Concurrent Computing (Computer Networks)

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨Daniel.Page@bristol.ac.uk⟩

April 18, 2016

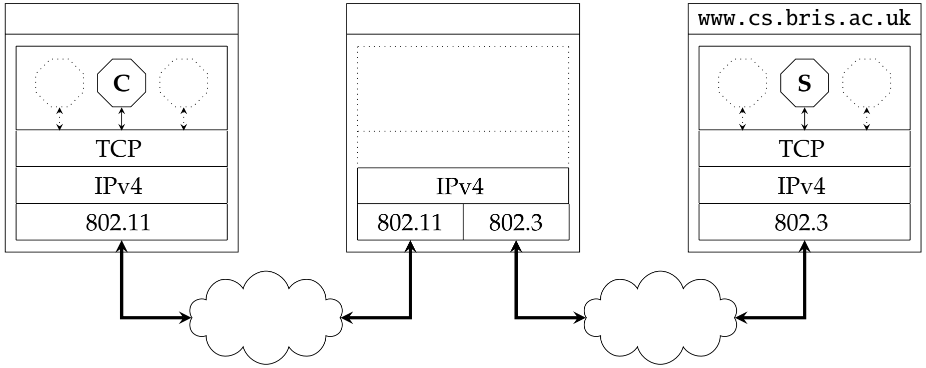
Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

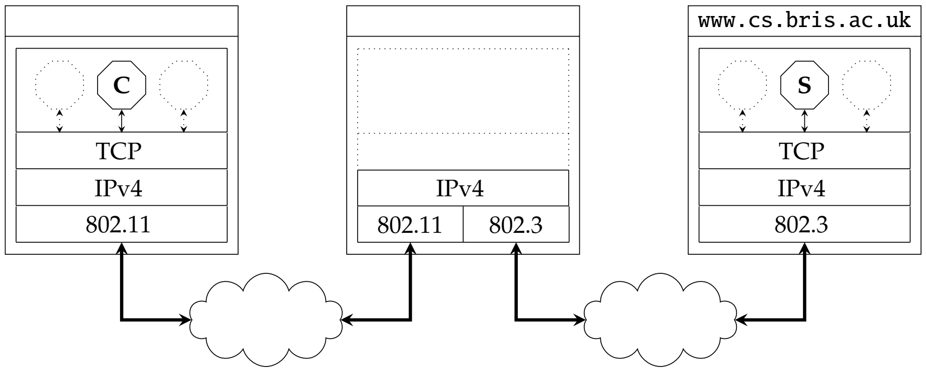
- Recall: we know how to realise



st. hosts can transmit IP packets to each other.

Notes:

- Recall: we know how to realise



st. hosts can transmit IP packets to each other ...

- ... *but*

1. how does the destination get an IP address in the first place,
2. how does the source find out the router MAC address, and
3. how are communication errors signalled by the router?

Notes:

Problem #1 ⇝ DHCP (1)

- Problem: how does some host, say $\mathcal{H}_2$, get assigned an IP address?
- Solution(s):
  1. manually assign one, *or*
  2. automatically assign one via
     - **Reverse Address Resolution Protocol (RARP)** [9],
     - **BOOTstrap Protocol (BOOTP)** [10], or
     - **Dynamic Host Configuration Protocol (DHCP)** [8]

  noting that

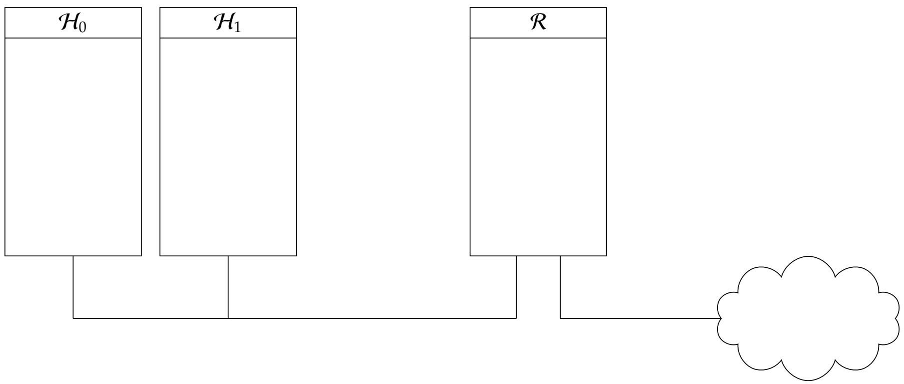
  - maintainability at scale, and
  - support for *re*assignment of addresses

  are important drivers for the latter.

Notes:

- In more detail, it's important to see that a) manually keeping track of the address assignment and then b) implementing it is fraught with logistical challenges, hence the desire for automation! Put another way, even given a relatively small address block, e.g., 137.222.102.0/24, means configuring upto 256 hosts: for larger blocks (i.e., smaller prefixes), this requires a *lot* of effort.
  Unlike MAC addresses, which can be fixed during manufacture of the NIC they are associated with, IP addresses depend on the sub-network a given host is attached to (via the NIC): they *demand* per host configuration (of the IP address *and* other parameters, e.g., the gateway router IP address), and potentially *re*configuration over time (e.g., as hosts are added and removed, which of course is *very* common in wireless networks).
- The possibility for reassignment of addresses can be important in contexts where there are a small number of addresses: this is true *full stop* in IPv4, but also for small address blocks. Imagine you have relatively small address block, e.g., 137.222.102.0/24, but more than 256 hosts. Provided that not all of the hosts want to connect at the same time, use of DHCP can allow a address to be reused once the host it is assigned to is disconnected.

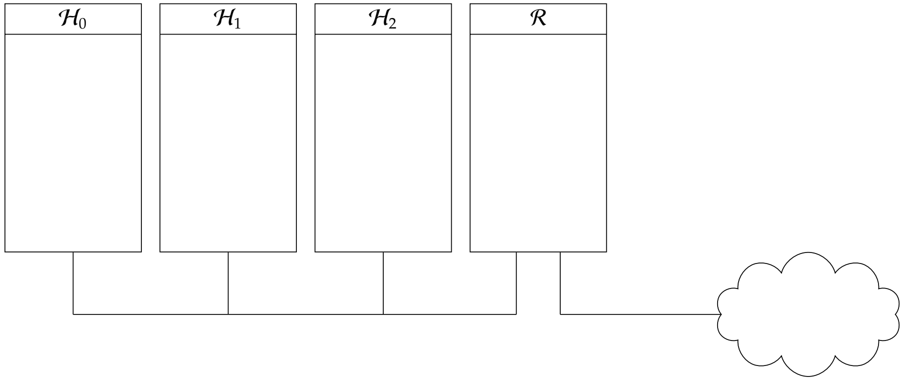
Problem #1 ⇝ DHCP (2)



Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

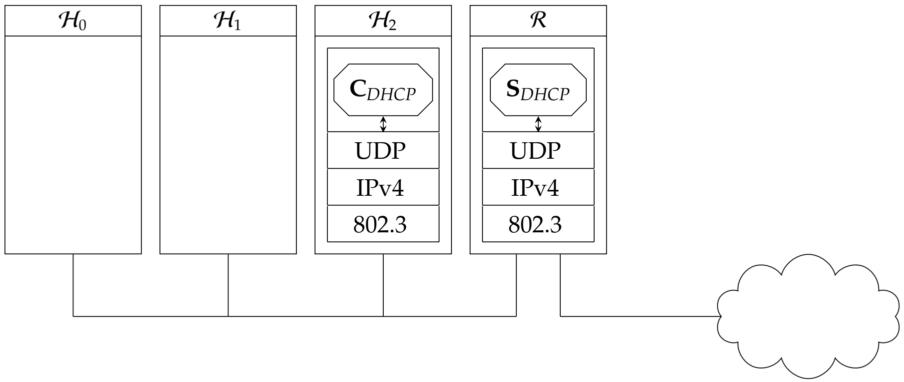
Problem #1 $\rightsquigarrow$ DHCP (2)



Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 $\rightsquigarrow$ DHCP (2)

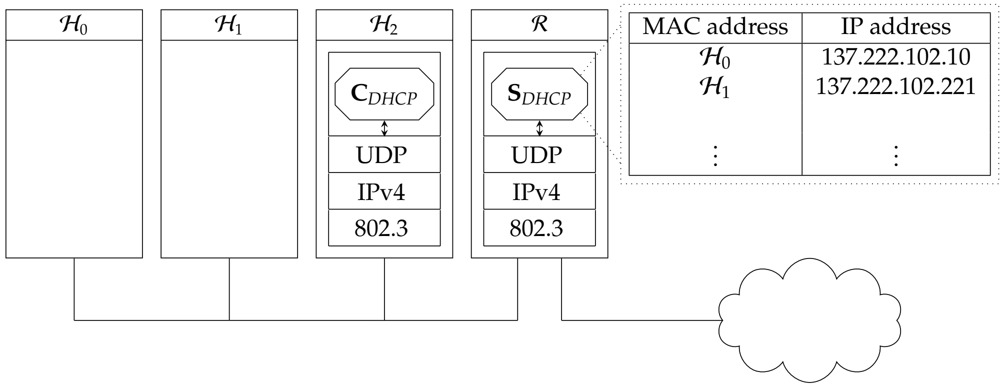


- Basic idea:
  - The DHCP protocol operates within the *application* layer, making use of UDP (and hence IP).
  - Each host executes a DHCP client that configures the network stack; a DHCP server executes somewhere on the same sub-network.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)



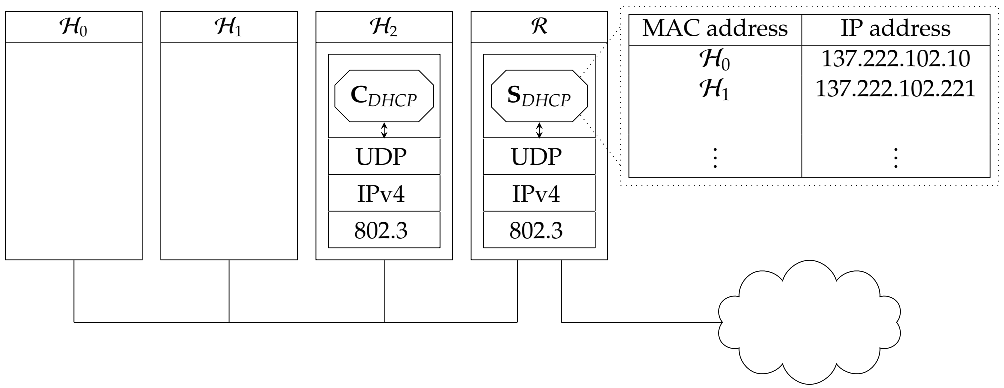
- Basic idea:
  - The server maintains a table (or **pool**) mapping keys (e.g., MAC addresses) to configurations (e.g., IP addresses); a given configuration is **leased** to a client for some period.
  - Several different strategies are possible:

Strategy	Assignment	Lease
Static	Static, pre-defined mapping	Permanent
Automatic	Dynamic, drawn from range(s)	Permanent
Dynamic	Dynamic, drawn from range(s)	Temporary

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)

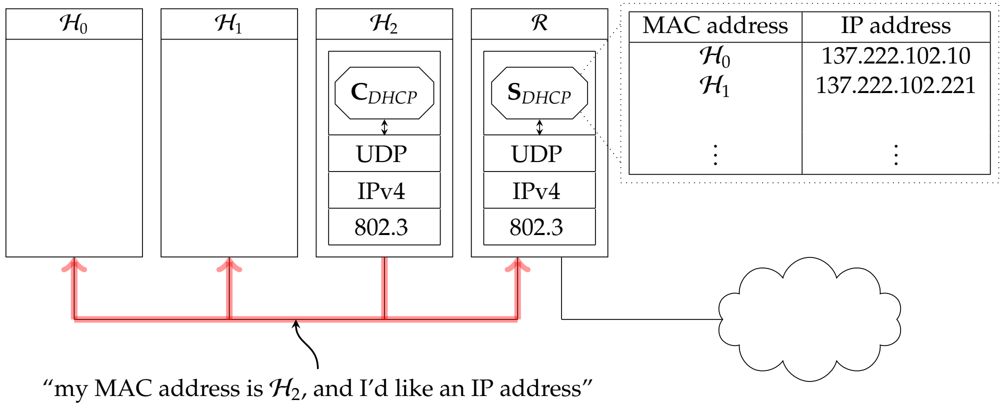


- Basic idea:
  - The end-points engage in a 4-step protocol:
    1. the client broadcasts a **discover** message,
    2. the server transmits an **offer** message,
    3. the client broadcasts a **request** message, and
    4. the server transmits an **acknowledgement** message.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)

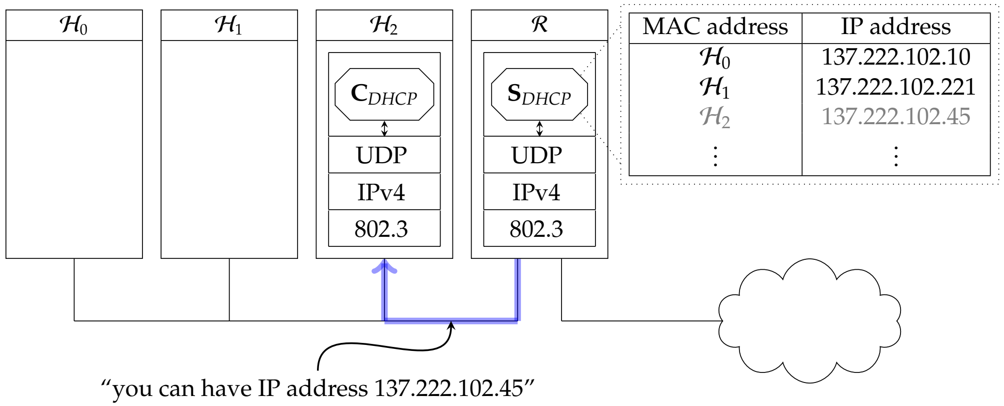


- **Basic idea:**
  - The end-points engage in a 4-step protocol:
    1. the client broadcasts a **discover** message,
    2. the server transmits an **offer** message,
    3. the client broadcasts a **request** message, and
    4. the server transmits an **acknowledgement** message.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)

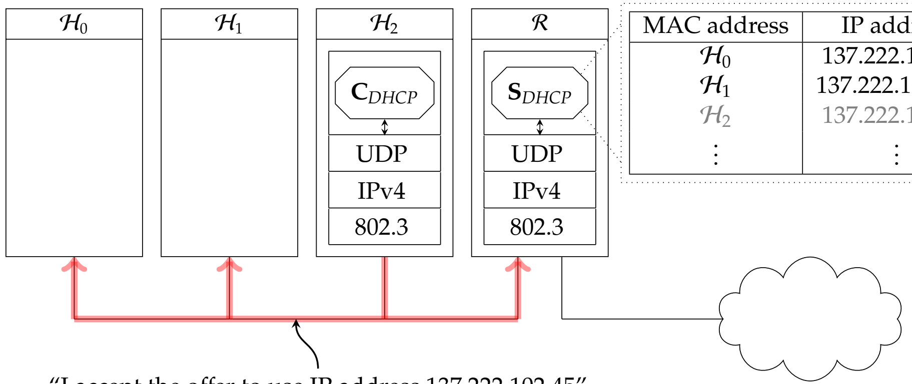


- **Basic idea:**
  - The end-points engage in a 4-step protocol:
    1. the client broadcasts a **discover** message,
    2. the server transmits an **offer** message,
    3. the client broadcasts a **request** message, and
    4. the server transmits an **acknowledgement** message.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)



MAC address	IP address
$\mathcal{H}_0$	137.222.102.10
$\mathcal{H}_1$	137.222.102.221
$\mathcal{H}_2$	137.222.102.45
⋮	⋮

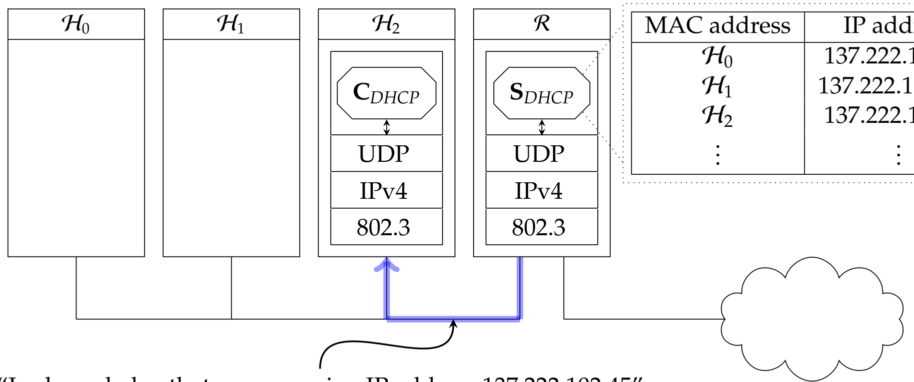
"I accept the offer to use IP address 137.222.102.45"

- Basic idea:
  - The end-points engage in a 4-step protocol:
    1. the client broadcasts a **discover** message,
    2. the server transmits an **offer** message,
    3. the client broadcasts a **request** message, and
    4. the server transmits an **acknowledgement** message.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (2)



MAC address	IP address
$\mathcal{H}_0$	137.222.102.10
$\mathcal{H}_1$	137.222.102.221
$\mathcal{H}_2$	137.222.102.45
⋮	⋮

"I acknowledge that you are using IP address 137.222.102.45"

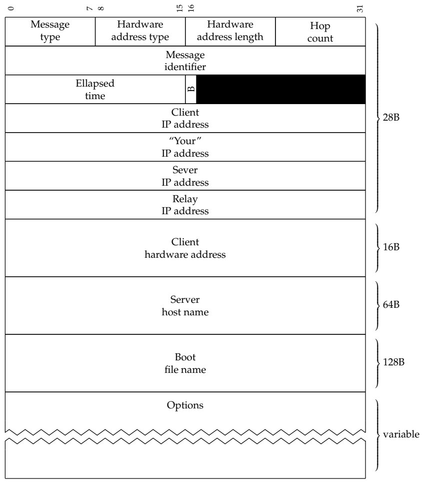
- Basic idea:
  - The end-points engage in a 4-step protocol:
    1. the client broadcasts a **discover** message,
    2. the server transmits an **offer** message,
    3. the client broadcasts a **request** message, and
    4. the server transmits an **acknowledgement** message.

Notes:

- The fact that DHCP operates in the application layer might *seem* a bit of an anomaly. The way to think about it is that from the network perspective DHCP is an application: even though it might be viewed as a protocol by the users, it configures the network stack (in the same way the user *could* do manually) rather being part of it.
- DHCP uses UDP known ports 67 and 68 for server and client respectively.
- The names used for each strategy can be a little misleading: the automatic and dynamic cases both avoid manual intervention, with a better distinction perhaps relating to the assignment being permanent or temporary (i.e., transient).
- Some caveats:
  1. Actually, there can be more than one server. This complicates the protocol a little (e.g., since a client may have to decide between more than one offer), but yields advantages such as robustness and load balancing.
  2. The server doesn't *have* to be on the same sub-network: use of so-called DHCP relay (or "helper") can act as a bridge, allowing one server to supply addresses for multiple sub-networks.

Problem #1 ⇝ DHCP (3)

Data Structure (**DHCP message** [8, Section 2])



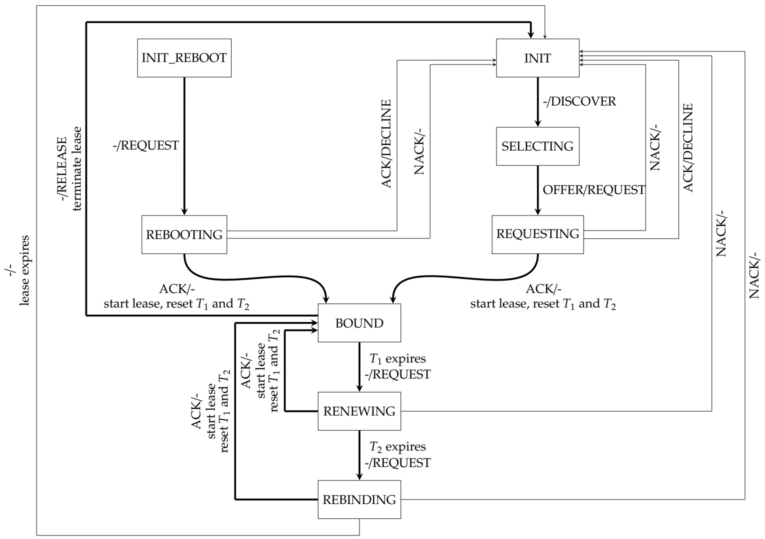
Notes:

- You might prefer the original ASCII art from [8, Section 2]:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     op (1)    |   htype (1)   |   hlen (1)    |   hops (1)    |
+---------------+---------------+---------------+---------------+
|                            xid (4)                            |
+-------------------------------+-------------------------------+
|           secs (2)            |           flags (2)           |
+-------------------------------+-------------------------------+
|                          ciaddr  (4)                          |
+---------------------------------------------------------------+
|                          yiaddr  (4)                          |
+---------------------------------------------------------------+
|                          siaddr  (4)                          |
+---------------------------------------------------------------+
|                          giaddr  (4)                          |
+---------------------------------------------------------------+
|                                                               |
|                          chaddr  (16)                         |
|                                                               |
|                                                               |
+---------------------------------------------------------------+
|                                                               |
|                          sname   (64)                         |
+---------------------------------------------------------------+
|                                                               |
|                          file    (128)                        |
+---------------------------------------------------------------+
|                                                               |
|                          options (312)                        |
+---------------------------------------------------------------+
```

Problem #1 ⇝ DHCP (4)

Algorithm (**DHCP state machine** [8, Figure 5])



Notes:

- This diagram encodes a *lot* of information:
  1. Matching the standard, it only includes states and actions wrt. the client (though there is clearly an analogous state machine for the server).
  2. The thick (resp. thin) lines denote normal (resp. unusual, or abnormal) transitions.
  3. Each transition is labelled with the event which caused it, plus the result of it (or a dash where there is no relevant result): basically the former is what the client sees, and the latter what it does.
  4. The top-right and top-left sub-graphs describes the assignment and reassignment processes; the bottom sub-graph describes the renewal and rebinding processes.
- The lease is tracked via two timers, T_1 and $T_2 > T_1$. When the T_1 timer expires, the client attempts to renew the lease with the server that offered it; when the T_2 timer expires, said server has not renewed so the client broadcasts the request to *any* server.

  Use of *two* timers caters for fact that renewal/rebinding takes some time, with typical defaults of T_1 and T_2 being $\frac{1}{2}$ and $\frac{7}{8}$ of the lease period respectively.
- Rather then successfully being (re)assigned an address, the client might fail because a) it receives an ACK but checks and the address is being used (so it then declines the address), or b) or receives a NACK (maybe because another client was successfully assigned that address before it responded).

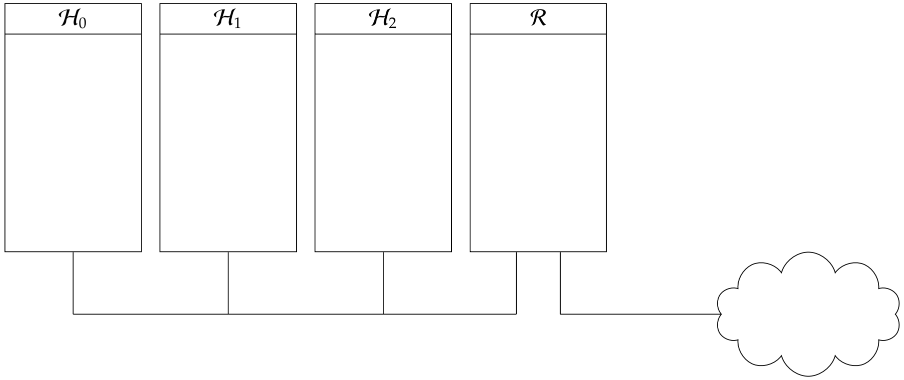
- Problem: imagine some host $\mathcal{H}_2$ wants to transmit an encapsulated IP packet

$$P = H_{802.3}[\mathrm{src} = \alpha, \mathrm{dst} = \beta] \;\|\; H_{IPv4}[\mathrm{src} = \gamma, \mathrm{dst} = \delta] \;\|\; D \;\|\; T_{802.3}$$

  to $\mathcal{H}_0$ on the same (sub-)network ... given
  - γ is the IP address of $\mathcal{H}_2$,
  - δ is the IP address of $\mathcal{H}_0$,
  - α is the MAC address of $\mathcal{H}_2$, and
  - β is the MAC address of $\mathcal{H}_0$

  how does $\mathcal{H}_2$ know, or how can it find out, β?
- Solution: it uses **Address Resolution Protocol (ARP)** [11].

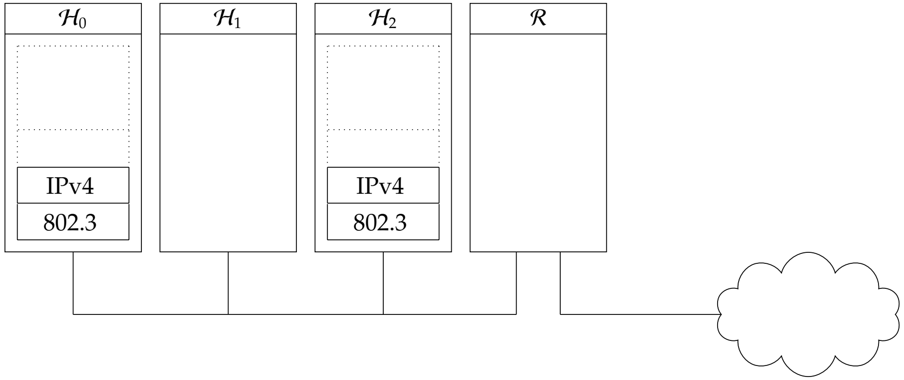
Notes:



Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 ⇝ ARP

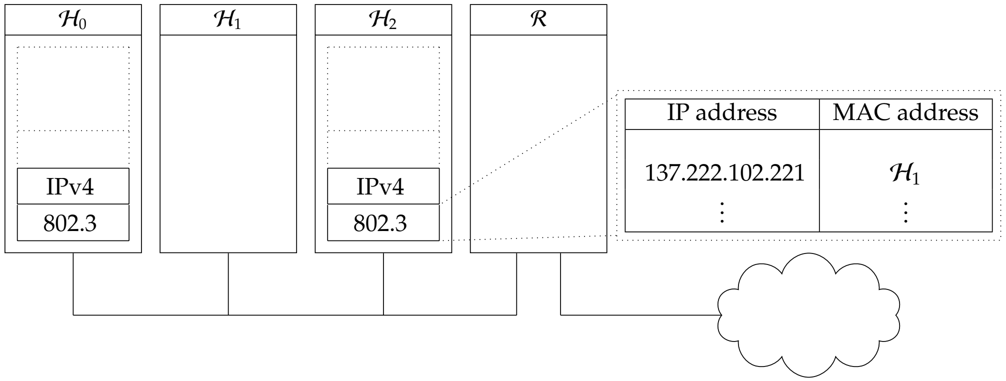


- Basic idea:
  - ARP operates within the *link* layer, servicing requests from the network layer (i.e., IP).
  - In this case, ARP messages will therefore be encapsulated in 802.3 frames and processed by the 802.3 MAC sub-layer.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 ⇝ ARP

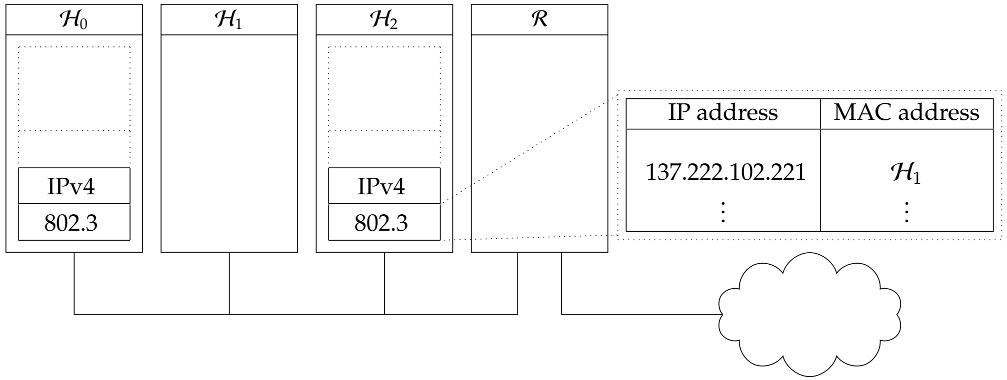


- Basic idea:
  - Using ARP for *every* IP address used would represent too high an overhead, so each host maintains a **cache**.
  - Cached entries can of course become stale (i.e., the mapping becomes invalid) over time, so are periodically refreshed.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 $\rightsquigarrow$ ARP



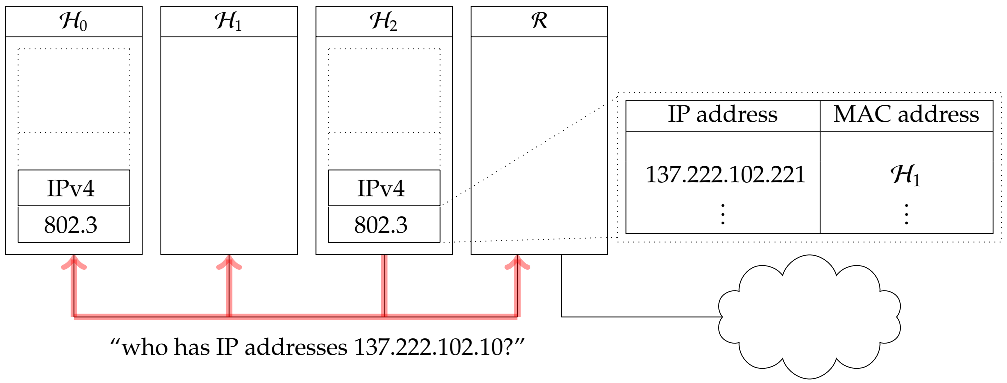
- Basic idea:
  - The end-points engage in a 1- or 3-step protocol:
    1. the requester can use a cached entry if available, otherwise
    2. the requester broadcasts a **request** message, and
    3. the requestee transmits a **response** message

    where a lack of response in step 3 prompts the requester to retry (including a back-off), then eventually give up.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 $\rightsquigarrow$ ARP



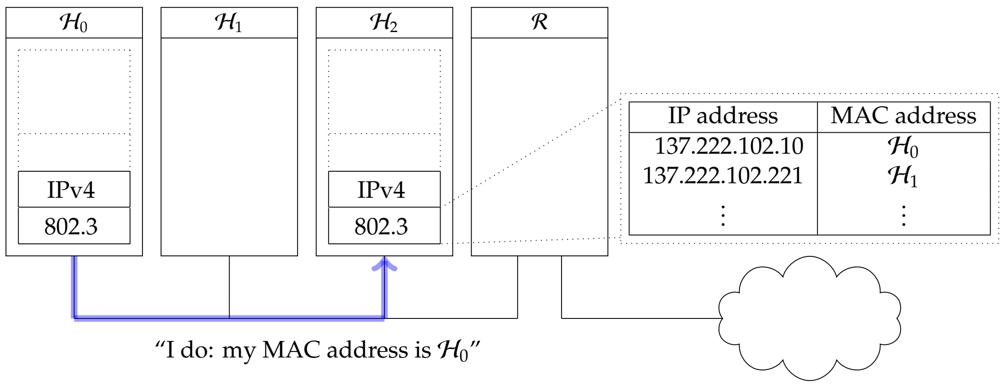
- Basic idea:
  - The end-points engage in a 1- or 3-step protocol:
    1. the requester can use a cached entry if available, otherwise
    2. the requester broadcasts a **request** message, and
    3. the requestee transmits a **response** message

    where a lack of response in step 3 prompts the requester to retry (including a back-off), then eventually give up.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 $\rightsquigarrow$ ARP



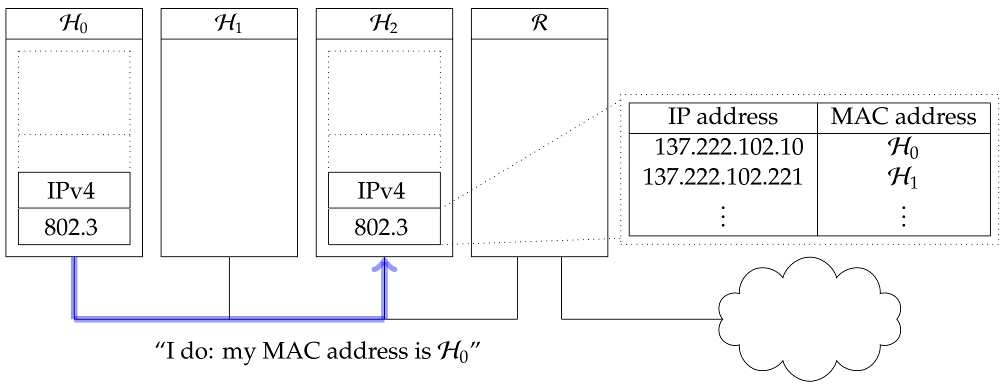
- Basic idea:
  - The end-points engage in a 1- or 3-step protocol:
    1. the requester can use a cached entry if available, otherwise
    2. the requester broadcasts a **request** message, and
    3. the requestee transmits a **response** message

    where a lack of response in step 3 prompts the requester to retry (including a back-off), then eventually give up.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 $\rightsquigarrow$ ARP



- Basic idea:
  - The end-points engage in a 1- or 3-step protocol:
    1. the requester can use a cached entry if available, otherwise
    2. the requester broadcasts a **request** message, and
    3. the requestee transmits a **response** message

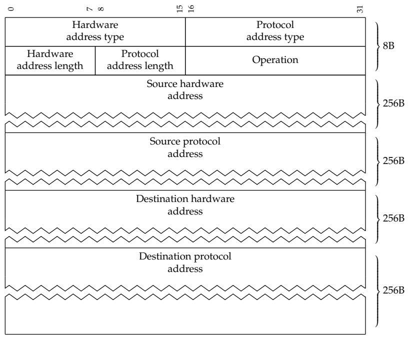
    where a lack of response in step 3 prompts the requester to retry (including a back-off), then eventually give up.

Notes:

- It isn't explicit on the slide, but of course when the requestee receives a request message *it* can cache the MAC address of the requester and hence (potentially) avoid the opposite request.
- There are various reasons a cache entry could become stale, including the fact that DHCP leases expire (so a given IP address may be reallocated to another hosts) and hosts may be replaced/upgraded (so their IP address is the same, even if the MAC address in their NIC changes).
- The lack of any central control, or methods of authentication, means there is nothing to prevent false (or "spoofed") ARP messages or for hosts to act maliciously (e.g., ignore requests).

Problem #2 $\leadsto$ ARP

Data Structure (ARP message [11])



Notes:

Problem #3 $\leadsto$ ICMP

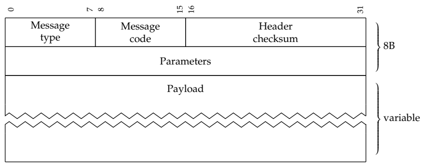
- Problem: how are communication errors signalled?
- Solution: via **Internet Control Message Protocol (ICMP)** [12], which actually covers
  1. error reporting,
  2. network control, *and*
  3. information retrieval

  operations.

Notes:

Problem #3 $\rightsquigarrow$ ICMP

Data Structure (ICMP message [12])



- Each ICMP message is encapsulated in an IP packet:
  - the source is the host/router where the error was detected, while
  - the destination is the host which transmitted the packet

  noting

  - the 16-bit message type and code specify the message type,
  - the payload of some messages captures the header *and* 64 bits of payload relating to the trigger packet.

Notes:

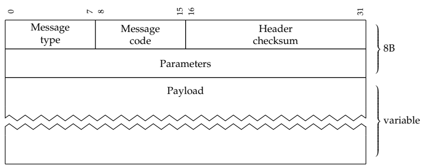
- IANA maintain a definitive set of assigned field values, e.g., for the message type and code fields, at

  `http://www.iana.org/assignments/icmp-parameters`

- One obvious issue with using IPv4 to transmit ICMP messages is that IPv4 is unreliable ... that might be how we got an error in the first place! Put another way, there is no stronger guerentee that ICMP messages will be transmitted correctly than the original packets they relate to.

Problem #3 $\rightsquigarrow$ ICMP

Data Structure (ICMP message [12])



- (Selected) examples:

$$
\left.\begin{array}{lcl}
03_{(16)} \parallel 00_{(16)} & \mapsto & \text{destination network unreachable} \\
03_{(16)} \parallel 01_{(16)} & \mapsto & \text{destination host unreachable} \\
03_{(16)} \parallel 06_{(16)} & \mapsto & \text{destination network unknown} \\
03_{(16)} \parallel 07_{(16)} & \mapsto & \text{destination host unknown} \\
03_{(16)} \parallel 04_{(16)} & \mapsto & \text{fragmentation required, but prevented} \\
0B_{(16)} \parallel 00_{(16)} & \mapsto & \text{time exceeded (TTL)} \\
0B_{(16)} \parallel 01_{(16)} & \mapsto & \text{time exceeded (fragmentation reassembly)}
\end{array}\right\} \text{error reporting}
$$

$$
\left.\begin{array}{lcl}
09_{(16)} \parallel 00_{(16)} & \mapsto & \text{router advertisement [7]} \\
10_{(16)} \parallel 00_{(16)} & \mapsto & \text{router solicitation [7]}
\end{array}\right\} \text{network control}
$$

$$
\left.\begin{array}{lcl}
08_{(16)} \parallel 00_{(16)} & \mapsto & \text{echo request} \\
00_{(16)} \parallel 00_{(16)} & \mapsto & \text{echo response}
\end{array}\right\} \text{information retrieval}
$$

Notes:

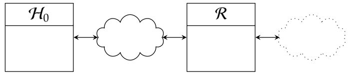
- IANA maintain a definitive set of assigned field values, e.g., for the message type and code fields, at

  `http://www.iana.org/assignments/icmp-parameters`

- One obvious issue with using IPv4 to transmit ICMP messages is that IPv4 is unreliable ... that might be how we got an error in the first place! Put another way, there is no stronger guerentee that ICMP messages will be transmitted correctly than the original packets they relate to.

Problem #3 $\rightsquigarrow$ ICMP

- Example: given some (inter-)connected hosts, e.g.,

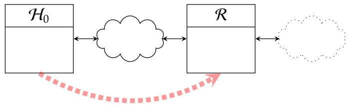


what happens if $\mathcal{H}_0$ transmits a packet to an unknown host $\mathcal{H}_?$?

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Example: given some (inter-)connected hosts, e.g.,



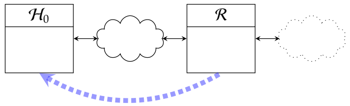
$P = H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_?] \parallel D$

what happens if $\mathcal{H}_0$ transmits a packet to an unknown host $\mathcal{H}_?$?

Notes:

Problem #3 ⇝ ICMP

- Example: given some (inter-)connected hosts, e.g.,



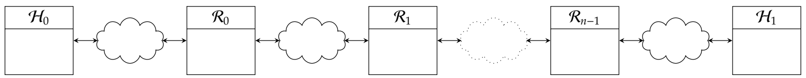
$P' = H_{IPv4}[\text{src} = \mathcal{R}, \text{dst} = \mathcal{H}_0, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{host unknown}, \text{payload} = P]$

what happens if $\mathcal{H}_0$ transmits a packet to an unknown host $\mathcal{H}_?$?

Notes:

Problem #3 ⇝ ICMP

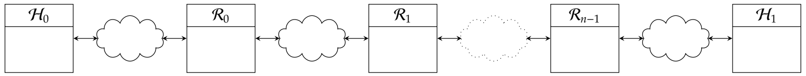
- Challenge: given some (inter-)connected hosts, e.g.,



how can $\mathcal{H}_0$ test whether $\mathcal{H}_1$ is contactable, and the RTT for communication with it?

Notes:

- Challenge: given some (inter-)connected hosts, e.g.,



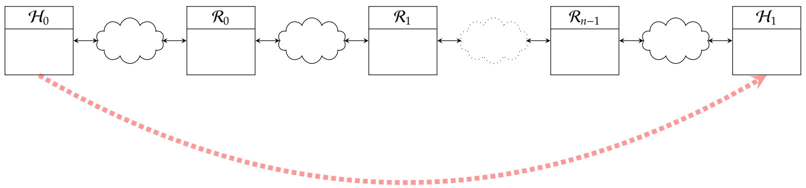
how can $\mathcal{H}_0$ test whether $\mathcal{H}_1$ is contactable, and the RTT for communication with it?

- Solution: `ping`.

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_1, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{echo request}]$

how can $\mathcal{H}_0$ test whether $\mathcal{H}_1$ is contactable, and the RTT for communication with it?

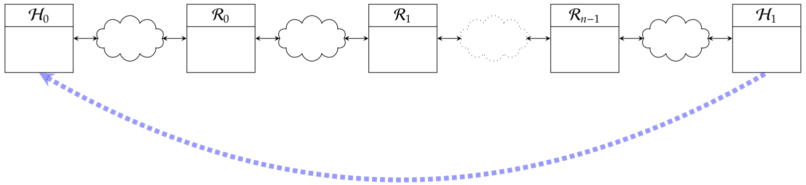
- Solution: `ping`.

Notes:

Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks) git # c2cb12a @ 2016-04-15 University of BRISTOL

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{H}_1, \text{dst} = \mathcal{H}_0, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{echo response}]$

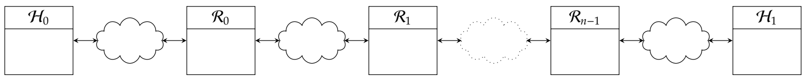
how can $\mathcal{H}_0$ test whether $\mathcal{H}_1$ is contactable, and the RTT for communication with it?

- Solution: `ping`.

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,

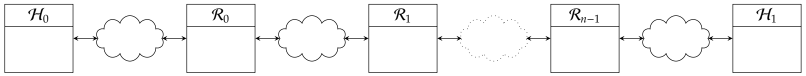


how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



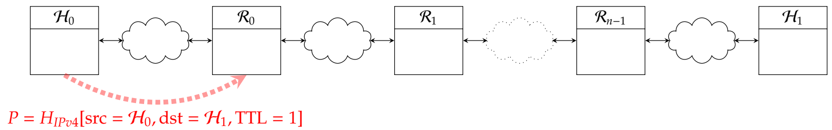
how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

- Solution: `traceroute`, which (ab)uses the TTL feature.

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



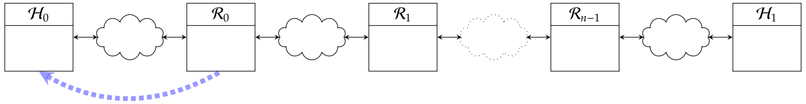
how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

- Solution: `traceroute`, which (ab)uses the TTL feature.

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{R}_0, \text{dst} = \mathcal{H}_0, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{TTL expired}]$

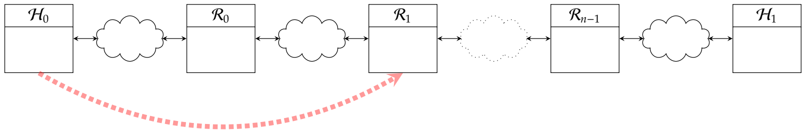
  how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0.$$

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_1, \text{TTL} = 2]$

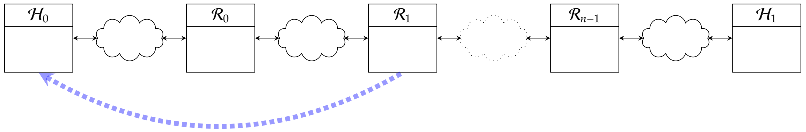
  how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0.$$

Notes:

Problem #3 ⇝ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{R}_1, \text{dst} = \mathcal{H}_0, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{TTL expired}]$

how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

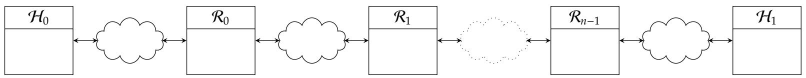
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1.$$

Notes:

Problem #3 ⇝ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



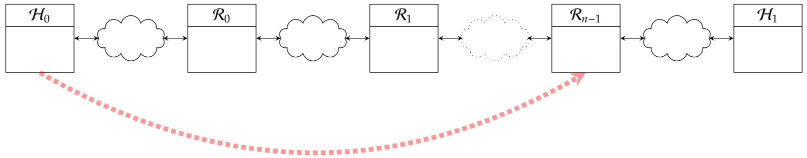
how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \cdots.$$

Notes:

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_1, \text{TTL} = n]$

how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

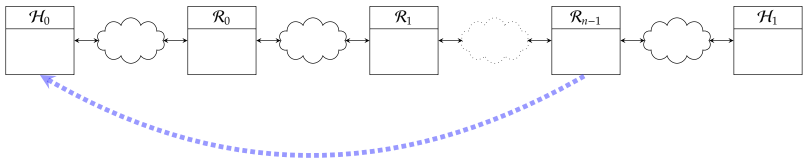
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \cdots.$$

Notes:

Problem #3 $\rightsquigarrow$ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



$P = H_{IPv4}[\text{src} = \mathcal{R}_{n-1}, \text{dst} = \mathcal{H}_0, \text{protocol} = \text{ICMP}] \parallel H_{ICMP}[\text{type} \parallel \text{code} = \text{TTL expired}]$

how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?

- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

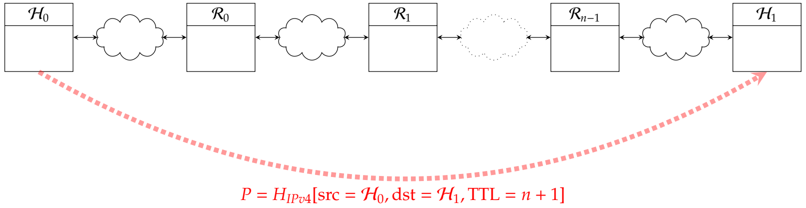
$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \cdots \rightarrow \mathcal{R}_{n-1}.$$

Daniel Page (Daniel.Page@bristol.ac.uk)
Concurrent Computing (Computer Networks) git # c2cb12a @ 2016-04-15 University of BRISTOL

Notes:

Problem #3 ⇝ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



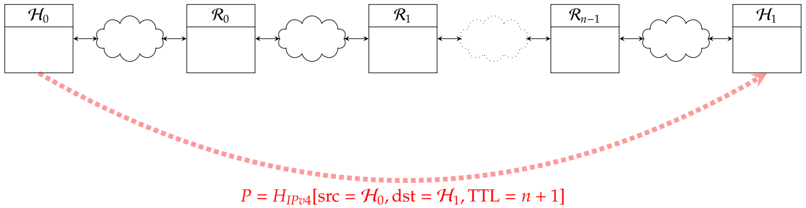
  how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \cdots \rightarrow \mathcal{R}_{n-1}.$$

Notes:

Problem #3 ⇝ ICMP

- Challenge: given some (inter-)connected hosts, e.g.,



  how can $\mathcal{H}_0$ discover the route packets take when transmitted to $\mathcal{H}_1$?
- Solution: `traceroute`, which (ab)uses the TTL feature, st. the path discovered is

$$\mathcal{H}_0 \rightarrow \mathcal{R}_0 \rightarrow \mathcal{R}_1 \rightarrow \cdots \rightarrow \mathcal{R}_{n-1} \rightarrow \mathcal{H}_1.$$

Notes:

Conclusions

Notes:

- Take away points:
  - The “glue” protocols outlined here solve issues that stem from practical deployment and use.
  - Applications such as `traceroute` are, in a sense, a by-product of flexibility in protocols such as ICMP.
  - There *are* scenarios, e.g., ARP, where host can disrupt (either maliciously, or by accident) normal operation ...
  - ... a set of requirements [6, 5] mandates what Internet hosts *must* implement.
  - DHCP and ARP are instances of more general **discovery protocols**; in such examples, broadcast capability is often an advantage.

References

Notes:

[1] Wikipedia: Address Resolution Protocol (ARP).
http://en.wikipedia.org/wiki/Address_Resolution_Protocol.

[2] Wikipedia: Dynamic Host Configuration Protocol (DHCP).
http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol.

[3] Wikipedia: Internet Control Message Protocol (ICMP).
http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[4] S. Alexander and R. Droms.
DHCP options and BOOTP vendor extensions.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 2132, 1997.
http://tools.ietf.org/html/rfc2132.

[5] R. Braden.
Requirements for Internet hosts – application and support.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1123, 1989.
http://tools.ietf.org/html/rfc1123.

[6] R. Braden.
Requirements for Internet hosts – communication layers.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1122, 1989.
http://tools.ietf.org/html/rfc1122.

[7] S. Deering.
ICMP router discovery messages.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1256, 1991.
http://tools.ietf.org/html/rfc1256.

[8] R. Droms.
Dynamic Host Configuration Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1541, 1993.
http://tools.ietf.org/html/rfc1541.

[9] R. Finlayson, T. Mann, J. Mogul, and M. Theimer.
A Reverse Address Resolution Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 903, 1984.
http://tools.ietf.org/html/rfc903.

[10] J. Gilmore.
Bootstrap protocol (BOOTP).
Internet Engineering Task Force (IETF) Request for Comments (RFC) 951, 1985.
http://tools.ietf.org/html/rfc951.

[11] D.C. Plummer.
An Ethernet address resolution protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 826, 1982.
http://tools.ietf.org/html/rfc826.

Notes:

[12] J. Postel.
Internet Control Message Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 792, 1981.
http://tools.ietf.org/html/rfc792.

Notes: