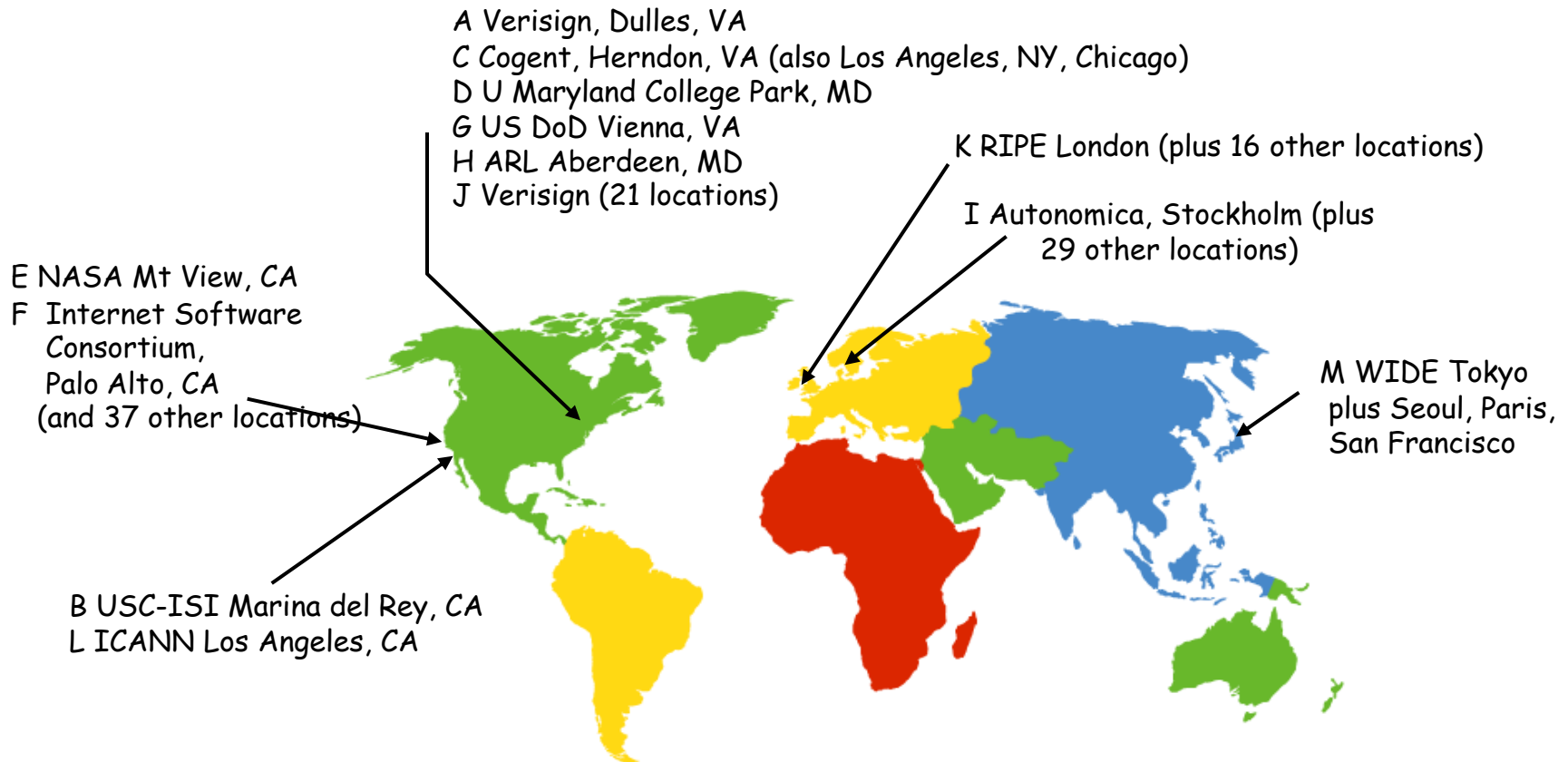
- Less burden on client
- **More burden on nameserver** (has to return an answer to the query)
- Most root and TLD servers will not answer (shed load)
 - Local name server answers recursive query

Iterative query

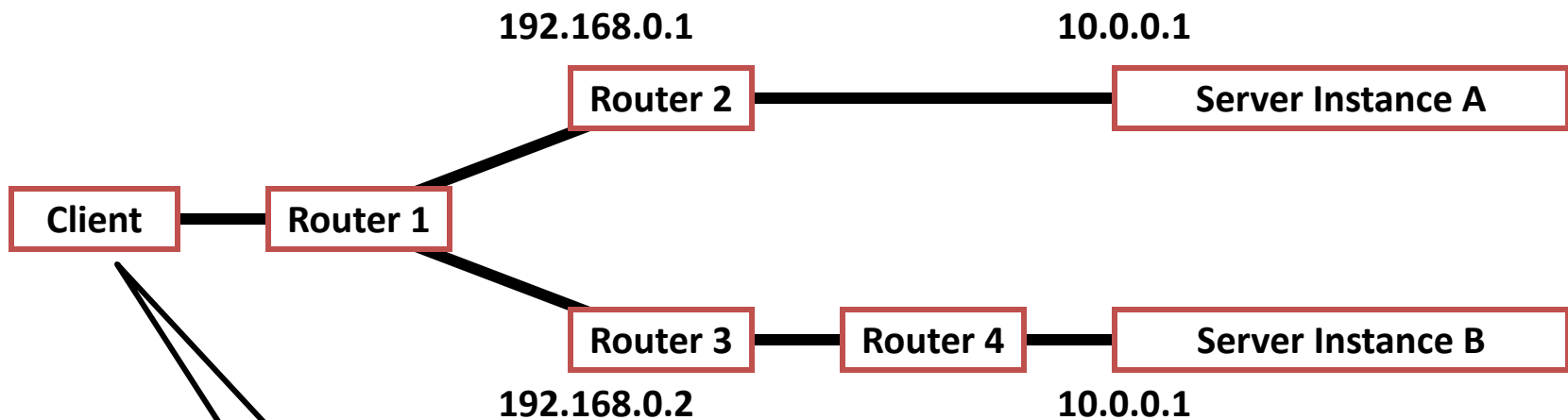
- **More burden on client**
- Less burden on nameserver (simply refers the query to another server)

DNS root nameservers

- 13 root servers (see <http://www.root-servers.org>)
 - Labeled A through M
- Each server is really a cluster of servers (some geographically distributed), replication via **IP anycast**



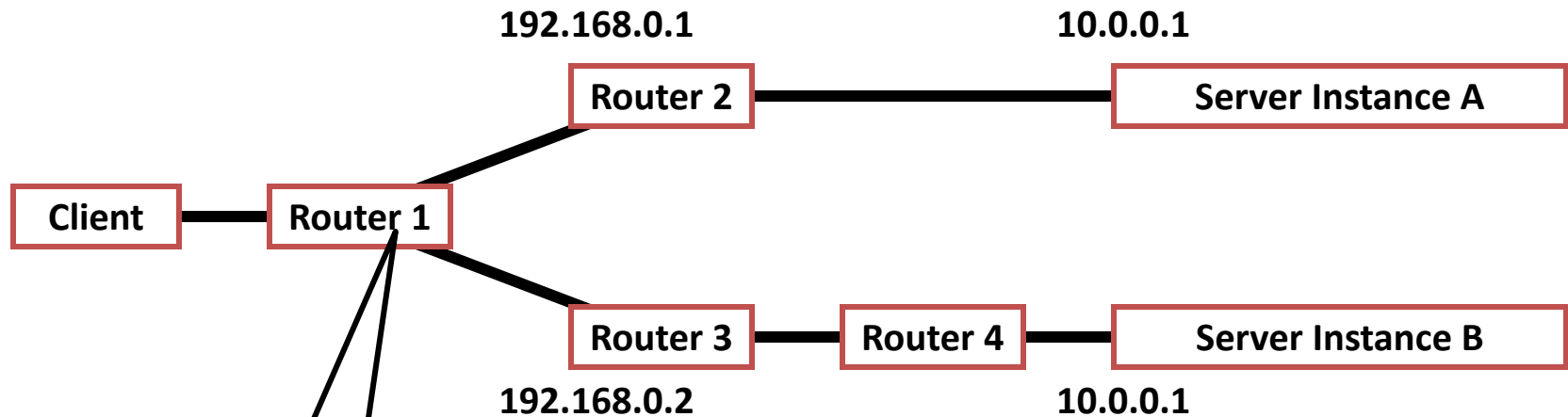
IP anycast in action



DNS lookup for `http://www.server.com/`
produces a single answer:

```
www.server.com. IN A 10.0.0.1
```


IP anycast in action



Routing Table from Router 1:

Destination	Mask	Next-Hop	Distance
192.168.0.0	/29	127.0.0.1	0
10.0.0.1	/32	192.168.0.1	1
10.0.0.1	/32	192.168.0.2	2

DNS resource record (RR): Overview

DNS is a distributed database storing resource records

RR includes: (name, type, value, time-to-live)

- **Type = A (address)**
 - name is hostname
 - value is IP address
- **Type = NS (name server)**
 - name is domain (e.g. cs.Princeton.edu)
 - value is hostname of authoritative name server for this domain
- **Type = CNAME**
 - name is an alias for some "canonical" (real) name
 - e.g. www.cs.Princeton.edu is really www-server.cs.Princeton.edu
 - value is canonical name
- **Type = MX (mail exchange)**
 - value is name of mail server associated with domain name
 - pref field discriminates between multiple MX records

Example: A real recursive query (1/3)

```
$ dig @a.root-servers.net www.freebsd.org +norecurse
; <<>> DiG 9.4.3-P3 <<>> @a.root-servers.net www.freebsd.org
+norecurse
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 57494
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 12

;; QUESTION SECTION:
;www.freebsd.org.      IN  A

;; AUTHORITY SECTION:
org.      172800 IN  NS  b0.org.afiliias-nst.org.
org.      172800 IN  NS  d0.org.afiliias-nst.org.

;; ADDITIONAL SECTION:
b0.org.afiliias-nst.org. 172800 IN  A  199.19.54.1
d0.org.afiliias-nst.org. 172800 IN  A  199.19.57.1

;; Query time: 177 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Wed Oct 28 07:32:02 2009
;; MSG SIZE rcvd: 435
```

"Glue" record

Example: A real recursive query (2/3)

```
$ dig @199.19.54 1 www.freebsd.org +norecurse
; <<>> Dig 9.4.3-P3 <<>> @a0.org.afilias-nst.org
  www.freebsd.org +norecurse
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39912
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 0

;; QUESTION SECTION:
;www.freebsd.org.      IN  A

:: AUTHORITY SECTION:
freebsd.org.      86400  IN  NS  ns1.isc-sns.net.
freebsd.org.      86400  IN  NS  ns2.isc-sns.com.
freebsd.org.      86400  IN  NS  ns3.isc-sns.info.

;; Query time: 128 msec
;; SERVER: 199.19.56.1#53(199.19.56.1)
;; WHEN: Wed Oct 28 07:38:40 2009
;; MSG SIZE rcvd: 121
```

No glue record provided for ns1.isc-sns.net, so need to go off and resolve (not shown here), then restart the query

Example: A real recursive query (3/3)

```
$ dig @ns1.isc-sns.net www.freebsd.org +norecurse
; <<>> DIG 9.4.3-P3 <<>> @ns1.isc-sns.net www.freebsd.org
+norecurse
; (1 server found)
;; global options:      printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17037
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 3,
ADDITIONAL: 5

;; QUESTION SECTION:
;www.freebsd.org.      IN A

;; ANSWER SECTION:
www.freebsd.org.      3600   IN A    69.147.83.33

;; AUTHORITY SECTION:
freebsd.org.         3600   IN NS   ns2.isc-sns.com.
freebsd.org.         3600   IN NS   ns1.isc-sns.net.
freebsd.org.         3600   IN NS   ns3.isc-sns.info.

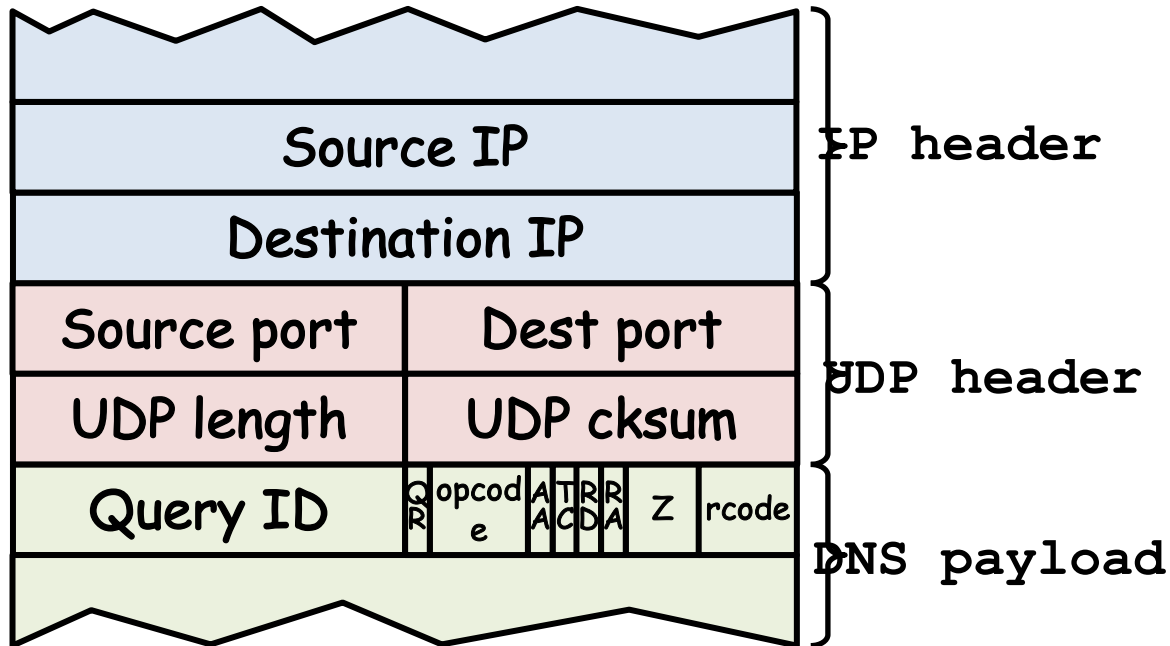
;; ADDITIONAL SECTION:
ns1.isc-sns.net.     3600   IN A    72.52.71.1
ns2.isc-sns.com.     3600   IN A    38.103.2.1
ns3.isc-sns.info.    3600   IN A    63.243.194.1
```

DNS Caching

- Performing all these queries takes time
 - And all this **before** actual communication takes place
 - *e.g.*, one-second latency before starting Web download
- **Caching** can greatly reduce overhead
 - The top-level servers very rarely change
 - Popular sites (*e.g.*, www.cnn.com) visited often
 - Local DNS server often has the information cached
- How DNS caching works
 - DNS servers cache responses to queries
 - Responses include a **time-to-live** (TTL) field
 - Server deletes cached entry after TTL expires

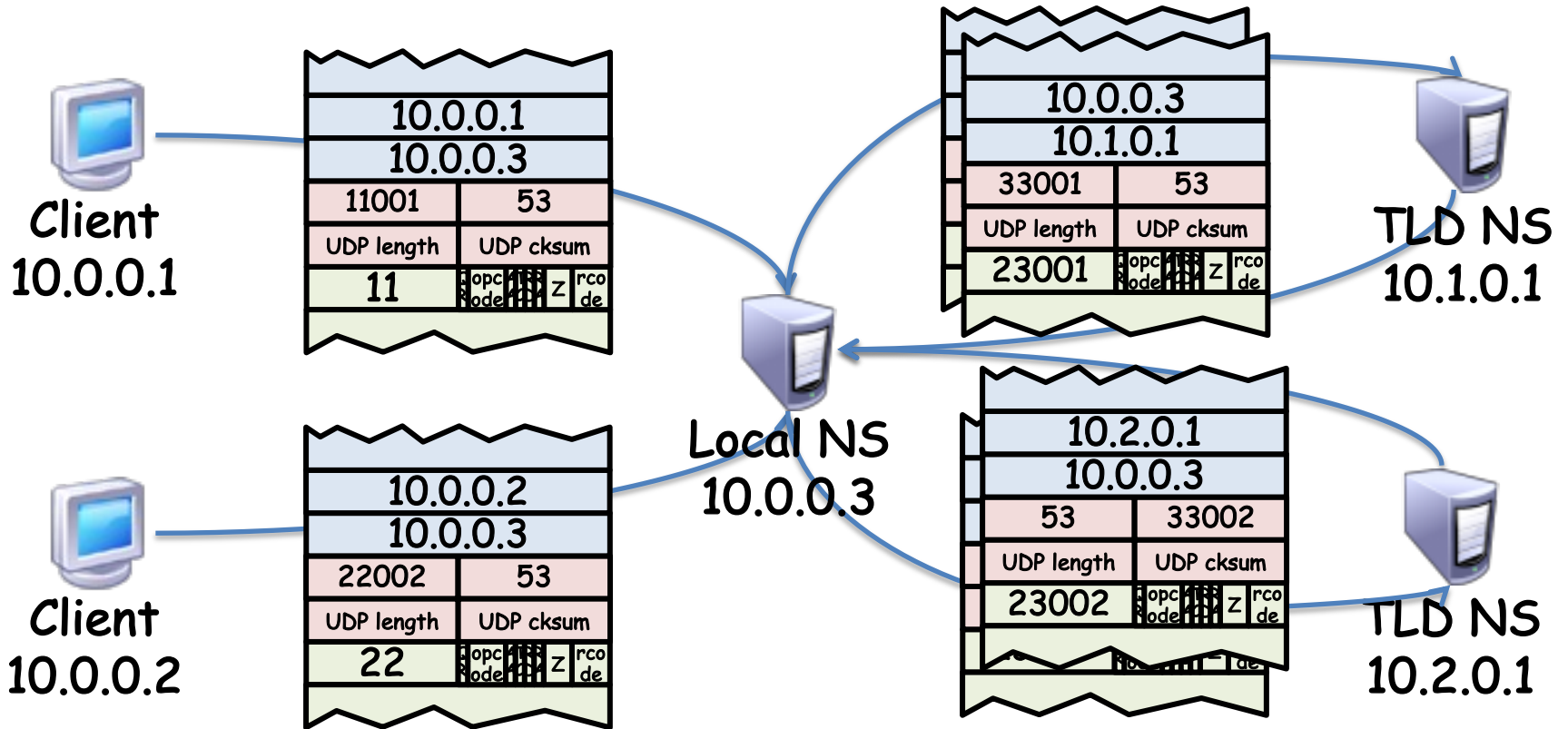
DNS protocol operation

- Most queries and responses via UDP, server port 53



DNS server state

UDP socket listening on port 53



Local NS maintains state associating incoming Query ID → ongoing query

Today

1. Review of Domain Name System (DNS)
2. **DNS security**

Implications of subverting DNS

1. Redirect victim's web traffic to rogue servers
 2. Redirect victim's email to rogue email servers (MX records in DNS)
- Does Secure Sockets Layer (SSL) provide protection?
 - **Yes**—user will get “wrong certificate warnings” if SSL is enabled
 - **No**—SSL not enabled or user ignores warnings
 - **No**—how is SSL trust established? **Often, by email!**

Security Problem #1: Coffee shop

- As you sip your latte and surf the Web, how does your laptop find `http://google.com`?
- Answer: it asks the local DNS nameserver
 - Which is run by the coffee shop / their contractor
 - Can return to you **any answer they please**
 - Including a bogus site that forwards your query to Google, gets reply to forward back to you, can **change anything** in **either** direction
- How can you know you're getting correct data?
 - Solution (mostly): Transport Layer Security (HTTPS)

Security Problem #2: Cache poisoning

- Suppose you are evil and **you control** the name server for foobar.com. You receive a request to resolve www.foobar.com and reply:

```
;; QUESTION SECTION:
;www.foobar.com.          IN      A

;; ANSWER SECTION:
www.foobar.com.          300     IN      A      212.44.9.144

;; AUTHORITY SECTION:
foobar.com.              600     IN      NS      dns1.foobar.com.
foobar.com.              600     IN      NS      google.com.

;; ADDITIONAL SECTION:
google.com.              5       IN      A      212.44.9.155
```

Evidence of the attack disappears
5 seconds later!

A foobar.com machine, *not* google.com

DNS cache poisoning (cont'd)

- Okay, but how do you get the victim to look up `www.foobar.com` in the first place?
- Perhaps you connect to their mail server and send
 - `HELO www.foobar.com`
 - Which their mail server then looks up to see if it corresponds to your source address (anti-spam measure)
- Perhaps you send many people at the victim organization phishing email, hope one clicks

Solution to simple DNS cache poisoning: Bailiwick checking


- DNS resolver **ignores** all RRs **not in or under the same zone as the question**
- Widely deployed since *ca.* 1997

```
;; QUESTION SECTION:
;www.foobar.com.          IN      A

;; ANSWER SECTION:
www.foobar.com.          300     IN      A      212.44.9.144

;; AUTHORITY SECTION:
foobar.com.              600     IN      NS      dns1.foobar.com.
foobar.com.              600     IN      NS      google.com.

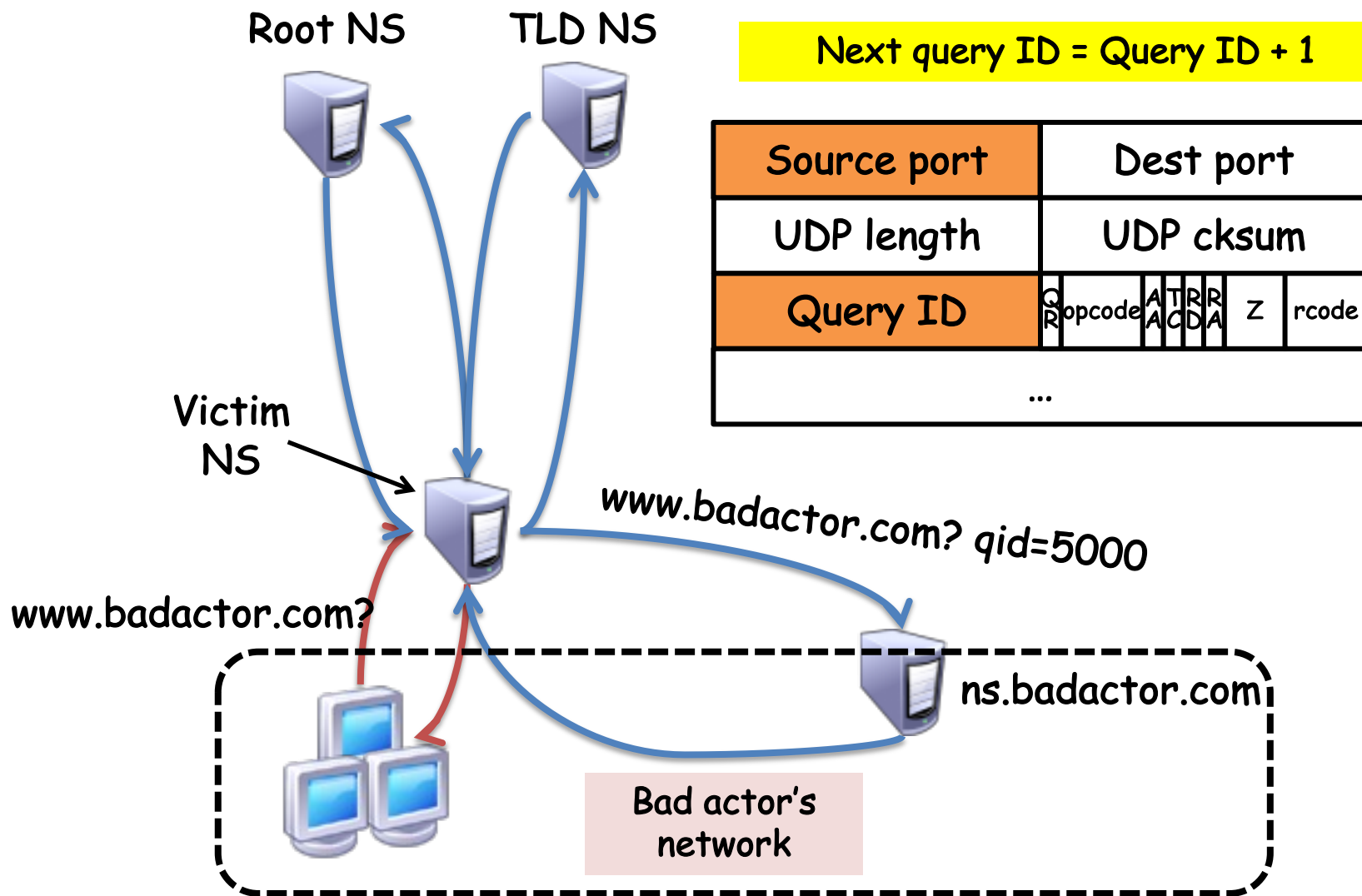
;; ADDITIONAL SECTION:
google.com.              5       IN      A      212.44.9.155
```



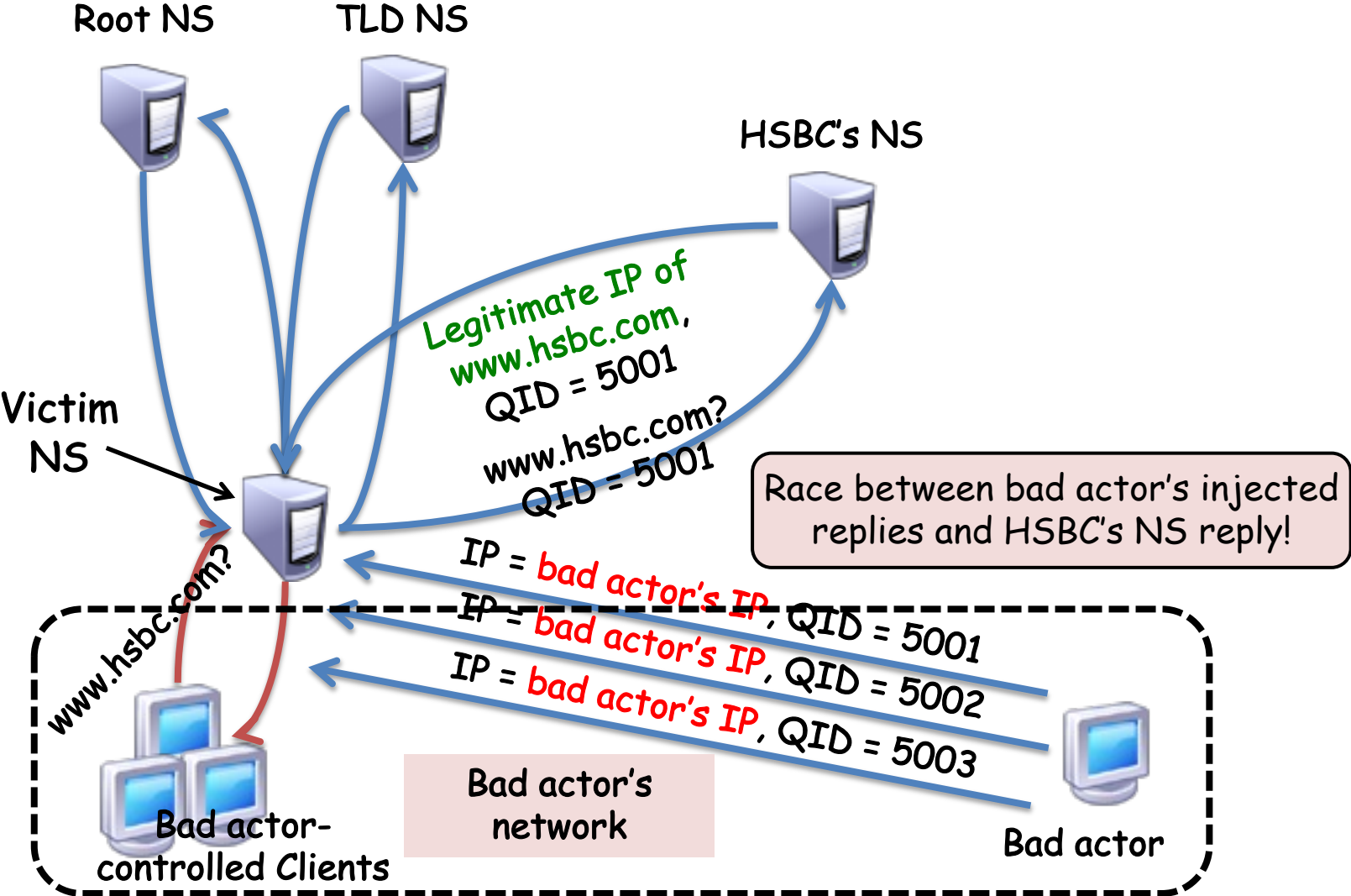
Poisoning the local nameserver remotely

- Let's get more sophisticated and try to target a local nameserver not under our control
- When does the nameserver accept a reply?
 - Reply's dest. UDP port = query's source UDP port
 - Matching question section
 - Matching (16-bit) query IDs
- So if the bad actor can achieve the above, they can inject incorrect data into nameserver's cache: let's see how

Predicting the next query ID



Remote Nameserver cache poisoning



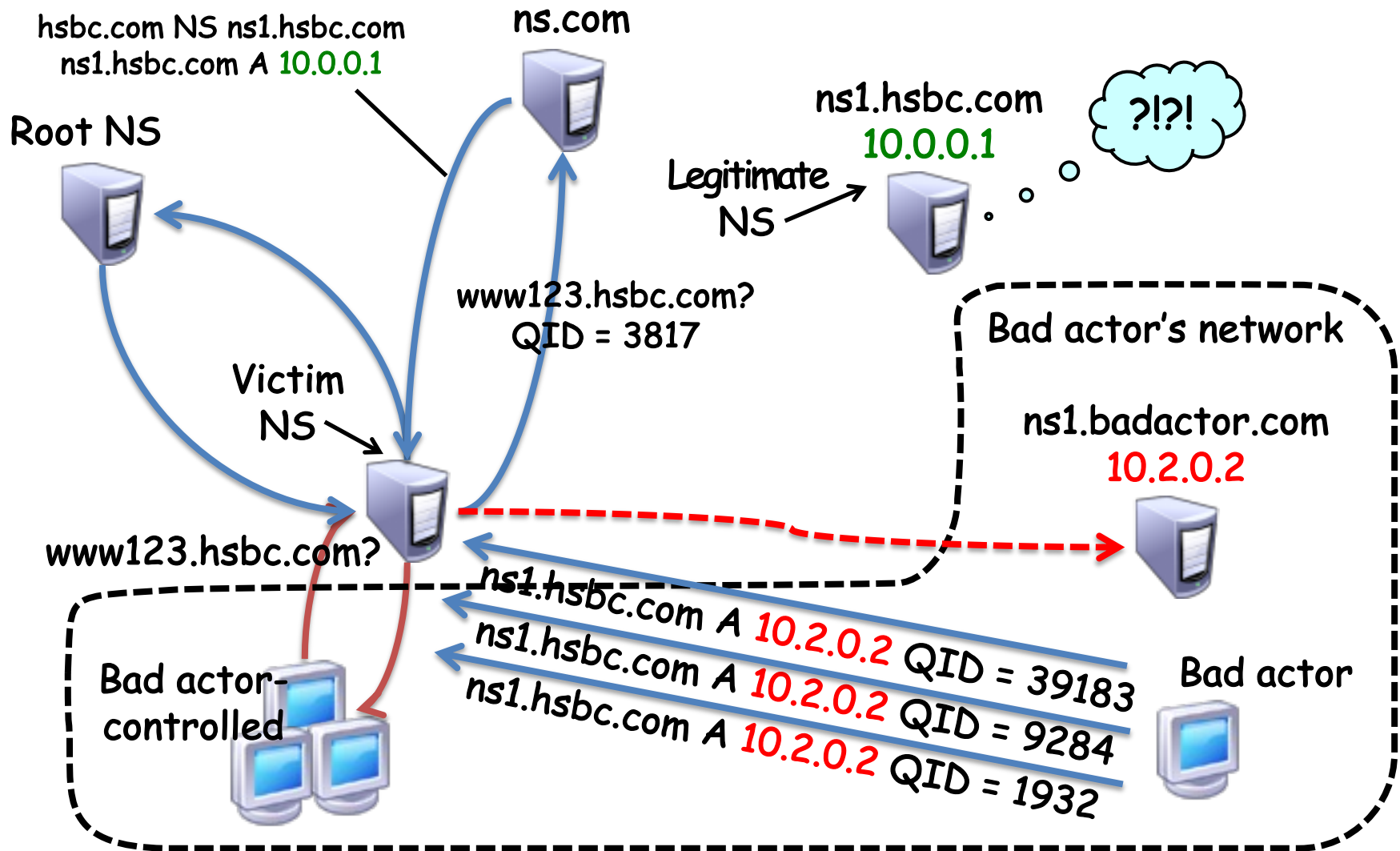
Requirements for a successful exploit

1. Attacker has to know UDP source port victim NS sent the query on (otherwise UDP drops the forged reply)
 - *ca. 2008*, most NSs used a well-known source port!
2. Attacker has to correctly guess 16-bit Query ID
 - Countermeasure: name servers now use pseudorandom query IDs
 - Although, older servers used an easily-guessable pseudorandom number generator
3. Forged replies have to arrive first
4. Name can't already be in victim's cache
5. Forged reply passes the bailiwick check (trivial)

Upping the ante: Kaminsky nameserver poisoning

- Now let's assume the nameserver uses query ID randomization
- Two main ideas behind Kaminsky DNS cache poisoning:
 1. **Compromise an entire domain** instead of just one IP
 - Now the attacker targets the NS **glue records**
 2. Launch multiple (K) simultaneous uncached queries to increase odds of success, for example:
 - www123.hsbc.com
 - www1234.hsbc.com
 - www12345.hsbc.com

Kaminsky nameserver poisoning I: One query



Kaminsky nameserver poisoning: Odds of success

- Now how likely is this attack to work?
 - The attacker is successful if they don't guess the query ID wrongly all K times

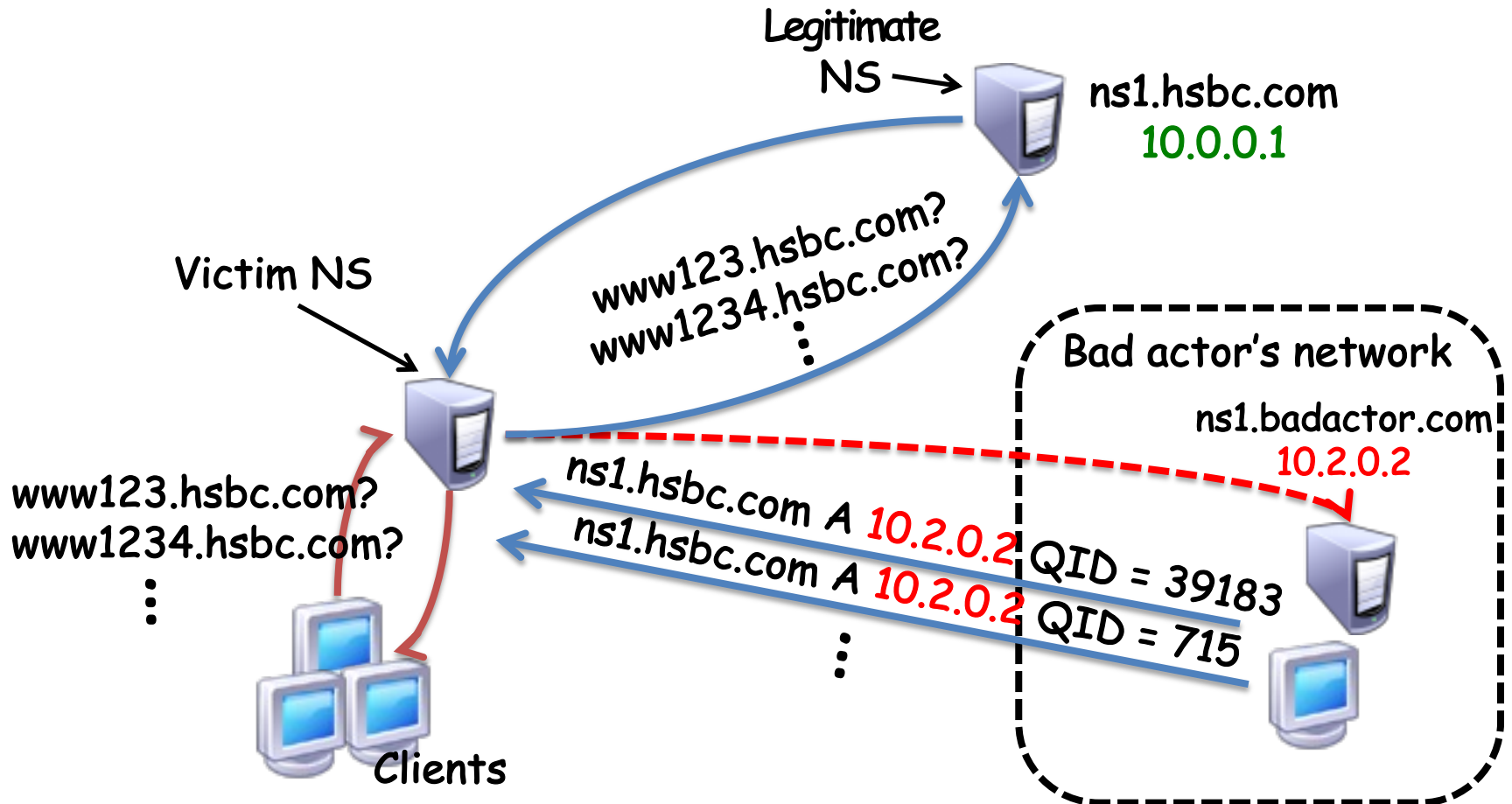
$$\Pr(\text{guess correct query id}) = \frac{1}{65,535}$$

$$\Pr(\text{guess wrong query id } K \text{ times}) = \left(1 - \frac{1}{65,535}\right)^K$$

K	$\Pr(\text{guess wrong query id } K \text{ times})$ $= 1 - \Pr(\text{success})$
4	0.99994
40	0.9994
400	0.994
4,000	0.94
40,000	0.54

Kaminsky nameserver poisoning II: Multiple queries and replies

Legitimate NS is now cached in the victim NS, but bad actor makes requests for new random names in victim's domain



Increasing the chances of success

- Suppose we send a burst of L queries and L forged responses
 - Random query IDs everywhere, L -choose-2 possibilities

$$\Pr(\text{one query/response pair matches}) = \frac{1}{65,535}$$

$$\Pr(\text{guess wrong query id } L \text{ times}) = \left(1 - \frac{1}{65,535}\right)^{\binom{L}{2}}$$
$$= \left(1 - \frac{1}{65,535}\right)^{\frac{L(L-1)}{2}}$$

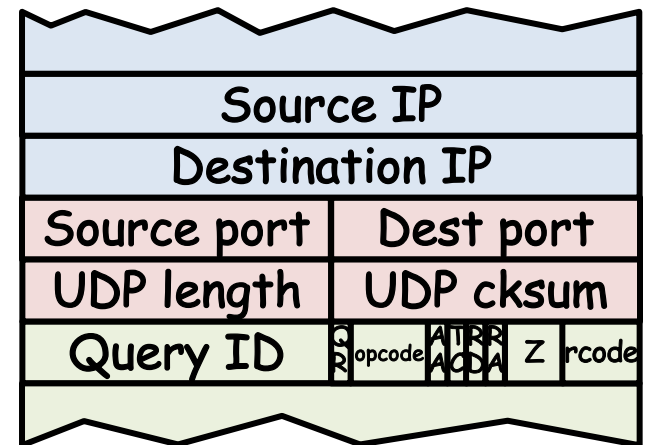
- In practice, takes about 10 minutes

L	Pr(Every forgery wrong)
10	0.9994
100	0.926
290	0.54

Mitigating Kaminsky nameserver poisoning

- **Solution: Randomize the query's UDP source port as well**
- Reply checking:
 1. Kernel network stack matches destination port of TLD server's reply with UDP source port of local NS's query
 2. DNS server matches query ID of reply with query id of request
- *e.g.* Msft DNS server pre-allocates 2,500 UDP ports for requests

$$\Pr(\text{correct guess}) = \left(\frac{1}{65,000}\right)\left(\frac{1}{2,500}\right) \approx 6 \times 10^{-9}$$



Conclusions

- DNS is core Internet infrastructure
- Need to keep it secure against attack
 - Many subtleties, attacks, and countermeasures
- In past decade, Transport Layer Security (HTTPS) has helped the situation