The Domain Name System

3035/GZ01 *Networked Systems*
Kyle Jamieson

Department of Computer Science
University College London

Today

1. **The Domain Name System (DNS)**

2. DNS security

3. Coursework 2 introduction

Host names versus IP addresses

- **Host names** (*e.g.* www.bbc.co.uk)
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about location
  - Examples: www.cnn.com and bbc.co.uk

- **IP addresses**
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location

Original design of the DNS

- Per-host file named **`/etc/hosts`**
  - Flat namespace: each line is an IP address and a name
  - SRI (Menlo Park, California) kept the master copy
  - Everyone else downloads regularly

- **But, a single server doesn't scale**
  - Traffic implosion (lookups and updates)
  - Single point of failure

- Need a distributed and hierarchical **collection** of servers

Domain Name System (DNS)

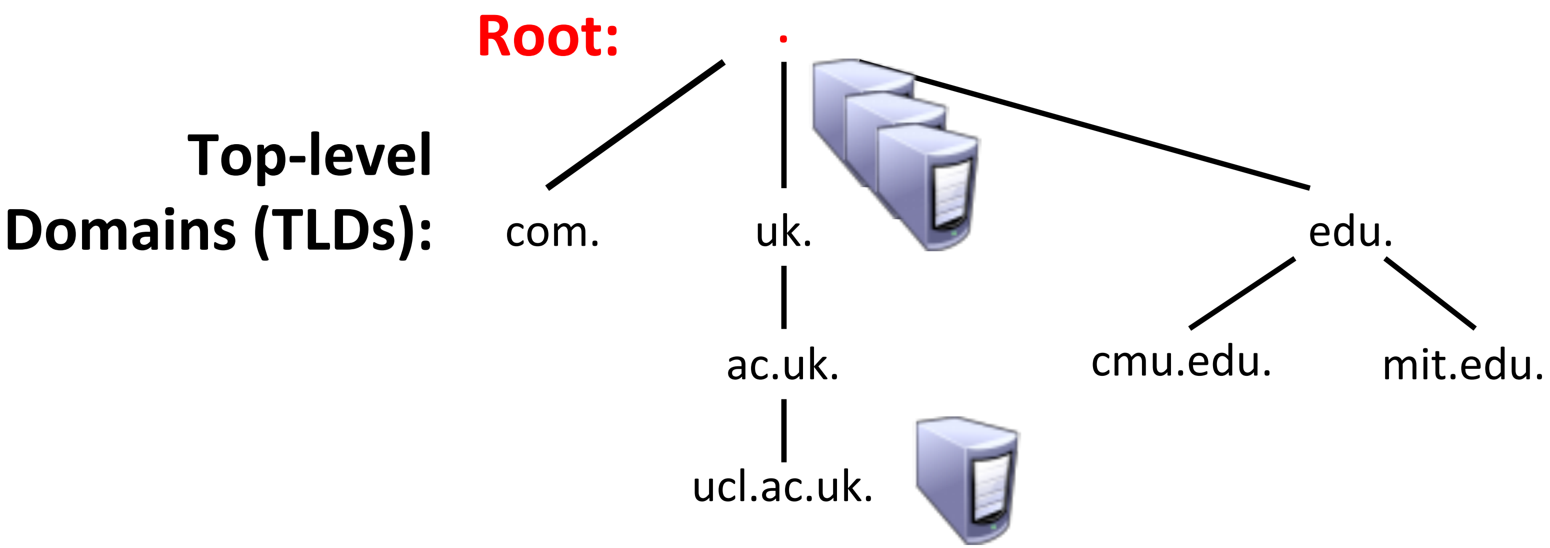
- **Hierarchical** name space divided into pieces called ***zones***

- Zones are distributed over a collection of DNS servers

- Hierarchy of DNS servers
  - ***Root*** servers (identity is hardwired into other servers)
  - ***Top-level domain (TLD)*** servers
  - ***Authoritative*** DNS servers

- Performing the translations
  - ***Local DNS servers*** located near clients
  - ***Resolver*** software running on clients

The DNS namespace is hierarchical



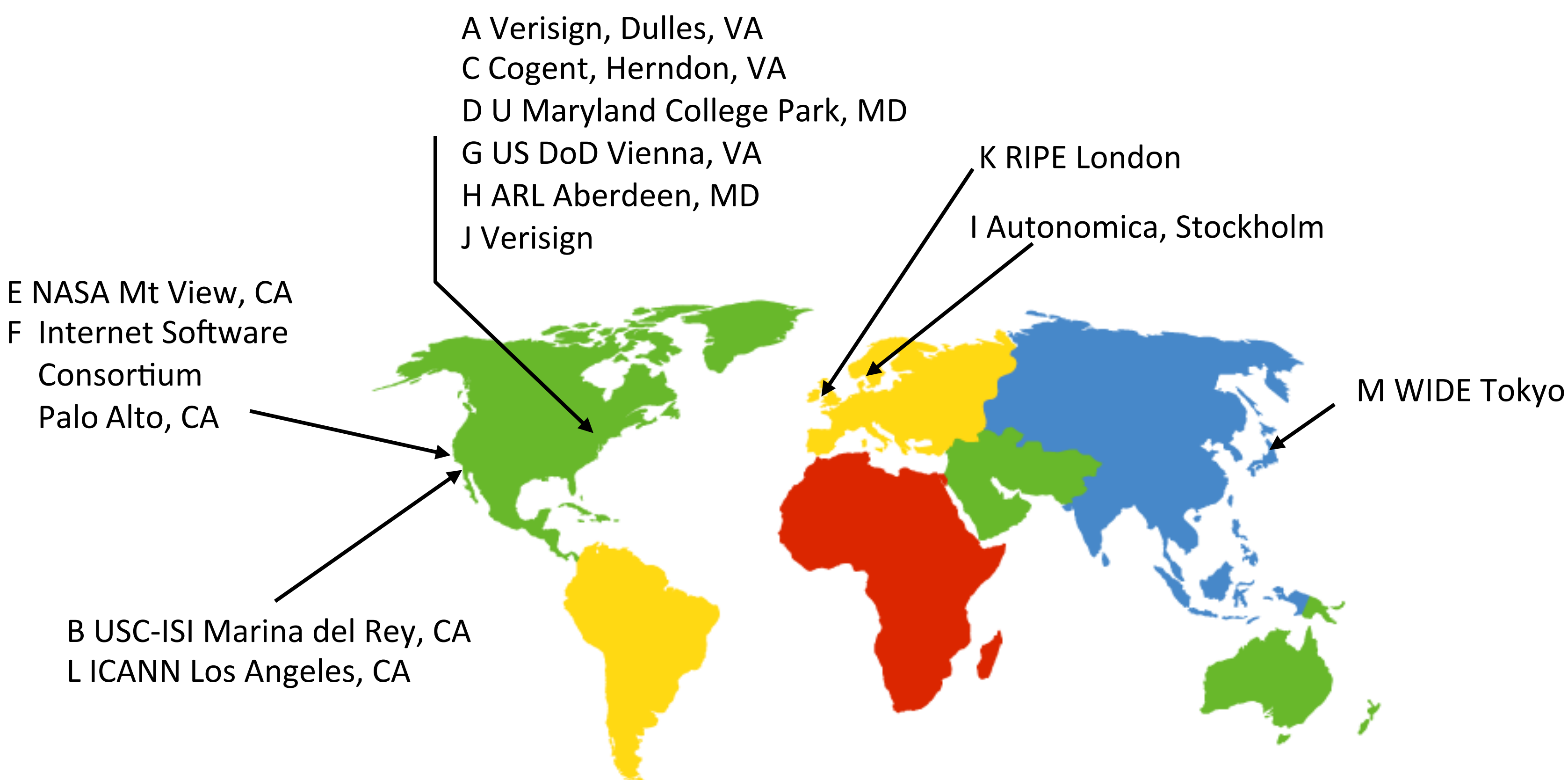
- Hierarchy of servers follows hierarchy of DNS zones

Many uses of DNS

- **Hostname to IP address** translation
- **IP address to hostname** translation (***reverse lookup***)
- ***Host name aliasing*** allows other names for a host
  - Can be arbitrarily many aliases
  - *Alias* host names point to canonical hostname
- **Mail server** location
  - Lookup zone's mail server based on zone name
- **Content distribution networks**
  - Load balancing among many servers with different IP addresses
  - Complex, hierarchical arrangements are possible

DNS root nameservers

- 13 root servers (see http://www.root-servers.org)
  - Labeled A through M
- Does **this** scale?



[Slide credit: Scott Shenker]

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**

[Slide credit: Scott Shenker]

TLD and Authoritative Servers

- ***Top-level domain (TLD)*** servers
  - Responsible for com, org, net, edu, etc, and all top-level country domains: uk, fr, ca, jp
  - *Network Solutions* maintains servers for com TLD
  - *Educause* for edu TLD

- ***Authoritative*** DNS servers
  - An organization's DNS servers, providing authoritative information for organization's servers
  - Can be maintained by organization or service provider

Local name servers

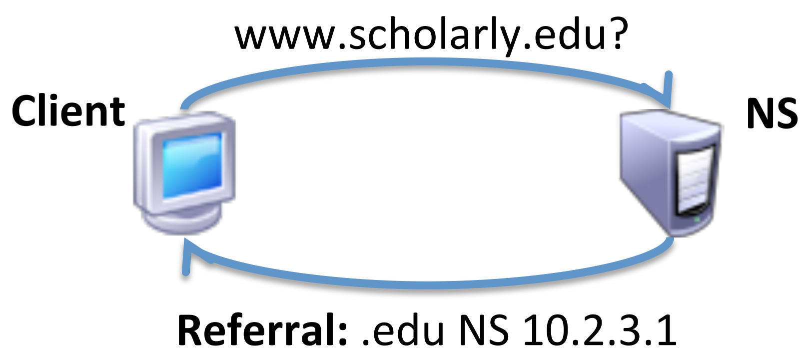
- Do not strictly belong to hierarchy
- Each ISP (company, university) has one
  – Also called ***default*** or ***caching*** name server
- When host makes DNS query, query is sent to its local DNS server
  – Acts as proxy, forwards query into hierarchy
  – Does work for the client

DNS in operation

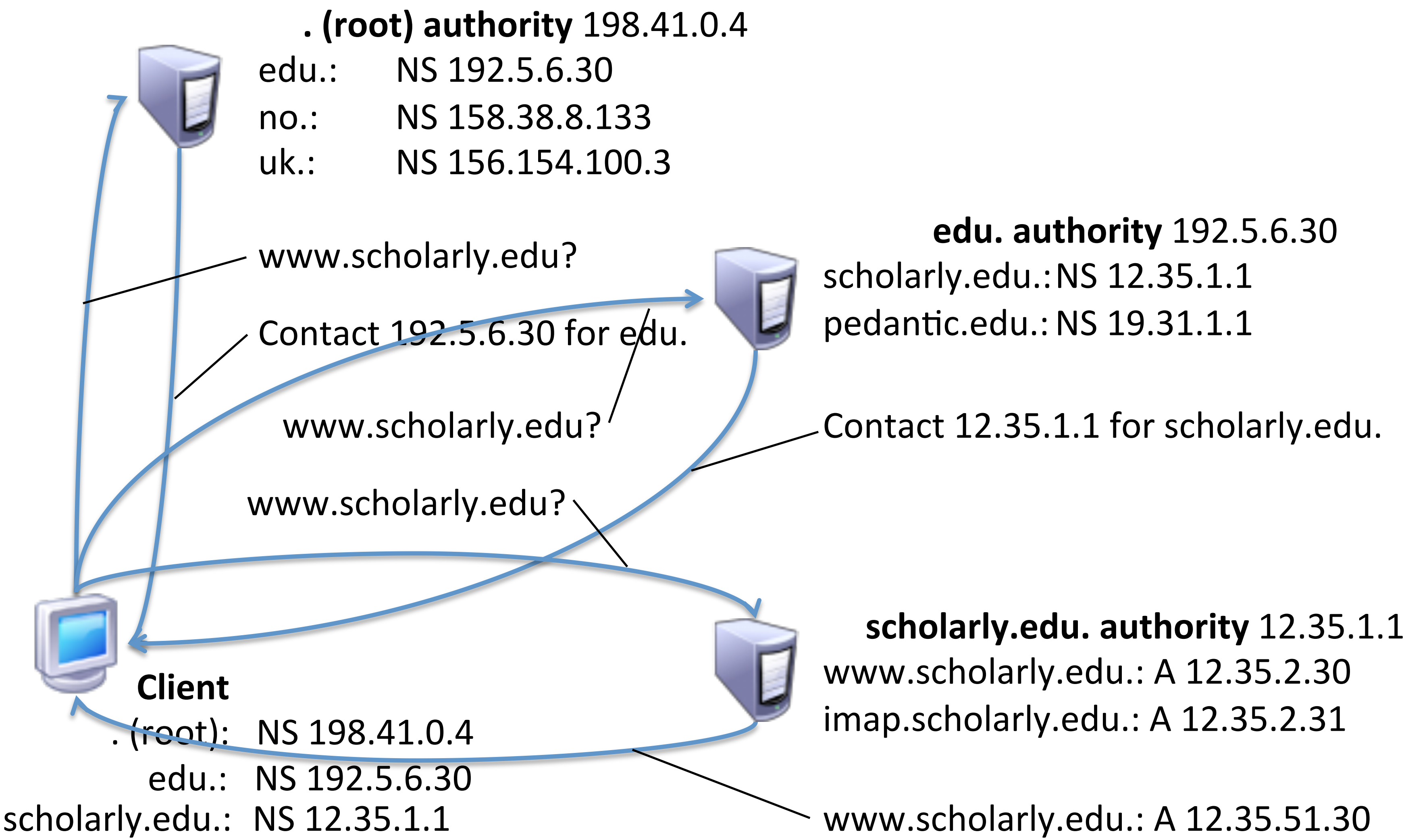
- Most queries and responses are UDP datagrams
- Two types of queries:

- **Recursive:**

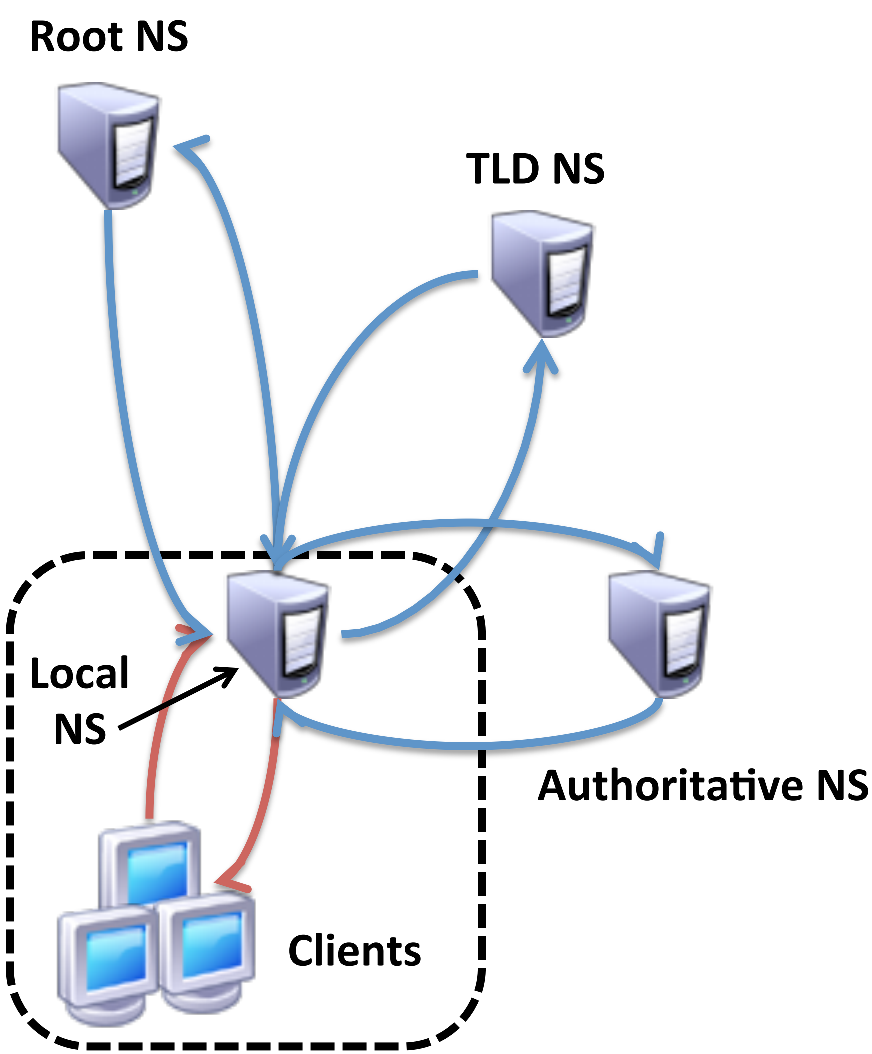
- **Iterative:**



A recursive DNS lookup (simplified)



Local NS does clients' work



1. Client's resolver makes a **recursive** query to local NS

2. Local NS processing:
   - Local NS sends **iterative** queries to other NS's
   - or, finds answer in cache

3. 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 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.ucl.ac.uk)
  - `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.ucl.ac.uk is really haig.cs.ucl.ac.uk
  - `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

```
$ 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.afilias-nst.org.
org.              172800 IN  NS  d0.org.afilias-nst.org.

;; ADDITIONAL SECTION:
b0.org.afilias-nst.org.  172800 IN  A   199.19.54.1
d0.org.afilias-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)

```
$ 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)

```
$ 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

[Slide credit: Scott Shenker]

Reverse mapping (IP to hostname)

- How do we go the other direction, from an IP address to the corresponding hostname?
  - Why do we care to? Troubleshooting, security, spam

- IP address already has natural "quad" hierarchy: **12.**34.56.78

- But: IP address has most-significant hierarchy element on the left, while www.cnn**.com** has it on the right

- Idea: **reverse** the quads = 78.56.34.12, and look **that** up in the DNS

- Under what top-level domain?
  - Convention: **in-addr.arpa**
  - So lookup is for `78.56.34.12.in-addr.arpa`

Inserting resource records into DNS

- Example: just created startup "FooBar"
- Get a block of address space from ISP, say 212.44.9.128/25
- Register **`foobar.com`** at Network Solutions (say)
  - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts RR pairs into the **`com`** TLD server:
    - (**`foobar.com`**, **`dns1.foobar.com`**, **`NS`**)
    - (**`dns1.foobar.com`**, **`212.44.9.129`**, **`A`**)
- Put in your (authoritative) server **`dns1.foobar.com`**:
  - Type A record for **`www.foobar.com`**
  - Type MX record for **`foobar.com`**

[Slide credit: Scott Shenker]

Setting up *foobar.com* (cont'd)

- In addition, need to provide reverse PTR bindings
  - *e.g.,* **`212.44.9.129`** → **`dns1.foobar.com`**

- Normally, these would go in **`9.44.212.in-addr.arpa`**

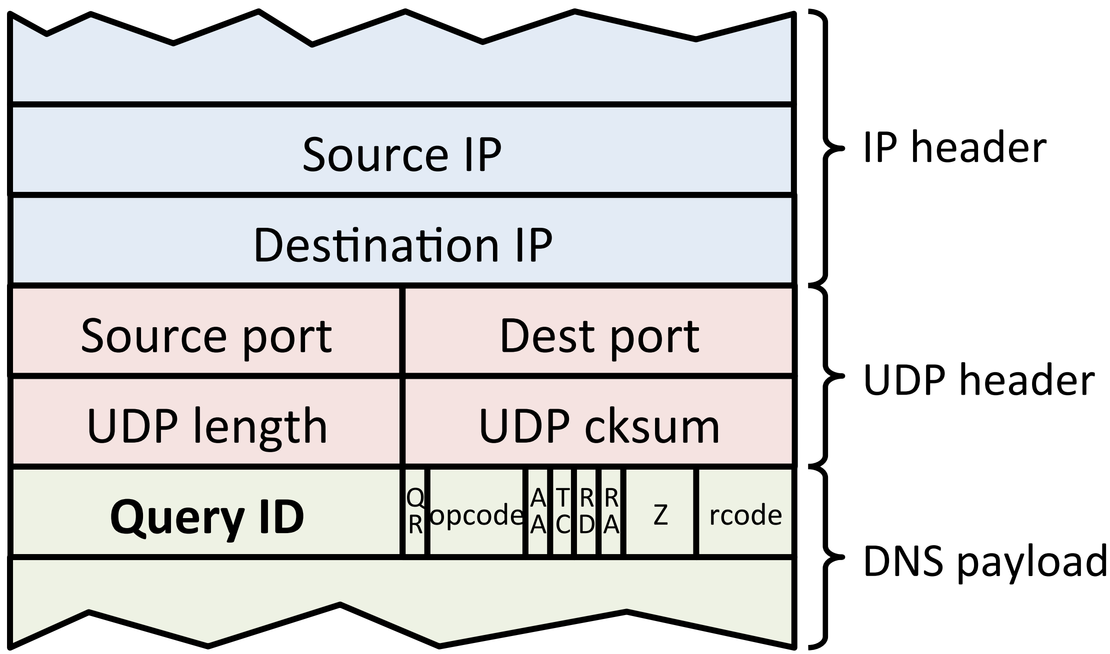
- Problem: you can't run the name server for that domain. Why not?
  - Because your block is 212.44.9.128/25, not 212.44.9.0/24
  - And whoever has 212.44.9.0/25 won't be happy with you owning their PTR records

- Solution: ISP runs it for you, but it's more of a headache to keep it up-to-date **`:-(`**

[Slide credit: Scott Shenker]

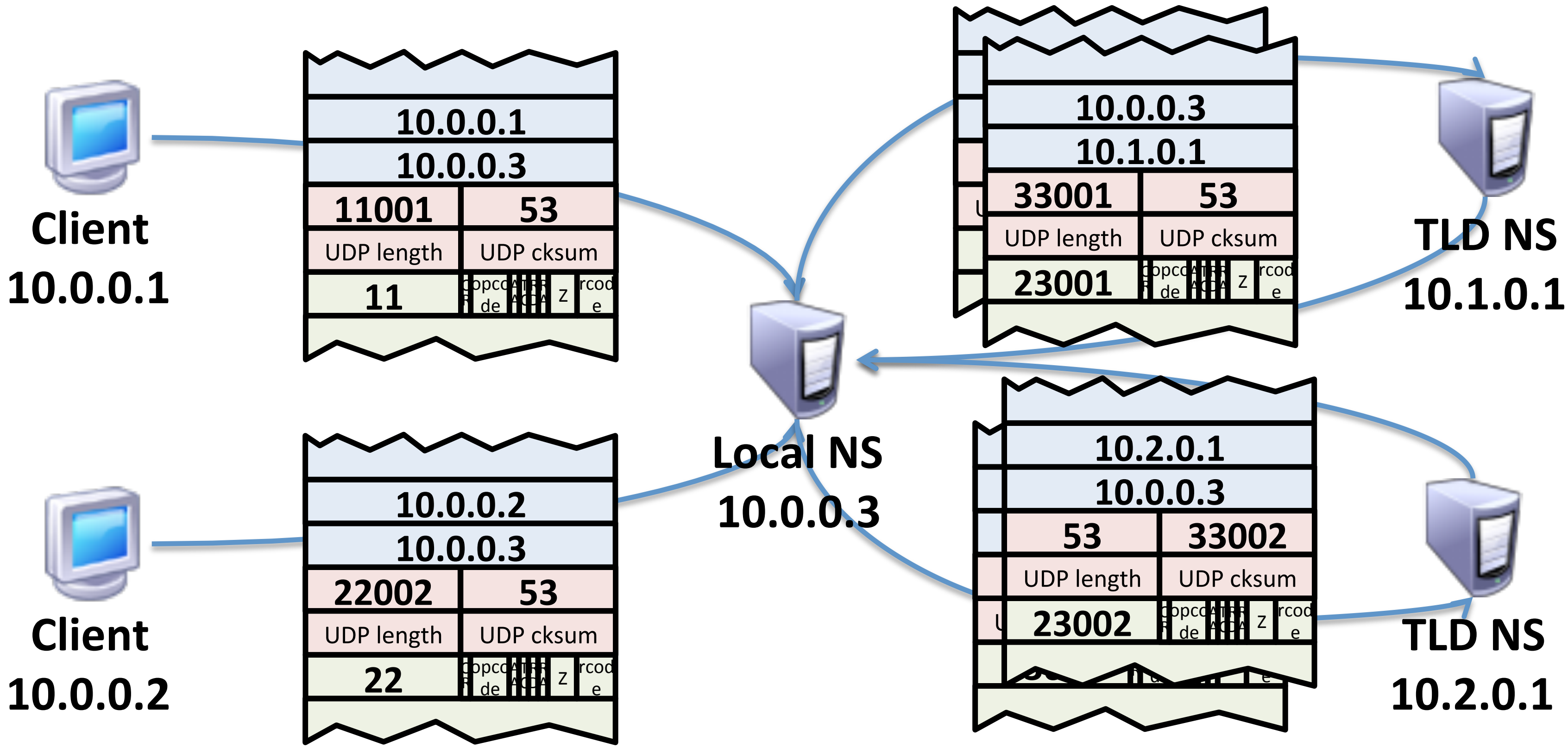
DNS protocol operation

- Most queries and responses via UDP, server port 53



DNS server state

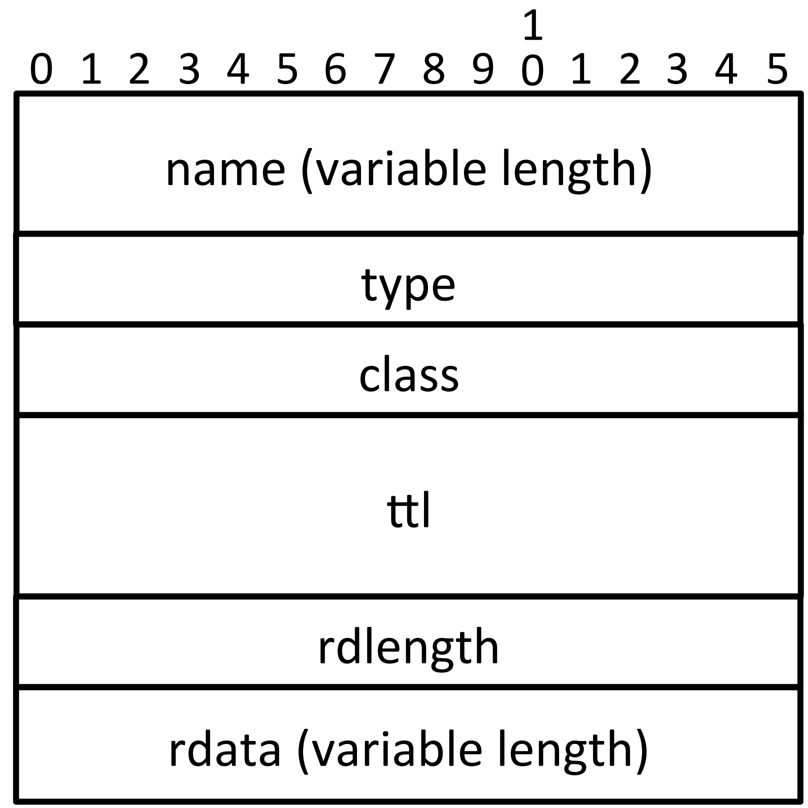
UDP socket listening on port 53



Local NS at least needs to keep state associating Query ID → which query (if any)

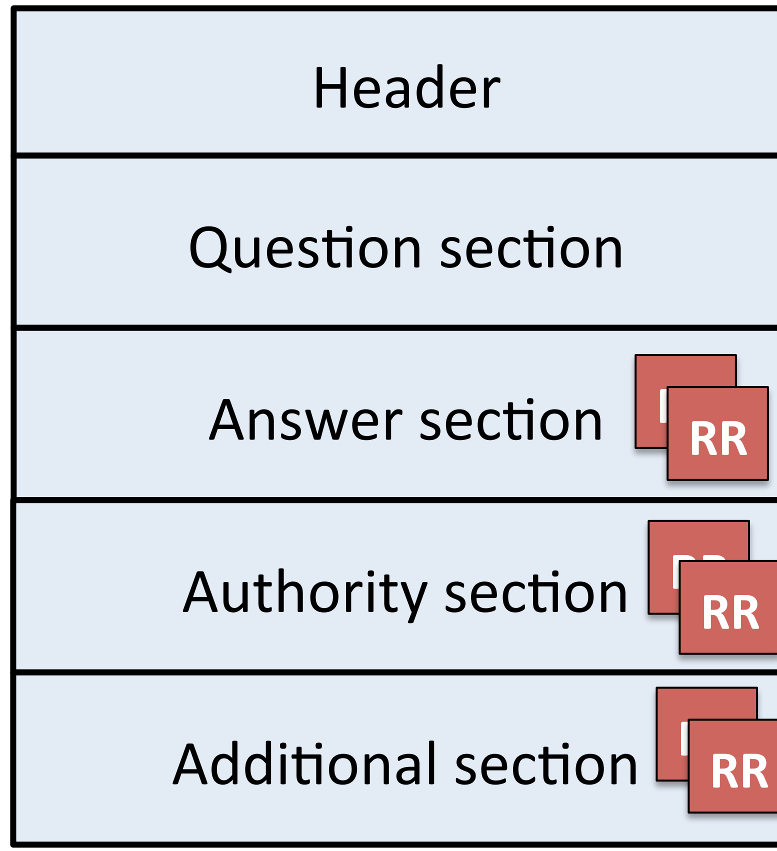
A DNS resource record (RR) in detail

- **type:** determines the meaning of rdata
- **class:** always IN (Internet)
- **rdata:** data associated with the RR



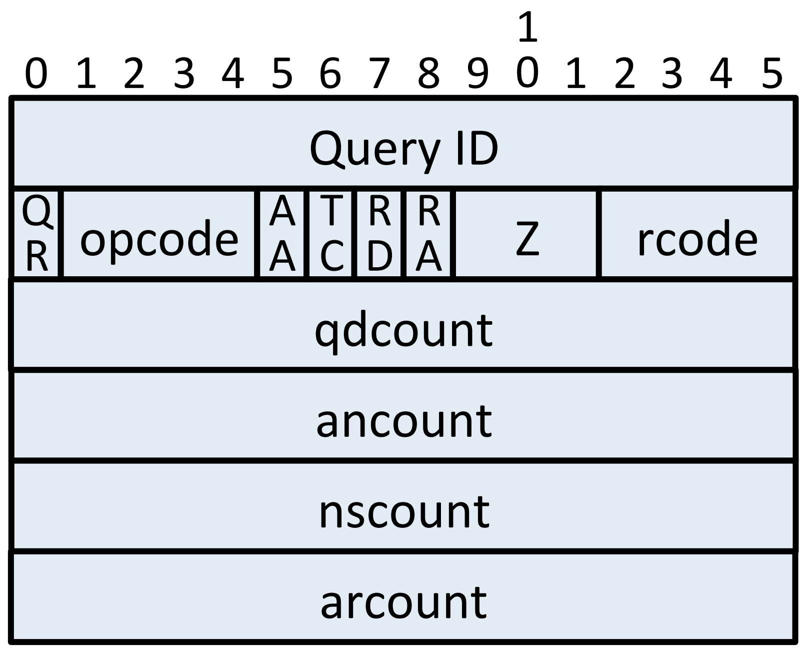
DNS protocol message

- Query and reply messages have identical format
- **Question section:** query for name server
- **Answer section:** RRs answering the question
- **Authority section:** RRs that point to an authoritative NS
- **Additional section:** “glue” RRs



DNS protocol header

- **Query ID:** 16-bit identifier shared between query, reply
- Flags word
  - **QR:** query (0) or response (1)
  - **opcode:** standard query (0)
  - **AA:** authoritative answer
  - **TC:** truncation
  - **RD:** Recursion desired
  - **RA:** Recursion available
  - **Z:** (reserved and zeroed)
  - **rcode:** response code; ok (0)



- **qdcount:** number of question entries (QEs) in message
- **ancount:** number of RRs in the answer section
- **nscount:** number of RRs in the authority section
- **arcount:** number of RRs in the additional section

Today

1. The Domain Name System (DNS)

2. **DNS security**

3. Coursework 2 introduction

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 `google.com`?

- Answer: it asks the local DNS nameserver
  - Which is run by the coffee shop or their contractor
  - And can return to you any answer they please
  - Including a "man in the middle" site that forwards your query to Google, gets the reply to forward back to you, yet can change anything they wish in either direction

- How can you know you're getting correct data?
  - Today, you can't. (Though if site is HTTP**S**, that helps)
  - One day, hopefully: **DNSSEC** extensions to DNS

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)
- Note, with compromised name server we can also lie about PTR records (address → name mapping)
  - *e.g.,* for 212.44.9.155 = 155.44.9.212.in-addr.arpa return google.com (or whitehouse.gov, or whatever)
    - If our ISP lets us manage those records as we see fit, or we happen to directly manage them

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
```

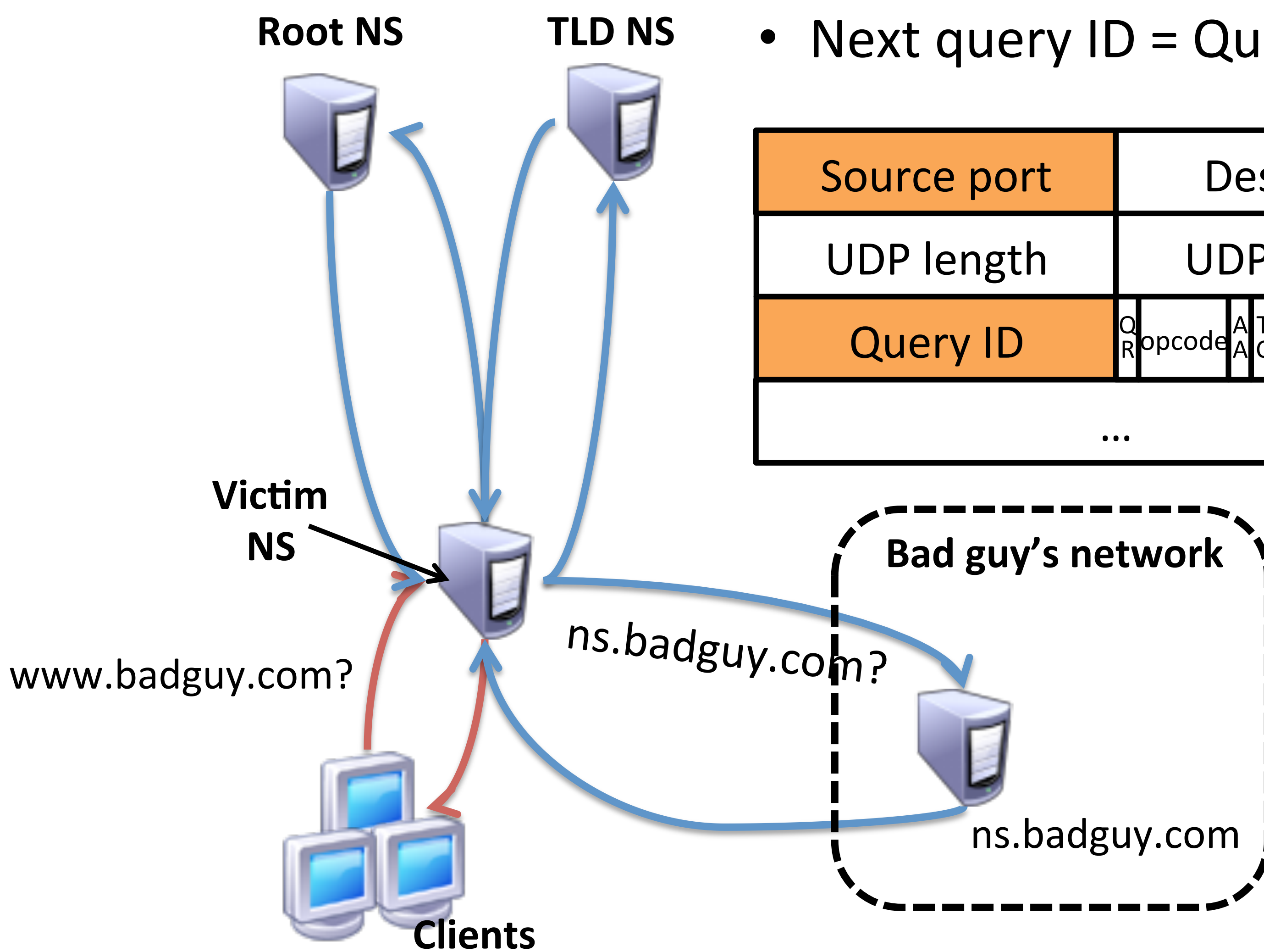
Poisoning the local nameserver

- Let's get more sophisticated and try to target the **local nameserver** instead of a single client
- 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 guy can achieve the above, he can inject incorrect data into a nameserver's cache**
  - Let's see how

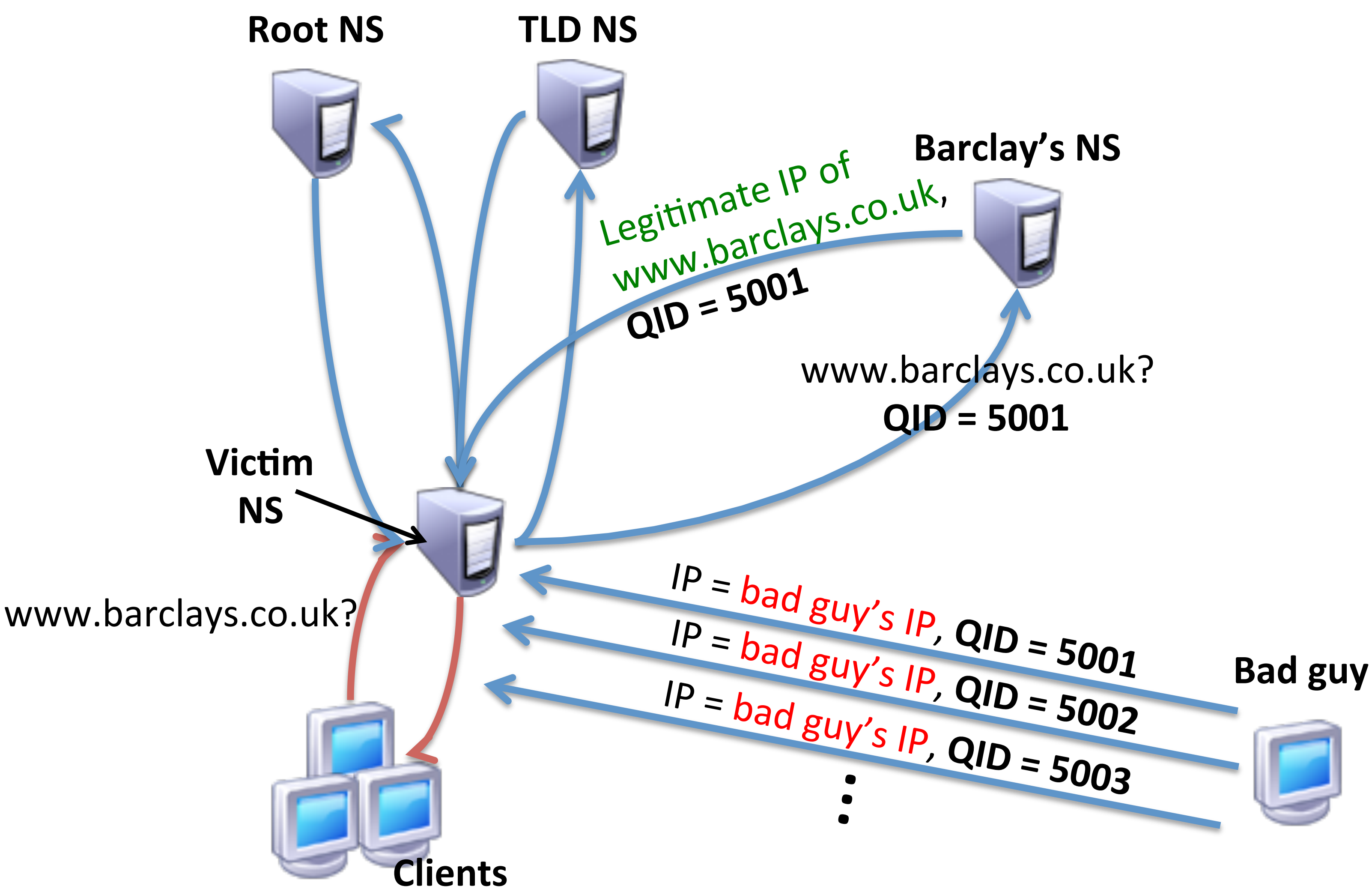
Predicting the next query ID

- Next query ID = Query ID + 1

Source port	Dest port
UDP length	UDP cksum
Query ID	QR · opcode · AA · TC · RD · RA · Z · rcode
...	



Nameserver cache poisoning



Requirements for a successful exploit

1. Attacker has to know the UDP source port the 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 the 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)

Kaminsky nameserver poisoning

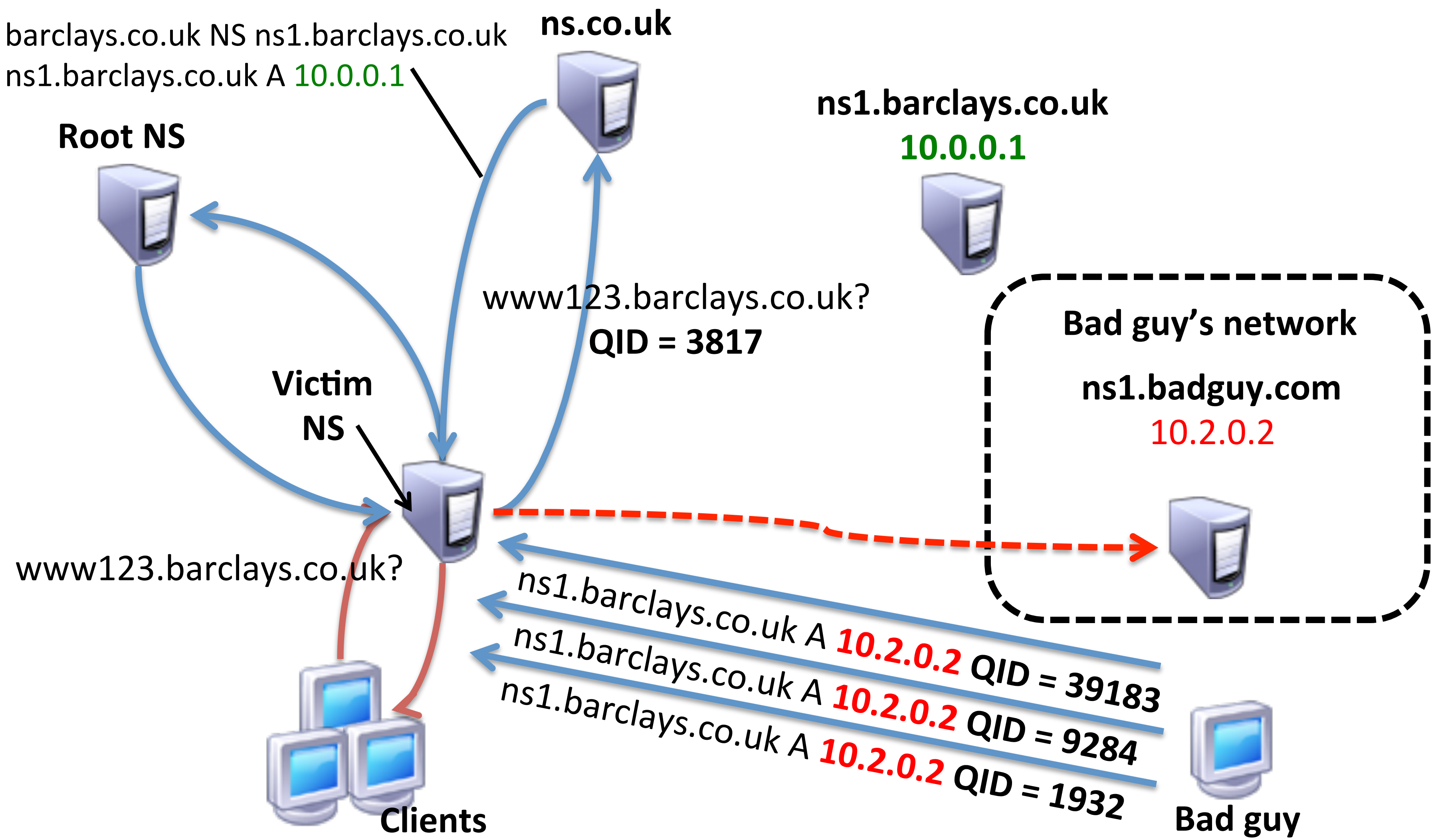
- **Now let's assume the nameserver uses query ID randomization**

- Two main ideas behind Kaminsky DNS cache poisoning:

1. Compromise an **entire domain** instead of just an IP
   – Now the attacker targets the **glue records**

2. Launch **multiple** (*K*) **simultaneous uncached queries** to increase odds of success, for example:
   – www123.barclays.co.uk
   – www1234.barclays.co.uk
   – www12345.barclays.co.uk

Kaminsky nameserver poisoning (1): One query



Kaminsky nameserver poisoning (2)

- Now how likely is this attack to work?
  - The attacker is successful if he **does not** guess the wrong query ID *K* times

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

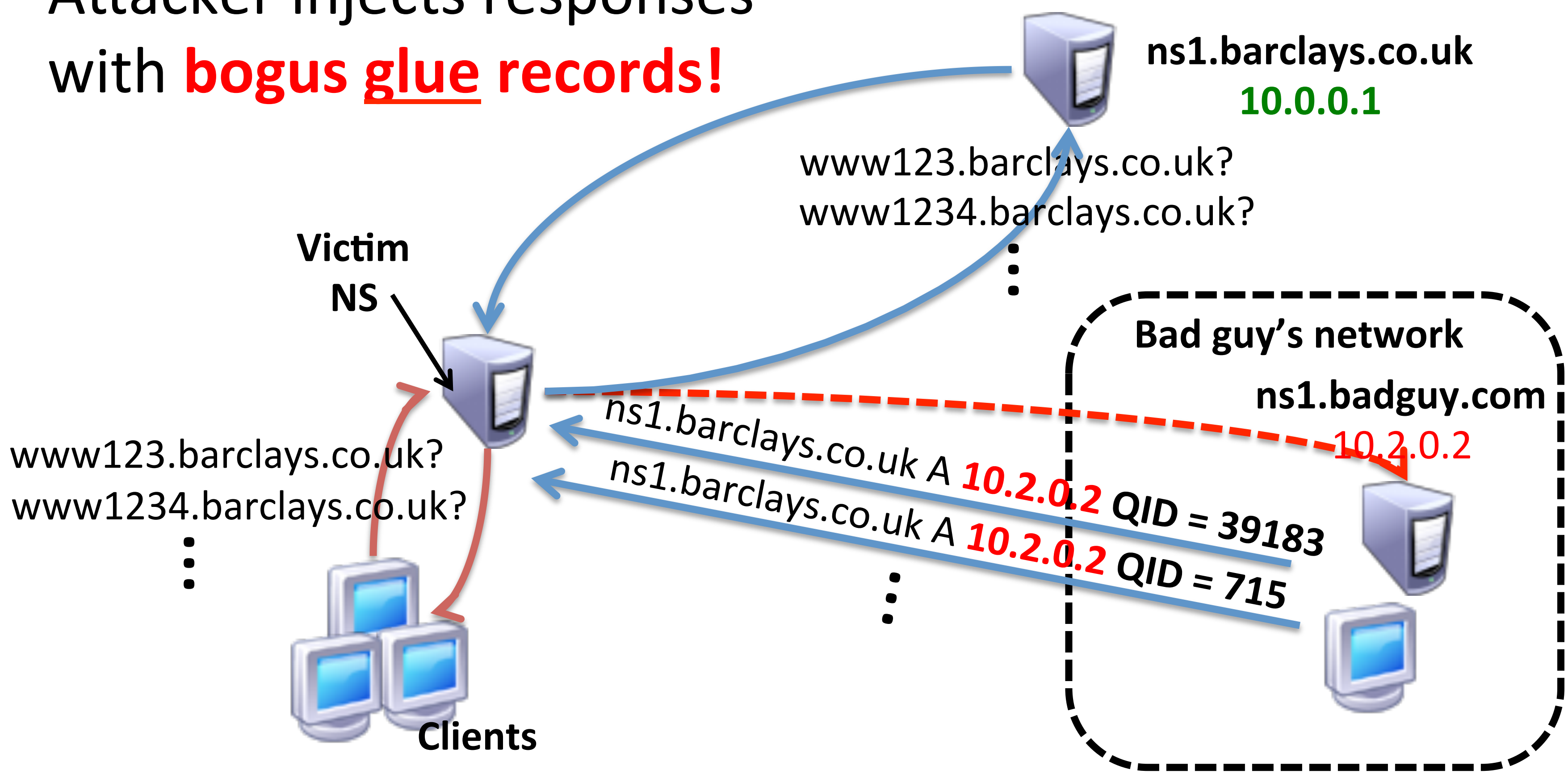
$$\Pr(\text{guess wrong query id } K \text{ times}) = \left(1 - \frac{1}{65{,}535}\right)^K$$

K	Pr(guess wrong query id *K* times)
4	0.99994
40	0.9994
400	0.994
4,000	0.94
40,000	**0.54**

Kaminsky nameserver poisoning (3)

Multiple queries and replies

- Legitimate NS is now cached in the victim NS, but victim NS still makes requests for new random names
- Attacker injects responses with **bogus glue records!**



Increasing the chances of success

- Suppose we send a burst of *L* queries **and** *L* forged responses
  - Random query IDs everywhere

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

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

$$= \left(1 - \frac{1}{65{,}535}\right)^{\frac{L(L-1)}{2}}$$

- In practice, takes about 10 minutes

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

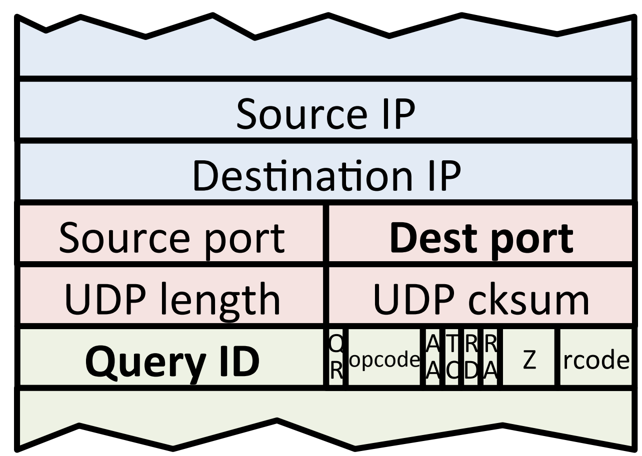
Mitigating nameserver poisoning

- **Solution: Randomize the query's UDP source port**

- 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

- MS DNS server pre-allocates 2,500 UDP ports for requests

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



Today

1. The Domain Name System (DNS)

2. DNS security

3. **Coursework 2 introduction**