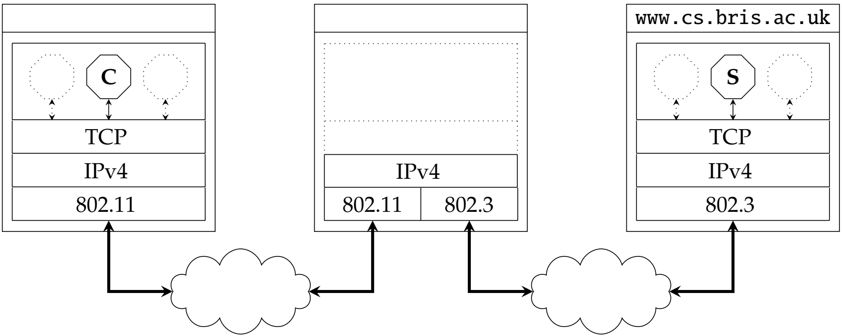
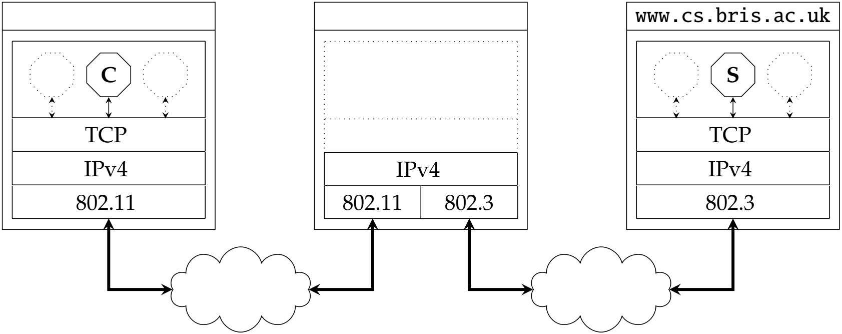
- Recall: we know how to realise



st. hosts can transmit TCP segments to each other.

- Recall: we know how to realise



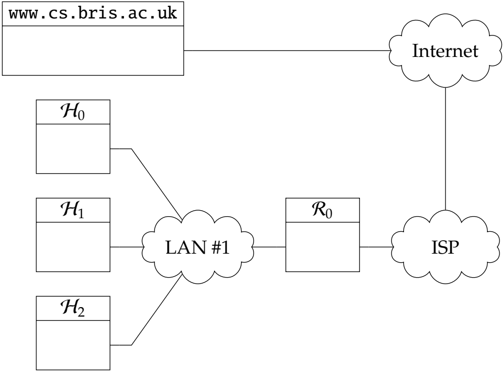
st. hosts can transmit TCP segments to each other ...

- ... *but*

1. how *could* both LANs use the same private IP address block, and
2. how does the client know the IP address of `www.cs.bris.ac.uk`?

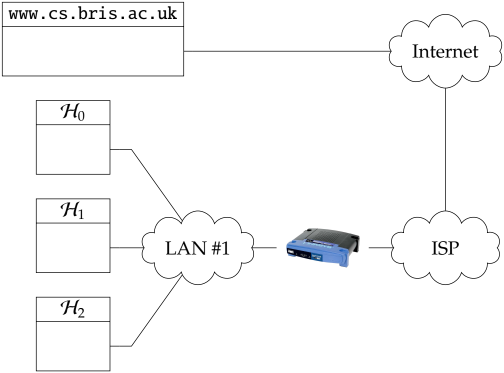
Problem #1 $\rightsquigarrow$ NAT (1)

- Problem: consider the following inter-network



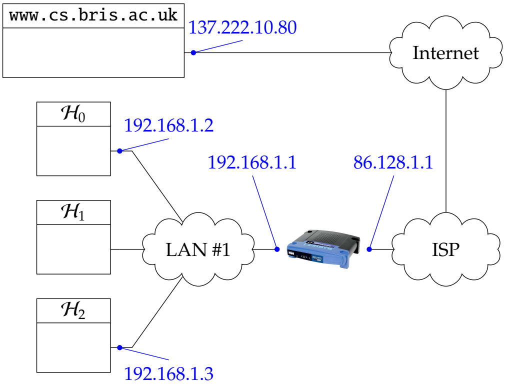
Problem #1 ⇝ NAT (1)

- Problem: consider the following inter-network



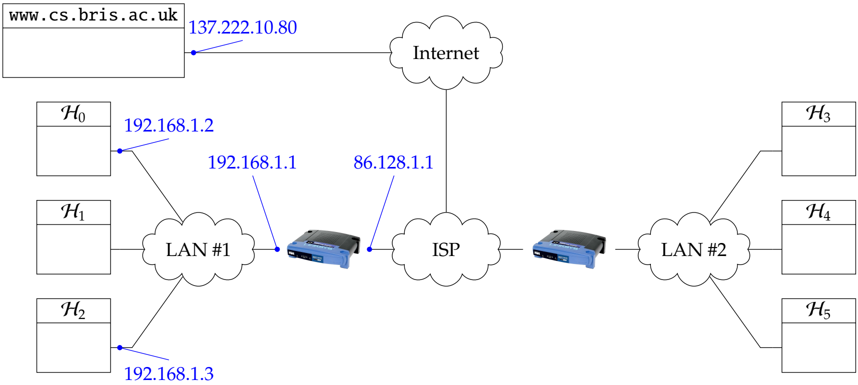
Problem #1 $\rightsquigarrow$ NAT (1)

- **Problem**: consider the following inter-network



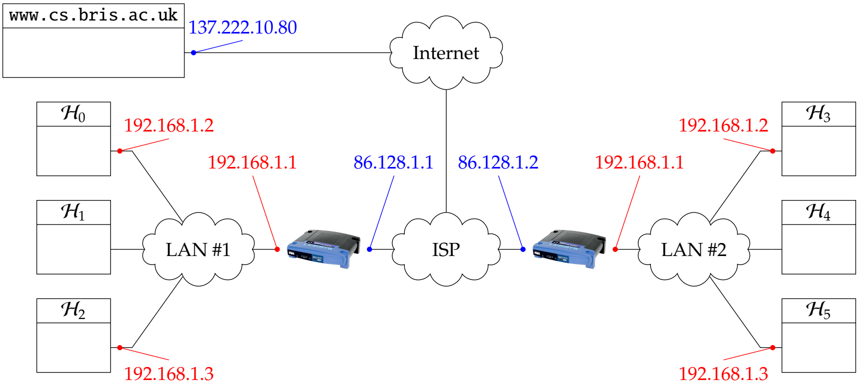
Problem #1 ⇝ NAT (1)

- Problem: consider the following inter-network



Problem #1 ⇝ NAT (1)

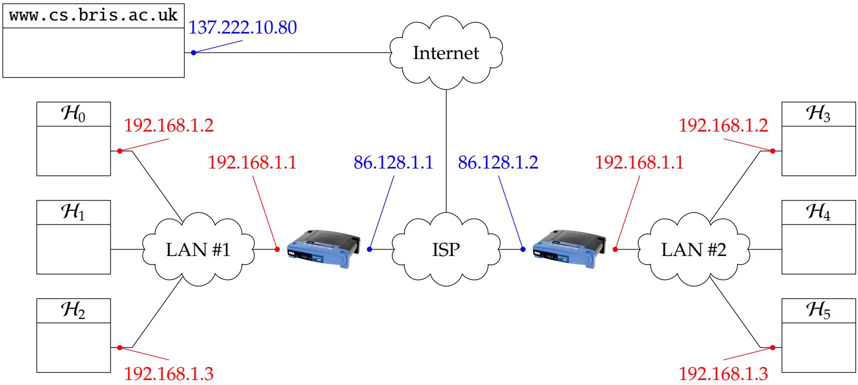
- Problem: consider the following inter-network



where LANs #1 *and* #2 (legitimately) use the private address block 192.168.0.0/16, so the host IP addresses conflict.

Problem #1 $\rightsquigarrow$ NAT (1)

- Problem: consider the following inter-network

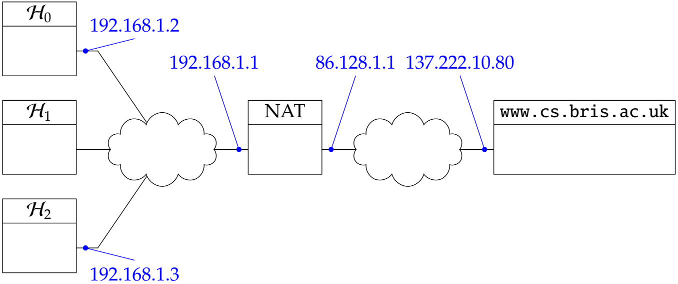


  where LANs #1 *and* #2 (legitimately) use the private address block 192.168.0.0/16, so the host IP addresses conflict.

- (A) solution: **Network Address Translation (NAT)** [7].

Problem #1 $\leadsto$ NAT (2)

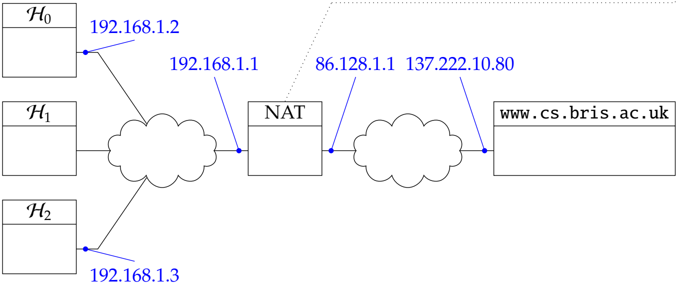
- Example:



Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

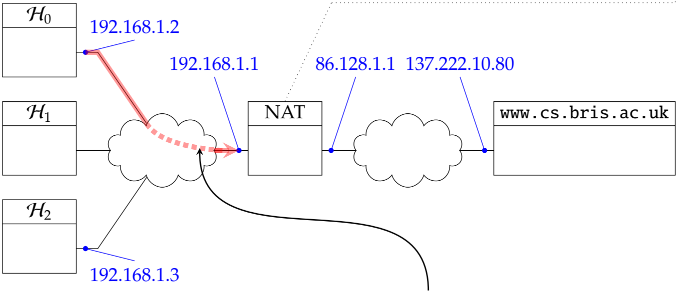
Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
⋮	⋮	⋮	⋮



Problem #1 $\leadsto$ NAT (2)

- Example:

Internal Address	Internal Port	External Address	External Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
⋮	⋮	⋮	⋮

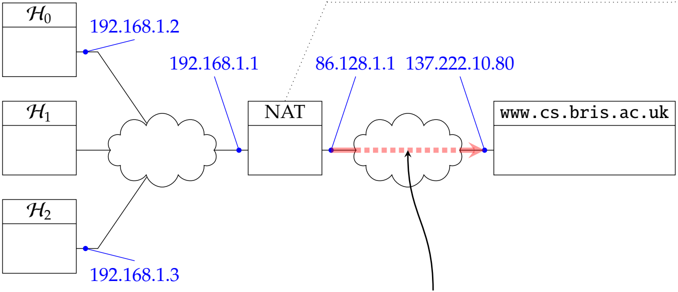


$H_{IPv4}[\text{src} = 192.168.1.2, \text{dst} = 137.222.10.80] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{src port} = 5678, \text{dst port} = 8$

Problem #1 ⇝ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

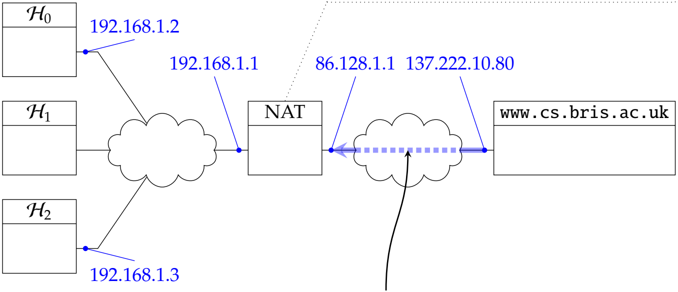


$H_{IPv4}[\text{src} = 86.120.1.1, \text{dst} = 137.222.10.80] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{src port} = 4853, \text{dst port} = 8$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

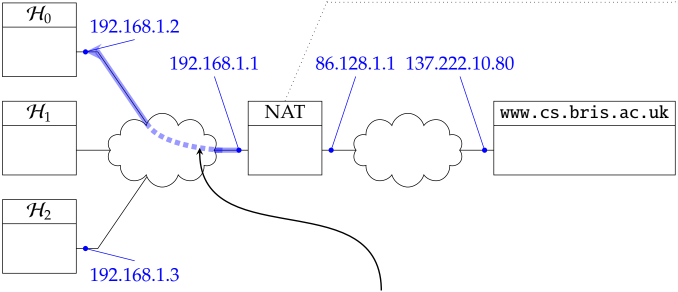


H_{IPv4}[src = 137.222.10.80, dst = 86.120.1.1] $\parallel$ H_{TCP}[syn = **true**, ack = **true**, src port = 80, dst port = 485

Problem #1 ⇝ NAT (2)

- Example:

Internal Address	Internal Port	External Address	External Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

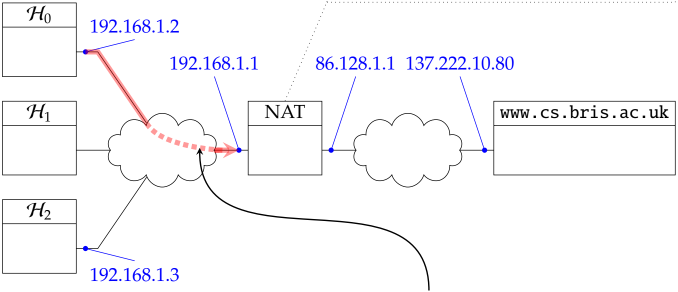


$H_{IPv4}[\text{src} = 137.222.10.80, \text{dst} = 192.168.1.2] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{ack} = \textbf{true}, \text{src port} = 80, \text{dst port} = 5678$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

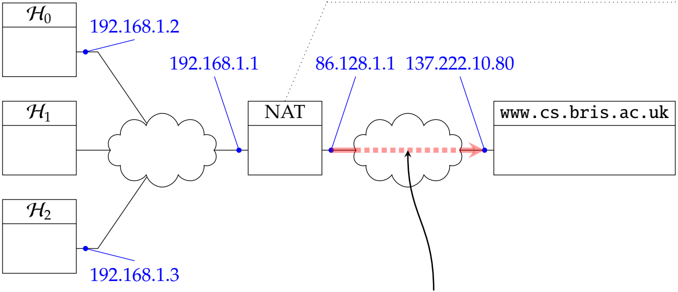


$H_{IPv4}[\text{src} = 192.168.1.2, \text{dst} = 137.222.10.80] \;\|\; H_{TCP}[\text{ack} = \textbf{true}, \text{src port} = 5678, \text{dst port} = 80$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

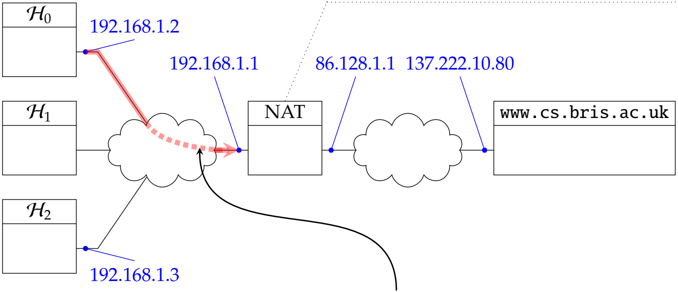


$H_{IPv4}[\text{src} = 86.120.1.1, \text{dst} = 137.222.10.80] \;\|\; H_{TCP}[\text{ack} = \textbf{true}, \text{src port} = 4853, \text{dst port} = 8$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

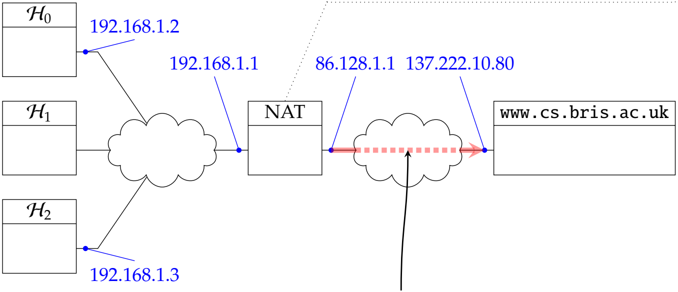


$$H_{IPv4}[\text{src} = 192.168.1.2, \text{dst} = 137.222.10.80] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{src port} = 5678, \text{dst port} = 80] \parallel \langle ..$$

Problem #1 $\leadsto$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

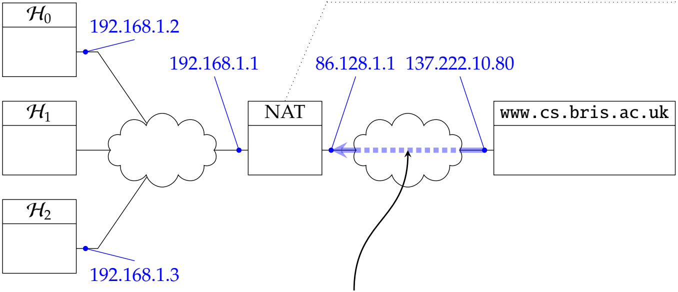


$H_{IPv4}[\text{src} = 86.120.1.1, \text{dst} = 137.222.10.80] \;\|\; H_{TCP}[\text{syn} = \textbf{true}, \text{src port} = 4853, \text{dst port} = 80] \;\|\; \langle ..$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮

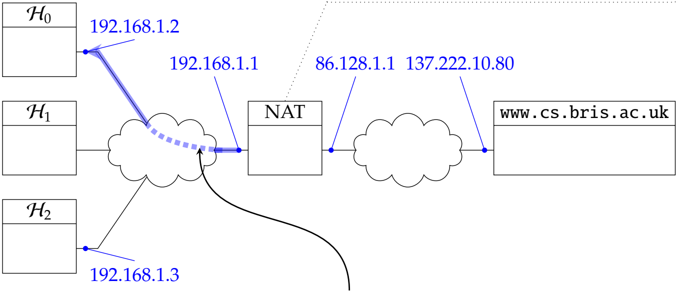


$H_{IPv4}[\text{src} = 137.222.10.80, \text{dst} = 86.120.1.1] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{ack} = \textbf{true}, \text{src port} = 80, \text{dst port} = 4853] \parallel \langle . .$

Problem #1 $\rightsquigarrow$ NAT (2)

- Example:

Internal		External	
Address	Port	Address	Port
192.168.1.3	1234	86.128.1.1	4923
192.168.1.3	5678	86.128.1.1	3811
192.168.1.2	1234	86.128.1.1	3947
192.168.1.2	5678	86.128.1.1	4853
⋮	⋮	⋮	⋮



$$H_{IPv4}[\text{src} = 137.222.10.80, \text{dst} = 192.168.1.2] \parallel H_{TCP}[\text{syn} = \textbf{true}, \text{ack} = \textbf{true}, \text{src port} = 80, \text{dst port} = 5678] \parallel \langle . .$$

- NAT appliances are examples of **middlebox** [6]:
  - exist "in" network (like routers), but
  - operate in more than network layer (i.e., don't *just* forward packets, like hosts), so
  - break layered model and connectivity, and can result in strange side-effects

  hence, NAT specifically

  - Good:
    - helps mitigate IPv4 address scarcity problem, and
    - can be easily, centrally deployed
  - Bad:
    - is difficult to extend beyond TCP,
    - requires initial outgoing connection, and
    - potentially fails if application layer depends on and/or exposes (internal) IP address

  plus, internal hosts are anonymised to some extent since their traffic flows are merged externally.

Problem #2 ⇝ DNS (1)

- Problem:

Definition (name, address and resolution)

- a (human-readable) **name** is used to *identify* the resource,
- an (machine-readable) **address** is used to *locate* the resource, and
- **resolution** (resp. **reverse resolution**) maps a name to an address (resp. address to a name).

so, we need some mechanism to perform

$$\texttt{www.cs.bris.ac.uk} \underset{\text{reverse resolve}}{\overset{\text{resolve}}{\rightleftarrows}} \text{137.222.10.80}$$

where

- the LHS is a **domain name**, and
- the RHS is an IP address.

Problem #2 ⇝ DNS (2)

- Solutions:
  1. fully centralised: pre ~ 1984 via ARPANET `hosts.txt` [8], e.g.,

     ```
     NET : 128.54.0.0 : UCSD :
     ```

     and

     ```
     HOST : 128.54.0.1 : SDCSVAX,UCSD : VAX-11/780 : UNIX : TCP/TELNET,TCP/FTP,TCP/SMTP,UDP :
     ```

  2. partially decentralised: via
     - in `/etc/networks`, e.g.,

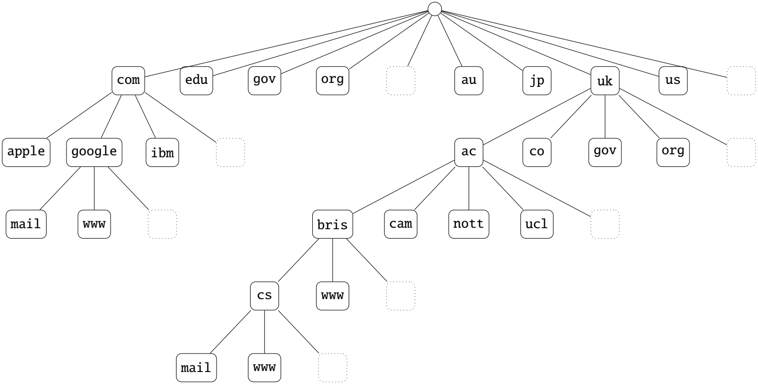
       ```
       loopback 127.0.0.0
       ```

       and
     - in `/etc/hosts`, e.g.,

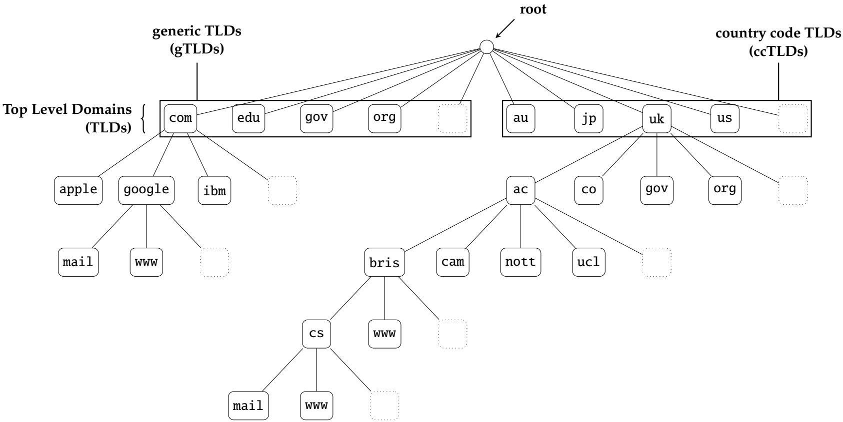
       ```
       127.0.0.1 localhost
       ```

     on a POSIX-style OS (e.g., Linux),
  3. fully decentralised: post ~ 1984 via **Domain Name System (DNS)** [9, 10] which includes
     3.1 a managed, hierarchical **name space**,
     3.2 a protocol, used to communicate queries and responses, plus
     3.3 an infrastructure, comprised of **name servers**, used to host the (distributed) database and respond to queries.

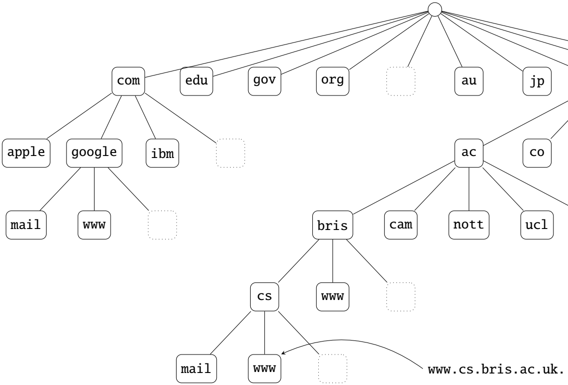
Problem #2 $\rightsquigarrow$ DNS (3)



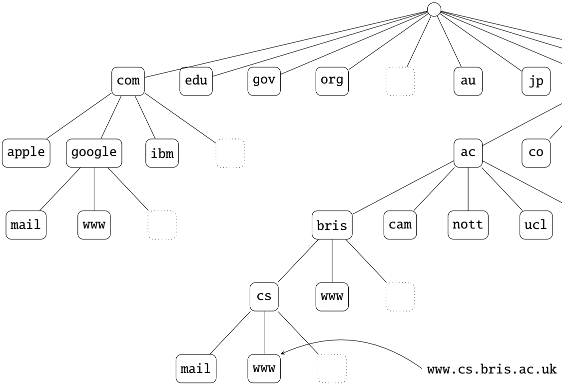
Problem #2 $\rightsquigarrow$ DNS (3)



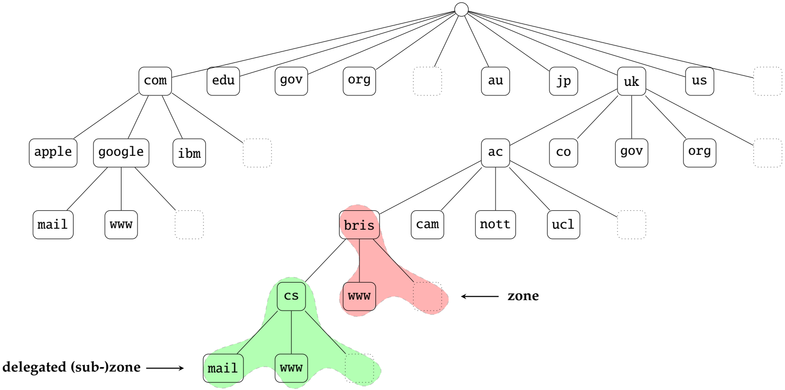
Problem #2 ⇝ DNS (3)



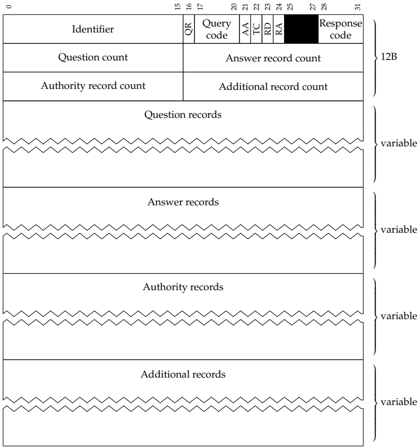
Problem #2 $\rightsquigarrow$ DNS (3)



Problem #2 ⇝ DNS (3)



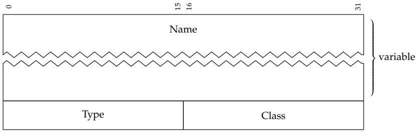
Data Structure (DNS message [10])



The data structure includes:

- A 16-bit identifier used to link query and response messages.
- 4-bit query and response codes.
- A set of flags, including
  - 1-bit **Query/Response (QR)** flag, to disambiguate queries from responses,
  - 1-bit **Authoritative Answer (AA)** flag, which marks responses that are authoritative,
  - 1-bit **Truncation Flag (TC)** which marks responses longer than the UDP 512B limit,
  - 1-bit **Recursion Desired (RD)** flag, used to request recursive resolution,
  - 1-bit **Recursion Available (RA)** flag, used to signal availability of recursive resolution.
- Some number of records (of each type).

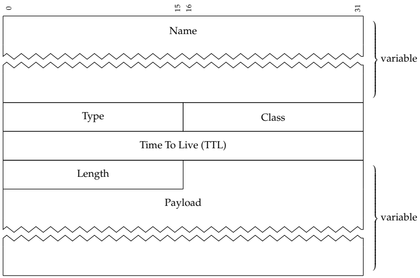
Data Structure (DNS question record [10])



The data structure includes:

- A variable length, length-prefixed name string.
- A 16-bit record type.
- A 16-bit record class.

Data Structure (DNS resource record [10])

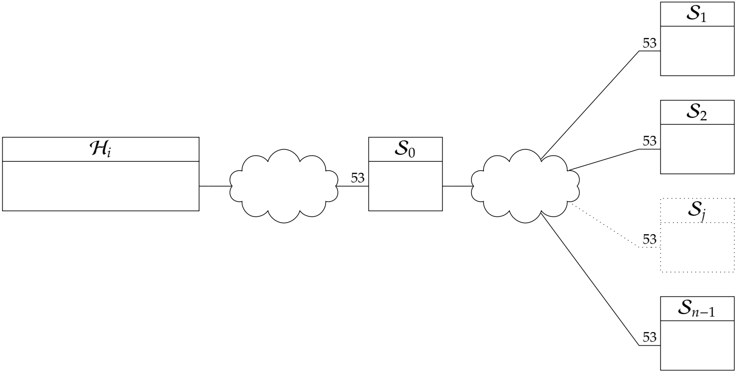


The data structure includes:

- A variable length, length-prefixed name string.
- A 16-bit record type.
- A 16-bit record class.
- A 32-bit **Time To Live (TTL)** field which controls cache retention.
- A 16-bit record length, and the record-specific payload, e.g.,
  1. a 32-bit IP address for A records,
  2. a variable length name for NS records, and
  3. a variable length name for CNAME records.

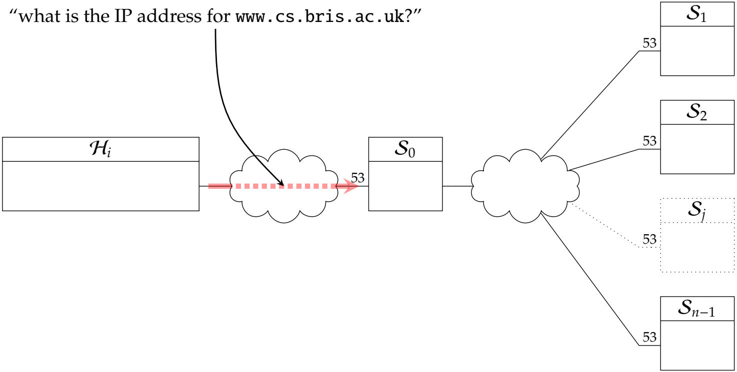
Problem #2 $\rightsquigarrow$ DNS (5)

- Example:



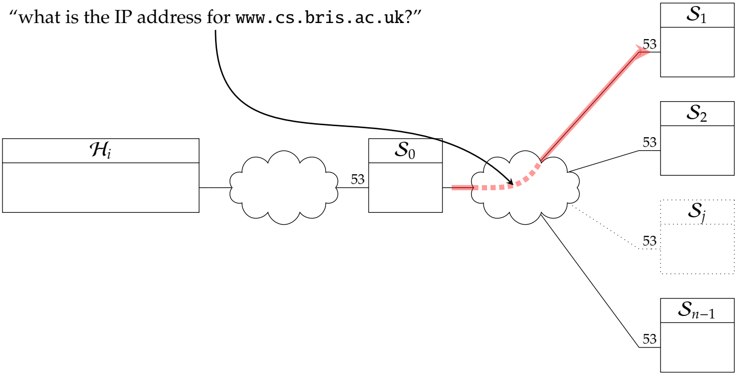
Problem #2 ⇝ DNS (5)

- Example:



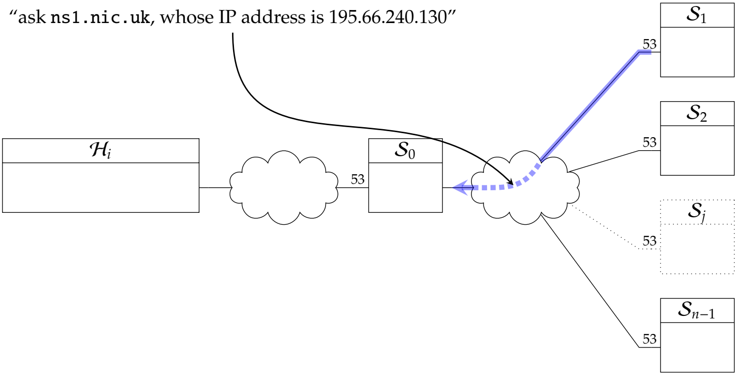
Problem #2 $\rightsquigarrow$ DNS (5)

- Example:

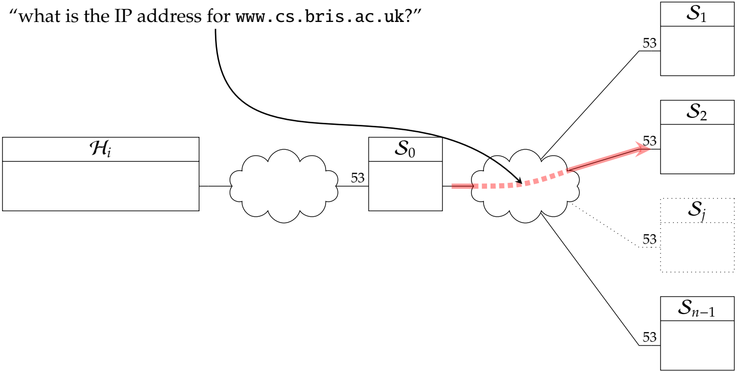


Problem #2 ⇝ DNS (5)

- Example:



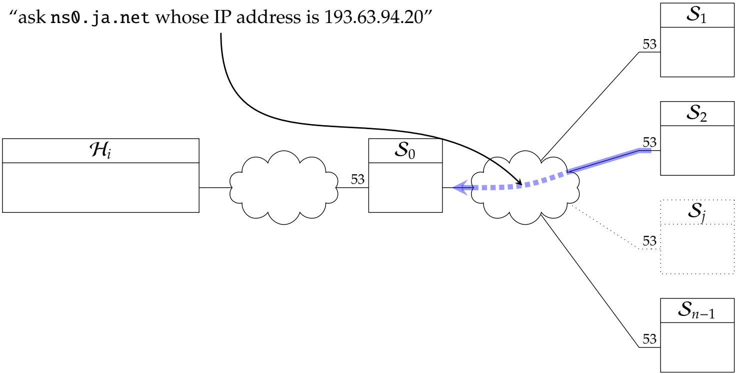
- **Example**:



"what is the IP address for `www.cs.bris.ac.uk`?"
$\mathcal{H}_i$
$\mathcal{S}_0$
$\mathcal{S}_1$
$\mathcal{S}_2$
$\mathcal{S}_j$
$\mathcal{S}_{n-1}$
53
53
53
53
53

Problem #2 ⇝ DNS (5)

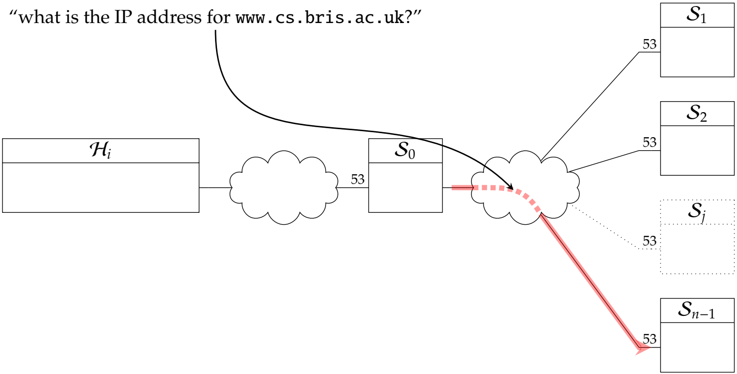
- Example:



Problem #2 ⇝ DNS (5)

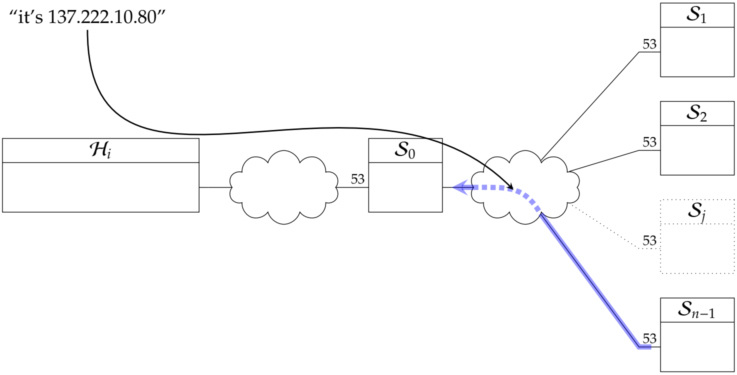
- **Example**:

“what is the IP address for `www.cs.bris.ac.uk`?”



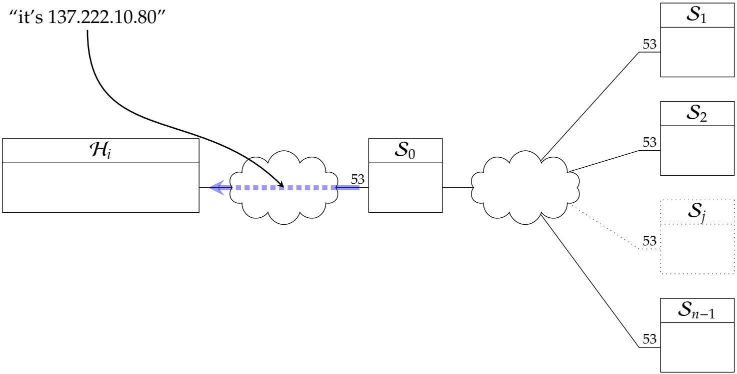
Problem #2 $\rightsquigarrow$ DNS (5)

- Example:

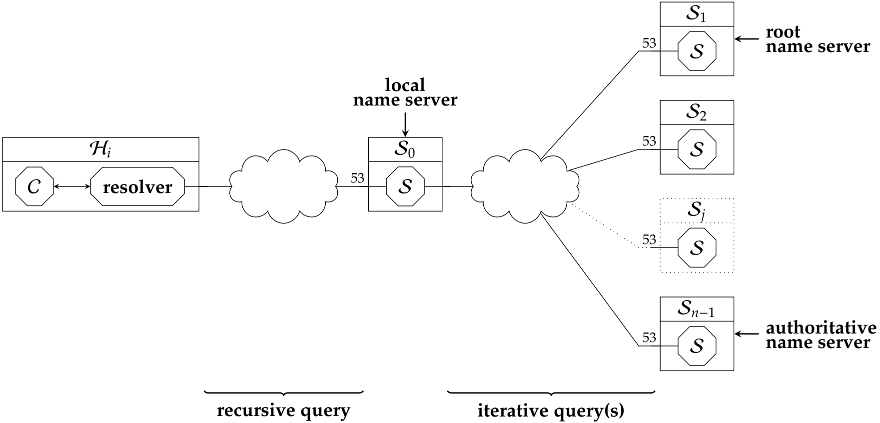


Problem #2 ⇝ DNS (5)

- Example:



- Example:


local
name server
root
name server
authoritative
name server
$\mathcal{H}_i$
$\mathcal{C}$
resolver
53
$\mathcal{S}_0$
$\mathcal{S}$
$\mathcal{S}_1$
$\mathcal{S}_2$
$\mathcal{S}_j$
$\mathcal{S}_{n-1}$
recursive query
iterative query(s)

Conclusions

- Take away points:
  - The "glue" protocols outlined here solve issues that stem from practical deployment and use.
  - DNS in particular is interesting, since
    - it highlights the tricky compromise enforced by lack of middle-ground between UDP and TCP,
    - like DHCP *users* consider it part of the network but in reality it is an application level mechanism,
    - although not *too* old, it already highlights the need for flexibility to meet changing requirements and use-cases,
    - versus other glue protocols, there is a much stronger efficiency requirement, plus
    - tangential requirements (e.g., security [5], cf. DNSSEC [4] etc.) make it a *much* larger and more challenging problem.

References

[1] Wikipedia: Domain name.
http://en.wikipedia.org/wiki/Domain_name.

[2] Wikipedia: Domain Name System (DNS).
http://en.wikipedia.org/wiki/Domain_Name_System.

[3] Wikipedia: Network address translation.
http://en.wikipedia.org/wiki/Network_address_translation.

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.
DNS security introduction and requirements.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 4033, 2005.
http://tools.ietf.org/html/rfc4033.

[5] D. Atkins and R. Austein.
Threat analysis of the Domain Name System (DNS).
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3833, 2004.
http://tools.ietf.org/html/rfc3833.

[6] B. Carpenter and S. Brim.
Middleboxes: Taxonomy and issues.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3234, 2002.
http://tools.ietf.org/html/rfc3234.

References

[7] K. Egevang and P. Francis.
The IP Network Address Translator (NAT).
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1631, 1994.
http://tools.ietf.org/html/rfc1631.

[8] E. Feinler, K. Harrenstien, Z.-S. Su, and V. White.
DoD Internet host table specification.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 810, 1982.
http://tools.ietf.org/html/rfc810.

[9] P. Mockapetris.
Domain names - concepts and facilities.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 882, 1983.
http://tools.ietf.org/html/rfc882.

[10] P. Mockapetris.
Domain names - implementation and specification.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 883, 1983.
http://tools.ietf.org/html/rfc883.