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 11, 2016

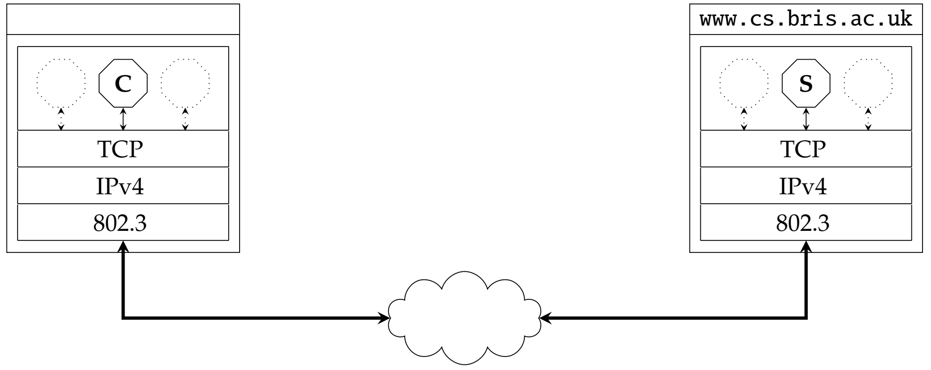
Keep in mind there are *two* PDFs available (of which this is the latter):

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:

COMS20001 lecture: week #21



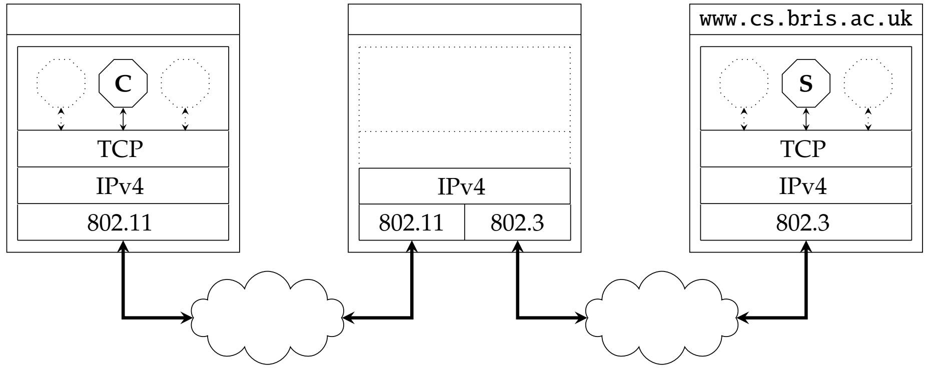
- Goal: investigate the **network layer** (or *inter*-networking more generally) e.g.,
  1. addressing,
  2. forwarding,
  3. routing

  st. we can transmit (structured) **packets** through an inter-network of multiple connected LANs.

Notes:

- One might ask why we need a network layer at all. For example, why not just construct an inter-network using switches? There are (at least) three good reasons *not* to do so:
  1. The use of switches does not scale, for (at least) two reasons: a) such an approach depends on the ability to broadcast (which is not realistic across the entire globe), and b) the mapping table which is maintained would grow very large (since it may need to include *all* hosts, rather than the limited number no a LAN).
  2. Modulo use of some types of bridge, switch-based LANs do not allow different network types (wrt. the MAC layer) to be connected: one cannot simple connect an 802.3 LAN with an 802.11 LAN in this way, which *is* obviously attractive in general.
  3. In order to make the most efficient use of available bandwidth, it is attractive to *plan* routes (globally). Put simply, even when fixed wrt. the issue of cycles, backward learning produces a solution rather than necessarily a *good* solution.

COMS20001 lecture: week #21



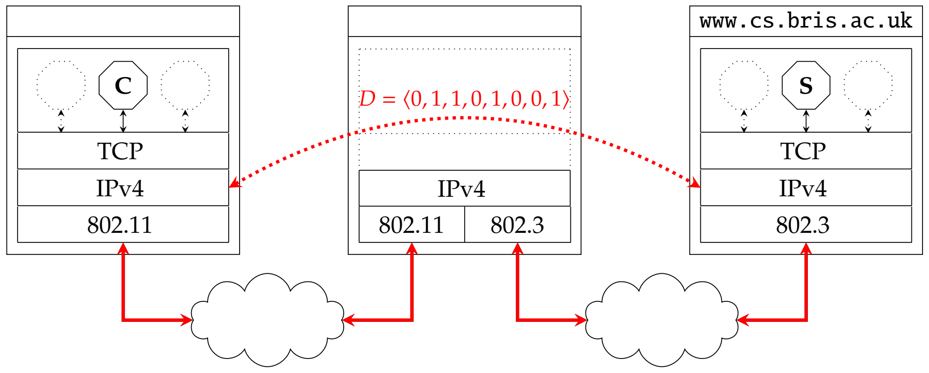
- Goal: investigate the **network layer** (or *inter*-networking more generally) e.g.,
  1. addressing,
  2. forwarding,
  3. routing

  st. we can transmit (structured) **packets** through an inter-network of multiple connected LANs.

Notes:

- One might ask why we need a network layer at all. For example, why not just construct an inter-network using switches? There are (at least) three good reasons *not* to do so:
  1. The use of switches does not scale, for (at least) two reasons: a) such an approach depends on the ability to broadcast (which is not realistic across the entire globe), and b) the mapping table which is maintained would grow very large (since it may need to include *all* hosts, rather than the limited number no a LAN).
  2. Modulo use of some types of bridge, switch-based LANs do not allow different network types (wrt. the MAC layer) to be connected: one cannot simple connect an 802.3 LAN with an 802.11 LAN in this way, which *is* obviously attractive in general.
  3. In order to make the most efficient use of available bandwidth, it is attractive to *plan* routes (globally). Put simply, even when fixed wrt. the issue of cycles, backward learning produces a solution rather than necessarily a *good* solution.

COMS20001 lecture: week #21



$F = H_{802.11} \parallel H_{IPv4} \parallel \langle 0,1,1,0,1,0,0,1 \rangle \parallel T_{802.11}$

- Goal: investigate the **network layer** (or *inter*-networking more generally) e.g.,
  1. addressing,
  2. forwarding,
  3. routing

  st. we can transmit (structured) **packets** through an inter-network of multiple connected LANs.

Notes:

- One might ask why we need a network layer at all. For example, why not just construct an inter-network using switches? There are (at least) three good reasons *not* to do so:
  1. The use of switches does not scale, for (at least) two reasons: a) such an approach depends on the ability to broadcast (which is not realistic across the entire globe), and b) the mapping table which is maintained would grow very large (since it may need to include *all* hosts, rather than the limited number no a LAN).
  2. Modulo use of some types of bridge, switch-based LANs do not allow different network types (wrt. the MAC layer) to be connected: one cannot simple connect an 802.3 LAN with an 802.11 LAN in this way, which *is* obviously attractive in general.
  3. In order to make the most efficient use of available bandwidth, it is attractive to *plan* routes (globally). Put simply, even when fixed wrt. the issue of cycles, backward learning produces a solution rather than necessarily a *good* solution.

An Aside: "This Jen, is *the Internet*"

http://www.youtube.com/watch?v=iDbyYGrswtg

Notes:

An Aside: "This Jen, is *the Internet*"

http://www.youtube.com/watch?v=iDbyYGrswtg

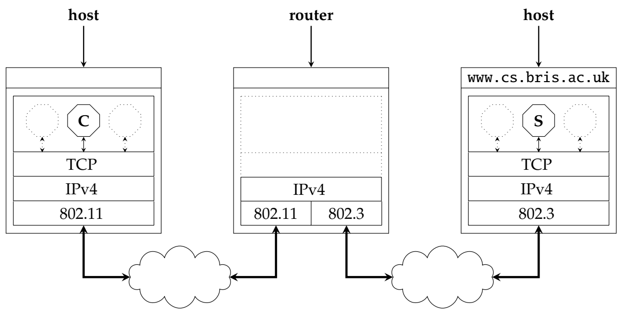
Notes:

Concepts (1)

Definition (host and router)

An inter-network is formed from nodes termed

1. **End Systems (ES)** (or **hosts**), and
2. **Intermediate Systems (IS)** (or **routers**).



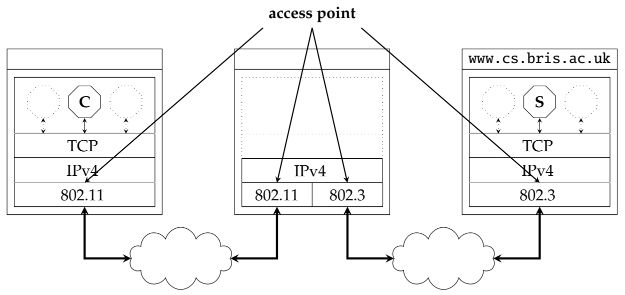
Notes:

Concepts (1)

Definition (address)

Each network layer **access point** is identified by an **address**:

- unlike link layer addresses, network layer addresses need to be *globally* unique,
- one address per host may be insufficient: we need an address per point of access, so per NIC for example.



Notes:

Concepts (1)

Definition (routing vs. forwarding)

Transmission of packets through an inter-network is based on two tasks, namely

1. **routing**, i.e., looking at the destination address in a packet and deciding on the next **hop** (toward said destination), and
2. **forwarding**, i.e., actually transmitting the packet to the next hop.

Notes:

IPv4 (1)

- **Internet Protocol (IP)** [13] is an inter-networking lingua franca ...
- ... rather than services per se, it deals with *abstraction*: it offers
  - unicast (one-to-one),
  - broadcast (one-to-all),
  - multicast (one-to-some), and
  - anycast (one-to-any)

  packet delivery models, each of which is
  - unreliable,
  - connection-less (i.e., stateless), and
  - "best effort" (i.e., no QoS guarantees).

  and, crucially, allows hererogenaity wrt. lower and higher layers.

Notes:

- The delivery models IP offers are interesting; the former are somewhat obvious, but the latter support less obvious use-cases. In all of the broadcast, multicast and anycast there is a *set* of destinations: for broadcast that set is *all* other hosts, with multicast it is a specified set of hosts. The final model, anycast, could be viewed as a combination of unicast and multicast. That is, a set of hosts is specified as potential destinations, but only one will actually receive a transmitted packet. The concept is arguable more general as written, but the most obvious metric to decide which destination is selected is distance: allowing a source to transmit packets to a single destination address but have them delivered to a geographically local alias allows fairly advanced, and attractive forms of load balancing (e.g., in DNS).
- A lingua franca is a "bridging" language used by people to communicate when they do no share a common language. If you recall the IP protocol suite diagram, this is essentially what IP is doing: it showed say n higher layers and m lower layers, but 1 IP layer at the center point through which they communicated encapsulated PDUs. IP can be described as
  - requiring little from lower, link layer, and
  - offering little to higher, transport layer.

  Put another way, rather than delivering services itself (other then the ability to transmit packets, and be informed when packets are received), the central goal of IP is to "glue together" the other layers and so enable whatever services *they* can deliver.
- Following this argument further, the properties of IP imply
  - packets might be delivered out-of-order, or not at all,
  - limited requirement from lower layers, meaning IP can run over more or less *any* link layer technology [21], and
  - higher layers must add services or functionality on top of IP.

  When we loop at the higher, transport layer (TCP in particular), some of the services or functionality introduced on top of IP will become clear: examples include flow and error (e.g., non-delivery of data) control, and support for connection-based communication.

IPv4 (2) – Addressing

Definition (IP address)

Each access point is assigned a unique identifier called an **IP address**. Note that

- a given address x is simply an n bit integer,
- for IPv4 we have $n = 32$,
- IP addresses are often written in dotted-decimal notation, i.e., 4 decimal integers $0 \le \hat{x}_i < 256$ each representing 8 bits of the address x.

Definition (IP prefix)

IP addresses are hierarchical: a **prefix** l defines a block of addresses, called a **sub-network** (or **sub-net**), by dividing the address into

1. an l-bit **network identifier**, and
2. an $(n-l)$-bit **host identifier**

All host identifiers within a block share the same prefix, i.e., have the same network identifier. Note that smaller l means more host identifiers, so is less specific, while larger l means fewer host identifiers, so is more specific.

Notes:

- The network identifier for a particular (sub-)network *cannot* be used as an address by a host: for this example, if $l = 24$ then 137.222.103.0 is not a usable IP address since it identifies the (sub-)network. Simply *ending* with 0 does not alone mean an address cannot be used: if we focused on $l = 16$ instead, 137.222.0.0 is the network identifier (and so is unusable) but 137.222.1.0, for example, *can* be used.
  You can think of the network identifier as the first address in the block determined by a given prefix. The *last* address also has a special purpose, and likewise cannot be used as an address by a host. In this case, the purpose is broadcast. If $l = 24$, 137.222.103.255 represents the (sub-)network broadcast address: traffic sent to this address will be broadcast to *all* hosts in the (sub-)network. Again note that just ending with 255 is not enough alone, so if $l = 16$ then 137.222.0.255 can be used as an address by some host (because the last address for the (sub-)network is now 137.222.255.255).
- A **(sub-)network mask** (or **netmask**) derived from l can be used to extract the network identifier from an address by AND'ing them. If you think of an address x as an n-bit integer, then the associated netmask will be

  $$m = (2^l - 1) \ll (n - l).$$

  Imagine we AND them together, i.e., compute $x \wedge m$: the result has 0 in the $n - l$ LSBs, while the l MSBs match x (i.e., provide the network identifier associated with x).
- Two important operations support later use of prefixes: we can
  1. aggregate (i.e., join) several prefixes into one, i.e., shorten the prefix to make it less specific st. it covers all the more specific, aggregated prefixes, or
  2. split one prefix into several i.e., lengthen the prefix to make multiple more specific prefixes; this is normally termed **sub-netting**.

  There are various reasons one might use either operation. An obvious one is to consider that a prefix 137.222.0.0/16 is assigned for use by some network (e.g., by a company). There is normally some internal structure, so from the network perspective it is sensible to split the prefix into multiple more specific sub-network prefixes, e.g., 137.222.0.0/17 and 137.222.128.0/17. However, from the external perspective it might be convenient to treat them as one prefix (e.g., for routing) which is sort of like (re-)aggregating them into the less specific 137.222.0.0/16.

IPv4 (3) – Addressing

Example

Given an IPv4 address 137.222.103.3, if we set $l = 24$ then we find

$$\underbrace{\overbrace{1\,0\,0\,0\,1\,0\,0\,1}^{137}\;.\;\overbrace{1\,1\,0\,1\,1\,1\,1\,0}^{222}\;.\;\overbrace{0\,1\,1\,0\,0\,1\,1\,1}^{103}}_{\substack{l\text{-bit}\\ \text{network identifier}}}\;.\;\underbrace{\overbrace{0\,0\,0\,0\,0\,0\,1\,1}^{3}}_{\substack{(n-l)\text{-bit}\\ \text{host identifier}}}$$

noting that

- $l = 24$ bit (sub-)network identifier is 137.222.103.0,
- the netmask is 255.255.255.0,
- there are at most $2^8 = 256$ addresses in this (sub-)network, and
- they range from 137.222.103.0 to 137.222.103.255.

Notes:

IPv4 (3) – Addressing

Example

Given an IPv4 address 137.222.103.3, if we set $l = 16$ then we find

$$\underbrace{\overbrace{1\,0\,0\,0\,1\,0\,0\,1}^{137}\;.\;\overbrace{1\,1\,0\,1\,1\,1\,1\,0}^{222}\;.\;\overbrace{0\,1\,1\,0\,0\,1\,1\,1}^{103}}_{\substack{l\text{-bit}\\ \text{network identifier}}}\;.\;\underbrace{\overbrace{0\,0\,0\,0\,0\,0\,1\,1}^{3}}_{\substack{(n-l)\text{-bit}\\ \text{host identifier}}}$$

noting that

- $l = 16$ bit (sub-)network identifier is 137.222.0.0,
- the netmask is 255.255.0.0,
- there are at most $2^{16} = 65536$ addresses in this (sub-)network, and
- they range from 137.222.0.0 to 137.222.255.255.

Notes:

IPv4 (4) – Addressing

- Question: how do we know the prefix l?

Notes:

- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  – the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  – the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

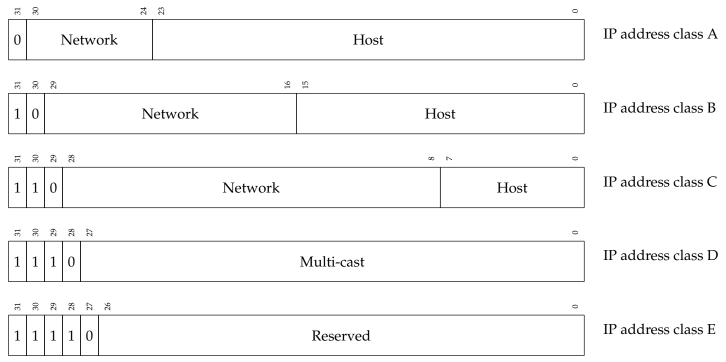
```
137.222.0.0
```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	↦	1 class A block
172.16.0.0/12	↦	16 class B blocks

IPv4 (4) – Addressing

- Question: how do we know the prefix l?
- Answer #1: pre-1993, using a class-based hierarchy, i.e.,



Notes:

- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  – the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  – the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

```
137.222.0.0
```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	↦	1 class A block
172.16.0.0/12	↦	16 class B blocks

IPv4 (4) – Addressing

- Question: how do we know the prefix l?

- Answer #2: post-1993, using **Classless Inter-Domain Routing (CIDR)** [16], st.

$$137.222.103.3/24$$

explicitly denotes the fact $l = 24$.

Notes:

- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  - the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  - the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

```
137.222.0.0
```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	$\mapsto$	1 class A block
172.16.0.0/12	$\mapsto$	16 class B blocks

IPv4 (4) – Addressing

- Question: how do we obtain an address x (or block thereof)?

Notes:

- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  - the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  - the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

```
137.222.0.0
```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	$\mapsto$	1 class A block
172.16.0.0/12	$\mapsto$	16 class B blocks

IPv4 (4) – Addressing

- Question: how do we obtain an address x (or block thereof)?
- Answer #1: use an assigned **public address block**
  1. network identifiers are assigned by **Internet Assigned Numbers Authority (IANA)**, e.g.,

     $$137.222.0.0/16$$

     was hierarchically assigned via IANA $\rightarrow$ RIPE $\rightarrow$ UoB, and
  2. host identifiers are assigned within the (sub-)network, e.g.,

     $$137.222.103.3$$

     was statically assigned to `snowy.cs.bris.ac.uk`

  which *is* globally unique.

Notes:

- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  - the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  - the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

  ```
  137.222.0.0
  ```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	$\mapsto$	1 class A block
172.16.0.0/12	$\mapsto$	16 class B blocks

IPv4 (4) – Addressing

- Question: how do we obtain an address x (or block thereof)?
- Answer #2: use *any* **private address block** [11] e.g.,

  $$192.168.0.0/16$$

  and *any* address in it, e.g.,

  $$192.168.1.123$$

  which *isn't* globally unique!

Notes:

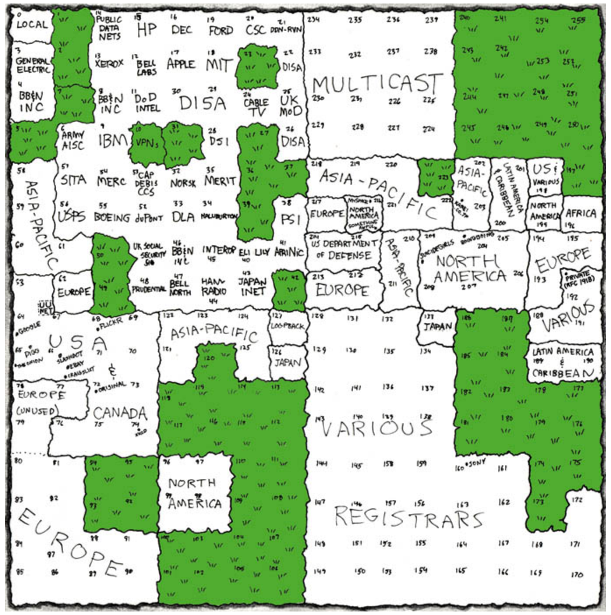
- Strictly speaking, there is a difference between address *assignment* and *allocation*. If an IP block is allocated, this means it can be used by and is under control of the associated (sub-)network. However, if it is assigned the sub-network can use the block *but* it remains under the control of another (sub-)network. This is a subtle difference, and only really relevant in a minority of contexts: to avoid confusion, we try to use the term assignment only.
- It should be clear that for $n = 32$ the $2^{32} = 4294967296$ possible IPv4 addresses does not represent a huge number. Even then, only *some* of those 2^{32} addresses can actually be used due to inefficiencies in the allocation process.
  In summary, IPv4 has an acute address scarcity problem: the small value of n means the address space will eventually be exhausted (the top-level authorities have already run out of blocks to assign), meaning there are too many hosts to uniquely identify. Larger, $n = 128$ bit addresses form a major motivation for the development and deployment of IPv6, although some interim mitigation strategies exist: reclamation of unused IPv4 addresses is one example, and use of technologies such as NAT is another.
- An address 137.222.103.3/24 in CIDR notation should be read as "137 dot 222 dot 103 dot 3 slash 24": it is a "slash 24" address. Some cases might seem odd at first glance, but do make sense (and will be of use when discussing routing):
  - the "slash 0" case specifies $2^{n-l} = 2^{32-0} = 2^{32}$ addresses, i.e., this captures *all* addresses.
  - the "slash 32" case specifies $2^{n-l} = 2^{32-32} = 1$ address i.e., this captures *one* address.
- The (non-)uniqueness of an address is clearly important. Iff. an address is globally unique can it be used as an identifier on inter-network as a whole. If an address is *not* unique, then it might be usable on a (sub-)network (e.g., on a LAN), but not on the inter-network (without a way to cope with the ambiguity introduced).
- It is important to realise the disadvantages of class-based assignment (in which l is implicit): they mainly step from a lack of granularity in terms of networks. For example, imagine we are assigned a class A block but do not use all the addresses in it (which seems likely given the larger number of them). With class-based assignment, we it simply is not possible to sub-divide the assigneded block or transfer addresses for someone else to use. Perhaps even more importantly, the lack of granularity means a fairly "flat" address space: the relative lack of hierarchy implies there are relatively few (sub-)networks and many hosts in each of them. In the context of routing, this becomes problematic because we cannot rely on a divide-and-conquer approach as easily.
- You can inspect some of the detail for a given IP address using the command `whois`. For example,

  ```
  137.222.0.0
  ```

  illustrates assignment of the 137.222.0.0/16 block to UoB.
- Local assignment of host identifiers, i.e., the concrete IP address for each host, can be static (i.e., hard-coded) or dynamic (e.g., via DHCP).
- Although [11] gives a more complete overview, some examples of private address blocks include

10.0.0.0/8	$\mapsto$	1 class A block
172.16.0.0/12	$\mapsto$	16 class B blocks

IPv4 (5) – Addressing

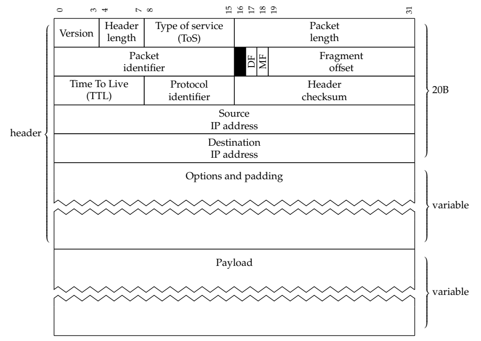


http://xkcd.com/195/

Notes:

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A packet version number (allowing protocol evolution).
- A header length (measured in 32-bit words), formally termed **Internet Header Length (IHL)**.
- A packet length (measured in 8-bit octets), which includes the header.
- A packet identifier.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

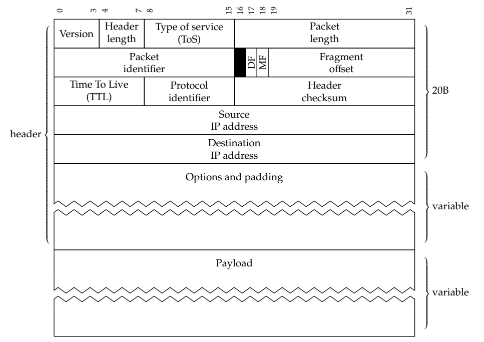
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

http://www.iana.org/assignments/protocol-numbers

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A set of service options, originally defined as
  - 2-bit reserved,
  - 3-bit priority level,
  - 1-bit reliability (normal or high),
  - 1-bit latency (normal or low),
  - 1-bit throughput (normal or high)

  but now depreciated in favour of

  1. **Differentiated Services (DS)** [12] and
  2. **Explicit Congestion Notification (ECN)** [15]

  fields.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

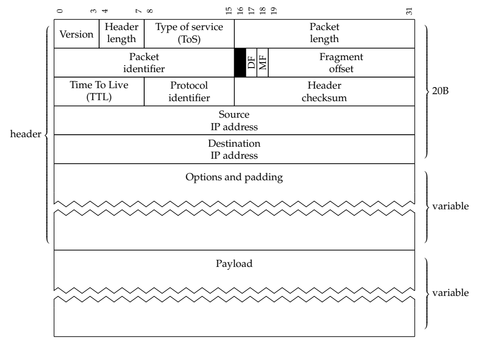
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

http://www.iana.org/assignments/protocol-numbers

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A set of flags, including
  - 1-bit **More Fragments (MF)** flag, which marks final or non-final fragment(s),
  - 1-bit **Don't Fragment (DF)** flag, which prohibits fragmentation by the IP layer.
- A fragment offset (measured in 64-bit words), specifying where this packet payload stems from in the original, unfragmented packet.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

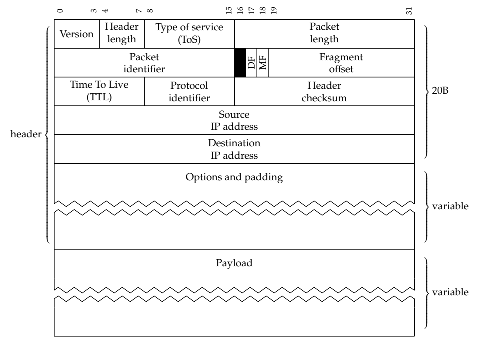
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

http://www.iana.org/assignments/protocol-numbers

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A **Time To Live (TTL)** field specifying the point at which the packet is discarded (if not yet delivered to the destination).
- A 16-bit checksum (on header only) used to detect errors.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

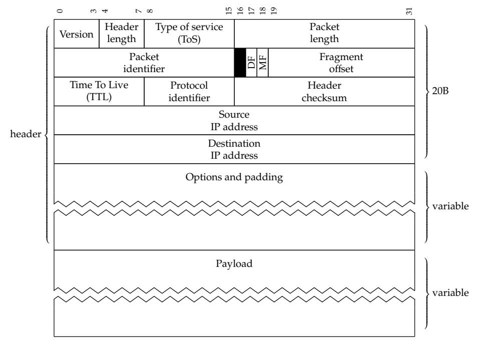
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

`http://www.iana.org/assignments/protocol-numbers`

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A 32-bit source IP address.
- A 32-bit destination IP address.
- The protocol identifier, specifying which higher layer will receive the packet once it reaches the destination.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

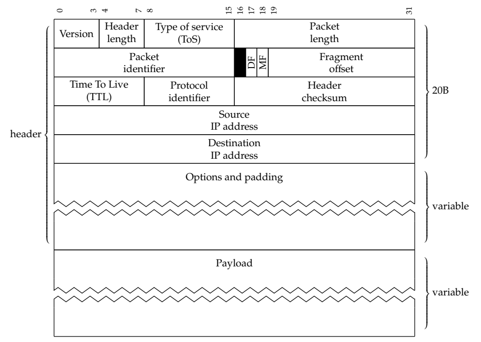
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

`http://www.iana.org/assignments/protocol-numbers`

IPv4 (6) – Packets

Data Structure (IPv4 packet [13, Section 3.1])



The data structure includes:

- A set of options (allowing protocol extensibility).
- Any padding required to ensure the header is a multiple of 32 bits.
- The payload.

Notes:

- Since we focus on IPv4, we clearly expect the protocol version field to be 4 in every packet here!
- The 16-bit packet length field dictates a maximum 65535B length. Since the minimum header length is 20B (if there are no options), this means a minimum and maximum payload length of 0B and 65515B respectively.
- Unlike the 802.3 checksum, which is based on a CRC, the IP checksum opts for a different design: [10] outlines in some detail how it is should be computed. Given some fields (e.g., TTL) in the header will change, the checksum will need to be *re*computed each time a packet is forwarded.
- RFC documents include several April Fools jokes, such as the so-called IP "evil bit" described in [8]: the "idea" proposed is to use one bit of the flags field to mark malicious packets, and hence make the job of network security Engineers a lot easier!
- You might prefer the original ASCII art from [13, Section 3.1]:

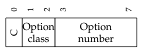
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- [17] lists standard protocol identifiers; important examples include UDP (whose identifier is $11_{(16)}$), TCP (whose identifier is $6_{(16)}$) and ICMP (whose identifier is $1_{(16)}$).
- IANA maintain a definitive set of assigned IP protocol identifiers at

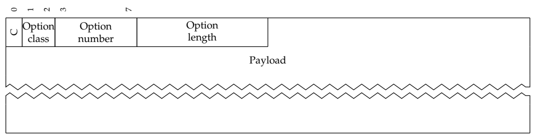
http://www.iana.org/assignments/protocol-numbers

IPv4 (7) – Packets

Data Structure (IPv4 packet option: case #1 [13, Section 3.1])



Data Structure (IPv4 packet option: case #2 [13, Section 3.1])



Notes:

- IANA maintain a definitive set of assigned IP options at

http://www.iana.org/assignments/ip-parameters

- The options basically form a list, which is terminated by an option with a special type (i.e., the EOOL option, whose class and number fields are zero).
- Each option can have an associated payload (case #2) or not (case #1). If there *is* a payload, the *entire* option length is specified using the associated 8-bit field.
- The options class field is filled with $00_{(2)}$ for control options, and $10_{(2)}$ for debugging and measurement options; $01_{(2)}$ and $11_{(2)}$ are reserved.
- The 1-bit copied field determines whether the option should be copied into any fragments resulting from this packet.

IPv4 (8) – Checksum

Quote

The checksum algorithm is:

> *The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.*

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

– Postel [13, Section 3.1]

Notes:

IPv4 (9) – Checksum

Algorithm (checksum)

Given a n-element sequence of octets

$$H = \langle H_0, H_1, \ldots, H_{n-1} \rangle$$

representing the header as input:

1. reinterpret H as a sequence of unsigned, 16-bit integers,
2. compute the sum of these integers, subtracting $2^{16} - 1$ each time the partial result is greater or equal to 2^{16}, then
3. the result is the bitwise complement, i.e., NOT, of that sum.

Algorithm (verify)

Noting that one's-complement has *two* representations of zero (positive *and* negative), to verify either

1. zero the checksum field in the header,
2. compute the checksum, then
3. compare result with original checksum (before being zero'ed),

or

1. compute the checksum, then
2. compare result with zero.

Notes:

IPv4 (10) – Forwarding

- **Goal**: forward a packet P from source $\mathcal{H}_i$ on next hop toward destination $\mathcal{H}_j$.

Notes:

- It's important to keep in mind that "maintain a table" is predicated on something a) filling it initially, and b) keeping it up to date: this is the task of routing, which we assume is taken care of (and then come back to later).
- In a sense, the table based on longest matching prefix is a trade-off that favours space over time: look-ups will take longer than a dense mapping of each address to a next hop, but the table can be much smaller as a result.
- If you think of each entry as being a rule, then it should be obvious why the table remains compact: we only need one rule to cover what might be a *huge* number of addresses within one block (i.e., identified by one prefix).
- There are various scenarios in which special-case routing behaviour might be useful; one example relates to specific applications, e.g., VoIP, where the traffic should be sent via a specific route to ensure quality of service.
- The prefixed address 0.0.0.0/0 might *seem* odd: what it says is that $l = 0$ bits are used for the network identifier, and $n - l = 32 - 0 = 32$ for the host identifier. That is, it captures *all* IPv4 addresses, i.e., acts as a default rule that will always match.

IPv4 (10) – Forwarding

- **Goal**: forward a packet P from source $\mathcal{H}_i$ on next hop toward destination $\mathcal{H}_j$.
- **Observation**:
  - all hosts on the same (sub-)network share a prefix, so
  - maintain a **forwarding table** that maps prefixes to next hop

  st. the table remains compact iff. sensible prefixing is used.

Notes:

- It's important to keep in mind that "maintain a table" is predicated on something a) filling it initially, and b) keeping it up to date: this is the task of routing, which we assume is taken care of (and then come back to later).
- In a sense, the table based on longest matching prefix is a trade-off that favours space over time: look-ups will take longer than a dense mapping of each address to a next hop, but the table can be much smaller as a result.
- If you think of each entry as being a rule, then it should be obvious why the table remains compact: we only need one rule to cover what might be a *huge* number of addresses within one block (i.e., identified by one prefix).
- There are various scenarios in which special-case routing behaviour might be useful; one example relates to specific applications, e.g., VoIP, where the traffic should be sent via a specific route to ensure quality of service.
- The prefixed address 0.0.0.0/0 might *seem* odd: what it says is that $l = 0$ bits are used for the network identifier, and $n - l = 32 - 0 = 32$ for the host identifier. That is, it captures *all* IPv4 addresses, i.e., acts as a default rule that will always match.

IPv4 (10) – Forwarding

- Goal: forward a packet P from source $\mathcal{H}_i$ on next hop toward destination $\mathcal{H}_j$.
- Problem: prefixes in the table might overlap.
- Solution: select the *longest* matching prefix that applies; this allows
  - special-case behaviours (via more-specific prefix, e.g., 137.222.103.3/32), and
  - default behaviours (via less-specific prefix, e.g., 0.0.0.0/0).

Notes:

- It's important to keep in mind that "maintain a table" is predicated on something a) filling it initially, and b) keeping it up to date: this is the task of routing, which we assume is taken care of (and then come back to later).
- In a sense, the table based on longest matching prefix is a trade-off that favours space over time: look-ups will take longer than a dense mapping of each address to a next hop, but the table can be much smaller as a result.
- If you think of each entry as being a rule, then it should be obvious why the table remains compact: we only need one rule to cover what might be a *huge* number of addresses within one block (i.e., identified by one prefix).
- There are various scenarios in which special-case routing behaviour might be useful; one example relates to specific applications, e.g., VoIP, where the traffic should be sent via a specific route to ensure quality of service.
- The prefixed address 0.0.0.0/0 might *seem* odd: what it says is that $l = 0$ bits are used for the network identifier, and $n - l = 32 - 0 = 32$ for the host identifier. That is, it captures *all* IPv4 addresses, i.e., acts as a default rule that will always match.

IPv4 (10) – Forwarding

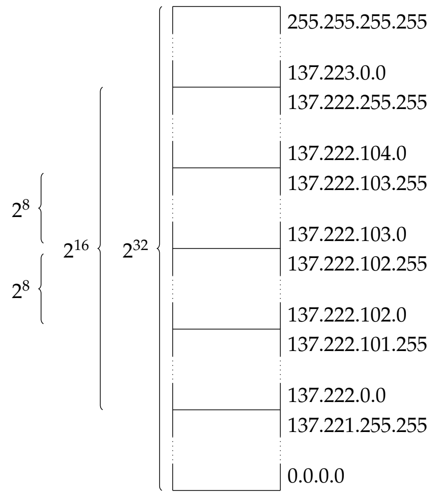
- Goal: forward a packet P from source $\mathcal{H}_i$ on next hop toward destination $\mathcal{H}_j$.
- Observation: we only need to know what the next hop is, so
  - have the routers make (global) routing decisions, and
  - have the hosts follow a simple rule:
    1. communicate locally with destination if it is on the same sub-network, otherwise
    2. forward to nearest router as next hop toward destination

  i.e., improve scalability by leveraging hierarchy within topology and addressing.

Notes:

- It's important to keep in mind that "maintain a table" is predicated on something a) filling it initially, and b) keeping it up to date: this is the task of routing, which we assume is taken care of (and then come back to later).
- In a sense, the table based on longest matching prefix is a trade-off that favours space over time: look-ups will take longer than a dense mapping of each address to a next hop, but the table can be much smaller as a result.
- If you think of each entry as being a rule, then it should be obvious why the table remains compact: we only need one rule to cover what might be a *huge* number of addresses within one block (i.e., identified by one prefix).
- There are various scenarios in which special-case routing behaviour might be useful; one example relates to specific applications, e.g., VoIP, where the traffic should be sent via a specific route to ensure quality of service.
- The prefixed address 0.0.0.0/0 might *seem* odd: what it says is that $l = 0$ bits are used for the network identifier, and $n - l = 32 - 0 = 32$ for the host identifier. That is, it captures *all* IPv4 addresses, i.e., acts as a default rule that will always match.

IPv4 (11) – Forwarding



- Consider a router with 3 NICs, and the forwarding table

Prefix	Next hop
137.222.102.0/24	···
137.222.0.0/16	···
0.0.0.0/0	···

- A given address 137.222.102.13
  - is within the prefixes in *all* rows, *but*
  - the prefix in the first row is longer, so this is the selected match.

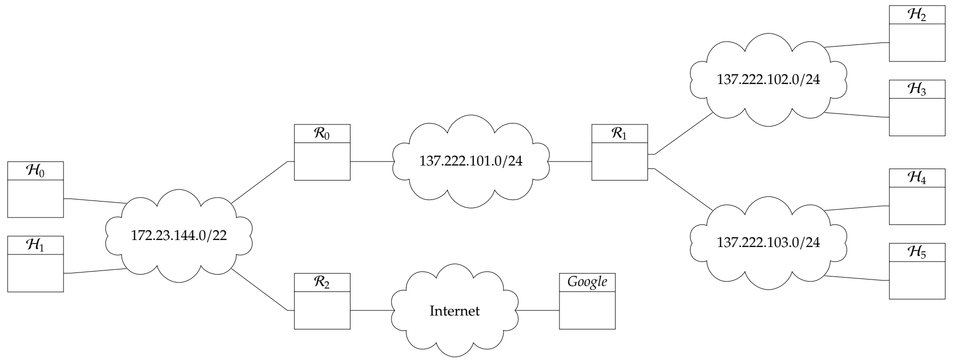
Notes:

IPv4 (12) – Forwarding

- Example: forward a packet

$$P = H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D$$

through some (purely hypothetical) inter-network.



Notes:

- We're focussing on the packet here: as it gets communicated through the various LANs, it'll obviously be framed by the associated link layers. For example, the first hop might see

$$P = H_{802.11} \parallel H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D \parallel T_{802.11}$$

  actually communicated.
- A reasonable question relates to how say $\mathcal{R}_0$ forwards the packet, whose destination is $\mathcal{H}_2$ via $\mathcal{R}_1$: $\mathcal{R}_0$ cannot just change the destination IP address, otherwise $\mathcal{R}_1$ will be unable to determine the final destination! The answer is of course that each hop requires local communication, which harnesses local addressing.
  Imagine that $\mathcal{H}_0$ to $\mathcal{R}_0$ have the MAC addresses

$$00:22:4D:99:03:0F$$

  and

$$E8:94:F6:0D:24:10$$

  respectively, and $\mathcal{H}_0$ wants to transmit some payload D. The first hop therefore sees the frame

$$H_{802.11}[\text{src} = 00:22:4D:99:03:0F, \text{dst} = E8:94:F6:0D:24:10] \parallel H_{IPv4}[\text{src} = \text{src} = 172.23.144.94, \text{dst} = \text{dst} = 137.222.102.13] \parallel D \parallel T_{802}$$

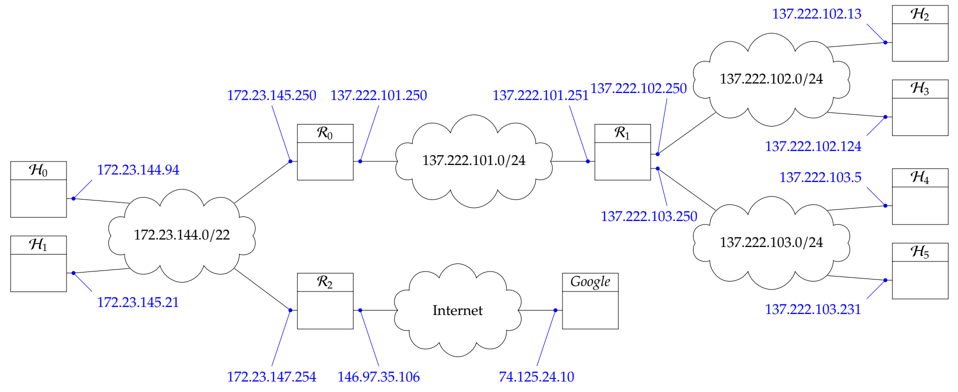
  transmitted via the 172.23.144.0/22 sub-network. $\mathcal{R}_0$ receives the frame, and decapsulates the IP packet: since the destination IP address does not match, it forwards it to $\mathcal{R}_1$ by encapsulating the packet in a different frame whose source and destination MAC addresses will differ (to match those used by $\mathcal{R}_0$ and $\mathcal{R}_1$ on the 137.222.101.0/24 sub-network).

IPv4 (12) – Forwarding

- Example: forward a packet

$$P = H_{IPv4}[\text{src} = 172.23.144.94, \text{dst} = 137.222.102.13] \parallel D$$

through some (purely hypothetical) inter-network.



Notes:

- We're focussing on the packet here: as it gets communicated through the various LANs, it'll obviously be framed by the associated link layers. For example, the first hop might see

$$P = H_{802.11} \parallel H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D \parallel T_{802.11}$$

actually communicated.

- A reasonable question relates to how say $\mathcal{R}_0$ forwards the packet, whose destination is $\mathcal{H}_2$ via $\mathcal{R}_1$: $\mathcal{R}_0$ cannot just change the destination IP address, otherwise $\mathcal{R}_1$ will be unable to determine the final destination! The answer is of course that each hop requires local communication, which harnesses local addressing.
Imagine that $\mathcal{H}_0$ to $\mathcal{R}_0$ have the MAC addresses

$$00:22:4D:99:03:0F$$

and

$$E8:94:F6:0D:24:10$$

respectively, and $\mathcal{H}_0$ wants to transmit some payload D. The first hop therefore sees the frame

$$H_{802.11}[\text{src} = 00:22:4D:99:03:0F, \text{dst} = E8:94:F6:0D:24:10] \parallel H_{IPv4}[\text{src} = \text{src} = 172.23.144.94, \text{dst} = \text{dst} = 137.222.102.13] \parallel D \parallel T_{802}$$

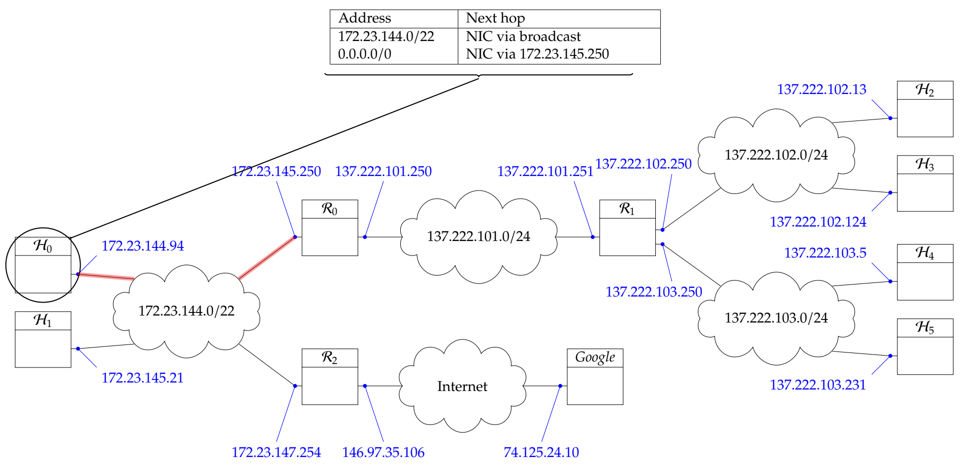
transmitted via the 172.23.144.0/22 sub-network. $\mathcal{R}_0$ receives the frame, and decapsulates the IP packet: since the destination IP address does not match, it forwards it to $\mathcal{R}_1$ by encapsulating the packet in a different frame whose source and destination MAC addresses will differ (to match those used by $\mathcal{R}_0$ and $\mathcal{R}_1$ on the 137.222.101.0/24 sub-network).

IPv4 (12) – Forwarding

- Example: forward a packet

$$P = H_{IPv4}[\text{src} = 172.23.144.94, \text{dst} = 137.222.102.13, \text{TTL} = 64] \parallel D$$

through some (purely hypothetical) inter-network.



Address	Next hop
172.23.144.0/22	NIC via broadcast
0.0.0.0/0	NIC via 172.23.145.250

Notes:

- We're focussing on the packet here: as it gets communicated through the various LANs, it'll obviously be framed by the associated link layers. For example, the first hop might see

$$P = H_{802.11} \parallel H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D \parallel T_{802.11}$$

actually communicated.

- A reasonable question relates to how say $\mathcal{R}_0$ forwards the packet, whose destination is $\mathcal{H}_2$ via $\mathcal{R}_1$: $\mathcal{R}_0$ cannot just change the destination IP address, otherwise $\mathcal{R}_1$ will be unable to determine the final destination! The answer is of course that each hop requires local communication, which harnesses local addressing.
Imagine that $\mathcal{H}_0$ to $\mathcal{R}_0$ have the MAC addresses

$$00:22:4D:99:03:0F$$

and

$$E8:94:F6:0D:24:10$$

respectively, and $\mathcal{H}_0$ wants to transmit some payload D. The first hop therefore sees the frame

$$H_{802.11}[\text{src} = 00:22:4D:99:03:0F, \text{dst} = E8:94:F6:0D:24:10] \parallel H_{IPv4}[\text{src} = \text{src} = 172.23.144.94, \text{dst} = \text{dst} = 137.222.102.13] \parallel D \parallel T_{802}$$

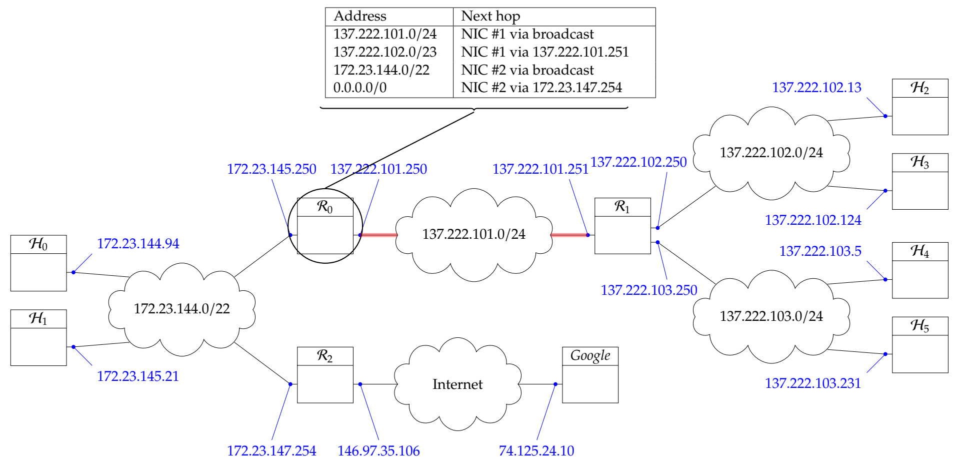
transmitted via the 172.23.144.0/22 sub-network. $\mathcal{R}_0$ receives the frame, and decapsulates the IP packet: since the destination IP address does not match, it forwards it to $\mathcal{R}_1$ by encapsulating the packet in a different frame whose source and destination MAC addresses will differ (to match those used by $\mathcal{R}_0$ and $\mathcal{R}_1$ on the 137.222.101.0/24 sub-network).

IPv4 (12) – Forwarding

- Example: forward a packet

$$P = H_{IPv4}[\text{src} = 172.23.144.94, \text{dst} = 137.222.102.13, \text{TTL} = 63] \parallel D$$

through some (purely hypothetical) inter-network.



Address	Next hop
137.222.101.0/24	NIC #1 via broadcast
137.222.102.0/23	NIC #1 via 137.222.101.251
172.23.144.0/22	NIC #2 via broadcast
0.0.0.0/0	NIC #2 via 172.23.147.254

Notes:

- We're focussing on the packet here: as it gets communicated through the various LANs, it'll obviously be framed by the associated link layers. For example, the first hop might see

$$P = H_{802.11} \parallel H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D \parallel T_{802.11}$$

actually communicated.

- A reasonable question relates to how say $\mathcal{R}_0$ forwards the packet, whose destination is $\mathcal{H}_2$ via $\mathcal{R}_1$: $\mathcal{R}_0$ cannot just change the destination IP address, otherwise $\mathcal{R}_1$ will be unable to determine the final destination! The answer is of course that each hop requires local communication, which harnesses local addressing.
Imagine that $\mathcal{H}_0$ to $\mathcal{R}_0$ have the MAC addresses

$$00:22:4D:99:03:0F$$

and

$$E8:94:F6:0D:24:10$$

respectively, and $\mathcal{H}_0$ wants to transmit some payload D. The first hop therefore sees the frame

$$H_{802.11}[\text{src} = 00:22:4D:99:03:0F, \text{dst} = E8:94:F6:0D:24:10] \parallel H_{IPv4}[\text{src} = \text{src} = 172.23.144.94, \text{dst} = \text{dst} = 137.222.102.13] \parallel D \parallel T_{802}$$

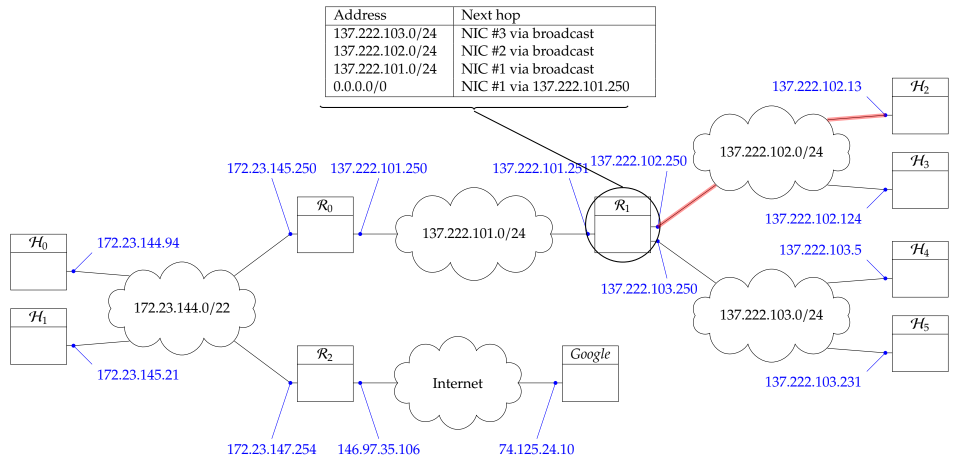
transmitted via the 172.23.144.0/22 sub-network. $\mathcal{R}_0$ receives the frame, and decapsulates the IP packet: since the destination IP address does not match, it forwards it to $\mathcal{R}_1$ by encapsulating the packet in a different frame whose source and destination MAC addresses will differ (to match those used by $\mathcal{R}_0$ and $\mathcal{R}_1$ on the 137.222.101.0/24 sub-network).

IPv4 (12) – Forwarding

- Example: forward a packet

$$P = H_{IPv4}[\text{src} = 172.23.144.94, \text{dst} = 137.222.102.13, \text{TTL} = 62] \parallel D$$

through some (purely hypothetical) inter-network.



Address	Next hop
137.222.103.0/24	NIC #3 via broadcast
137.222.102.0/24	NIC #2 via broadcast
137.222.101.0/24	NIC #1 via broadcast
0.0.0.0/0	NIC #1 via 137.222.101.250

Notes:

- We're focussing on the packet here: as it gets communicated through the various LANs, it'll obviously be framed by the associated link layers. For example, the first hop might see

$$P = H_{802.11} \parallel H_{IPv4}[\text{src} = \mathcal{H}_0, \text{dst} = \mathcal{H}_2] \parallel D \parallel T_{802.11}$$

actually communicated.

- A reasonable question relates to how say $\mathcal{R}_0$ forwards the packet, whose destination is $\mathcal{H}_2$ via $\mathcal{R}_1$: $\mathcal{R}_0$ cannot just change the destination IP address, otherwise $\mathcal{R}_1$ will be unable to determine the final destination! The answer is of course that each hop requires local communication, which harnesses local addressing.
Imagine that $\mathcal{H}_0$ to $\mathcal{R}_0$ have the MAC addresses

$$00:22:4D:99:03:0F$$

and

$$E8:94:F6:0D:24:10$$

respectively, and $\mathcal{H}_0$ wants to transmit some payload D. The first hop therefore sees the frame

$$H_{802.11}[\text{src} = 00:22:4D:99:03:0F, \text{dst} = E8:94:F6:0D:24:10] \parallel H_{IPv4}[\text{src} = \text{src} = 172.23.144.94, \text{dst} = \text{dst} = 137.222.102.13] \parallel D \parallel T_{802}$$

transmitted via the 172.23.144.0/22 sub-network. $\mathcal{R}_0$ receives the frame, and decapsulates the IP packet: since the destination IP address does not match, it forwards it to $\mathcal{R}_1$ by encapsulating the packet in a different frame whose source and destination MAC addresses will differ (to match those used by $\mathcal{R}_0$ and $\mathcal{R}_1$ on the 137.222.101.0/24 sub-network).

IPv4 (13) – Fragmentation

- Problem:
  - each (sub-)network has a **Maximum Transmission Unit (MTU)**,
  - if $\mathcal{H}_i$ transmits an n-octet packet to $\mathcal{H}_j$, what if n is larger than some (intermediate) MTU?

Notes:

- Strictly speaking, $\mathcal{H}_i$ cannot transmit packets of any length even if fragmentation is used because *it* needs to respect the MTU of the LAN to which it is connected. The *application* that invoked the IPv4 layer on $\mathcal{H}_i$ can send data of any length; the IPv4 layer on $\mathcal{H}_i$ itself *or* any intermediate router between $\mathcal{H}_i$ and $\mathcal{H}_j$ may then need to fragment the resulting packet.
- The discovery process needs to be repeated intermittently (because the network topology and hence path MTU can change over time), and relies on the use of ICMP to send "probe" packets into the network: based on the existence or absence of an associated error, we can tell if a particular packet length will "fit" or not.
- In more detail, the maximum IPv4 packet length is is 65535B meaning a maximum payload length of 65535 − 20 = 65515B (if there are no options). If you contrast this with
  - 802.2 (Ethernet), whose MTU is 1500B, or
  - 802.11 (WiFi), whose MTU is 7981B

  clearly a maximum-length IPv4 packet will require encapsulation into many frames.
- Clearly the link layer may have some restriction on frame length relating to the implementation technology: in 802.3, for instance, a *minimum* frame length is required to satisfy the demands of collision detection. Beyond this, however, and ignoring the issue of the field size (i.e., an n-bit length field will limit the maximum frame length to $2^n - 1$), why select a specific MTU? There are a large number trade-offs, meaning a single "perfect" selection is impossible: as just two examples only, consider that
  - with a larger MTU, communication is more efficient in the sense of requiring less overhead (less header for instance) per-octet of the payload,
  - with a smaller MTU, communication is more efficient than recovery from errors in a given payload requires retransmission of a smaller replacement.

IPv4 (13) – Fragmentation

- Problem:
  - each (sub-)network has a **Maximum Transmission Unit (MTU)**,
  - if $\mathcal{H}_i$ transmits an n-octet packet to $\mathcal{H}_j$, what if n is larger than some (intermediate) MTU?
- "Solution" #1: produce an error, and drop the packet!

Notes:

- Strictly speaking, $\mathcal{H}_i$ cannot transmit packets of any length even if fragmentation is used because *it* needs to respect the MTU of the LAN to which it is connected. The *application* that invoked the IPv4 layer on $\mathcal{H}_i$ can send data of any length; the IPv4 layer on $\mathcal{H}_i$ itself *or* any intermediate router between $\mathcal{H}_i$ and $\mathcal{H}_j$ may then need to fragment the resulting packet.
- The discovery process needs to be repeated intermittently (because the network topology and hence path MTU can change over time), and relies on the use of ICMP to send "probe" packets into the network: based on the existence or absence of an associated error, we can tell if a particular packet length will "fit" or not.
- In more detail, the maximum IPv4 packet length is is 65535B meaning a maximum payload length of 65535 − 20 = 65515B (if there are no options). If you contrast this with
  - 802.2 (Ethernet), whose MTU is 1500B, or
  - 802.11 (WiFi), whose MTU is 7981B

  clearly a maximum-length IPv4 packet will require encapsulation into many frames.
- Clearly the link layer may have some restriction on frame length relating to the implementation technology: in 802.3, for instance, a *minimum* frame length is required to satisfy the demands of collision detection. Beyond this, however, and ignoring the issue of the field size (i.e., an n-bit length field will limit the maximum frame length to $2^n - 1$), why select a specific MTU? There are a large number trade-offs, meaning a single "perfect" selection is impossible: as just two examples only, consider that
  - with a larger MTU, communication is more efficient in the sense of requiring less overhead (less header for instance) per-octet of the payload,
  - with a smaller MTU, communication is more efficient than recovery from errors in a given payload requires retransmission of a smaller replacement.

IPv4 (13) – Fragmentation

- Problem:
  - each (sub-)network has a **Maximum Transmission Unit (MTU)**,
  - if $\mathcal{H}_i$ transmits an n-octet packet to $\mathcal{H}_j$, what if n is larger than some (intermediate) MTU?

- Solution #2:
  1. allow $\mathcal{H}_i$ to transmit packets of any length,
  2. have each router **fragment** any outgoing packet that is larger than the associated MTU,
  3. reassemble packet fragments at destination.

Notes:

- Strictly speaking, $\mathcal{H}_i$ cannot transmit packets of any length even if fragmentation is used because *it* needs to respect the MTU of the LAN to which it is connected. The *application* that invoked the IPv4 layer on $\mathcal{H}_i$ can send data of any length; the IPv4 layer on $\mathcal{H}_i$ itself *or* any intermediate router between $\mathcal{H}_i$ and $\mathcal{H}_j$ may then need to fragment the resulting packet.
- The discovery process needs to be repeated intermittently (because the network topology and hence path MTU can change over time), and relies on the use of ICMP to send "probe" packets into the network: based on the existence or absence of an associated error, we can tell if a particular packet length will "fit" or not.
- In more detail, the maximum IPv4 packet length is is 65535B meaning a maximum payload length of $65535 - 20 = 65515$B (if there are no options). If you contrast this with
  - 802.2 (Ethernet), whose MTU is 1500B, or
  - 802.11 (WiFi), whose MTU is 7981B

  clearly a maximum-length IPv4 packet will require encapsulation into many frames.
- Clearly the link layer may have some restriction on frame length relating to the implementation technology: in 802.3, for instance, a *minimum* frame length is required to satisfy the demands of collision detection. Beyond this, however, and ignoring the issue of the field size (i.e., an n-bit length field will limit the maximum frame length to $2^n - 1$), why select a specific MTU? There are a large number trade-offs, meaning a single "perfect" selection is impossible: as just two examples only, consider that
  - with a larger MTU, communication is more efficient in the sense of requiring less overhead (less header for instance) per-octet of the payload,
  - with a smaller MTU, communication is more efficient than recovery from errors in a given payload requires retransmission of a smaller replacement.

IPv4 (13) – Fragmentation

- Problem:
  - each (sub-)network has a **Maximum Transmission Unit (MTU)**,
  - if $\mathcal{H}_i$ transmits an n-octet packet to $\mathcal{H}_j$, what if n is larger than some (intermediate) MTU?

- Solution #3:
  1. force $\mathcal{H}_i$ to discover the **path MTU** between $\mathcal{H}_i$ and $\mathcal{H}_j$, then
  2. limit the size of packets transmitted by $\mathcal{H}_i$ so fragmentation is avoided.

Notes:

- Strictly speaking, $\mathcal{H}_i$ cannot transmit packets of any length even if fragmentation is used because *it* needs to respect the MTU of the LAN to which it is connected. The *application* that invoked the IPv4 layer on $\mathcal{H}_i$ can send data of any length; the IPv4 layer on $\mathcal{H}_i$ itself *or* any intermediate router between $\mathcal{H}_i$ and $\mathcal{H}_j$ may then need to fragment the resulting packet.
- The discovery process needs to be repeated intermittently (because the network topology and hence path MTU can change over time), and relies on the use of ICMP to send "probe" packets into the network: based on the existence or absence of an associated error, we can tell if a particular packet length will "fit" or not.
- In more detail, the maximum IPv4 packet length is is 65535B meaning a maximum payload length of $65535 - 20 = 65515$B (if there are no options). If you contrast this with
  - 802.2 (Ethernet), whose MTU is 1500B, or
  - 802.11 (WiFi), whose MTU is 7981B

  clearly a maximum-length IPv4 packet will require encapsulation into many frames.
- Clearly the link layer may have some restriction on frame length relating to the implementation technology: in 802.3, for instance, a *minimum* frame length is required to satisfy the demands of collision detection. Beyond this, however, and ignoring the issue of the field size (i.e., an n-bit length field will limit the maximum frame length to $2^n - 1$), why select a specific MTU? There are a large number trade-offs, meaning a single "perfect" selection is impossible: as just two examples only, consider that
  - with a larger MTU, communication is more efficient in the sense of requiring less overhead (less header for instance) per-octet of the payload,
  - with a smaller MTU, communication is more efficient than recovery from errors in a given payload requires retransmission of a smaller replacement.

IPv4 (13) – Fragmentation

- Problem:
  - each (sub-)network has a **Maximum Transmission Unit (MTU)**,
  - if $\mathcal{H}_i$ transmits an n-octet packet to $\mathcal{H}_j$, what if n is larger than some (intermediate) MTU?

noting that IPv4 hosts and routers

- must support reassembly [9, Section 3.3.2], and
- may support fragmentation [9, Section 3.3.3]

but modern implementations typically use path MTU discovery.

Notes:

- Strictly speaking, $\mathcal{H}_i$ cannot transmit packets of any length even if fragmentation is used because *it* needs to respect the MTU of the LAN to which it is connected. The *application* that invoked the IPv4 layer on $\mathcal{H}_i$ can send data of any length; the IPv4 layer on $\mathcal{H}_i$ itself *or* any intermediate router between $\mathcal{H}_i$ and $\mathcal{H}_j$ may then need to fragment the resulting packet.
- The discovery process needs to be repeated intermittently (because the network topology and hence path MTU can change over time), and relies on the use of ICMP to send "probe" packets into the network: based on the existence or absence of an associated error, we can tell if a particular packet length will "fit" or not.
- In more detail, the maximum IPv4 packet length is is 65535B meaning a maximum payload length of $65535 - 20 = 65515$B (if there are no options). If you contrast this with
  - 802.2 (Ethernet), whose MTU is 1500B, or
  - 802.11 (WiFi), whose MTU is 7981B

  clearly a maximum-length IPv4 packet will require encapsulation into many frames.
- Clearly the link layer may have some restriction on frame length relating to the implementation technology: in 802.3, for instance, a *minimum* frame length is required to satisfy the demands of collision detection. Beyond this, however, and ignoring the issue of the field size (i.e., an n-bit length field will limit the maximum frame length to $2^n - 1$), why select a specific MTU? There are a large number trade-offs, meaning a single "perfect" selection is impossible: as just two examples only, consider that
  - with a larger MTU, communication is more efficient in the sense of requiring less overhead (less header for instance) per-octet of the payload,
  - with a smaller MTU, communication is more efficient than recovery from errors in a given payload requires retransmission of a smaller replacement.

IPv4 (14) – Fragmentation

Algorithm (fragment)

At the source or an intermediate router, imagine there is a need to fragment some packet P:

1. If the DF flag in P is **true**, drop P.
2. Otherwise divide the payload into n fragments, F_i for $0 \leq i < n$, each of whose length (including header) is less than the MTU.
3. Copy the header from P into each F_i, and update both the offset and MF flags based on i.
4. Transmit each F_i.

Algorithm (reassemble)

At the destination *only*:

1. Buffer fragments with same identifier until they're all received, noting
   - they may be received out-of-order,
   - a reassembly time-out prevents indefinite buffering, and
   - a fragment F_i whose MF flag is **false** is the last one, so yields the total length.
2. Reconstruct the original packet P (at least the payload).
3. Process P as per normal, i.e., provide it to whatever transport layer protocol P indicates.

Notes:

- In some more detail:
  - there will be $n = \lceil (l_p - l_h)/(m - l_h) \rceil$ fragments in total,
  - this gets more complicated if the original header includes options that disallow copying into the fragments,
  - the first $n-1$ fragments will have an $m - l_h$ octet payload, with the last, $(n-1)$-th fragment being partially filled,
  - The i-th fragment will have a header that looks like

$$H_{IPv4}\left[\begin{array}{lcl} \text{length} & = & \left\{\begin{array}{rl} (l_p - l_h) \bmod (m - l_h) & \text{if } i = n-1 \\ (m - l_h) & \text{otherwise} \end{array}\right. \\ \\ \text{offset} & = & i \cdot (m - l_h) \\ \\ \text{MF} & = & \left\{\begin{array}{ll} 0 & \text{if } i = n-1 \\ 1 & \text{otherwise} \end{array}\right. \end{array}\right]$$

  and a payload equal to

$$D_{i \cdot (m - l_h), \ldots, \min(l_p - l_h, (i+1) \cdot (m - l_h)) - 1}.$$

- The fact that *any* intermediate router can apply fragmentation (to cope with whatever MTU the next hop dictates) means it can potentially occur *multiple* times: the algorithm shown allows this naturally.
- The fact that reassembly *only* occurs at the destination disallows some forms of optimisation, but, crucially,
  1. removes any requirement for buffering by intermediate routers, and
  2. avoids the complication that intermediate router may be unable to perform reassembly if some fragments are routed through a different path to others (meaning it may never receive them).
- Hosts are *not* required to buffer and reassemble arbitrarily large packets: an upper bound is, by default, 576B for IPv4 as described in [14].
  This represents a limit on what can be asked of the host, so in a sense is a limit on what can be transmitted; it *doesn't* mean the host cannot opt to deal with larger packers if it wants to, i.e., it isn't a limit on what can be recieved.
- Clearly there is some overhead associated with fragmentation beyond the extra execution time and memory required for the buffer(s): assuming we fragment the packet into n fragments, we get $n-1$ *extra* copies of the header transmitted.
- Since the offset is specified in multiples of 64-bit words, one needs to take care to divide the payload into fragments wrt. such boundaries (which isn't done here).
- Selection of the identifier field turns out to be quite a thorny issue. The packet is *actually* identified via a combination of source and destination IP addresses, plus the packet identifier field. Therefore, if the identifier field is unique the communication between source and destination can be identified (within general traffic, e.g., some other communication between different hosts, or some different communication between the same hosts).
  The higher-layer that uses IP, e.g., UDP or TCP, selects the identifier field. One approach is to use a counter: modulo the size of field [illegible]

IPv4 (15) – Fragmentation

Example

Assume some $\mathcal{H}_k$ needs to forward a packet

$$P = H_{IPv4}[\text{length} = 3040, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}] \parallel D$$

with a 20B header and 3020B payload where the MTU is 1500B.

Notes:

IPv4 (15) – Fragmentation

Example

Assume some $\mathcal{H}_k$ needs to forward a packet

$$P = H_{IPv4}[\text{length} = 3040, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}] \parallel D$$

with a 20B header and 3020B payload where the MTU is 1500B.
To fragment the packet:

1. Divide the payload into three fragments st.

$$D = D_{0,\ldots,1479} \parallel D_{1480,\ldots,2959} \parallel D_{2960,\ldots,3019}.$$

2. Copy the header from P into the fragments to get

$$\begin{array}{lcl} F_0 & = & H_{IPv4}[\ \text{length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}\ ] \parallel D_{0,\ldots,1479} \\ F_1 & = & H_{IPv4}[\ \text{length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}\ ] \parallel D_{1480,\ldots,2950} \\ F_2 & = & H_{IPv4}[\ \text{length} = 80, \quad \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}\ ] \parallel D_{2960,\ldots,3019} \end{array}$$

3. Update the fields in each F_i, i.e.,

$$\begin{array}{lcl} F_0 & = & H_{IPv4}[\ \text{length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 0, \quad \text{MF} = 1\ ] \parallel D_{0,\ldots,1479} \\ F_1 & = & H_{IPv4}[\ \text{length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 185, \text{MF} = 1\ ] \parallel D_{1480,\ldots,2950} \\ F_2 & = & H_{IPv4}[\ \text{length} = 80, \quad \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 370, \text{MF} = 0\ ] \parallel D_{2960,\ldots,3019} \end{array}$$

   noting that the offset is measured in 64-bit words.

4. Transmit each F_i.

Notes:

IPv4 (15) – Fragmentation

Example

Assume some $\mathcal{H}_k$ needs to forward a packet

$$P = H_{IPv4}[\text{length} = 3040, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}] \parallel D$$

with a 20B header and 3020B payload where the MTU is 1500B.
To reassemble the packet:

1. Buffer fragments with same identifier until they're all received

$$\begin{array}{lcl} F_1 & = & H_{IPv4}[\text{ length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 185, \text{MF} = 1 \text{ }] \parallel D_{1480,\ldots,2950} \\ F_2 & = & H_{IPv4}[\text{ length} = 80, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 370, \text{MF} = 0 \text{ }] \parallel D_{2960,\ldots,3019} \\ F_0 & = & H_{IPv4}[\text{ length} = 1500, \text{IHL} = 5, \text{ident} = 3, \text{DF} = \textbf{false}, \text{offset} = 0, \text{MF} = 1 \text{ }] \parallel D_{0,\ldots,1479} \end{array}$$

   noting that we know P should be $370 \cdot 8 + 80 - 20$ octets long.

2. Reconstruct the original packet

$$D = D_{0,\ldots,1479} \parallel D_{1480,\ldots,2959} \parallel D_{2960,\ldots,3019}$$

   and provide this to the transport layer.

Notes:

Conclusions

Continued in next lecture ...

Notes:

References

[1] Wikipedia: Internet protocol (IP).
http://en.wikipedia.org/wiki/Internet_Protocol.

[2] Wikipedia: IP address.
http://en.wikipedia.org/wiki/IP_address.

[3] Wikipedia: IP fragmentation.
http://en.wikipedia.org/wiki/IP_fragmentation.

[4] Wikipedia: Network layer.
http://en.wikipedia.org/wiki/Network_layer.

[5] Wikipedia: Packet forwarding.
http://en.wikipedia.org/wiki/Packet_forwarding.

[6] Wikipedia: Routing.
http://en.wikipedia.org/wiki/Routing.

[7] Wikipedia: Routing table.
http://en.wikipedia.org/wiki/Routing_table.

[8] S. Bellovin.
The security flag in the IPv4 header.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3514, 2003.
http://tools.ietf.org/html/rfc3540.

Notes:

References

[9] 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.

[10] R. Braden, D. Borman, and C. Partridge.
Computing the Internet checksum.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1071, 1988.
http://tools.ietf.org/html/rfc1071.

[11] IANA.
Special-use IPv4 addresses.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3330, 2002.
http://tools.ietf.org/html/rfc3330.

[12] K. Nichols, S. Blake, F. Baker, and D. Black.
Definition of the Differentiated Services field (or DS field) in the IPv4 and IPv6 headers.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 2474, 1998.
http://tools.ietf.org/html/rfc2474.

[13] J. Postel.
Internet Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 791, 1981.
http://tools.ietf.org/html/rfc791.

Notes:

References

[14] J. Postel.
The TCP maximum segment size and related topics.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 879, 1982.
http://tools.ietf.org/html/rfc879.

[15] K. Ramakrishnan, S. Floyd, and D. Black.
The addition of Explicit Congestion Notification (ECN) to IP.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 3168, 2001.
http://tools.ietf.org/html/rfc3168.

[16] Y. Rekhter and T. Li.
An architecture for IP address alloation with CIDR.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1518, 1989.
http://tools.ietf.org/html/rfc1518.

[17] J. Reynolds and J. Postel.
Assigned numbers.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1060, 1990.
http://tools.ietf.org/html/rfc1060.

[18] W. Stallings.
Chapter 19: Internet protocols.
In *Data and Computer Communications* [20].

Notes:

References

[19] W. Stallings.
Chapter 20: Internetwork operation.
In *Data and Computer Communications* [20].

[20] W. Stallings.
Data and Computer Communications.
Pearson, 9th edition, 2010.

[21] D. Waitzman.
A standard for the transmission of IP datagrams on avian carriers.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1149, 1998.
http://tools.ietf.org/html/rfc1149.

[22] G. Ziemba, D. Reed, and P. Traina.
Security considerations for IP fragment filtering.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1858, 1988.
http://tools.ietf.org/html/rfc1858.

Notes: