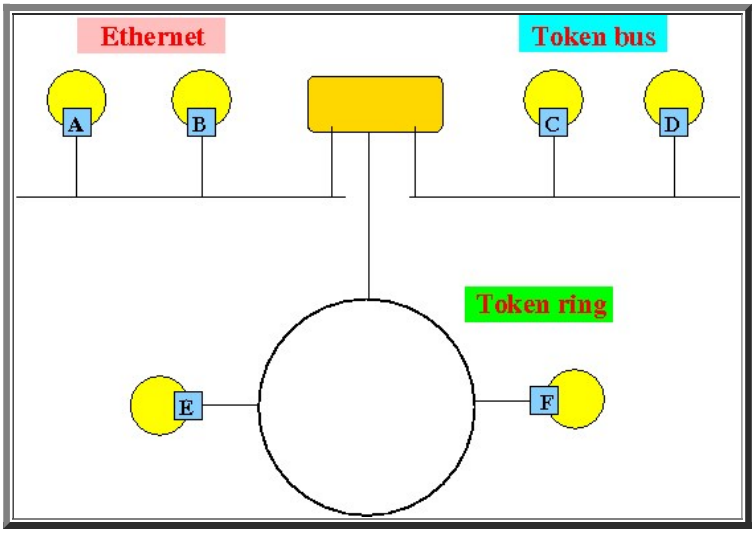
Intro to interconnection *heterogeneous* networks

- **Interconnecting *heterogeneous* networks**
  - We will **now** consider the **problem** of **inter-connecting *heterogeneous* networks**.

  - **Example:**



- ***Problems* when we interconnect *heterogeneous* Networks**
  - **Problems**

1. The **identification problem**:
   - ***Different* networks** use ***different* network numbers (identification)**
   - **How** can we ***uniquely* identify** a **node** on the ***entire* network**:

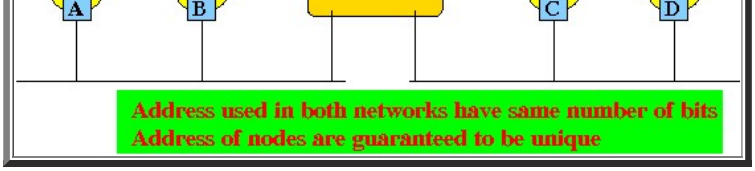
     (**Computers need** to use the ***same* type** of **identifiers** !!!)
2. The **heterogeneity problem**:
   - ***Different* networks** use ***different* frame formats**

     (**Computers need** to use the ***same* format (= code)** !!!)
   - **How** can we make ***different* networks agree** on the ***same* encoding** ???
3. **Scalability** (same issue as in a **homogeneous network**)

- ***Identification* in a *homogenous* network**
  - **Fact:**
    - ***Every* Ethernet address** is **48 bits**
    - ***Every* Ethernet network card** that has been made has a ***unique* Ethernet (network) address**

  - **Therefore:**
    - **Nodes** on ***different* Ethernet networks** have **network addresses** that are **48 bits** long (we know what to use to identify a node)
    - ***Each* node** (even on ***different* networks**) is ***uniquely* identified** by its **Ethernet Address**

  Graphically illustrated:



- ***Identification* in a *heterogenous* network**
  - **Fact:**

    - An **Ethernet Address** is **48 bits**
    - A **Token bus address** is ***also* 48 bits**

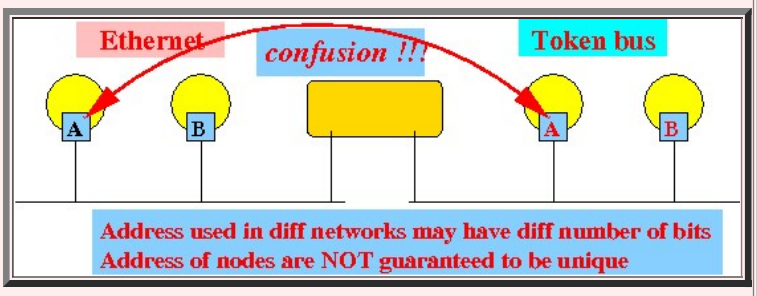
    Claim:

    - **Even** when the **address format** are ***identical***, we **cannot** use the **Ethernet/Token bus address** as **identifiers** !!!

    Reason:

    - **Two (2) different nodes** on the ***heterogeneous* network** can have the ***same* address** !!!

      Example:

      An **Ethernet/Token bus address** is **no longer *unique*** !!!

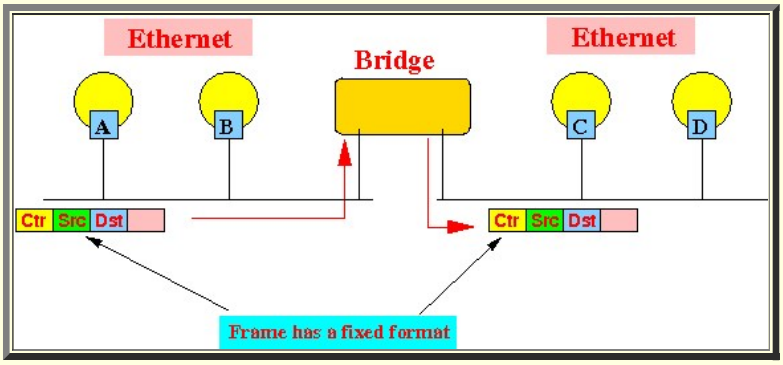
  - **Analogy:**

    - The **licence plate** of a **car** is ***unique*** within ***one* state** (e.g., some imaginary **Georgia licence plate XXX YYY** is **unique** in **Georgia**

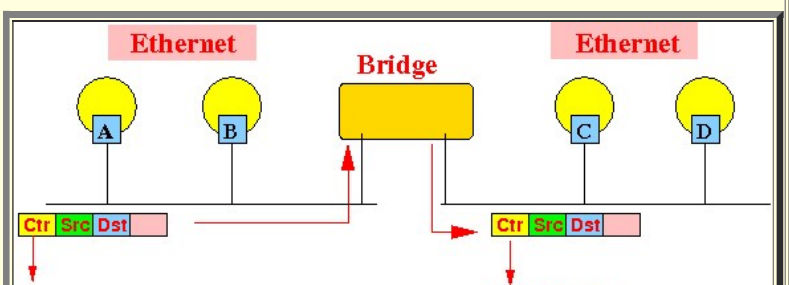
    - But this is ***not* true** for **licence plates** in ***two* different states**

- **The hereogeneity (frame *format*) problem**
  - In a **homogeneous network**:

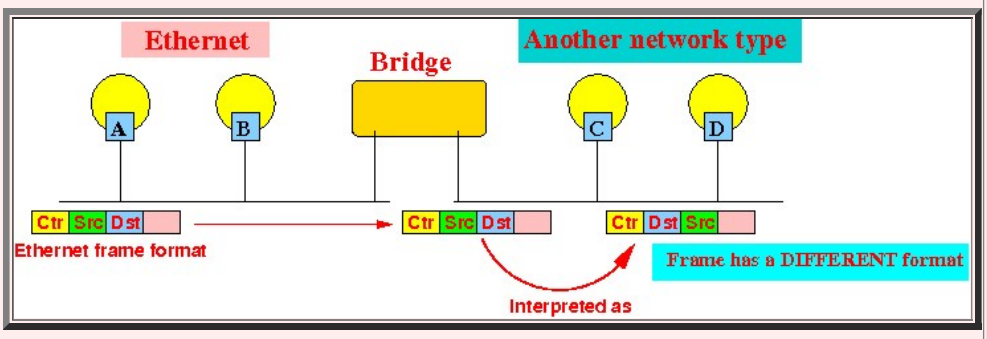
    - In ***homogeneous* networks**, ***all* frames** has the ***same* format**:

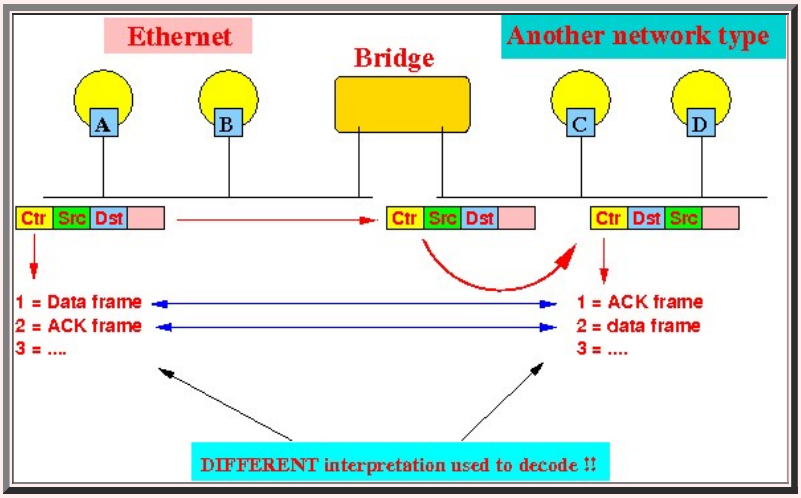
    - In ***homogeneous* networks**, node uses the ***same* encoding** to ***interprete* meaning**:

- The *different* **frame formats** used in ***heterogeneous* networks** will cause a ***Babel* problem**:
  - In ***heterogeneous* networks**, the **fields** in the **frame** can be **interpreted *incorrectly***:

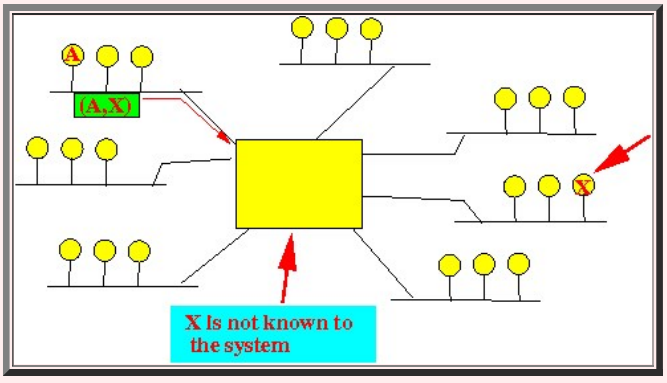


  - In ***heterogeneous* networks**, the **codes** in the ***control* field** can be **interpreted *incorrectly***:

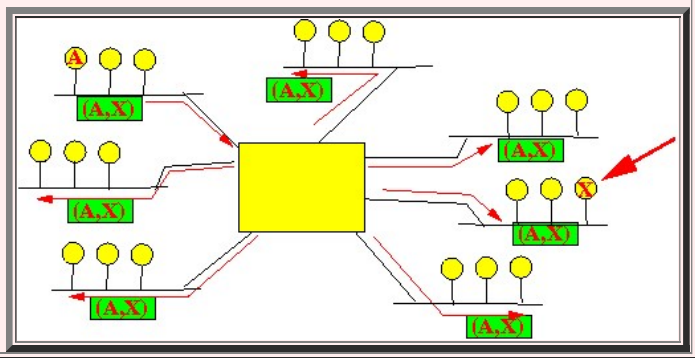


- **Analogy:**
  - **Each *type* of network "speak" a *different* language**
  - The **interconnecting *device*** must be ***smarter*** than a ***ordinary* bridge** because the **interconnecting device** must perform:
    - **Protocol *translation* !!!**

- **The *scalability* issue**
  - **Recall: *Flooding*:**
    - When an **Ethernet switch** does **not know** the **location** of a **destination**:



the **Ethernet switch** will ***flood*** **all networks**:



- **Scalability**:
  - **Network operation** must ***minimize*** the **use** of ***flooding*** !!!

- **Summary of the solutions (very brief)**
  - **Summary of solutions:**
    - **Establish** a ***uniform*** **addressing method** (and **assignment**):
      - We need a **world-wide Social Security Number (identification)** system for **every computer** on the **Internet**
    - **Establish** a ***common*** **packet format**
      - I.e.: we need a **network "Esperanto"** ***language*** spoken by **every computer** on the **Internet** !!!
    - **Minimzie** the **use** of ***flooding***:
      - **Avoid** the **use** of **promiscuous interfaces** (which encourage broadcasting)
      - **Retain** (simplified) **forwarding information** to ***all*** **destinations** on the **Internet**
  - We will discuss **each problem/solution** in more details **next**

    (Not neccessary in the order of the problems presented above)

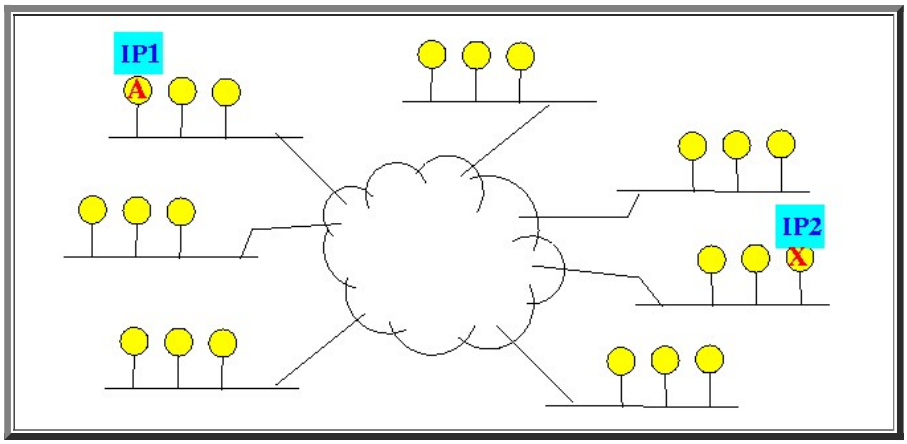
Logical identifiers

- **Solving the identification problem**
  - **How to *identify* a node (computer) *uniquely***:

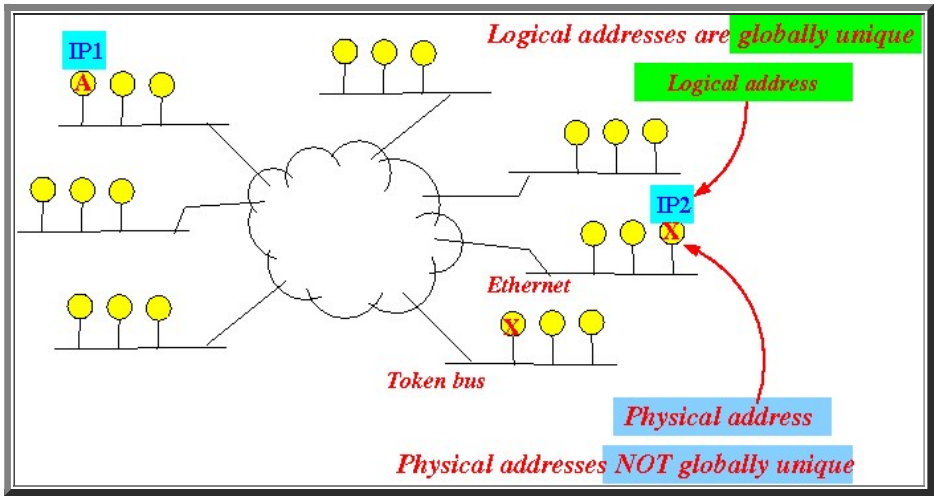
    - We **invent/assign** a set of ***logical* host identifiers (= numbers)**
      - These **identifiers/numbers** are called: ***IP addresses***

    Each node is **assigned** a **unique ID (IP address)**

  - **Example:**



- ***Important* note**
  - ***Each* node** now has **2 identifier's**:



Every **node** has:

- A ***physical* network address** that is **unique** per **network *type***

  Example:

  - An **Ethernet address** is ***unique*** among ***all* Ethernet nodes**
  - A **Token ring address** is ***unique*** among ***all* Token ring nodes**

- A ***logical* ("made up") IP address** that is ***unique*** in the ***whole* wide world** !!

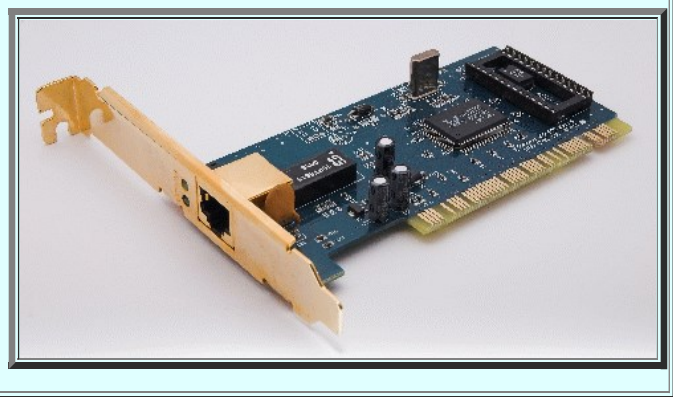
  - **Note:**

    - **Network addresses *and* IP addresses** are ***binary* numbers** !!!
      - I use **letters (symbols)** to **denote** the ***type*** of **addresses** !!!

Physical networks

- ***Physical* networks**
  - **Physical networks:**
    - **Physical network** = a **network** that is **established** by using **hardware (network cards)**

    Examples:
    - **Ethernet**
      - You can **buy** an **Ethernet network card** as ***hardware***:

        
    - **Token ring**
    - **Token bus**
    - **Ethernet segments** interconnected by **Ethernet switches**

- **Terminology for the Physical network**
  - ***Physical* (Network) Address:**
    - ***Physical* (network) address** = the **network address** of the ***physical* network**
      - The ***physical* address** is **embedded** inside the **network interface card (NIC)**
      - **Changing** a **physical address** is ***very hard*** to do (well, you can **spoof** (= masquerade), that's **illegal**)
  - ***Frames*:**
    - ***Frame*** = a **message** that is **transmitted** on a **physical network**
  - **ISO level:**
    - The ***physical* network** is **layer 1 and 2** in the **ISO reference model**:

Application layer	make sure the program communicate according to proper procedure (+ error recovery)
Presentation layer	make sure that all data are acceptable to all participants (encrypt, translate)
Session layer	make sure that all participants are aware of each other
Transport layer	make sure final destination receive data from source reliably
Network layer	determine which neighbor node to forward message to reach final destination
Datalink layer	make sure node receive data reliably from neighbor node
Physical layer	connect to your neighbor nodes

The *logical* network

- ***Logical* netwrk**
  - **Logical network:**
    - **Logical network** = a **network created** by the **use** of ***software***

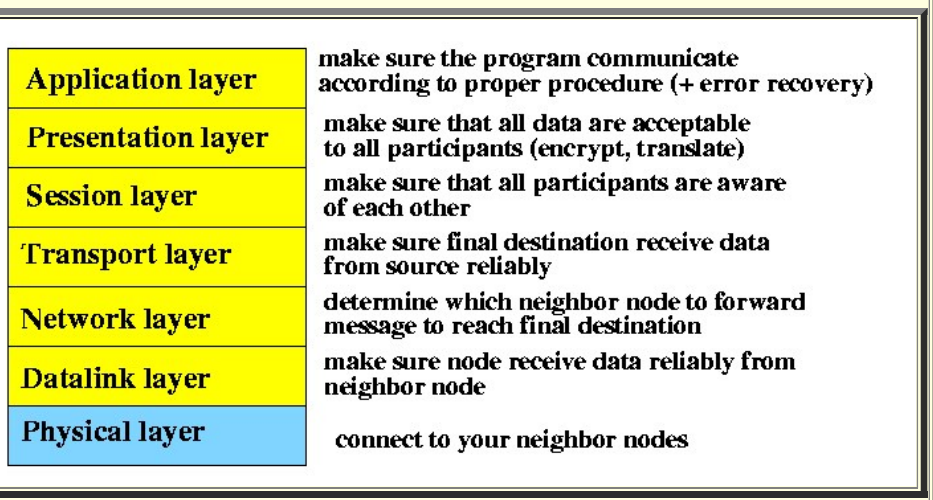
  - **More accurate definition**:
    - **Logical network** = a **network** that we **create (using software)** ***on top*** of multiple ***physical* network**

    - The **logical network** will ***use*** the **physical networks** to:
      - **Transmit *logical network* messages (= packets)**

- **Terminology for the Logical network**
  - ***Packets***:
    - **Packet** = a **message** that is **transmitted** between **nodes** in a ***logical* network**

      Note:
      - The ***logical* network** use ***use*** the ***physical* network** to transmitt **messages** !!!

  - **ISO level:**
    - The ***logical* network** is **layer 3** in the **ISO reference model**:



- **Logical and *virtual* networks**
  - ***Virtual* network**:
    - ***Virtual* network** = a **network** that does **not** exists ***physically*** but **created** by ***software***
      - **Logical networks** are ***virtual* networks** !!!

- **The Internet**
  - **Fact:**
    - The **Internet** is a ***virtual* network** (created by ***software*** !!!)

    Case in point:

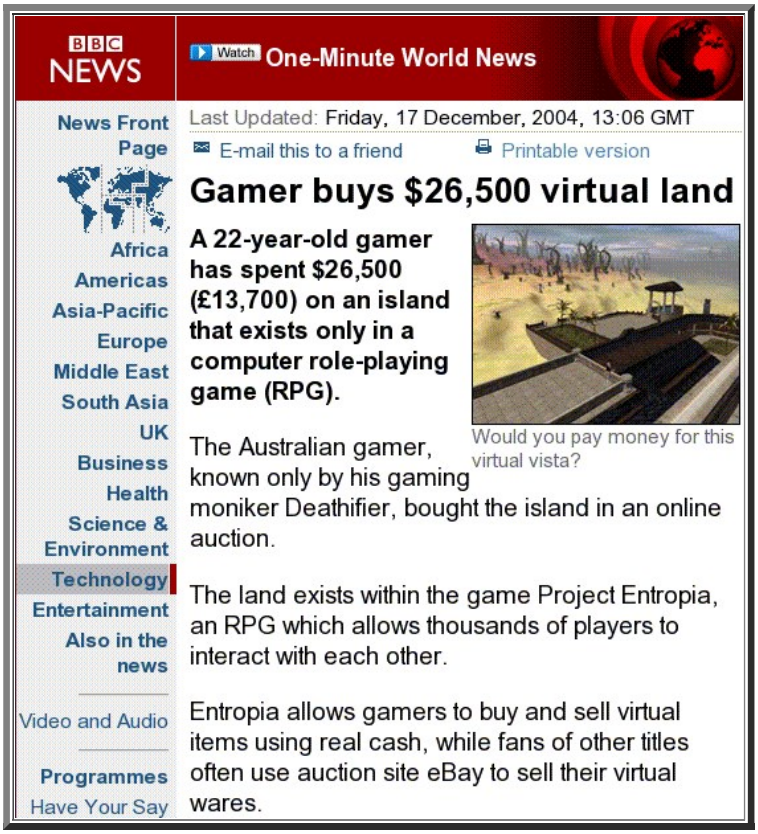
- You **cannot** buy a **network card** that uses **IP addressess**

- **Fact:**

  - There are many **virtual stuff** that are **pretty real**...

  Example:

  You can even **sell virtual islands** for **real money**:



BBC NEWS — Watch One-Minute World News

News Front Page | Africa | Americas | Asia-Pacific | Europe | Middle East | South Asia | UK | Business | Health | Science & Environment | Technology | Entertainment | Also in the news | Video and Audio | Programmes | Have Your Say

Last Updated: Friday, 17 December, 2004, 13:06 GMT

E-mail this to a friend Printable version

Gamer buys $26,500 virtual land

A 22-year-old gamer has spent $26,500 (£13,700) on an island that exists only in a computer role-playing game (RPG).

Would you pay money for this virtual vista?

The Australian gamer, known only by his gaming moniker Deathifier, bought the island in an online auction.

The land exists within the game Project Entropia, an RPG which allows thousands of players to interact with each other.

Entropia allows gamers to buy and sell virtual items using real cash, while fans of other titles often use auction site eBay to sell their virtual wares.

- ***Structure* of Internet packets**

  - **Complete structure** of an **IPv4 packet**:

bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (if Header Length > 5)				
160 or 192+	Data				

  - For this **discussion**, we use an ***abreviated*** **structure** of an **IP packet**:

```
  4 bytes 4 bytes   32      <= 64K
+-------+--------+-----+----------------+
| SrcIP | DestIP | CRC |      Data      |
+-------+--------+-----+----------------+
```

  For **now**, all we need to **know** is:

  - An **IP packet** has:
    - a **source IP address** **and**
    - a **destination IP address**

Note:

- An **IP packet** is **called**:
  - a ***Layer 3*** **"frame"**

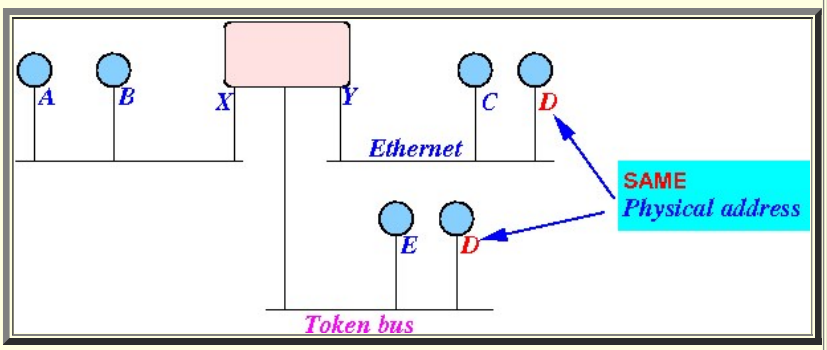
  (because the **IP layer** is a **Network layer (= layer # 3 in ISO model)**

- A ***Layer 2*** **frame** is:
  - a **frame** that is processed by a **Datalink layer (layer 2)**

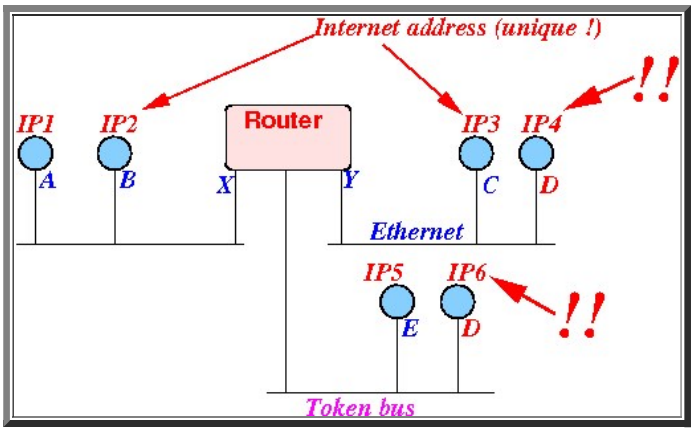
    (**Layer 2** is the ***physical*** **network** !!!)

Identifying nodes uniquely

- **Assigning unique logical network addresses**
  - **Suppose:**
    - ***Multiple* nodes** on the ***physical* network** have ***same* network address**:



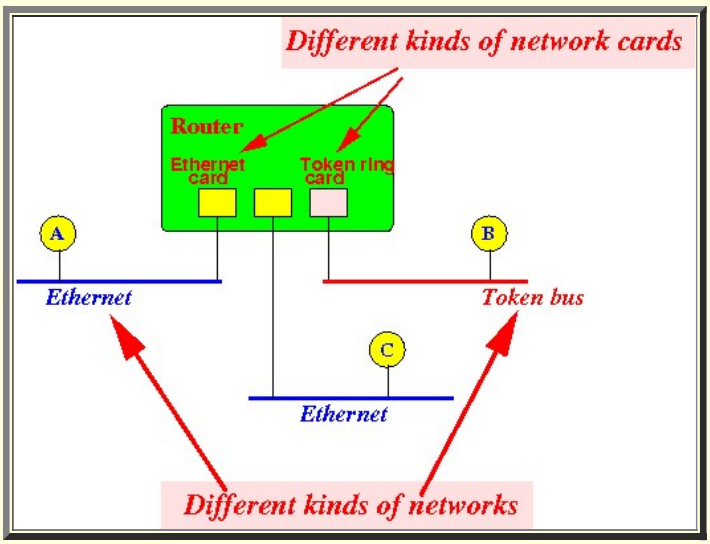
  - We can **assign *each* node** with a ***unique* network level (IP) address**:



Operation of a *router*

- **Routers**
  - **Router:**

    - **Router** = a **networking device** that can **inter-connect** multiple ***heterogeneous*** (and **also homogeneous**) **networks**



  - **Furthermore:**

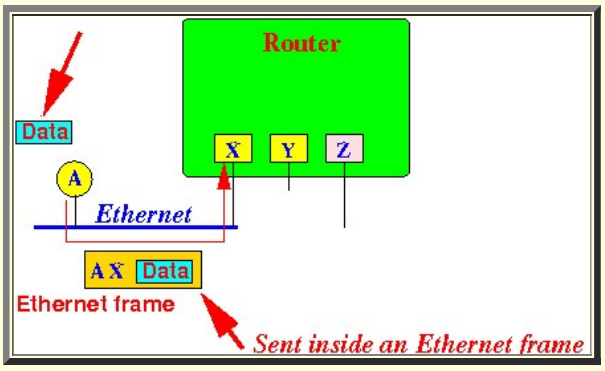
    - A **router** ***knows*** (= **programmed**):
      - The **structure** of a ***layer 3*** **frame (= IP packet)**

- **Terminology**
  - **Receiving data**:

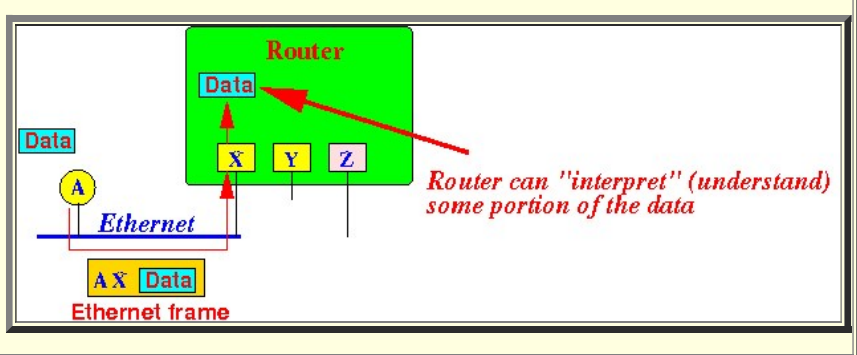
    - **Receive data** = having the **ability** to **extract** the **data** from a **frame** and **interpret** ***some*** **portion** of the **data**

    Illustrated:

    - **A** sends some **data** in an **Ethernet frame**:

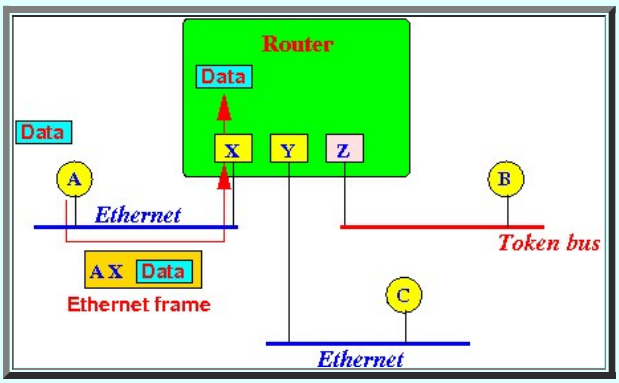


    - **Router** can **receive** the **data** if the **router** can ***extract*** the **data** and ***interpret*** some ***portion*** of the **data**:



- **Send and receive capabilities of a router**
  - **Capabilities** of a **router**:

    - A **Router** can **receive layer 2 frames** from ***any*** **(attached) network**:

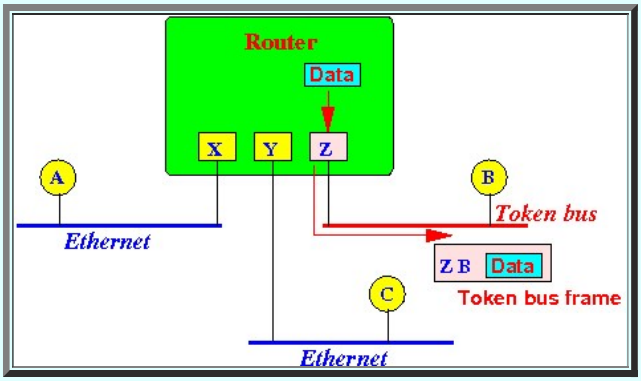
      Note:

      - **Data** will be an **IP packet** (see later)

    - The **router** can ***interpret*** **(= understand)** the **header information** inside the **data (= IP packet)**

      - The **header portion** of the **data (= IP packet)** contains the **destination IP address**
      - The **router** will use the **destination address** to **decide where (= which output)** to **send** the **data (= IP packet)** next

    - A **Router** can **transmit the *data* (= IP packet)** in a **layer 2 frame** on ***any*** **(attached) network**:

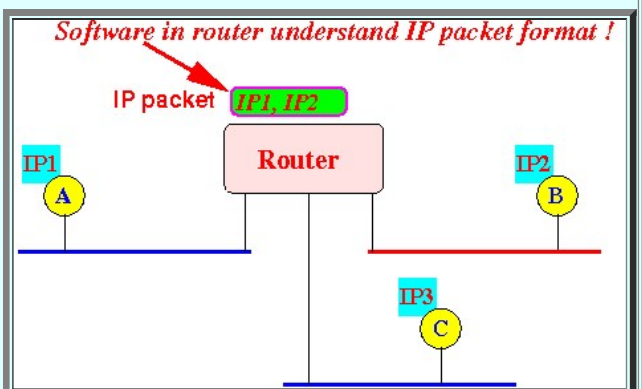
      

- **A router knows the IP packet format**
  - A **router** ***knows*** the ***structure*** of an **IP packet**:

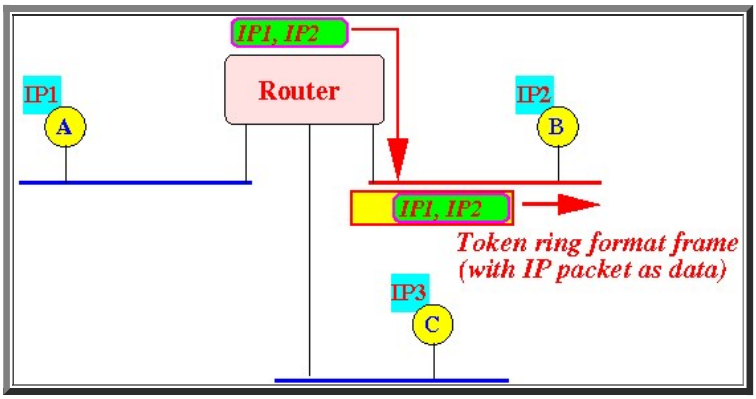
    - A **router** can **read**:

      - the **source IP address** in an **IP packet**
      - the **destination IP address** in an **IP packet**

      Graphically:

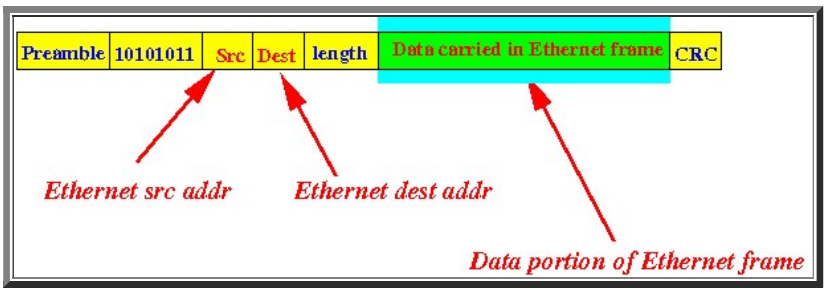
      

- The **router** will **transmit** the **IP packet (unchanged)** as ***data*** in a **level 2 (Ethernet/Token ring) frame**:



Packet transmission on the logical network

- ***How* to transmit *packets***
  - How to **transmit any kind** of **data** in **Ethernet**:



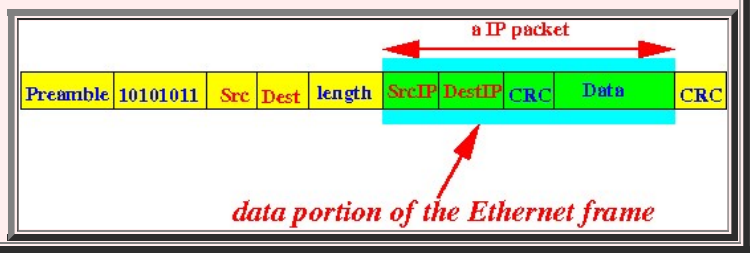
***Note*:**

- You can **transmit *any* kind** of **data** in the **data portion** of the **Ethernet frame**
  - We will **transmit** an ***IP packet* *using*** an **Ethernet frame** !!!

  - **How to** to **transmit** a **(*logical* network) IP packet**:

- We **put** an ***IP* packet** inside the ***data* portion** of the **Ethernet frame**

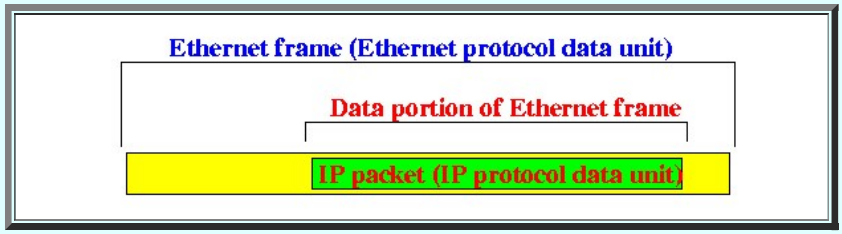
Graphically:



- **Encapsulation**
  - **Encapsulation:**

- **Encapsulation** = **transmission technique** where an **entire packet** of a **protocol A (= *Internet* packet)** is **carried *inside*** a packet of **protocol B (= *Ethernet* frame)**

Graphically:



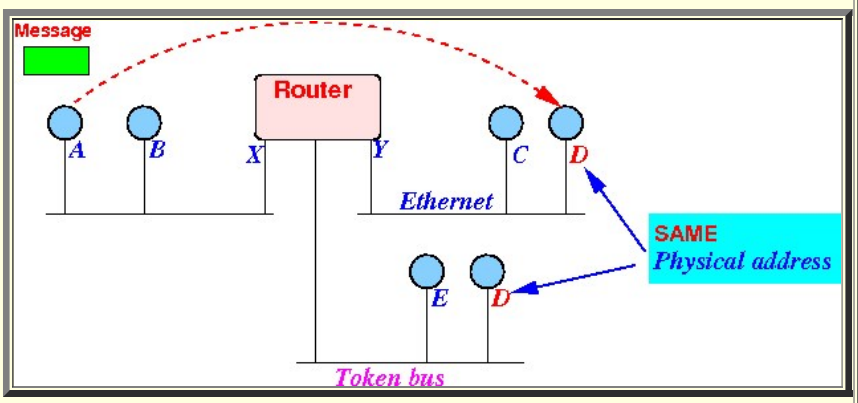
It's like a **capsule**:

  - **Real life encapsulation:** the **FedEx** service

We can **put** a **letter (= packet A)** inside a **FedEx envelop (= packet B)**

- **Packet forwarding in the logical network**
  - **Scenario:**

    - **Node *A*** wants to **transmit** a **message** to ***node D***:

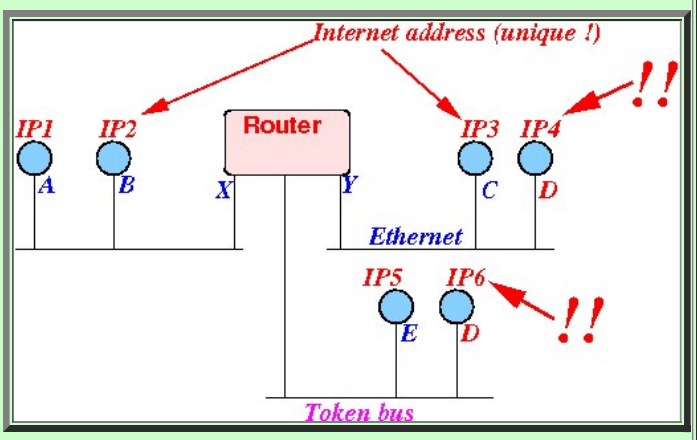


Notice:

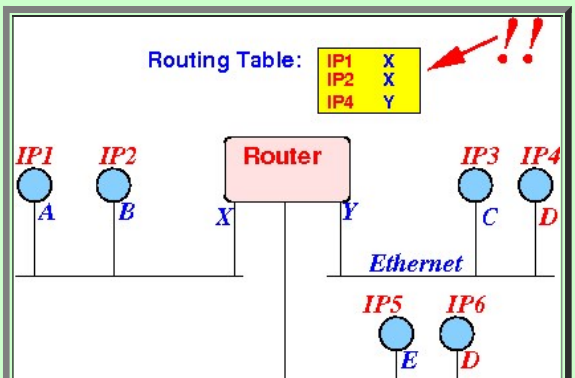
- There are ***two* (2) different node** with the ***same* physical address (*D*)**

  - **Prelude:**

    - ***Each* node** is **assigned** a ***unique* IP address**:



- The ***router*** contains ***information* (= routing table)** on **where** to **forward IP packets**:

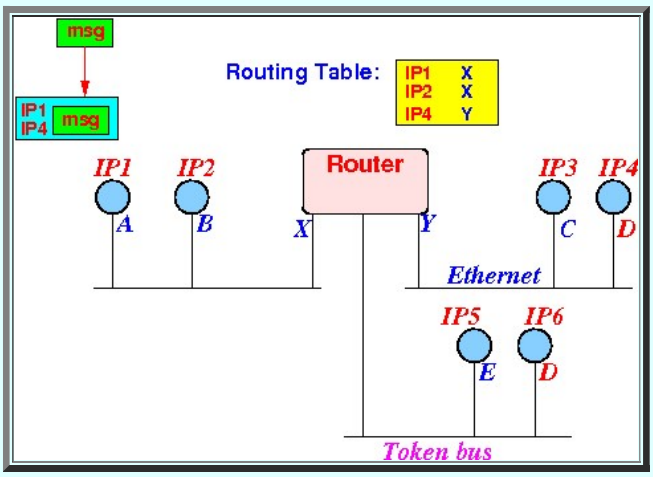


- **How** the ***logical*** **network** forward **packets**:

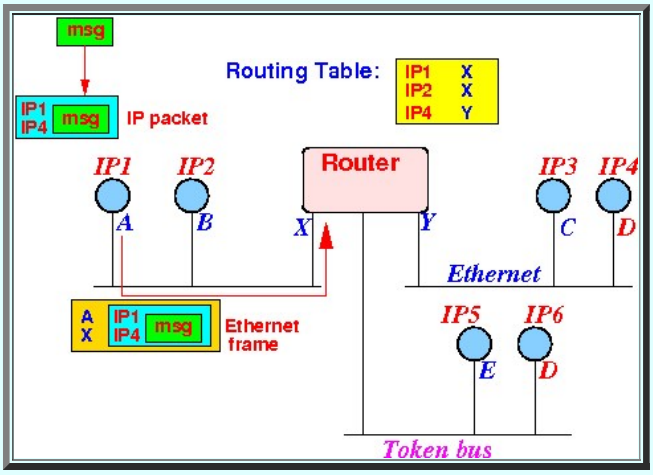
  - The **messsage** is **put** inside an **IP packet** with

    - **source IP address** = **IP1**
    - **destination IP address** = **IP4**

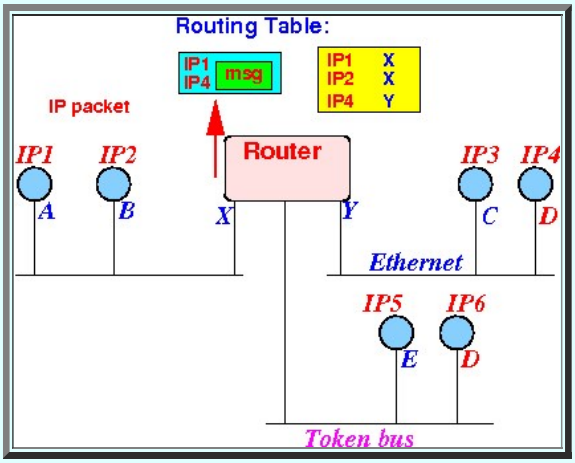
    Graphically:



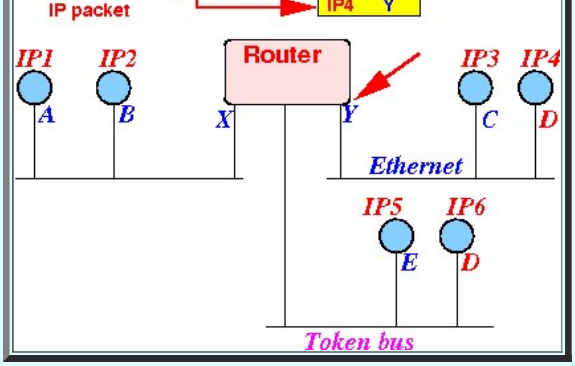
  - The **IP packet** is **transmitted** ***inside*** a **Ethernet frame** to the **router**:



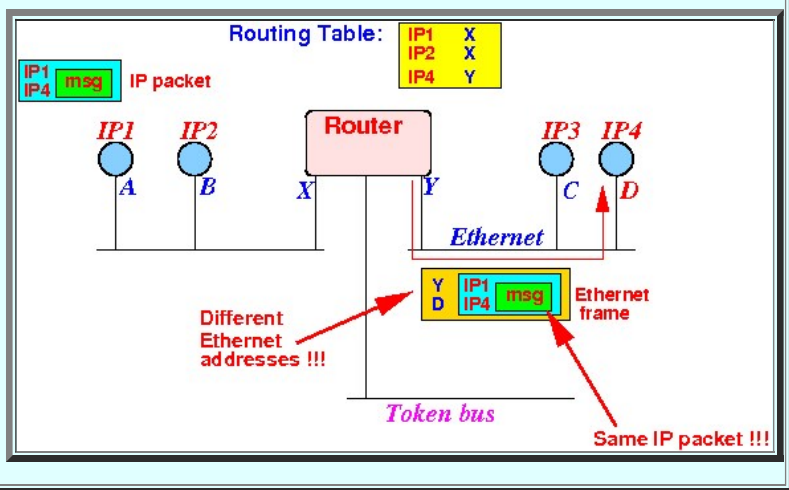
  - The **IP packet** is ***extracted*** from the **Ethernet frame**:



  - The **router** uses the ***destination*** **IP address** to find the **output port**:



- **Then forwards** the **IP packet** ***inside*** ***another*** **Ethernet (or some other type) frame**:



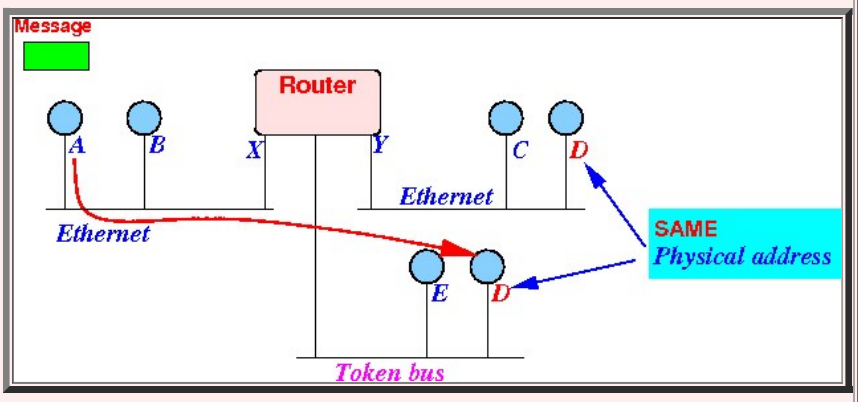
Solving the *heterogeneity* problem

- **Heterogeneous networking**
  - **Connecting** to ***different*** **types** of **networks**:

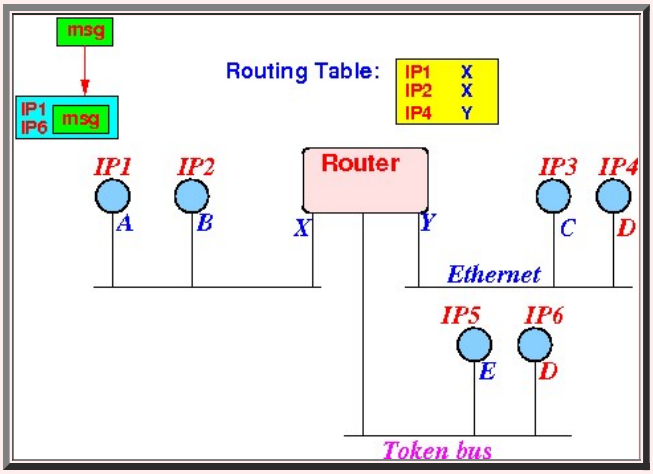
    - It should be **clear** that the **forwarding method** will ***also*** work on ***different*** **types** of ***physical*** **network**

    Example:

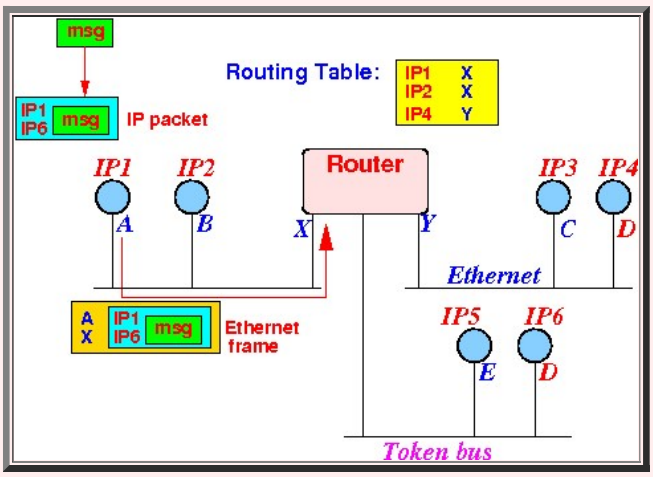
    - **Suppose** node ***A*** wants to **send** a **message** to ***D***:

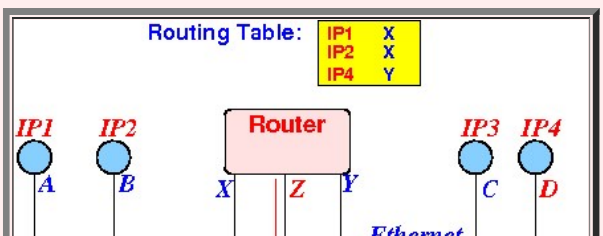
    - **Node** ***A*** will transmit the **follow** **IP packet**:

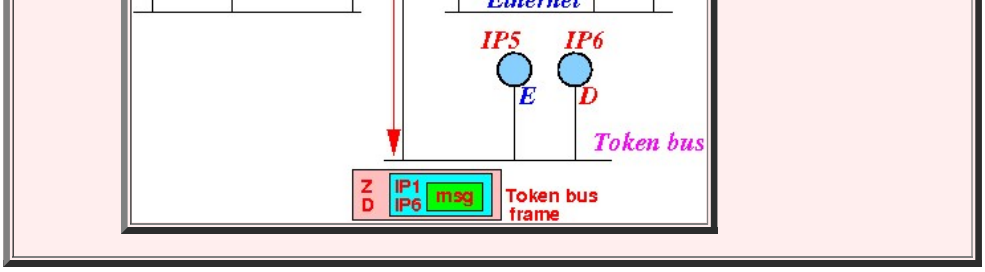
      

    - The **IP packet** will be **transmitted** with an **Ethernet frame** to the **router**:

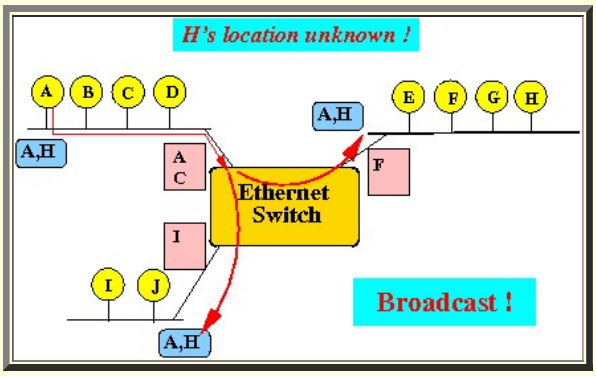
    - The **IP packet** will be **forwarded** to ***D*** using a **Token bus frame**:


Ethernet
IP5
IP6
E
D
Token bus
Z
D
IP1
IP6
msg
Token bus
frame

Key to scalability -- minimize intrusion

- **Key to achieving scalability**
  - **Key** to **achieving scalability:**
    - **Avoid** the **intruding** on ***non-dstination*** **networks**
  - **Why connecting multiple Ethernet LANs** is **not scalable**:
    - When the **location** of the **destination** is ***unknown***, a **Ethernet switch** will ***broadcast*** the **frame** on ***all*** **attached networks**



- **How to avoid broadcasting**
  - **Avoid** having to use **broadcasting**:
    - **Each node** must **maintain** the **location information** of ***every*** **node** (on the **entire Internet** !!!
  - **Note:**
    - The **amount** of **location information** is ***too*** **much** for ***any*** **single node** to **store**.
    - The **information** will be **stored** in a ***distributed*** **manner** to **reduce storage size** !!!

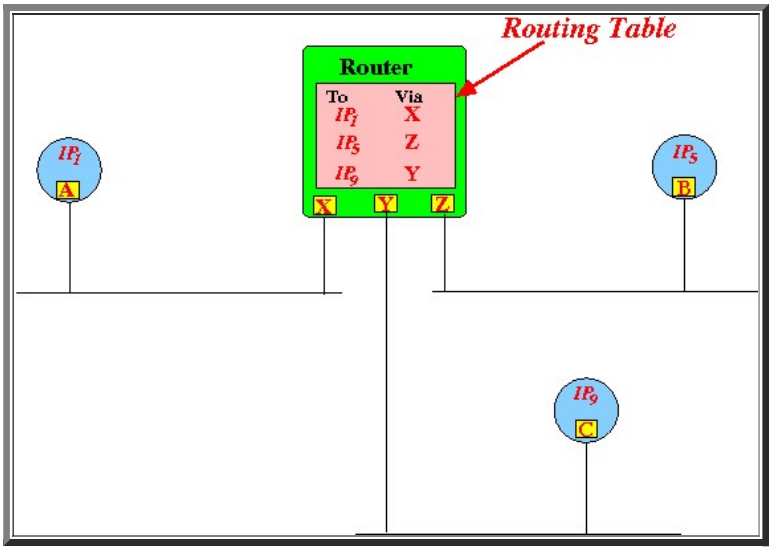
- **Routing Table**
  - **Routing Table:**
    - **Routing table** = contains **location information** on ***every*** **IP destination**
    - **How to** store **routing table** in a **distributed manner:**
      - ***Most*** **router only** store **location information** of their ***"close-by"*** **IP destinations**
      - **Most router** has a **default router** (that store a ***larger*** **routing table**)
        - If a **destination** is **unknown (= not found in routing table)**, the **router** will **send** the **IP packet** to its **default router** for **processing**
  - **(*Simplified*) structure** of the **routing table:**
    - The **routing table** contains **entries** of the form:

```
   Destination  |  Output port
  --------------+---------------
     IP-addr1   |    port1
     IP-addr2   |    port2
     ...        |    ...
```

    (We will **discuss** the **complete structure** **later**)

(The **routing table** ***actually*** store ***IP network*** **addresses**.)

- **Example:**



Explanation:

- If **router** receives an **IP packet** destined for node IP_1, it will send the **packet** out on its **port** ***X***
- If **router** receives an **IP packet** destined for node IP_5, it will send the **packet** out on its **port** ***Z***
- If **router** receives an **IP packet** destined for node IP_9, it will send the **packet** out on its **port** ***Y***

- **Command** to **view** the **routing table** in some **computer**:

```
netstat  -r     (r = routing table)

Destination     Gateway          Genmask         Flags   MSS Window  irtt Iface
170.140.150.0   *                255.255.254.0   U         0 0          0 eth0
link-local      *                255.255.0.0     U         0 0          0 eth0
default         defaultrouter.m  0.0.0.0         UG        0 0          0 eth0
```

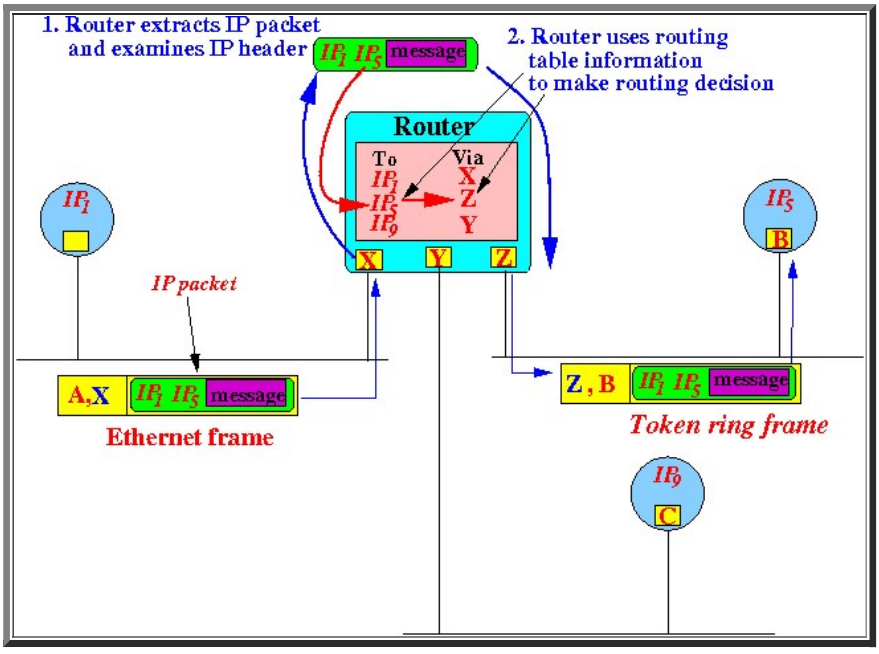
You can **view** the **routing table** in **numerical format** with:

```
netstat -r  -n

Destination     Gateway          Genmask         Flags   MSS Window  irtt Iface
170.140.150.0   0.0.0.0          255.255.254.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0          255.255.0.0     U         0 0          0 eth0
0.0.0.0         170.140.150.254  0.0.0.0         UG        0 0          0 eth0
```

- **How a *router* make its routing *decisions***

- **Recall**: **how to forward** an **IP packet** by a **router**



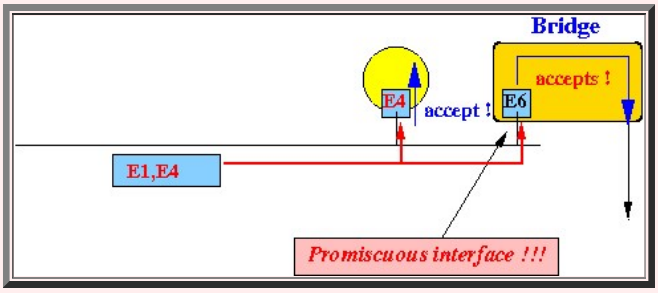
Explanation:

- The **router** will **accept** the **frame** because the **(Ethernet) destination** is its **network address**
- The **router** will first **extract** the **data portion** from the **Ethernet frame**
- The **router** then **examine** the **destination IP address** in the **IP packet** (which is in the **data portion**)
- The **router** finds the **destination** in its **routing table**
- **Finally**, the **router** uses the **appropriate port** to transmit the **frame** (in the **network format** of that **port**)
    - The **network interface card** knows the **proper frame format**

Difference between a bridge and a router

- **How a bridge operates**
  - **Properties** and **operation** of a ***bridge***:

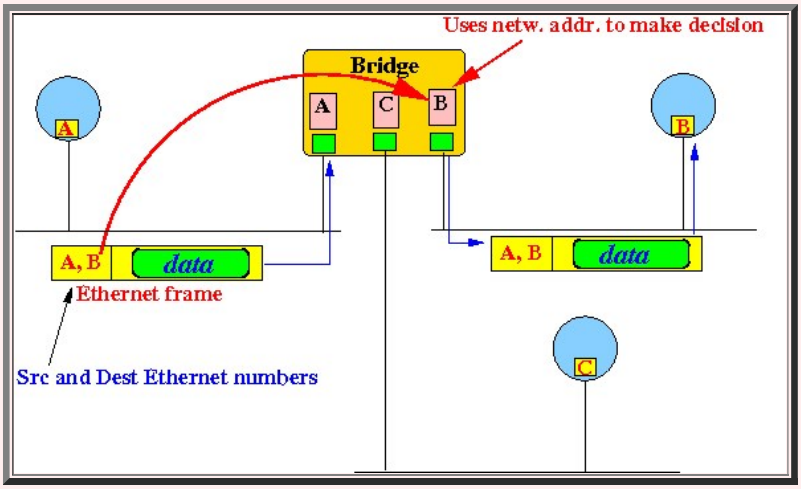
    - A **bridge** has ***promiscuous*** **ports**:

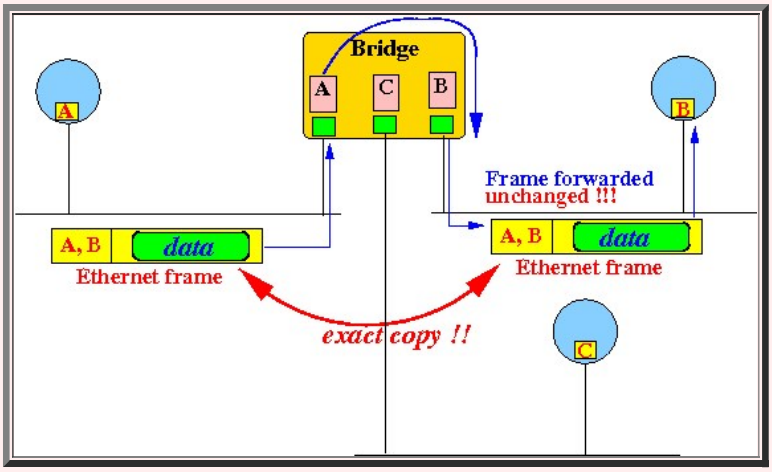
      Promiscuous port:

      - **Promiscuous port** = a **network port** that **receives *every* (Ethernet) frame** on the **network**

    - A **bridge** makes its **forwarding decisions** using the ***physical*** **(Ethernet) addresses**

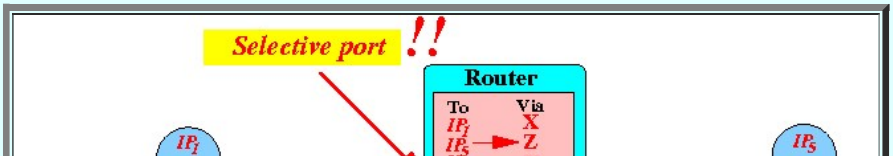
      

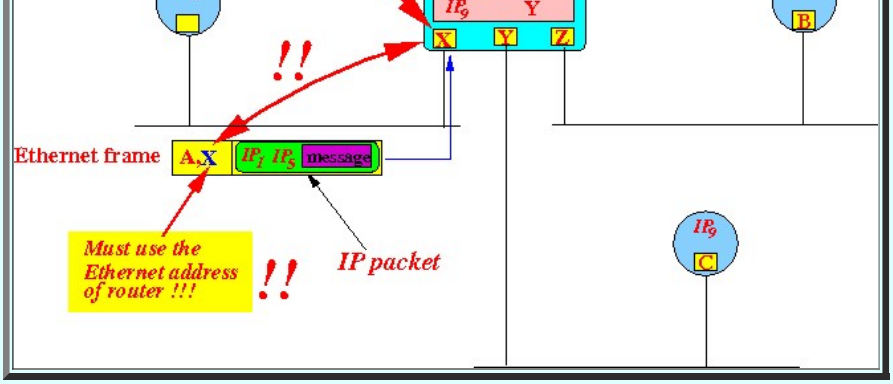
    - A **bridge** will ***repeat*** a **(received) frame** ***verbatim***

- **How a router operates**
  - A ***router*** operates as **follows:**

    - A **router** has ***selective*** **(= *non*-promiscuous) ports**:

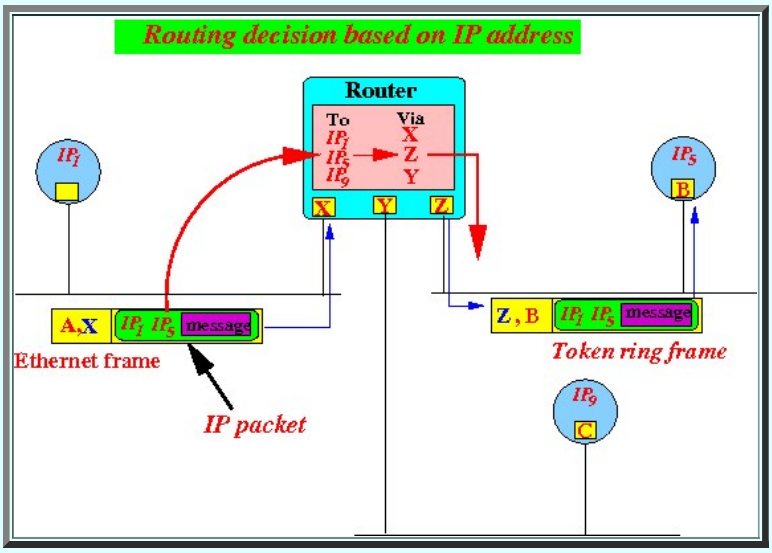
      



Selective port:

- *Selective* **port** = a **network port** that ***only*** **receives** an **(Ethernet) frame** that contains its **own address** as ***destination***

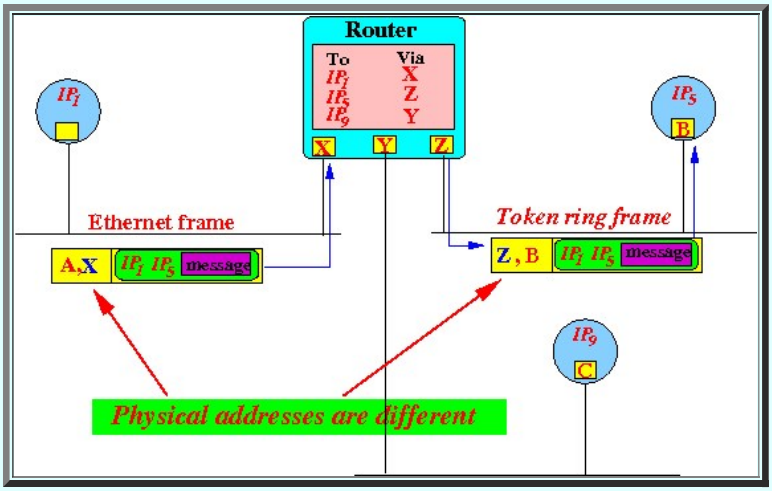
- A **router** makes its **routing decision** using ***logical*** **(IP) addresses**



Note:

- The **logical IP address** is found in the ***data*** **portion (IP packet)** of the **frame** !!!

- The ***received*** **frame** and the ***transmitted*** **frame** (by the **router**) contain ***different*** **physical addresses**:



Notice that:

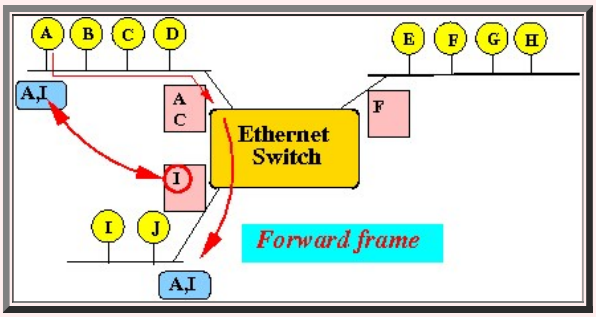
- The ***physical*** **addresses** are **different** !!!

- **But** the ***logical*** **addresses** (in the **IP packet**) remains the ***same***

- **Network jargon: Routing and Forwarding**
  - **Forwarding:**

- **Forwarding** = sending a **message** to another network using ***physical* addresses** (e.g., **Ethernet address**)

  Example:



- **Forwarding** will ***not*** change the **frame**

  - In the **above figure**, you can see that the **frame (A,I)** is **repeated *verbatim*** onto a ***different* Ethernet segment** !!!

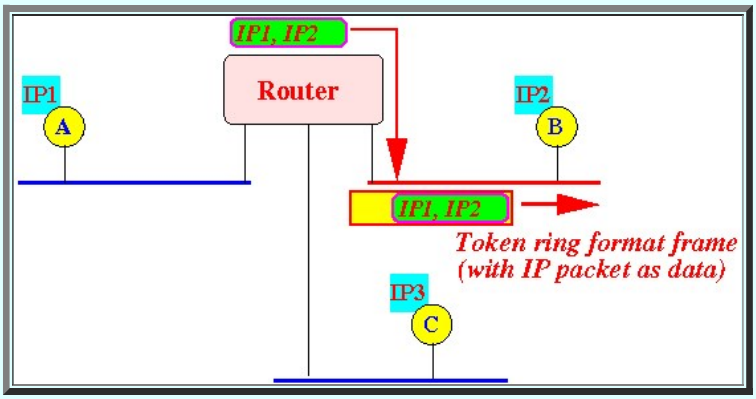
- **Forwarding** will **therefore *only* work** with ***homogeneous* networks**

  - Because there is ***no* protocol translation** taking place !

- **Routing:**

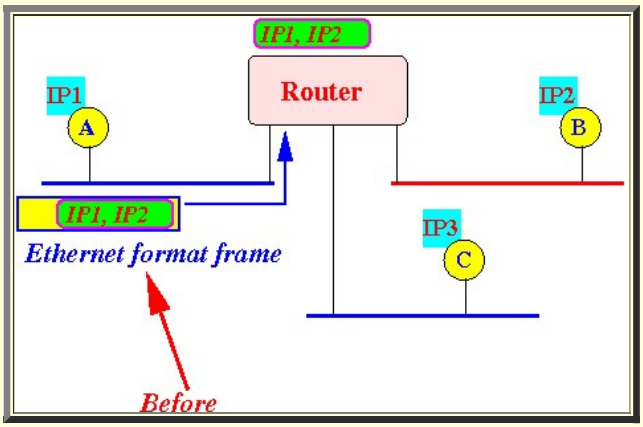
  - **Routing** = sending a **message** to another network using ***logical* addresses** (e.g., **IP address**)

    Example:

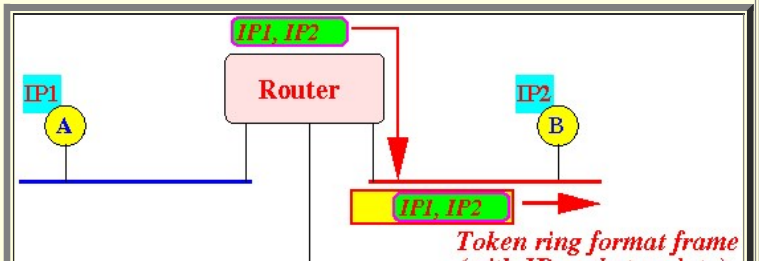


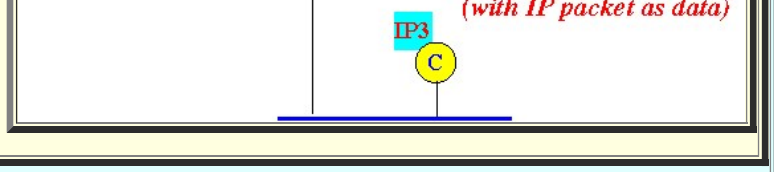
  - **Routing *will*** change the **frame**:

    - **Before:** the **frame** was an **Ethernet frame**



    - **After:** the **frame** is a **Token bus frame**



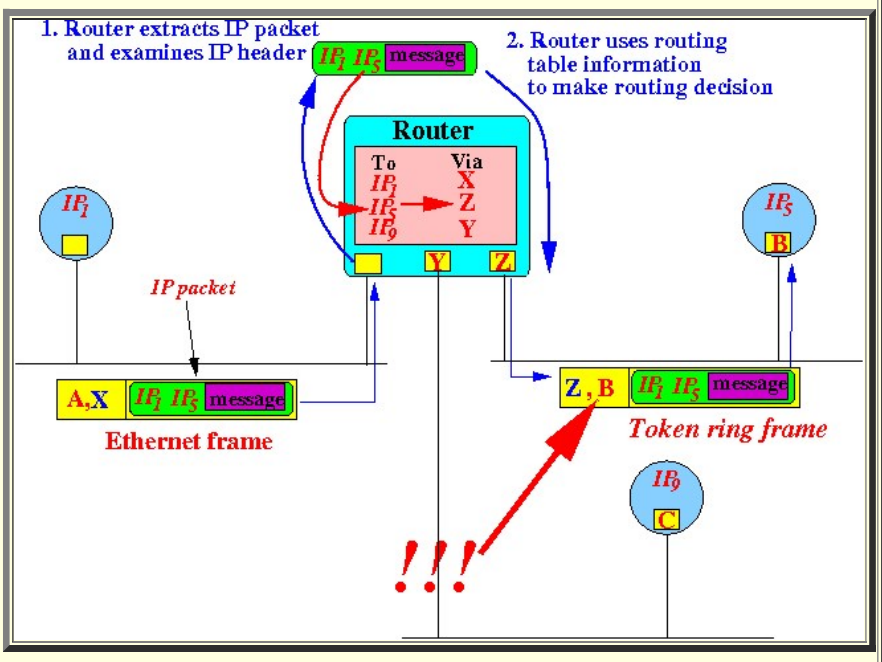


- **Routing** will **work** even on ***heterogeneous*** **networks** !!!

Important *omission* in the discussion of *routing*

- **Important fact**
  - **Fact:**

    - When a **router** **transmit** an **IP packet** to the **destination** **(= IP_5)**:



    the **router** must **use**:

    - The ***physical*** **address** **(= B)** of the **destination** **(= IP_5)** in the **Ethernet frame** !!!

- **Important omission**
  - I have **deliberately** **omitted** the **following** **detail** to **keep** the discussion **simple**:

    - ***How*** does the **router** know the ***physical*** **address** of a ***destination*** **(= IP_5)** ???

      (This **information** is ***not*** **stored** in the **routing table**)

  - We will **answer** this **question** **later** in the **Section** on the **Address Resolution Protocol (ARP)**

    --- if you can't wait, then: **click here**

Intro to the IP protocol

- **Purpose of the Internet Protocol**
  - **Before** the **IP protocol**:
    - There was a **proliferation** of **network protocols** interconnecting ***many*** **different kinds** of ***physical*** **networks**

  - **Goal** of the **IP protocol**:
    - The **IP protocol** was designed to ***interconnect*** ***any*** **type of networks**

      I.e.:
      - "One network to ***rule*** them (networks) all" :)

    "*Any* type of network" would **include** even:
    - **physical network types** developed in the ***future***

    How to **achieve** this **goal**:
    - The **IP protocol** requires ***very*** **little capability** of the ***physical*** **network.**
      - As long as the ***physical*** **network** can **deliver** a **frame** to the **destination**, the **physical network** can run the **IP protocol** !!!

  - **Case in point:**
    - The **IP protocol** can even be and ***in fact has been*** **implemented** using **carrier** ***pigeons***: **click here**

IP over Avian Carriers

From Wikipedia, the free encyclopedia

In computer networking, **IP over Avian Carriers** (IPoAC) is a humorously-intended proposal to carry Internet Protocol (IP) traffic by birds such as homing pigeons. IP over Avian Carriers was initially described in RFC 1149, a "Request for Comments" (RFC) issued by the Internet Engineering Task Force (IETF) written by D. Waitzman and released on 1 April 1990 (April Fools' Day). It is one of several April 1 RFCs.

A homing pigeon can carry Internet Protocol traffic.

Waitzman described an improvement of his protocol in RFC 2549, *IP over Avian Carriers with Quality of Service* (1 April 1999).

IPoAC has been successfully implemented, but for only nine packets of data, with a packet loss ratio of 55% (due to user error[1]), and a response time ranging from 3000 seconds to over 6000 seconds. Thus, this technology suffers from poor latency. Nevertheless, for large transfers avian carriers are capable of high average throughput when carrying flash memory devices.

- **IP version 4 and IP version 6**
  - **Fact:**
    - There are **2 versions** of **Internet Protocols** in **existence** today:
      - **IPv4** ("version 4")
      - **IPv6** ("version 6")

- **IPv4:**
  - **IPv4** was the ***original*** **IP protocol** that became **well-known** over the world
  - But due to its **popularity**, **IPv4** was **about** to **run out** of **IPv4 addresses** (**IPv4** has $2^{32} = 4 \times 10^9$ addresses)
- **IPv6:**
  - **IPv6** was **standardized** in **1998**
  - **IPv6** has $2^{128} = 3.4 \times 10^{38}$ addresses !!!
- **Fact:**
  - **Most computers/routers** today have **both** **IPv4** and **IPv6** in their **Operating System**
    - Such **devices** are known as ***dual*** **stacked**
- **IPv4 addresses** are officially **exhausted**:
  - **IPv4** ran out of **addresses** in **Sept 2015**:
    - click here

- **Datagram**
  - **Terminology:**
    - An **IP packet** is a **datagram**
  - **Properties** of a **datagram**:
    - **Unreliable delivery**
      - An **IP packet** can be ***lost*** (= **not received** by the **receiver**)
    - ***No Acknowledgement***
      - The **receiver** does ***not*** **acknowledge** an **IP packet** when the **packet** is **received**
    - ***Duplicate*** **reception** !!!
      - The **receiver** can receive ***multiple*** **copies** of the ***same*** **datagram** !!!

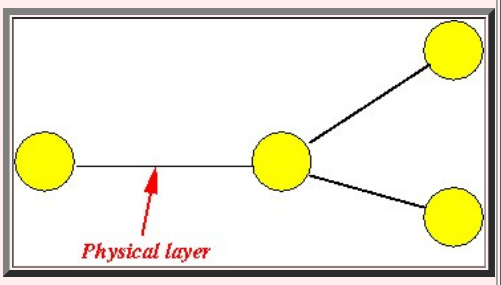
- **A bigger picture view**
  - **Services** proviced by the **IP layer**:
    - The **IP layer** provides ***similar*** **transmission service** as the:
      - **Physical layer** !!!!

  Compare the **IP datagram** and a **transmission** in the **physical layer**:

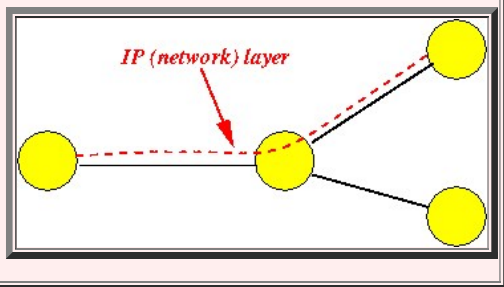
  - **Both** can be **lost**
  - **Neither** requires **ACKs** (the **Datalink layer** uses **ACKs**, not the **physical layer** !!!)

- **Difference** between the **IP layer** and the **physical layer**:

  - The **physical layer** connects **two *neighboring* nodes**:

  - The **IP (network) layer** connects ***any* two nodes**:

- The **reliable communication** function will be **implemented** in the ***next* communication layer**:

  - The **Datalink layer**:

    - **uses** the **unreliable frame delivery service** of the ***Physical* layer** to implement **reliable communication** between ***neighboring* nodes**

  - The **Transport layer** (discussed later):

    - **will use** the **unreliable *packet* delivery service** of the ***Network* (IP) layer** to implement **reliable communication** between ***any* 2 nodes**

The IP packet format (structure)

- **How to create the virtual (IP) network**
  - **How** to **implement** the **Internet protocol**:
    - A **router** or **computer** that **implements** the **Internet Protocol** contains **software (program)** that:
      - process **messages (= packets)** in the **IP packet *format***
    - The **software** will:
      - **Find** the **source/destination IP addresses** in the **IP packet**
      - **Lookup** the **output interface** based on the **destination address**
      - **Transmit** the **IP packet** on an **output interface**
  - The **Internet**:
    - **Internet** = a **network** of **routers** and **computers** that exchange **messages ("packets")** that conform to the ***IP packet* format**.
- **The IP Packet Format**
  - There are **nowadays 2 types** of **IP packet formates**:
    - **IP version 4 (IPv4)** --- this is the **original IP format**
    - **IP version 6 (IPv6)** --- this is the ***new* IP format** that will replace **IPv4** hopefully, one day in the future....
- **The IPv4 packet format**
  - The **IPv4 packet** consists of:
    - The **IPv4 header**
      - The **header** contains **control information**
    - The **payload**
      - The **payload** (= **data portion**) is the **message** carried inside an **IP packet**
      - The **most common payload** carried inside an **IP packet** is:
        - A **TCP protocol message**
  - The **IPv4 packet format**:

bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (if Header Length > 5)				
160					

or 192+	Data

How the **fields** in the **IP packet** is **used**:

- **Version:**
  - **0100 (4)** (4 bits) - identifies the ***version*** of the **IP protocol**

- **Header Length:**
  - **number of 32-bit words** in the **header**.

- **Type of Service (a.k.a: Differentiated Services):**
  - specify a preference for **how** the datagram would be **handled** (a kind of ***priority*** indicator)

  Comment:
  - This field is now defined by **RFC 2474** for **Differentiated services (DiffServ)** and
  - by **RFC 3168** for **Explicit Congestion Notification (ECN)**

- **Total Length**:
  - **number of bytes** of the entire IP datagram, in bytes (16 bits)

  Comments:
  - The **smallest IP packet** with **no data** (only the **IP header**) is **20 bytes** long
  - The **largest IP packet** has **65,535 (2^{16} - 1) bytes** (total length)

- **Identification**:
  - This **field** is used in the **fragmentation operation**

    (See ***next*** **webpage**)

- **Flags**:
  - The **flags** is a **three-bit field** used to **control or identify fragments**.

    (See ***next*** **webpage**)

  Comment:
  - **Bit 1**: Reserved; must be zero (meaning: can't find a use for the bit yet :-))
  - **Bit 2**: **Don't Fragment (DF)**
  - **Bit 3**: **More Fragments (MF)** (explained below)

- **Fragment Offset:**
  - specifies the **offset of this fragment** relative to the **beginning** of the **original *unfragmented* IP datagram**.

    (Explained in the ***next*** **webpage**)

- **Time To Live (TTL):**
  - Used to limits the **range of the packet**

  How the **TTL field** is **used**:
  - **Each router** that routes an **IP packet** will ***decrement*** the **TTL** field by **one**.

  - When the **TTL field** becomes **zero**, the **IP packet** is **discarded**.
    - This **feature** will **stop** an **IP packet** from **looping forever** !!!

- **Protocol:**

  - Specifies the **Transport layer protocol (= type of message)** that is carried in the **data portion** of the **IP datagram**.

  Common "protocol" values:

  - **6** = **TCP**
  - **17** = **UDP**

  More details, see: **click here**

- **Header Checksum:**

  - This is the **CRC code**

  Comment"

  - Checks ***only*** the **header**

    (Does not include the **data portion** in the **check sum**)

  Note:

  - **Because** the **TTL field** is ***decremented*** **(changed)** by the **router**, the **checksum** must be **recomputed** each time by the **router**.

- **Source and Destination address:**

  - **IP address** of the **source and destination**

- **Options:**

  - **Extra fields** --- such as additional **header fields**

  (**Not of interest** to this course, will not discuss further)

- **The IPv6 packet header**
  - The **IPv6 packet format:**

Octet Offset		0								1								2								3							
	Bit Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Traffic Class								Flow Label																			
4	32	Payload Length																Next Header								Hop Limit							
8	64	Source Address																															
C	96																																
10	128																																
14	160																																
18	192	Destination Address																															
1C	224																																
20	256																																
24	288																																

How the **fields** in the **IPv6 packet** is **used**: **Notes:**

- **Version:**:

  - **0110 (6)** (4 bits) --- identifies verion **6**

- **Traffic class**:

  - contains the **priority** of this **IP packet** (8 bits)

- **Flow label**:
  - Used to indicate the **QoS (Quality of Service) management** function (20 bits)

  Comment:

  - **Originally** created togive ***real-time*** **(video) applications** special service, but currently **unused**.

- **Payload length:**
  - The **size of the payload (= data portion)** in **# of bytes** (16 bits).

- **Next header:**
  - Contains the **location** of the **next protocol data** (8 bits).
    - ***Normally*** **(usually)**, the **next header** will be **location** of the the **transport layer protocol data**

    - Under **special circumstances**, the **sender** can **add** ***options*** to the **IPv6 header**

- **Hop limit:**
  - is the **same filed** as the **time to live (TTL) field** of **IPv4** (8 bits).

- **Source and destination addresses:**
  - same as **IPv4**, except the **IPv6 addresses** are **128 bits** each.

- **Packet fragmentation/re-assembly in IPv6**
  - **Fact:**
    - The **IPv6 header** does **not** contain **fragmentation/re-assembly fields**

    - **IPv6 does** have **fragmentation/re-assembly**:
      - **IPv6** provdies **fragmentation/re-assembly** using the **fragment** ***option*** **header**

        See: **click here**

The Datagram delivery service of IP

- **Services provided by the Internet Protocol**
  - The **IP protocol** ***software*** ***must*** **privide** the following **(*minimal*) services**:

1. ***datagram*** **packet delivery**
   - An **IP node** must **forward** an **IP packet** towards the **destination node**
   - The **delivery service** is ***unreliable*** **(datagram)**

2. ***Fragmentation*** **and** ***Re-assembly***
   - **Fragmentation**:
     - **One (large) IP packet** can be **split up** into **many (smaller) IP packets**

     (and each smaller **IP packet** is **transmitted separately**)
   - **Re-assembly**:
     - The **smaller packets** that are **parts of a larger packet** are **put back together** at the **destination node** to form the **original packet**.

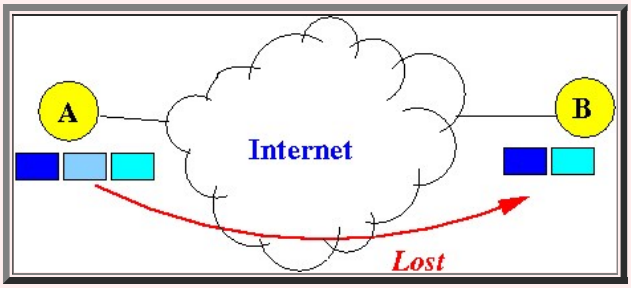
- **Datagram Delivery Service**
  - **Recall:** ***reliable*** **communication** service:
    - A **series of messages** is **delivered** ***reliably*** if:
      - **Each message** delivered ***exactly once***
      - **Each message** delivered ***in the order*** **sent**

  - The **datagram delivery service**:
    - is ***unreliable***

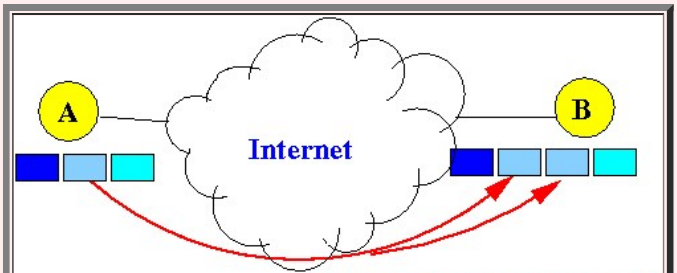
  I.e.: it can **violate** ***both*** **qualities** of ***reliable*** **communication** !!!

  - **What** can **happen** in ***unreliable*** **(datagram) delivery**:

1. A **message (IP packet)** can be **lost** (i.e., ***not*** **delivered**):

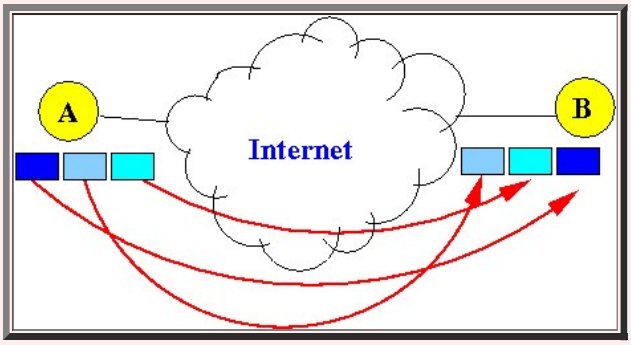


2. A **message (IP packet)** can even be **delivered** ***multiple*** **times**:

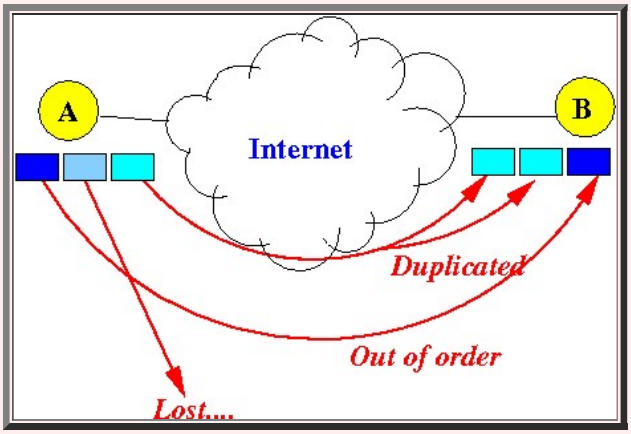


Delivered multiple times !!!

3. **Messages (IP packets)** can be delivered **out of order**:



4. **And** any ***combination*** of the above **errors**:



- In this example, the second packet is lossed, the first packet is delivered multiple times and packets are delivered out of order...

- **Observation:**

  - The **datagram delivery service** is a ***very simple*** **delivery service**
    - It is ***so*** **simple** that **most networks** (even **pigeons**) can provide this **capability**

The **requirement** is set ***deliberately*** to such a **low level**, so that:

- ***all*** **existing** (and possibly future) **networks** can connected to the **Internet**

Fragmentation and Re-assembly services

- **Maximum payload size in physical networks**
  - **Fact:**
    - ***Each* type of *physical* network has a *maximum* frame size.**
  - **Example:**
    - The **payload (= data portion) size** of an **Ethernet frame** is at most **1500 bytes**:

      Graphically:

      ```
          7 bytes   start     48 bit  48 bit  16 b     n <= 1500 B   32b
      +-----------+----------+-------+------+--------+-------------+-----+
      | Preamble  | 10101011 |  Src  | Dest | Length |    Payload  | CRC |
      +-----------+----------+-------+------+--------+-------------+-----+
      ```

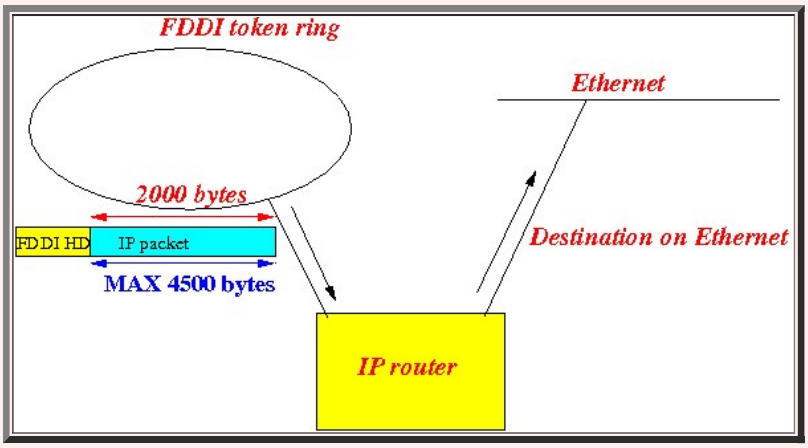
    - The **payload (= data portion) size** of the ***FDDI* token ring frame** is at most **4500 bytes**.

- **The need to break up an IP packet**
  - The ***need*** to **fragment** an **IP Packet**:
    - When an **IP packet** is **transmitted** across ***different types* of networks**
      - that has ***different* maximum sizes payloads**

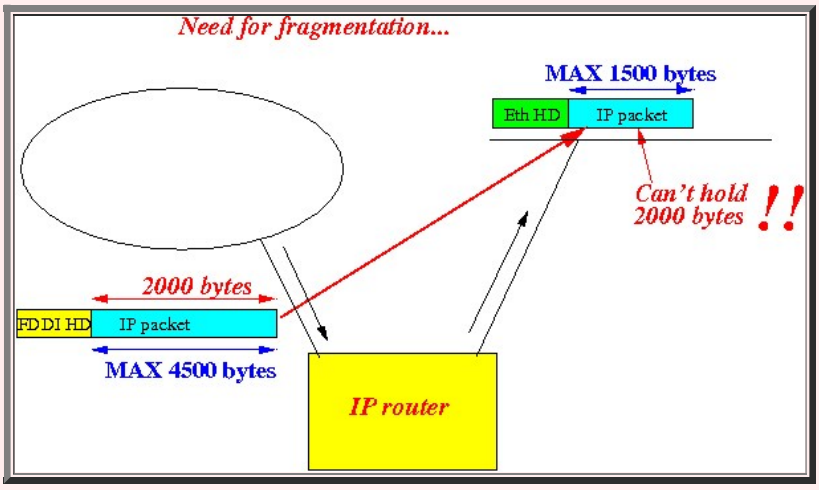
      we may **need** to ***fragment* (divide)** the **payload**

    Example:

    - A **node** on an ***FDDI* Token ring network** transmits an **IP packet** of size **2000 bytes** to a **destination** on an **Ethernet network**:

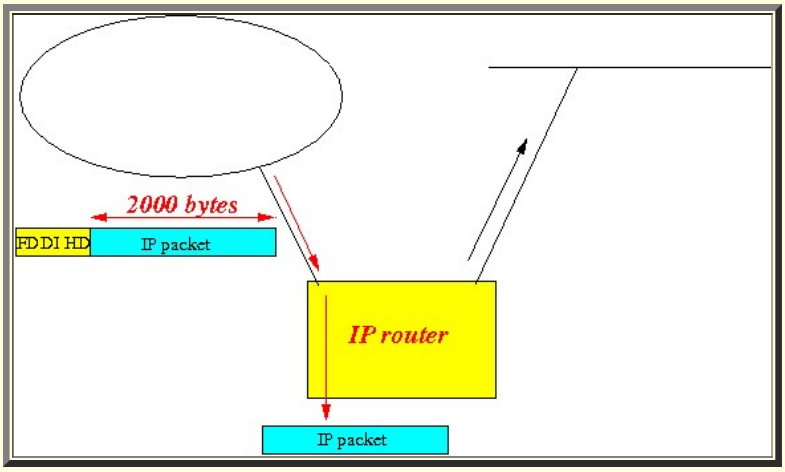
      

    - The **IP packet (payload)** is ***too big*** to be **transmitted** inside an **Ethernet frame**:

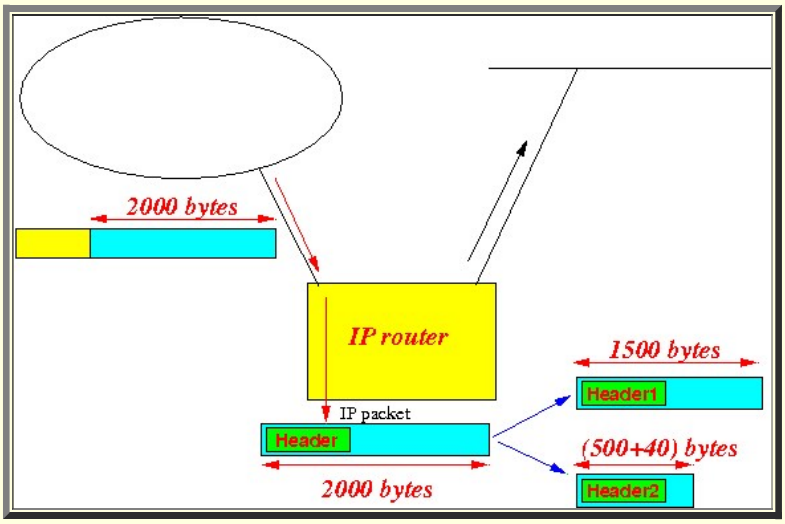
      

- **How to transmit over-sized IP packets**
  - **How** to **transmit** an **over-sized IP packet**:

    - The **IP packet** is **received** by a **router**:

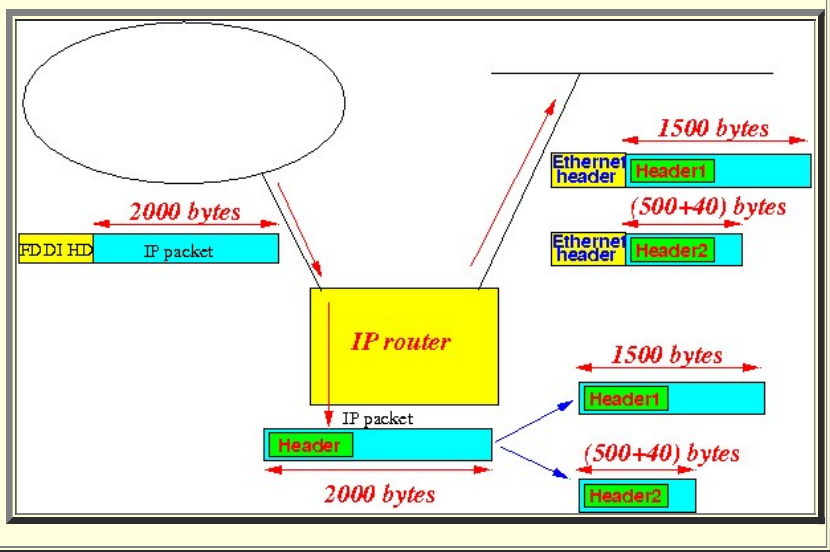


    - The **IP packet** is **divided** into ***multiple* IP packets**:



      The **IP header (= 40 bytes)** is ***duplicated*** in ***every* IP packet** (with some **modification** --- see **below**)

    - ***Each* IP packet** is is **transmitted** in a **Ethernet frame**:



- ***Where* does fragmentation and re-assembly take place**
  - **Packet *fragmentation*** happens **at**:

    - **In** a ***router*** that **forward** an **IP packet**:

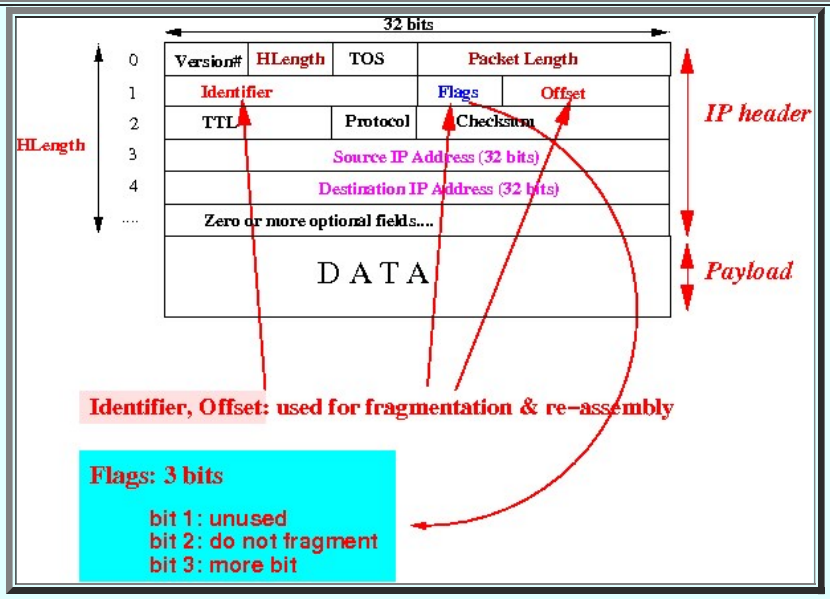
- The **destination network** uses a ***smaller*** **maximum payload size**

- **IP packet *re-assembly*** happens **at**:
  - **Packet re-assembly *only*** takes place at the **destination (computer)**

- **Note:**
  - **Every IP protocol *software* module** must **include** the ***fragmentation* *and* *re-assembly*** functions.

The IP Fragmentation and Re-assembly Algorithm

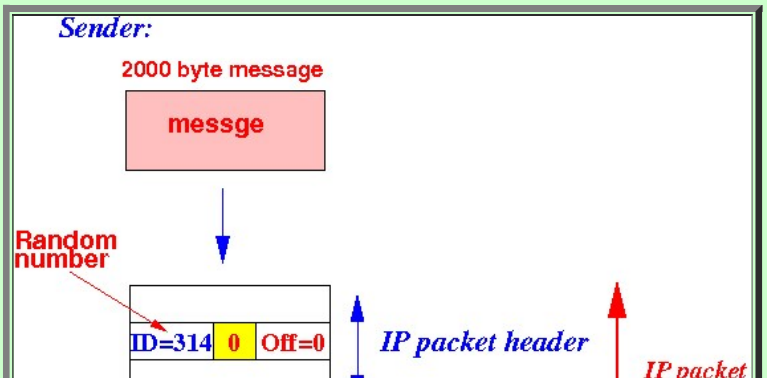
- **The IP *header* fields used in fragmentation and re-assembly**
  - ***Information* field** in the **IPv4 packet header used** to implement the **fragmentation/re-assembling procedure**:



- **Identifier**:
  - The **identifier** field contains a **random number**
    - The **random number** is **generate** by the **sender** of the **IP packet**
  - The **identifier** field is used to **identify the *fragments*** from the ***same* IP packet**:
    - ***All* fragments** from the ***same* (original) IP packet** will have the ***same* value** for the **identifier field**
- The **MORE bit**:
  - **1** = this **fragment** is ***not*** the ***last* fragement**
  - **0** = this **fragment** is the ***last* fragment** of the **IP packet**.
- **Offset**:
  - The **offset** field contains the **offset (position)** of the **IP packet fragment** within the **original (whole) message**.
    - The ***different* fragments** can be **re-assembled** using the **value** of the **offset values** in this **information field**

- **The IP *Fragmentation* procedure**
  - **Example: Fragmentation**

- A **sender** transmits a **message** consisting of **2000 bytes**:

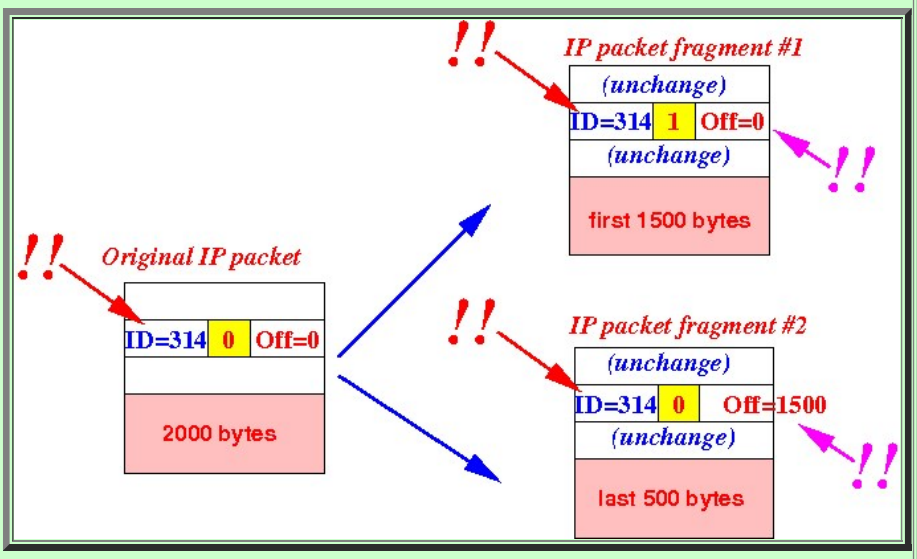


The **sender** has **generate** the **random number** ***314*** as **identifier**

- The ***original*** **IP packet** will contain:

  - **identifier** = **314**
  - **More bit** = **0** (because the **current fragment** is the **last fragement**)
  - **Offset** = **0** (because the **current fragment** is ***first*** **piece**)

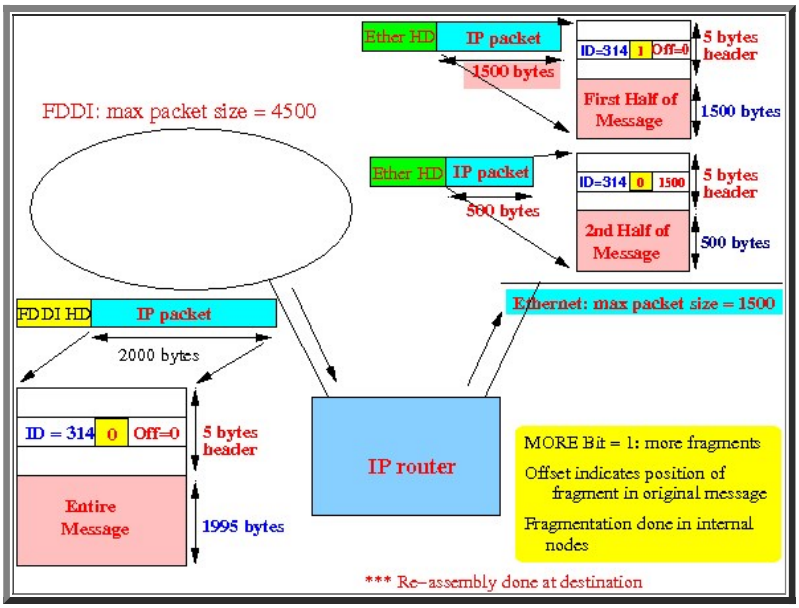
- The **IP packet** can be **fragmented** as follows into ***2 (two)*** **IP packets**:



Notice that:

- The **identifier**, **More bit** and **offset** will allow the **receiver** to:
  - ***Re-assemble*** the **different fragments** back into the ***original*** **message** !!!

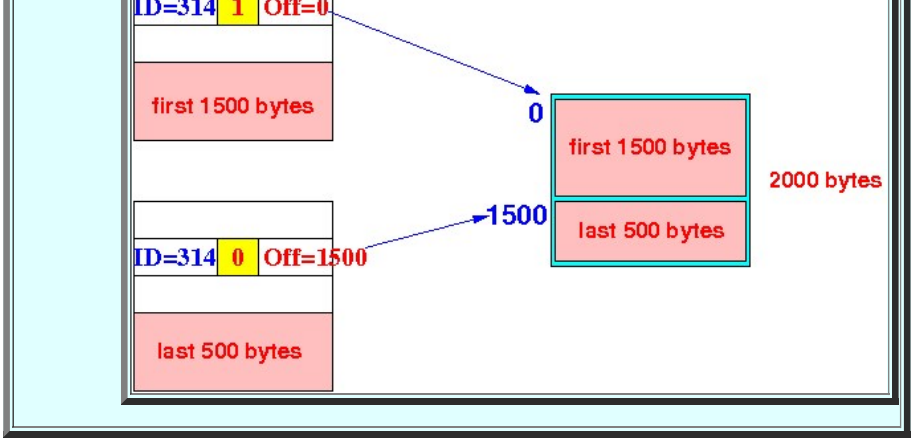
- **How** the **IP fragments** will be **transmitted** in **Ethernet frames**:



- **The IP *Re-assembly* procedure**

  - **Example:**

    - **IP fragments** are **concatenated** in an **assembly buffer** using the ***offset*** in the **IP header**:



- **Fragmentation and Reassembly in IPv6**
  - The **IPv6** ***header***:

Octet Offset		0								1								2								3							
	Bit Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Traffic Class								Flow Label																			
4	32	Payload Length																Next Header								Hop Limit							
8	64	Source Address																															
C	96																																
10	128																																
14	160																																
18	192	Destination Address																															
1C	224																																
20	256																																
24	288																																

Notice that:

- The **identifier** and ***other*** **fields** used for **fragmentation/re-assembly** are ***missing*** !!!

- **Fact:**

- The **IPv6** protocol **header** does **not** contains **fragmentation/re-assembly** information

---

- **How IPv6** supports **fragmentation/re-assembly**:

- **Fragmentation/re-assembly** is an **extension (option)** in the **IPv6 protocol**

- **Extensions** in **IPv6:**

- **Extensions** are **specified** using a **extension header**
- The **extension headers** are located **immediately** ***after*** the **IPv6 header**

  Recall:

  - The `Next Header` field in the **IPv6 header** will **point** to the ***first*** **extension header**

IP network address and IP host address

- **IP (IPv4) Address**
  - **IPv4 address:**

    - **IPv4 address** = a **32 bit binary number** that is **made up** and **used** to **identify** a **node** in the ***Internet*** **uniquely**

      - There are 2^{32} = 4,294,967,296 different **IPv4 addresses**

  - **IPv6 address:**

    - **IPv6 address** = a **128 bit binary number**

- **Specifying IPv4 addresses**
  - ***Dotted*** **decimal notation**:

    - An **IPv4 addres** is **often** written as **4 groups** of **decimal numbers**:

```
xx.xx.xx.xx

   where each xx is a decimal number
```

      Example:

```
  170.140.150.1

represents:

     170 = 10101010 (binary representation of 170)
     140 = 10001100
     150 = 10010110
       1 = 00000001

  170.140.150.1  = 10101010 10001100 10010110 00000001
```

- **Specifying IPv6 addresses**
  - **IPv6 address notation**:

    - An **IPv6 addres** is **often** written as **8 groups** of **4 hexadecimal digits**, separated by **':'**:

```
xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx

   where each x is a hexadecimal digit
```

      The **initial zero's** are **suppressed** and **2 or more 0000 groups** may be **compressed** by **::**

      Example:

```
  2001:0db8:0a0b:12f0:0000:0000:0000:0001

is writted as:

  2001:db8:a0b:12f0:0:0:0:1

and comprssed to:

  2001:db8:a0b:12f0::1
```

- **Assigning IP address to hosts (nodes) --- in theory**
  - **Assigning IP addresses** to **nodes**:

    - ***In theory***

      - You **could** assign **IP addresses** in an ***arbitrary*** **manner** (= without any **constraint**) to **nodes**

    (As long as **2 different nodes** have ***different*** **IP addresses**)

- ***Problem*:**

  - **Since:**

    - A **node** in the **Internet** **must** know the **location** of ***every*** **Internet nodes**

    The **routing table** can **potentially** have:

    - 2^{32} = 4,294,967,296 entries !!!

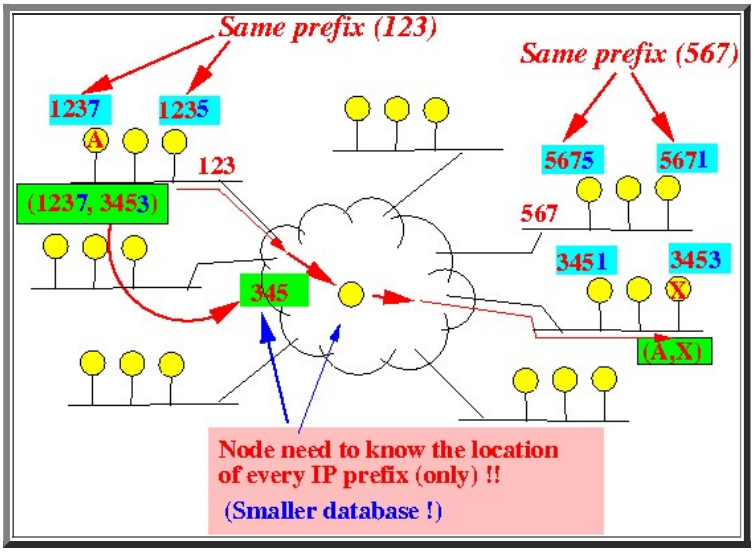
    That's a **lot** of **memory** (especially in **1980** !!!)

- **Assigning IP address to hosts (nodes) --- in practice**

  - In order to **reduce** the **size** of the **routing table**:

    - **Nodes (= computers and routers)** on the **same *physical* network** must have the **same *prefix*** for **IP address**

    Example:

  - **Experiment:**

```
host lab1a:

   lab1a.mathcs.emory.edu has address 170.140.151.110
   lab1b.mathcs.emory.edu has address 170.140.151.111
   lab5h.mathcs.emory.edu has address 170.140.151.149

   compute.mathcs.emory.edu has address 170.140.150.8
```

    Notice that:

    - ***Every*** **node** on the **MathCS department network** has the **prefix**:

```
   10101010 10001100 10010110 xxxxxxxx       (max 255 nodes)
      170      140      150
```

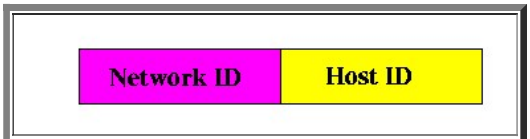
- **Structure of IP addresses**

  - **Fact:**

    - **IP addresses** are ***structured***

      (**Not** arbitrary)

  - The **format** of an **IP address**:

where:

- **Network ID** part:
  - **identifies** a **network "domain"**
- **Host ID** part:
  - **identifies *one* specific node (host)** inside a **network domain**

- **Consequences of the structure of IP addresses**
  - **Consequences** of the (imposed) **structure** of **IP addresses**:

1. **IP routing** is **based** on ***network IDs***

   Routing table contains:

```
  Network ID     |  Output port
-----------------+-------------------
  Netw Addr 1    |   port 1
  Netw Addr 2    |   port 2
   ....          |    ....
```

   (This will **reduce** the **routing table size considerably**)

2. ***All* IP hosts** on the ***same* IP network must** have:
   - the ***same* Network ID** !!!

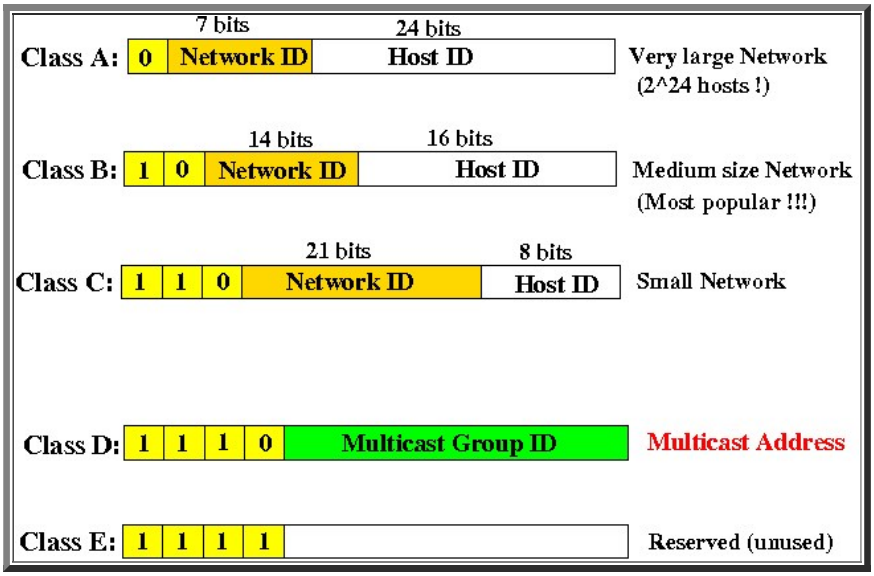
   This **consequence** will make ***mobile* IP hosts** a ***tougher* problem** than it **ought to be**....

Original (pre-1993 --- "classful") IP Address structure

- **IP-address structure ---- prior to the use of CIDR in 1993**
  - **Originally** (before the big **shortage** on **IPv4** addresses), the **IP addresses** were divided into **5 classes**:

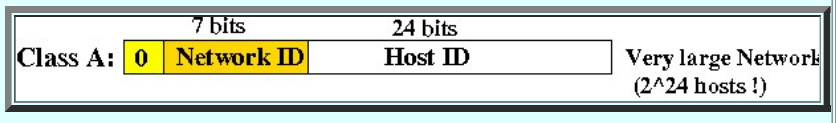
    - **class A** IP network address
    - **class B** IP network address
    - **class C** IP network address
    - **class D** IP network address
    - **class E** IP network address

  - The **IP address class** was organized by the ***prefix* of the address**:



- **Use of the classes of IP addresses**
  - **Class A IP network address**:

    - **Each class A** network can have upto **16,777,216 nodes**:



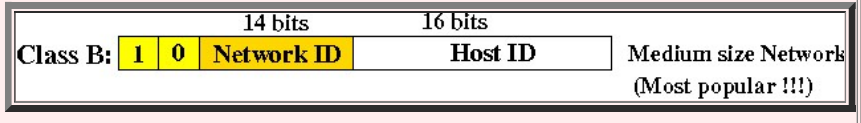
    - The **Class A networks** are used to:

      - **Connect** the ***large* number** of **nodes** that forms the **Internet "backbone"**

        Wikipedia: click here

  - **Class B IP network address**:

    - **Each class B** network can have upto **65,536 nodes**:



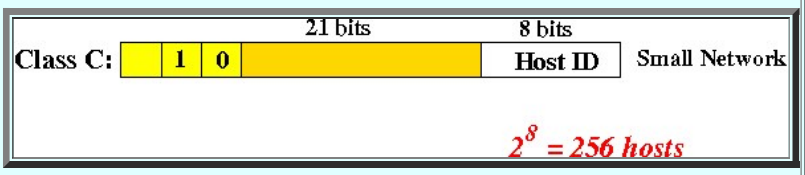
    - The **Class B networks** are used to:

      - **Connect** the ***moderate* number** of **nodes (computers)** in a **company/organization**

        (**Emory University** used to have a **class B** network ID)

  - **Class C** IP address:

- Each class C network can have upto 256 nodes:



- The Class C networks are used to:
  - Connect a computers of a *small* company/organization

- Class D addresses: **Comments:**
  - Class D address are used to identify a *communication group*

  - The communication *group* is called: a *multicast* group
  - The class D addresses are called *Multicast* addresses
  - IP Multicasting was **introduced** around 1992 by Steve Deering
    - A multicast *group* is identified by a multicast address

- Class E addresses:
  - The class E addresses were reserved for future applications

- **Historical note.....**
  - Why did we **design** IPv6:
    - US companies discovered the Internet around 1995
      - Thanks to the killer application Mosaic, *many* companies want their website to advertise

      This period was called the .COM boom - because the **websites** has the .com suffix...
    - Class C networks were *too* small for these companies....
    - Many requests were made for Class B addresses during the .COM boom.
      - The Internet growth at that time hit a crisis level because the Internet was starting to run out Class B addressess !!!
    - That's why a *new* Internet protocol with a large address was standardized:
      - There were 3 competing proposals
      - The winner was the "IPNG" (IP Next Generation) protocol
      - Since it was candidate 6, the **new version of IP protocol** is called IPv6 (version 6): click here

- **A temporary fix that lasted until 2015**
  - **Fact:**

    - The **agency** that **manages** the **IP address assignments** detects the **alarming address drain** around **1993**
    - They **knew** that a ***new* IP standard** will take a **long time** to **implement**
    - So **while** a ***new* standard** we being **developed**:
      - A ***temporary* fix** was **also** developed

      The ***temporary* fix** was ***so* good**, it ***slowed*** the **deployment** of **IPv6**...

  - The ***temporary* fix**: **re-assigning IP addresses** in **chunks** of **arbitrary size**
  - The ***real*** of **address shortage problem**:

    - **Actually:**
      - The **Internet** has ***plenty*** of **unused IP address**....

      The **Internet** was ***not* running out** of ***IP addresses*** in general.
    - More **accurately**:
      - The **Internet** was **running out** of ***class B*** addresses...

      Everyone that wanted to join the **Internet**, ***needed*** to have a ***Class B* network**
      - In fact, **Emory** had a **Class B** network back in **1993**....
  - The **cause** of the **address shortage problem**:
    - ***Wastage*** !!!

    Explanation:

    - A **class B network** has 2^{16} **= 65536 IP-addresses**
    - **Companies** usually has about **1000 computers**
    - **Therefore:**
      - A **company** receives **65536 IP-addresses** when it is **assigned** a **class B** network address
      - But the **company** will ***only* use 1000** of these **IP-addresses** !!!

        Percentage *wasted*:

        ```
          64536
        ------- = 98.5% !!!
          65536
        ```

      So a **lot** of **IP addresses** were ***not* usable** due to the **IP address *structure*** !!!
  - A ***temporary* fix:**
    - **CIDR - classless Inter-Domain Routing**

    The **temporary fix** (**CDIR**) was **so successful** that in **2016**, we are ***still*** running **IPv4** in ***most* of the US** !!!
    - **Emory** still run **IPv4**...

CIDR: "Classless" Inter-Domain Routing

- **CIDR: Classless Inter-Domain Routing...**
  - **Class*less***:
    - In **CIDR**, there are ***no*** **network *classes***:
      - **No class A**
      - **Nor class B**
      - **Etc**

  - **Specifying** a **network domain** in **CIDR**:
    - A **CIDR IP domain** is **specified** by:
      1. a **32 bit IP address**
      2. a **IP network prefix** (= **number of bits** that constitute the **network ID**)

      Example: Emory's CIDR IP domain is:

```
170.140.0.0/16
            ^^ IP network prefix

     The number 16 indicates
     the first 16 bits is the network ID
```

- **How the CIDR methods solved the address shortage problem**
  - **Fact:**
    - The **CIDR** Internet domain **specification** allow you to **assign** the **"rightsize"** to an **IP network**

    Example:

```
  CIDR Address              IP address range covered by CIDR address
==================        ========================================
 255.255.255.0/24         from  11111111 11111111 11111111 00000000
                          to    11111111 11111111 11111111 11111111

        ==>     from 255.255.255.0 to 255.255.255.255
                              (= 2^8 = 256 addresses)

 255.255.255.0/23         from  11111111 11111111 1111111 0 00000000
                          to    11111111 11111111 1111111 1 11111111

        ==>     from 255.255.254.0 to 255.255.255.255
                              (= 2^9 = 512 addresses)
```

  - **History:**
    - When **CIDR** was **adopted**:
      - a large number of **companies and universities** (including **Emory**) who owned a **class B addresses** needed to **return their IP domain**
      - Some were given ***smaller*** **IP domains**

    Result:
      - A **lot** of **IP addresses** became **available** for use !!!

  - **Disadvantage of CIDR:**
    - The **number** of **IP domains** also ***increased***
      - **Router** needed to have ***more*** **memory**

    (It was **OK**, by then, the **computer memory** got **much larger** and **memory prices** has has **dropped** significantly...)

IPv6 address structure

- **IPv6 address formats**
  - **IPv6 address** are classified as **follows:**

    - **Unicast** addresses
    - **Anycast** addresses
    - **Multicast** addresses

- **Transmission methods**
  - **Unicasting**:

    - **Unicast transmission** = **sending** of **messages** to a ***single*** **network destination** (that is **identified** by a ***unicast*** **address**)

    (This is the ***normal*** **way** to **transmit** a **message**)

  - **Anycasting**:

    - **Anycast transmission** = **sending** of **(datagram) messages** to ***any*** **one member** of a **group** of **network destinations** (that is **identified** by an ***anycast*** **address**)

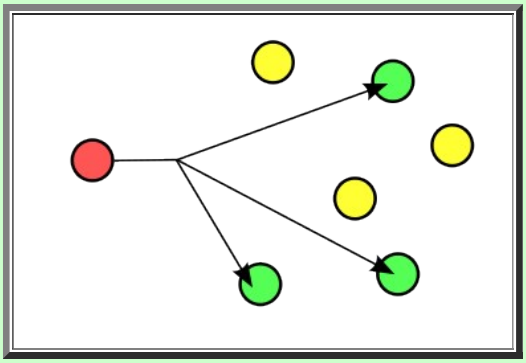
    Example:

    - **Google/Yahoo/Facebook** uses ***anycasting*** to **direct** your **requests** to:

      - The ***nearest*** **server** **or**
      - Their ***lightly*** **loaded** server...

  - **Multicasting**:

    - **Multicast transmission** = **sending** of **(datagram) messages** to ***all*** **members** of a **group** of **network destinations** (that is **identified** by a ***multicast*** **address**)

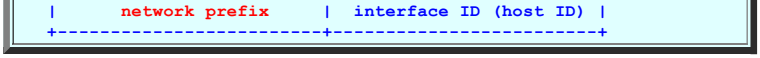
    (This is the **group communication** method developed by **Steve Deering**)

- **Multicast address format**
  - **IPv6 multicast address format**:

```
    8 bits   4 bits  4 bits           112 bits
+----------+-------+-------+------------------------------+
| 11111111 | flags | scope | <---- multicast group ID ----> |
+----------+-------+-------+------------------------------+
```

- **Unicast and anycast IPv6 addresses**
  - **IPv6 unicast/anycast address format**:

```
  <------- 64 bits -------> <------- 64 bits ------->
+-------------------------+-------------------------+
```



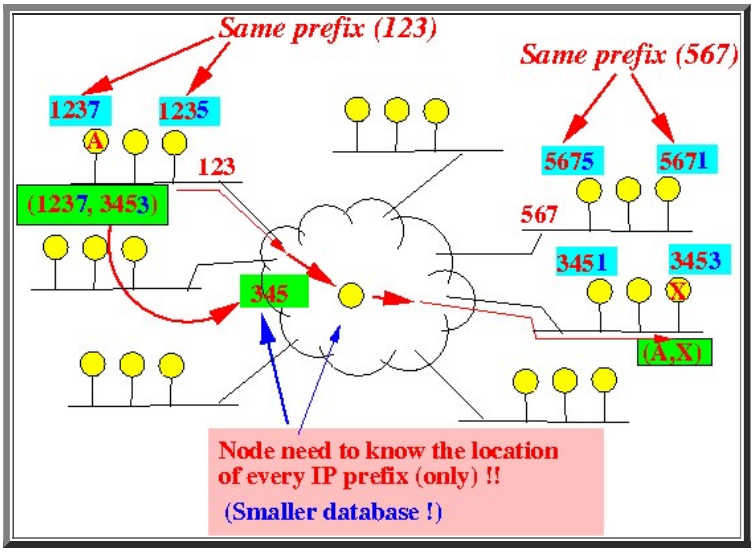
- **Further reading**
  - **Wikipedia: click here**

Multi-homed nodes

- **IP address assignment**
  - **Recall:**

    - The **network prefix** of an **IP address** for ***all*** **nodes** in the ***same*** **IP domain** are ***equal*** **(same prefix !!)**

    Example:

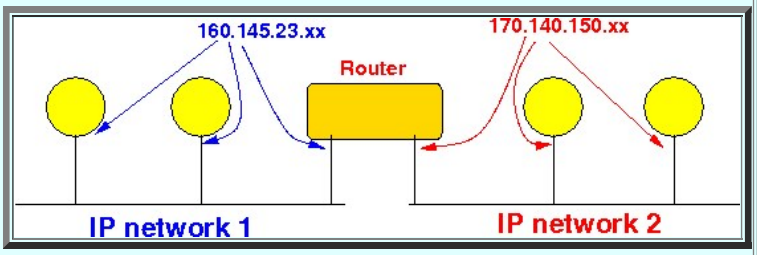


- **IP hosts attached on *multiple* IP networks**
  - **Fact:**

    - ***Some*** **IP hosts (routers !!!)** are located on ***2 or more*** **IP-networks**.

  - **In fact:**

    - A **routers** is ***always*** **attached** to **2 or more IP-networks**:



  - **Multi-homed IP node:**

    - **Multi-homed node** = A **node** that is **attached** to ***multiple*** **IP networks**

  - **Fact:**

    - A ***multi-homed*** **host/node** will have:
      - ***multiple*** **IP addresses** (**one** IP address per **IP network**)

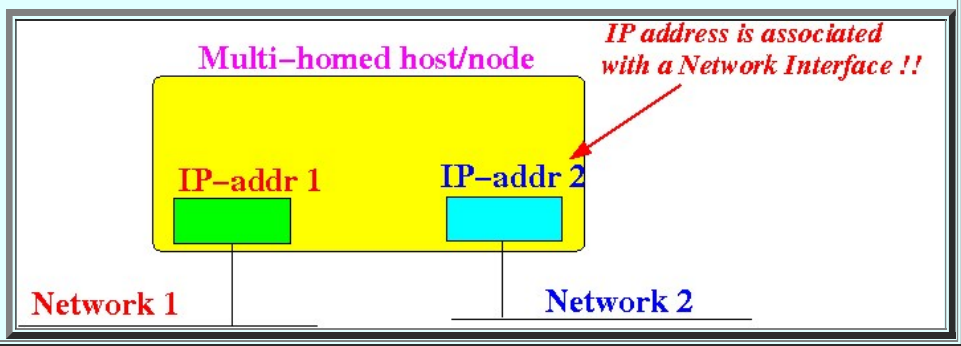
    (Because the **network prefix** of the **networks** are **different**)

  - **Note:**

    - It's **best** to think that **IP addresses** are ***assigned*** to:
      - The **network *interfaces*** of a **node**

    (instead of being assigned to the **node itself**)

Graphically:



- **Unix command to find out the IP addresses of a host**
  - Try this:

```
host www.yahoo.com

    fd-fp3.wg1.b.yahoo.com has address 98.139.183.24
    fd-fp3.wg1.b.yahoo.com has address 98.139.180.149
```

Information used to forward/route IP packets

- *Information* used to forward an IP packets (to its destination)
  - Information maintained by a router:
    - The network IDs of networks that the router is *attached* to
      - This information is used to transmit IP packets *directly* to IP destinations on the *same* IP network
    - The network IDs of networks that the router is *not* attached to
      - This information is used to transmit IP packets to *remote* IP destinations on *other* IP networks
  - Information maintained by a node to forward (transmit) IP packets:
    - A (layer 2) forwarding table
      - Structure of the layer 2 forwarding table:

```
   destination network ID     |  Output port
------------------------------+----------------
     IP netw ID 1             |  port 1
     IP netw ID 2             |  port 2
       ....                   |   ...      (and so on)
```

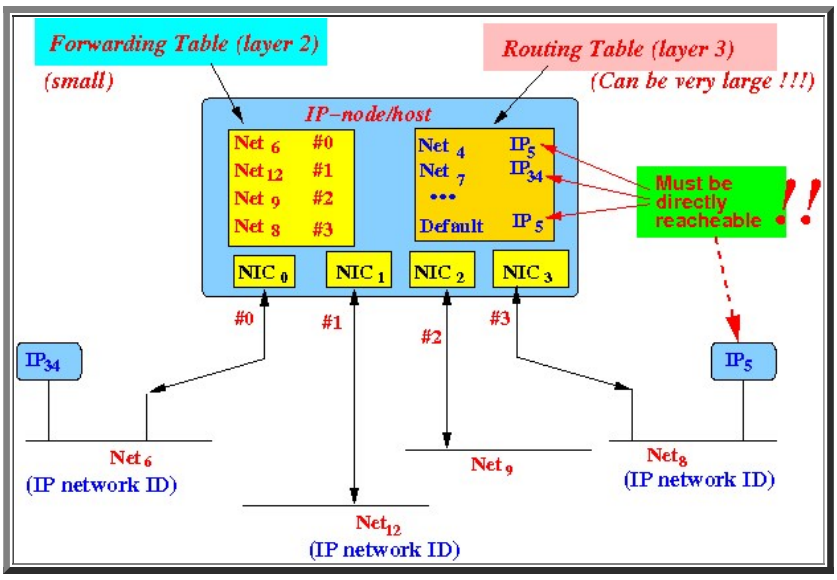
      - This table is used to check if a destination is on the *same* network as the IP node

        (I.e., destination is immediately reachable)
        - If destination is on the *same* network, the node will send the IP packet to the destination *directly*
    - A (layer 3) routing table
      - Structure of the layer 3 forwarding table:

```
   destination network ID     |  Next IP node
------------------------------+----------------
     IP netw ID 1             |  IP address 1
     IP netw ID 2             |  IP address 2
       ....                   |   ...      (and so on)
```

      - This table is used to *pass* the packet to the *next* router if a destination is *not* directly reachable.

Example:

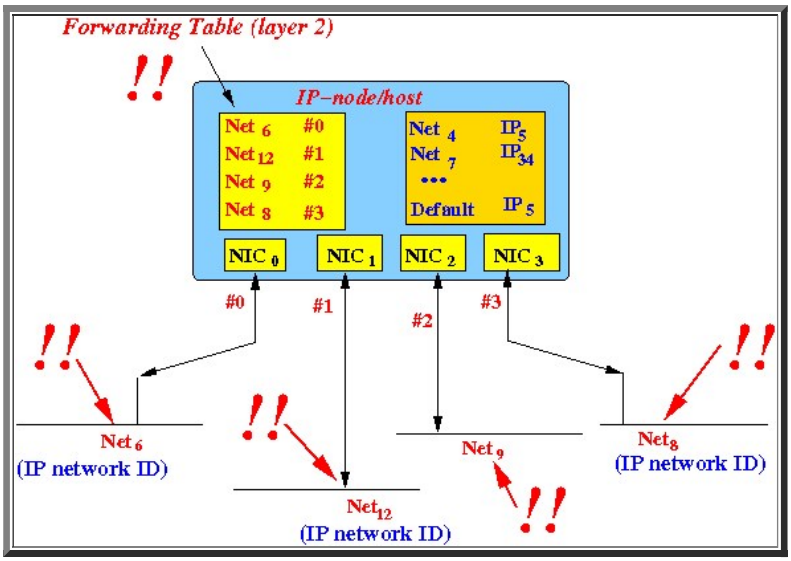


Situation:

- The node is connected to 4 different IP networks:

- Net_6 (represents some **IP network address**)
- Net_{12}
- Net_9
- Net_8

- **Using the forwarding table**
  - The ***(level 2)* forwarding table** contains ***IP* network IDs** to which the **node (host or router)** is ***attached to***:



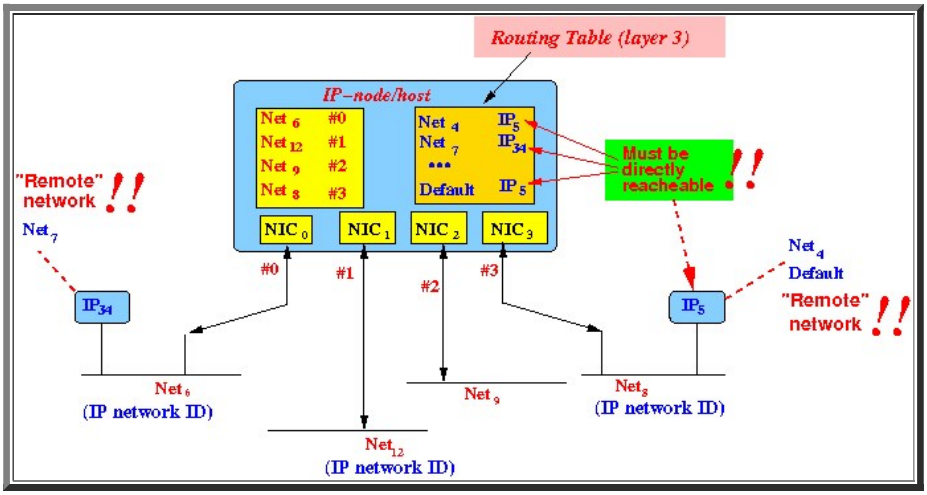
Entry in the **forwarding table** has the **form**:

```
(Attached) IP Network ID        Port#
```

The **forwarding table** is used to

- **Forward** the **IP-packets** destinated for **IP destination** on the ***same* IP network** as the **node *itself***

- **Using the routing table**
  - A **(level 3) routing table** containing ***IP* network IDs** of **"*remote*" networks**:



Entry in the **routing table** has the **following form**:

```
(Remote) IP network ID        next-Router IP-addr
```

Usage:

- The `next-Router IP addr` is the **IP address** of a **router** that is **immediately *reacheable***

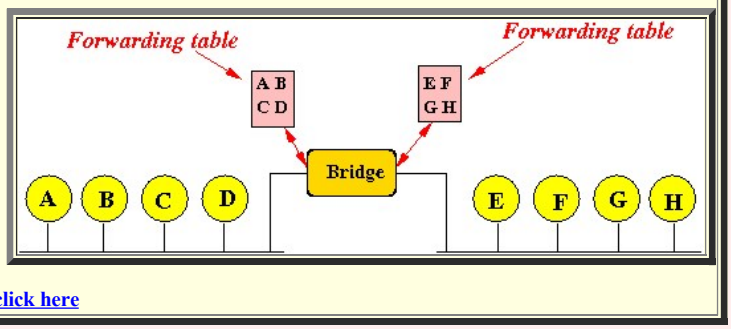
- **IP packets** destined for a **destination** of that **(remote) IP network** will be ***forwarded*** to the `next-Router IP addr` **node** for ***further* processing**

- **Nomenclature.....**

- **Note:**

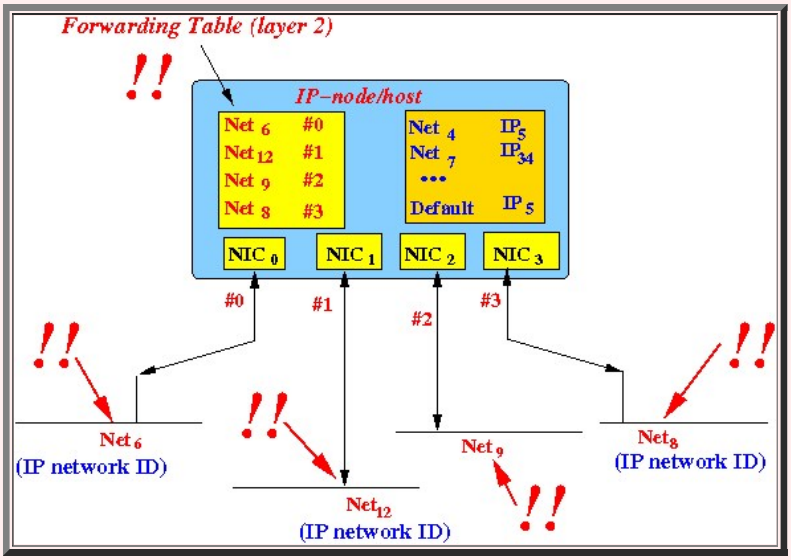
  - The **name** of the ***level 2* forwarding table** is some what **unfortunate** because:

    - We have **seen another** kind of ***forwarding* table** inside a **bridge** previously:

      See: click here

    - The ***level 2* forwarding table** inside a **router** contains ***different* information** than the **forwarding table** in a **bridge**:

      Namely:

      - The **forwarding table** in a **bridge** contains ***physical* addresses (Ethernet)**
      - The **forwarding table** in a **router** contains ***IP (logical)* network IDs (IP domains)**

- ***Default* route**

  - **Default route:**

    - The **(level 3) routing table *may* (often)** contain a **special *default* route entry**:

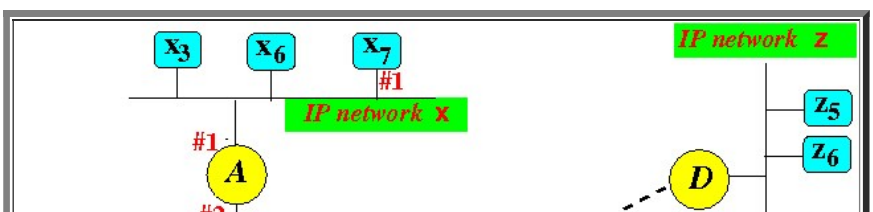
      ```
      destination network ID     |  Next IP node
      ---------------------------+----------------
          IP netw ID 1           |  IP address 1
          IP netw ID 2           |  IP address 2
            ....                 |    ...      (and so on)
          Default                |  IP address X
      ```

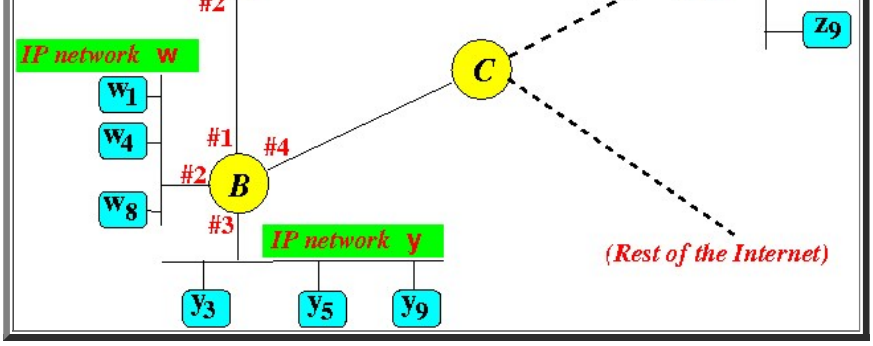
    - The **default route entry** will be **used** for **routing** an **IP packet** if:

      - its **destination network ID** is ***not* found** in the **routing table**.

- **Example of a network configuration and the corresponding forwarding/routing information**

  - Consider the following **IP network** that consists of **4 IP domains**:



The **IP networks (domains)** are:

- **IP network w**
- **IP network x**
- **IP network y**
- **IP network z**

(w, x, y and z are **IP network *prefixes***)

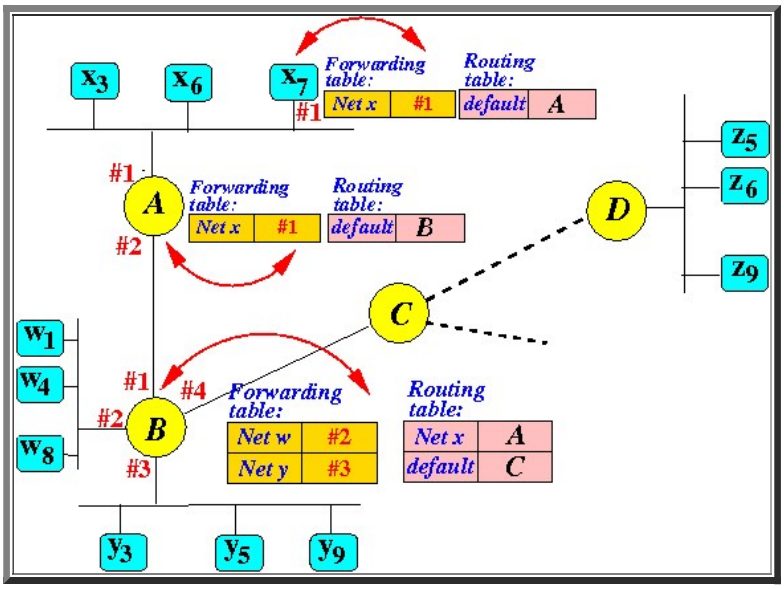
Remember that:

- **Nodes** on the ***same* IP network (= IP domain)** has the ***same* network prefix** !!!

Example:

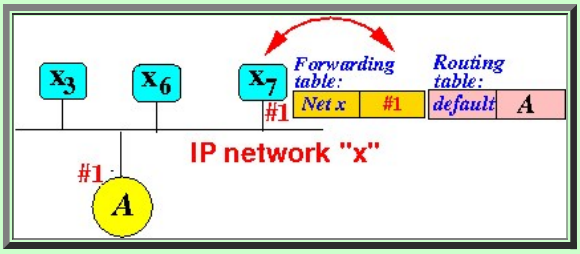
- The **nodes** x_3, x_6 and x_7 have the ***same* IP network prefix x**

- **Content** of the **forwarding table** and the **routing table** of **some** of the **nodes:**



Explanation:

- The **tables** at **node** x_7:



Comments:

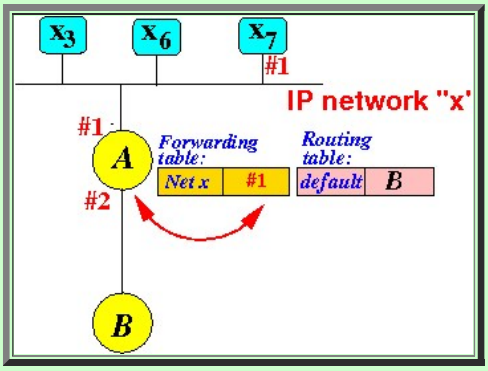
- x_7 is on **IP network x**
  - Hence the **forwarding table** of x_7 contains **IP network x**

- There is **only *one* way out** of **network x**:
  - Through the **router A**

So x_7's **routing table** only contains the ***default* router entry**

(Most **end destinations hosts (computers)** that are connected to the **Internet** by **one single router** would have this **configuration**)

- The **tables** at **router A**:



Comments:

- **Router A** is the ***only* entrance ("gateway")** to the **network x**

  Because the **router A** is **attached** to the **network x**:

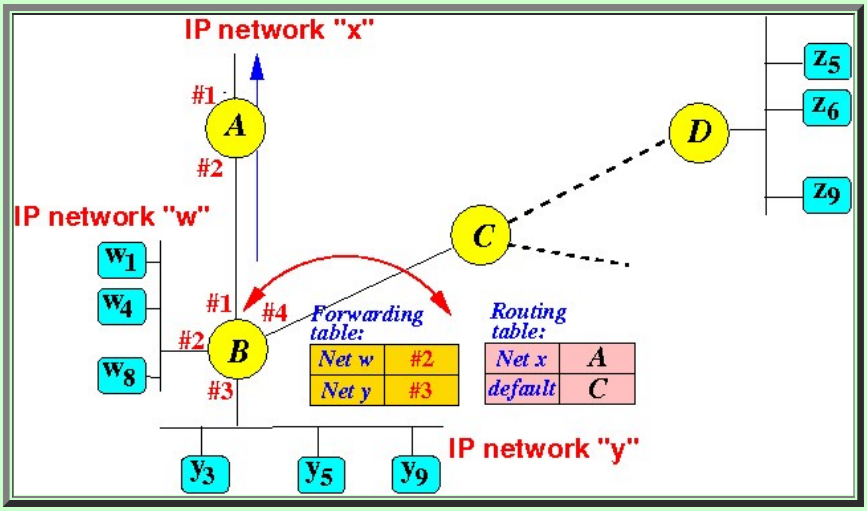
  - **Router A's forwarding table** contains the **entry** for **network x**

- Because there is ***only* one route** through the **router *B*** to ***all* other networks**:

  - **A's routing table *only*** contains the ***default* router entry** (to **router *B***)

  (Most **routers** that connects ***one* network** to the **Internet** would have this **configuration**)

- The **tables** at **router B**:



Comments:

- **Router *B*** is **attached** to **2 IP networks (w and y)**:

  - **Router *B*'s forwarding table** contains **rntries** for **networks w and y**

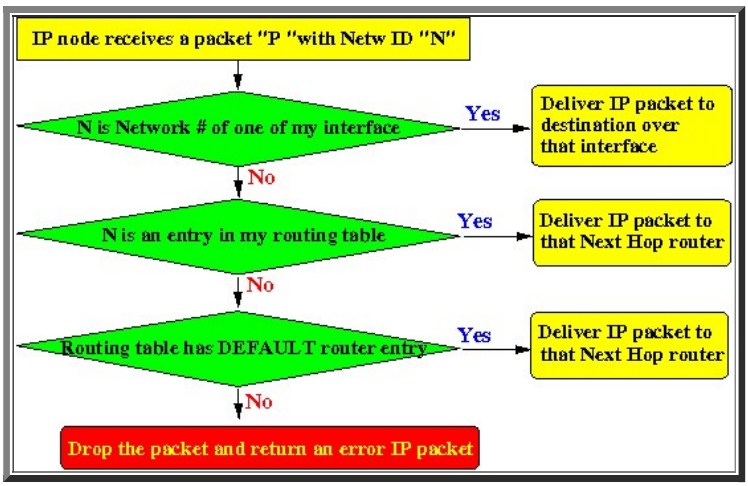
- **Router *B*** is **connected** to **2 different routers**

  Router *B*'s routing table indicates that:

  - It uses **router *A*** to reach **network x**
  - It uses **router *C*** to reach all ***other* networks**

The IP-Forwarding Algorithm

- **The IP Forwarding Algorithm:**
  - **Flow chart** of the **IP forwarding algorithm:**



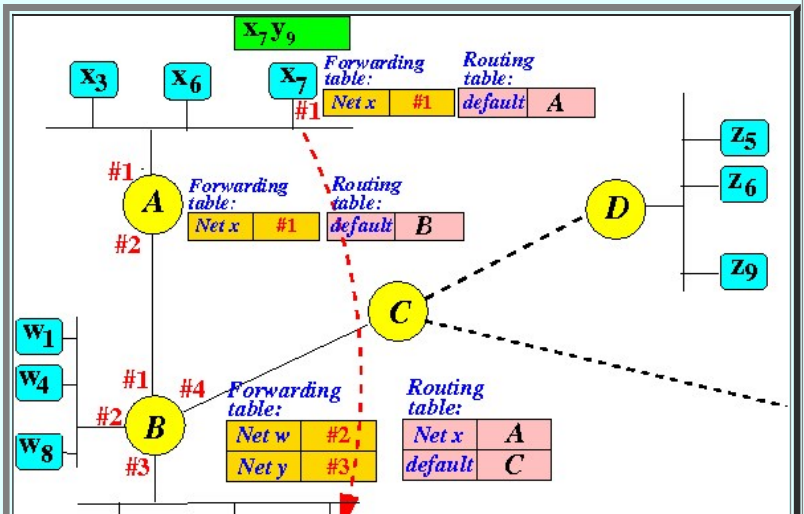
  - **Explanation:**

1. **Node** (computer or router) receives an **IP packet** (inside an **Ethernet frame** !!!)
   - It will first **extract** the **IP packet** and
   - **Examine** the **destination IP address** (in the **IP protocol header**)
2. First, it check whether the **destination IP address** is one of the **IP addresses** in the **IP forwarding table**
   - If so, the **IP packet** is sent of the **corresponding network interface**

     (Because the **IP destination** is on **one** of the **IP networks** attached to the **node**)
3. Otherwise, check whether the **destination IP address** is one of the **IP addresses** in the **routing table**
   - If so, the **IP packet** is sent of the **corresponding router**
4. Otherwise, check if the **routing table** has a ***default*** **entry**
   - If so, the **IP packet** is sent of the **corresponding *default* router**
5. Final option: **drop the packet**.... (**cannot** find a **route** to the **destination**)

- **IP forwarding example 1**
  - **Example 1:**

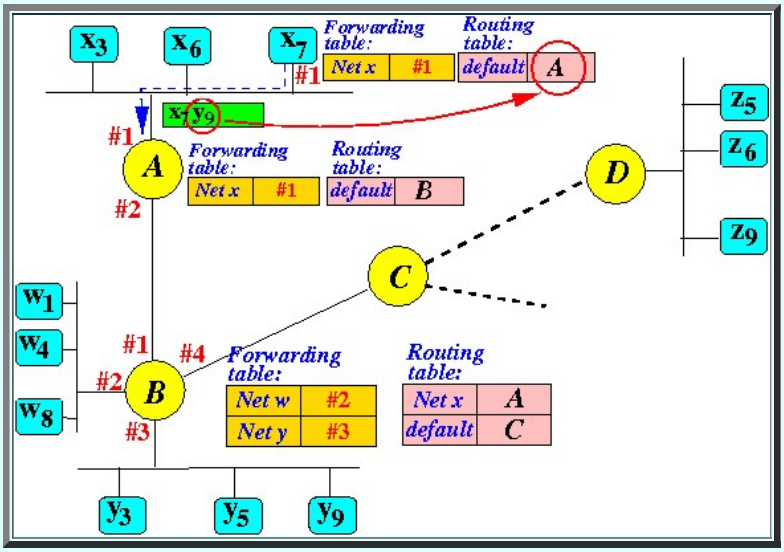
- **IP host/node** x_7 sends a message to **IP destination host/node** y_9:



- x_7 runs the **IP forwarding algorithm**:

1. Destination y_9's **network ID** is **y**
2. **Look up y** in the **forwarding table**
   - **Network ID y** is **not** in the **forwarding table**....
     So it will check the **routing table**
3. **Look up y** in the **routing table**
   - **Network ID y** is **not** in the **routing table**
     So it try to use the ***default* route**
4. Host x_7 will use the **default route**: **(default, *A*)** for routing

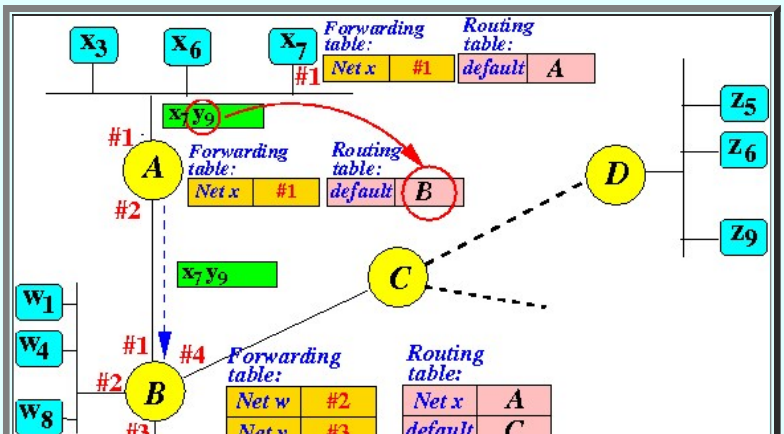
Result: x_7 sends the message to ***A***



- **Router *A*** runs the **IP forwarding algorithm**:

1. Destination y_9's **network ID** is **y**
2. **Look up y** in the **forwarding table**
   - **Network ID y** is **not** in the **forwarding table**....
     So it will check the **routing table**
3. **Look up y** in the **routing table**
   - **Network ID y** is **not** in the **routing table**
     So it try to use the ***default* route**
4. **Router *A*** will use the **default route**: **(default, *B*)** for routing

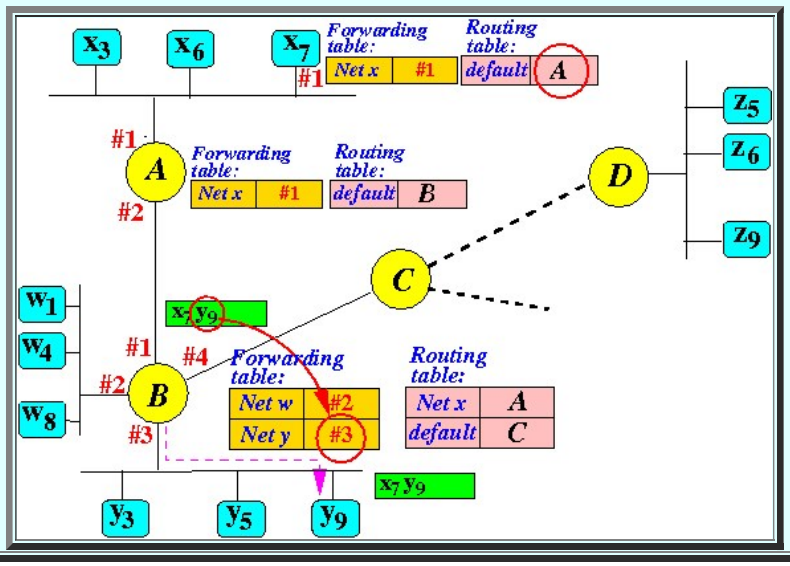
Result: **router *A*** sends the message to **router *B***



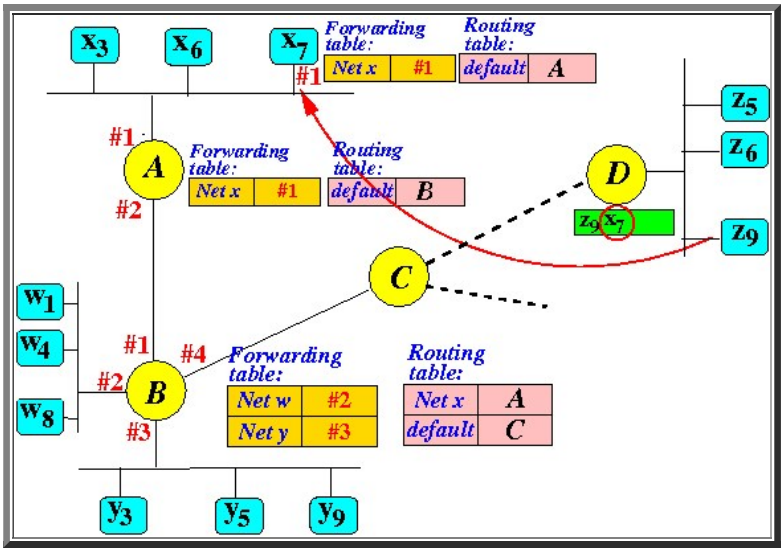
- **Router *B*** runs the **IP forwarding algorithm**:

1. Destination y_9's **network ID** is **y**
2. **Look up y** in the **forwarding table**
   - **Network ID y** is ***found*** in the **forwarding table** !!!!

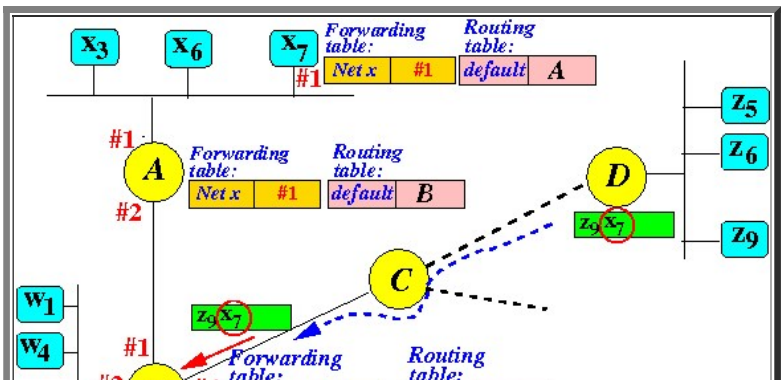
Result: **router *B*** sends the message ***directly*** on the **network interface *#3***:

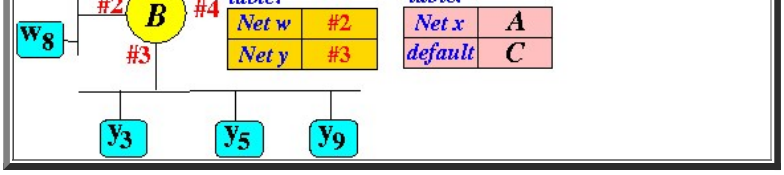


- **IP forwarding example 2**
  - **Example 2:** node z_9 sends a message to node x_7:

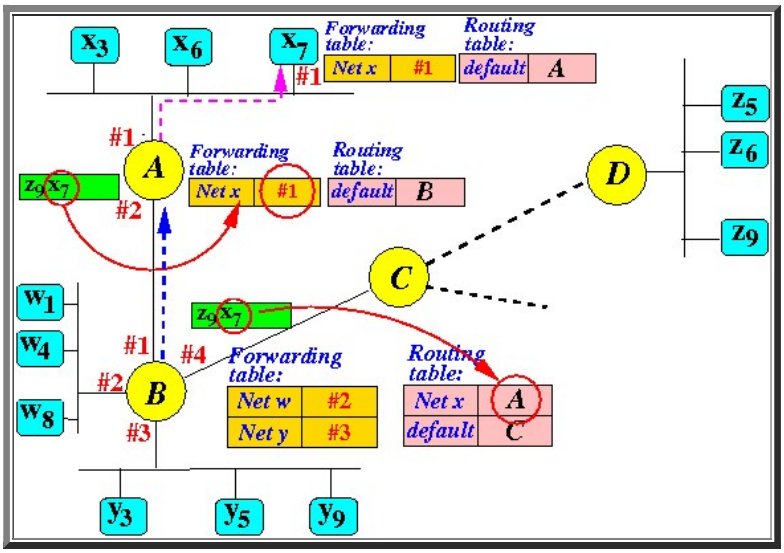


  - We will **start** looking at this **example** when the **IP packet** has **arrived** at **router *B***:





- The following **diagram summarizes** the **routing decisions**:



(Shorten) explanation:

- **IP forwarding algorithm** run at router **B**

  1. Destination x_7's **network ID** is **x**

     x is not in the **forwarding table** so it uses the **routing table**

  2. Router **B** will use the entry: **(Net x, *A*)** for routing

  Result: **B** sends the message to ***A***

- **IP forwarding algorithm** run at router **A**:

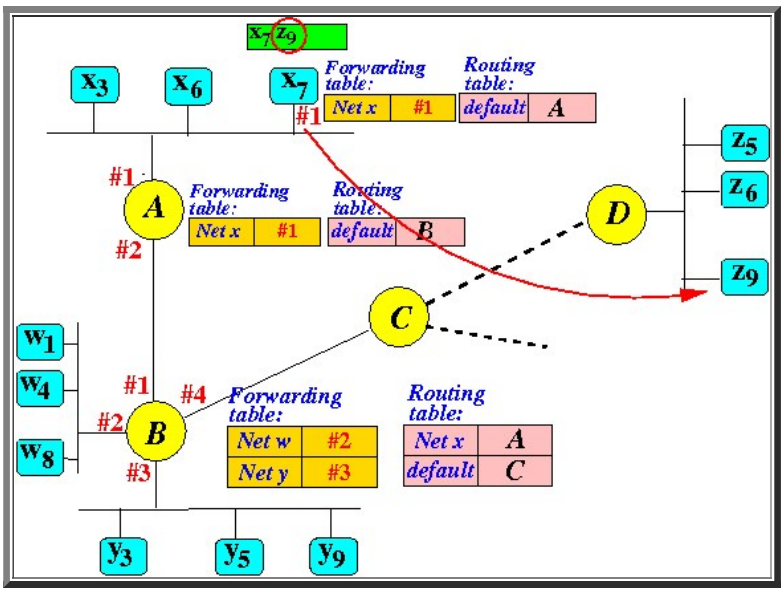
  1. Destination x_9's **network ID** is **x**

     x is **found** in the **forwarding table** !!!

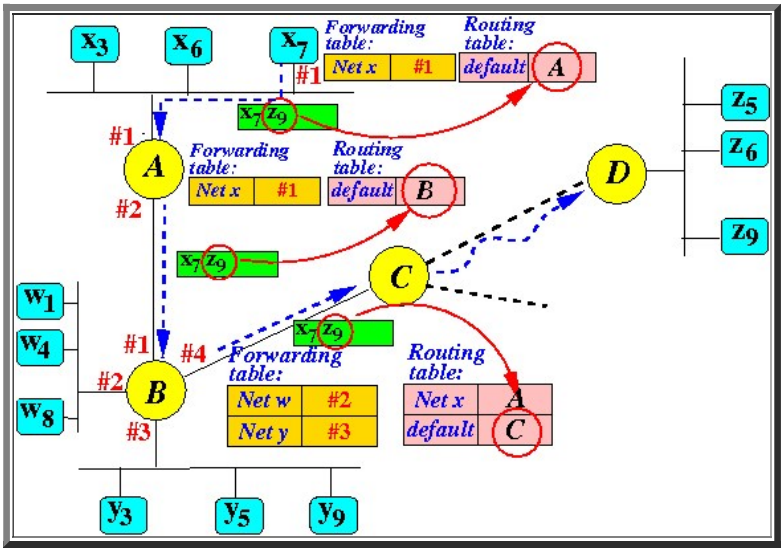
  Result: router **A** sends the message to the **network** attached to its ***port #1***

- **IP forwarding example 3**

  - **Forwarding example 3:** x_7 sends a message to z_9



- The following **diagram summarizes** the **routing decisions**:



Notes:

- **IP forwarding algorithm** run at x_7

  1. Destination z_9's **network ID** is **z**

     z is not in the **forwarding table** so it uses the **routing table**
  2. Host x_7 will use the entry: **(default, *A*)** for routing

  Result: x_7 sends the message to ***A***

- **IP forwarding algorithm** run at router **A**:

  1. Destination z_9's **network ID** is **z**

     z is not in the **forwarding table**, so it uses the **routing table**
  2. Router **A** will use the entry: **(default, *B*)** for routing

  Result: router **A** sends the message to router ***B***

- **IP forwarding algorithm** run at router **B**:

  1. Destination z_9's **network ID** is **z**

     z is not in the **forwarding table**, so it uses the **routing table**
  2. Router **B** will use the entry: **(default, *C*)** for routing

  Result: router **B** sends the message to **C**

The message will eventuall get to **D** and then to the network **z**....

- **Summary:**
  - **Key points:**
    - A **node** (computer or router) can **always** tell if the **destination of an IP packet** is located on its **own network** by the **network IDs** in its ***Forwarding Table***
      - If the **destination** is on one of its **own networks**, it will send the **IP-packet** ***directly* to the destination** in a **physical (Ethernet or Token bus) frame**.

    - If the **destination** is ***not* on its own network**, the node will send the **IP-packet** to **a neighboring router** (a router that is connected to node)
      - This **neighboring router** is always given in the **routing table**.

  - **Note**:
    - Some **text book** considers the **IP *forwarding* table** as as ***special* section** of the **IP *routing* table**

Some loose ends in IP-forwarding

- **A few loose ends...**
  - **Things** that we **still** need to **discuss**:

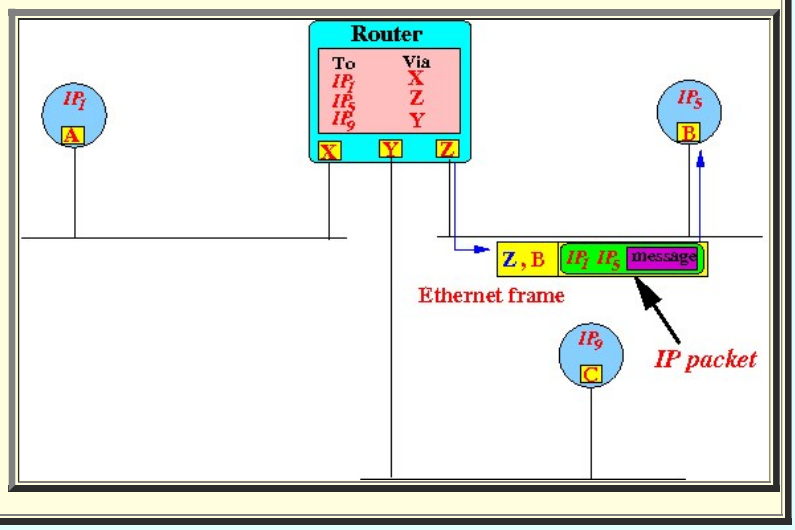
1. **Constructing** the **Routing Table**

   - In the examples, the **content** of the **routing table** (and the **forwarding table**) have been **setup** ***already***)
   - We will discuss **how** the **routing tables** are setup - this will be discussed in a **later lecture** (routing): **click here**

---

2. **Discovering** the **physical (e.g., Ethernet) address** of a **host**

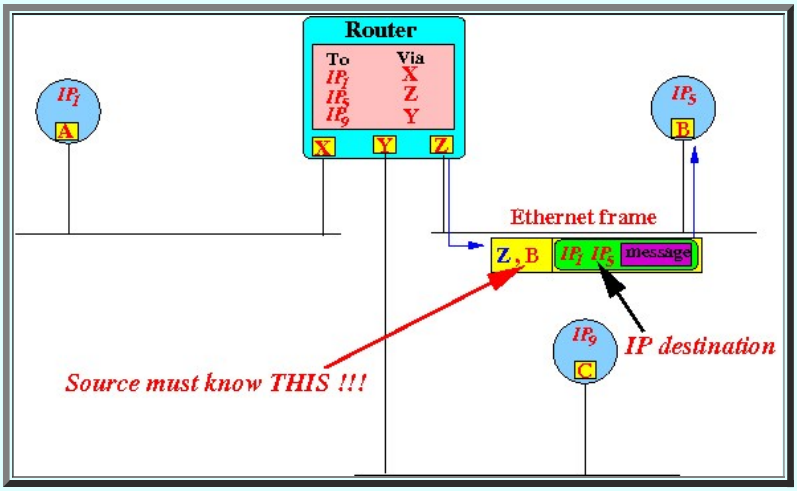
   - **IP packets** are transmitted **inside** a **physical network frame** (e.g., an **Ethernet frame**):



   In order to transmit the **IP packet** to a **node**:

   - A **source** must **know** the ***physical*** **(Ethernet) address** for a given **IP address** !!!

   Graphically:



   Problem:

   - How can the **router B** find the **Ethernet address** of a host that has the **IP address** IP_5 ????

   This problem will be solved by the **Address Resolution Protocol (ARP)** - discussed later.

---

3. **Assign** an **IP address** and **configure** the **IP forwarding/routing table** of a **node** ***automatically*** (when a **node** starts up)

   - **Manual** configuration:
     - Give a **computer** a **unique IP address** (must have the **correct network prefix**)
     - Set up the **IP forwarding table**
     - Set up the **IP routing table**

   But that is **very time consuming** when you have to setup **hundred of computers**

We will discuss an ***automated*** **way** to **assign IP addresses** and **configure** the **IP forwarding/routing table**

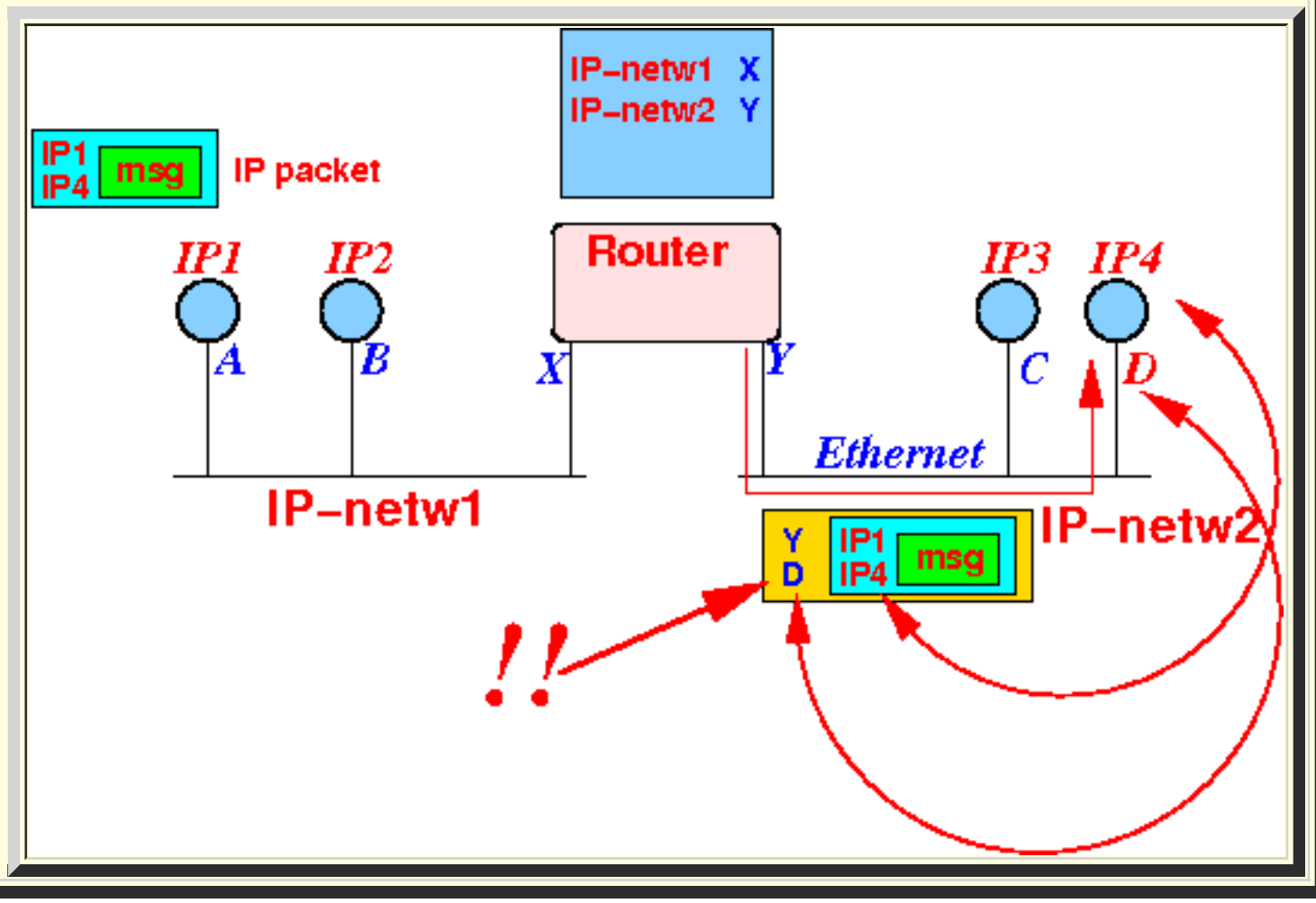
- The solution is the **Dynamic Host Configuration Protocol (DHCP)**...

The *Address Resolution Protocol* (ARP)

- **Review: IP packet forwarding**
  - **Recall** that:

    - An **IP packet must** be **transmitted** in a **frame** that contain the ***physical*** **address** of the **IP node**

      Graphically:



Question:

- **How** can a **node** find the ***physical*** **address** for the **(logical) IP address** ???
  - The **router** does **not** store the **physical address** of the **IP nodes** in its **forwarding/routing tables** !!!

- **Address Resolution**
  - **"Address Resolution"** :

    - **Address resolution** = **find** the ***physical (real)*** **address** that **corresponds to** a ***logical (IP)*** **address**

- **Data structure used to resolve *IP addresses* to *Physical addresses***
  - The **mapping information** for a **logical (IP) address** to **physical address** is stored in:

    - The **Address Resolution Protocol Cache (ARP cache)**

    Example: (**ARP cache**)

```
    Logical (IP) address   |  Physical (Ethernet) address
-------------------------+-------------------------------
  170.140.150.186        |  00:13:72:a3:17:e7
  170.140.150.128        |  00:03:ba:2f:d0:21
    ....                 |    .....
```

  - **UNIX command** to **display** the **content** of the **ARP cache**:

```
arp  -n
```

    Example:

```
cheung@aruba> arp -n

(IP) Address              HWtype  HWaddress           Flags Mask     Iface
=============================================================================
170.140.151.110           ether   3c:d9:2b:76:d9:16   C              eth0
170.140.151.7             ether   00:14:4f:1f:cc:ca   C              eth0
170.140.150.253           ether   00:16:36:e0:7e:a4   C              eth0
170.140.150.8             ether   00:03:ba:08:d1:a4   C              eth0
170.140.151.20            ether   00:1e:68:2f:71:ae   C              eth0
170.140.150.9             ether   02:08:20:6b:a8:9e   C              eth0
170.140.150.2             ether   00:03:ba:13:8f:e1   C              eth0
170.140.150.1             ether   00:03:ba:72:e3:4d   C              eth0
170.140.150.60            ether   00:14:4f:a8:31:08   C              eth0
170.140.150.254           ether   00:00:0c:9f:f0:00   C              eth0
```

  - The **address** in **red** is an **Ethernet Address**

- **Maintaining the ARP cache**
  - **Fact:**

    - The **ARP cache** is **filled/maintained** by using:

      - The **Address Resolution Protocol (ARP)**

- **Important fact:**

  - The **ARP protocol requires** the **use** of:

    - The ***broadcast* search** capability of the to **physical network**

- **ARP: finding the Physical (Ethernet) address for an IP address**

  - **Purpose** of the **ARP protocol**:

    - **Find** the ***physical* address** that **corresponds** to a given ***logical* IP address**

  - **The ARP protocol:**

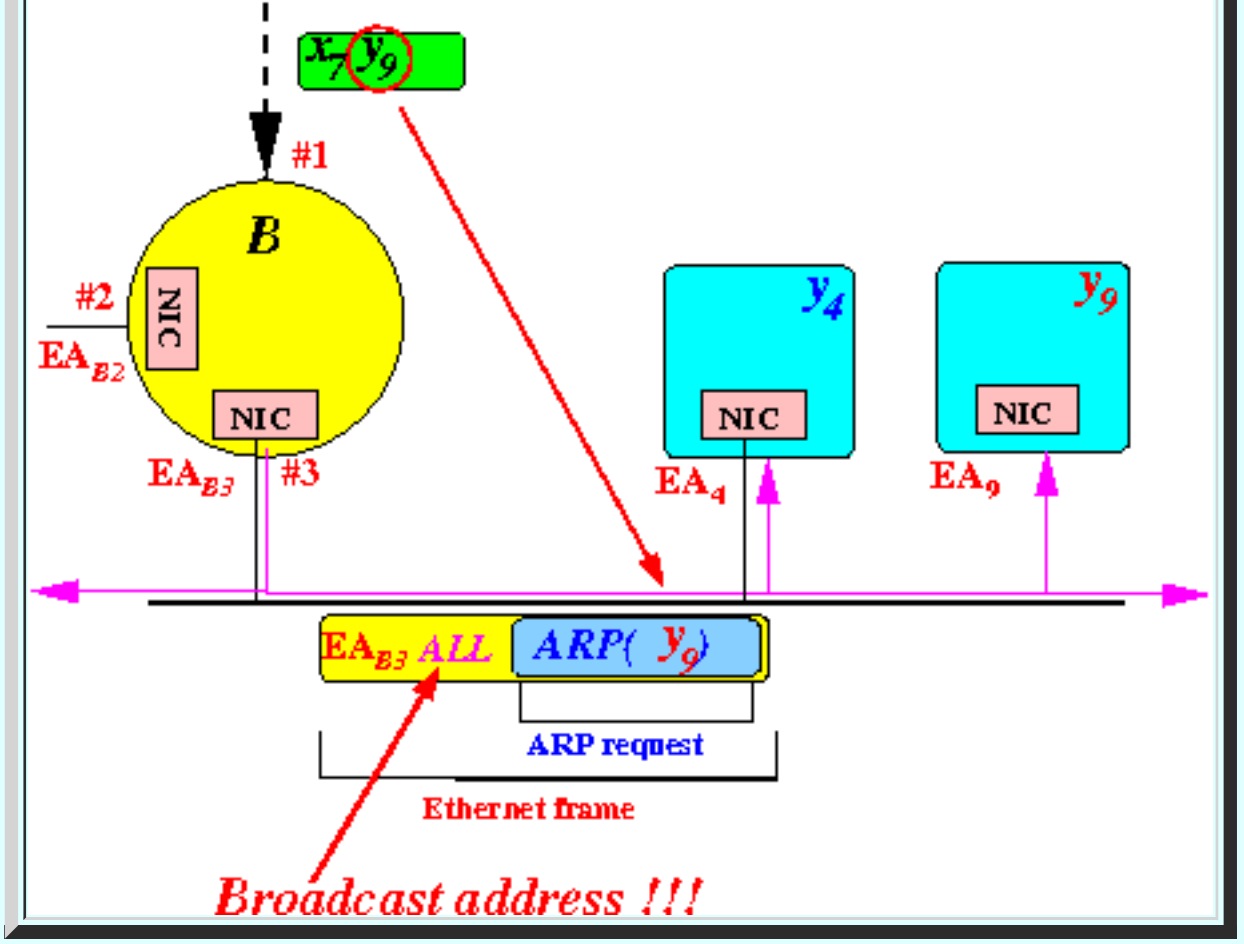
    - The **sender**:

      - Sends a **broadcast Ethernet frame** containing a **ARP-request message** (as **frame type**

        - The **broadcast Ethernet frame** contains the **broadcast destination address** (which is all 1-bits)

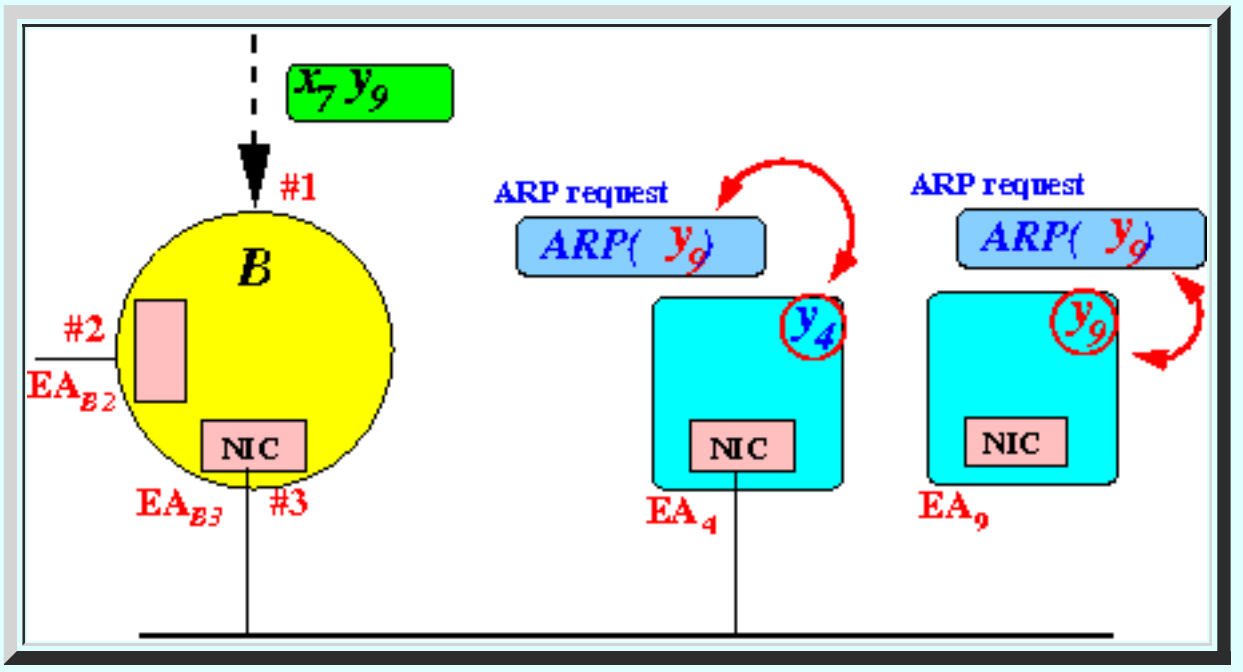
    - The **ARP-request message body** contains the **IP address**

Example: **node *B*** wants to find the ***physical* (Ethernet) address** of the **IP address y_9**



Note: the **Ethernet frame** uses the **broadcast address 111111....1**

- ***All* hosts** on the **network** will **receive** the **ARP request** (because the **frame** has the **broadcast address**):

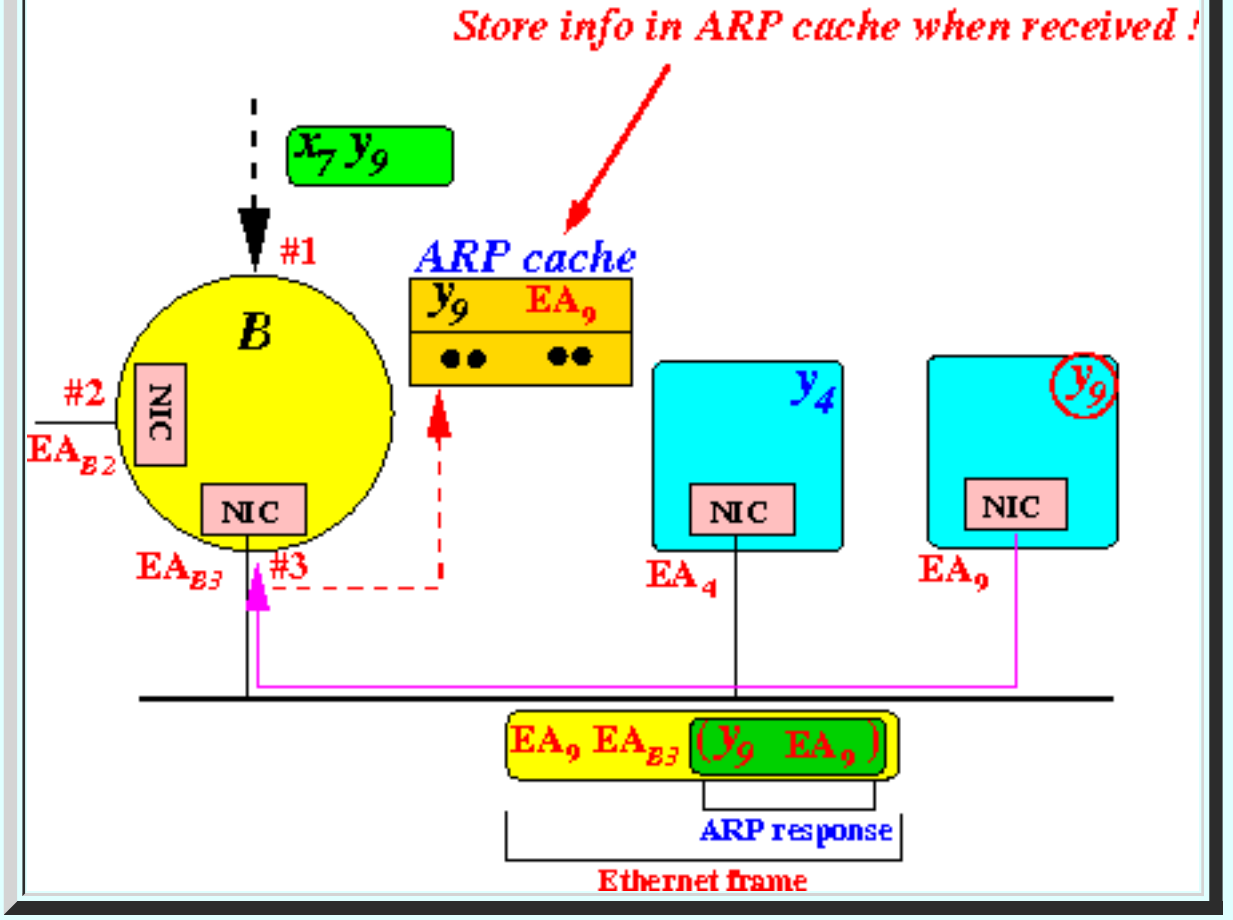


Each host (computer) will than ***match*** the requested **IP address** against its ***own* IP address**

- The **node** whose **IP address *matches*** the **requested IP-address** will reply with an **ARP-*response* message**

  - The **ARP response** contains the ***physical* (Ethernet) address** !!!

Example:



(The **ARP response** is carried in the **data portion** of the **Ethernet frame**)

- **The ARP Cache: minimize the usage of the ARP protocol**
  - **Fact:**

    - An **ARP request** is very ***intrusive*** (= disturb **many nodes**) because it is transmitted using **broadcast**:

      - ***All*** **nodes** in the **network** will **receive** and ***must*** **process** the **ARP message**

  - **Reducing** the **number** of **ARP requests:**

    - The **information** returned by the **ARP protocol** is ***saved***:

      - in the **ARP cache**

    - The **ARP protocol** is ***only*** invoked when:

      - The **IP address** can ***not*** be **found (= resolved)** using the **ARP cache** information

- ***Security Problems* with ARP**
  - **Fact:**
    - The **ARP protocol** has some **major security risks**...
  - But this topic (**security**) is **beyond the scope** of this **course**...
  - If you want to **explore** some **spoofing techniques**, read this excellent article: **click here**

    (I have a local copy: **click here**)

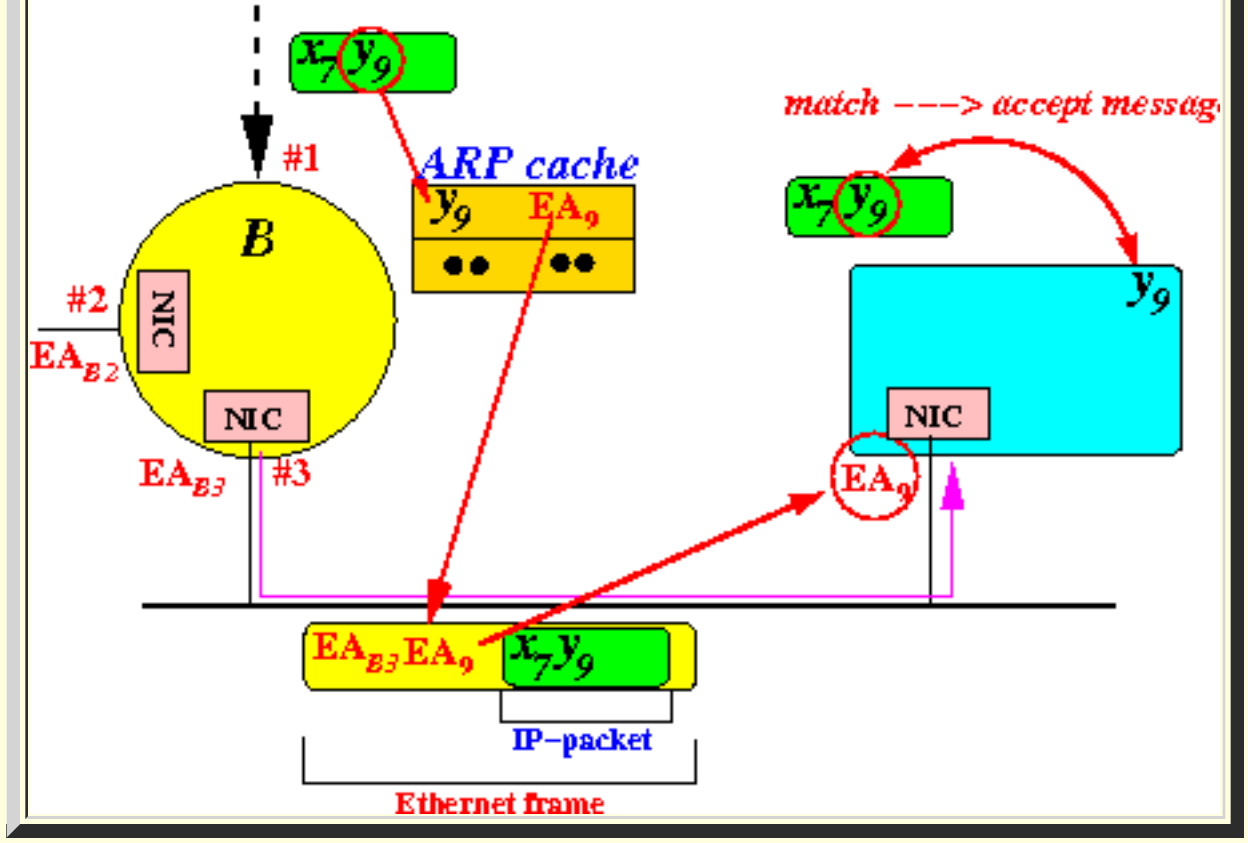
Time out and flushing of the ARP cache

- **A *problem* of using ARP cache: when an IP host changes its IP address**
  - **Fact:**
    - The ***mapping*** between **IP addresses** and **Physical (e.g., Ethernet) addresses** can ***change*** over time

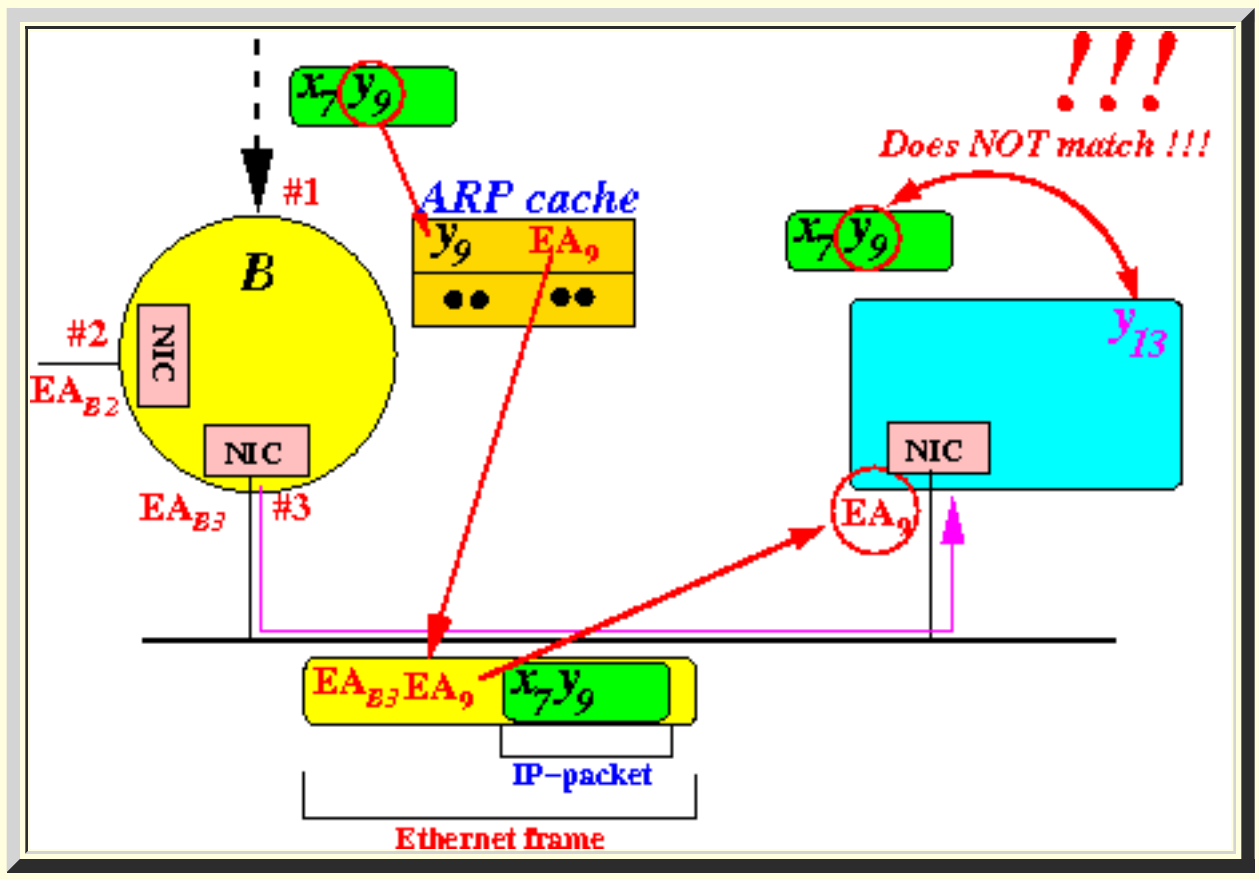
  - **Reasons** that can cause a **change**:
    - A **node** can **change** its ***physical* address**:
      - A **node** might experience a **hardware failure** and needs a ***new* network card**
    - **More common:** a **node** can **change** its ***logical* (IP) address**:
      - When a **node (computer)** is **rebooted**, it ***can*** receive a ***different* IP addresses**

  - **Fact:**
    - If the ***physical* address or** the **logical (IP) address** is ***changed***, then:
      - The **ARP mapping** will be ***incorrect*** (***mapping* error**)

    Example:
    - ***Before*** the **node y_9** has ***physical* address $\mathtt{EA}_9$**:



- **After** the **node** is **rebooted**, it is given the **IP address** y_{13}:



Result:

- **IP packet** destined for y_9 will ***not*** be **accepted** by the ***new*** **configuration** !!!

  (Because the **IP packet** has a ***wrong*** **destination IP address** !!!)

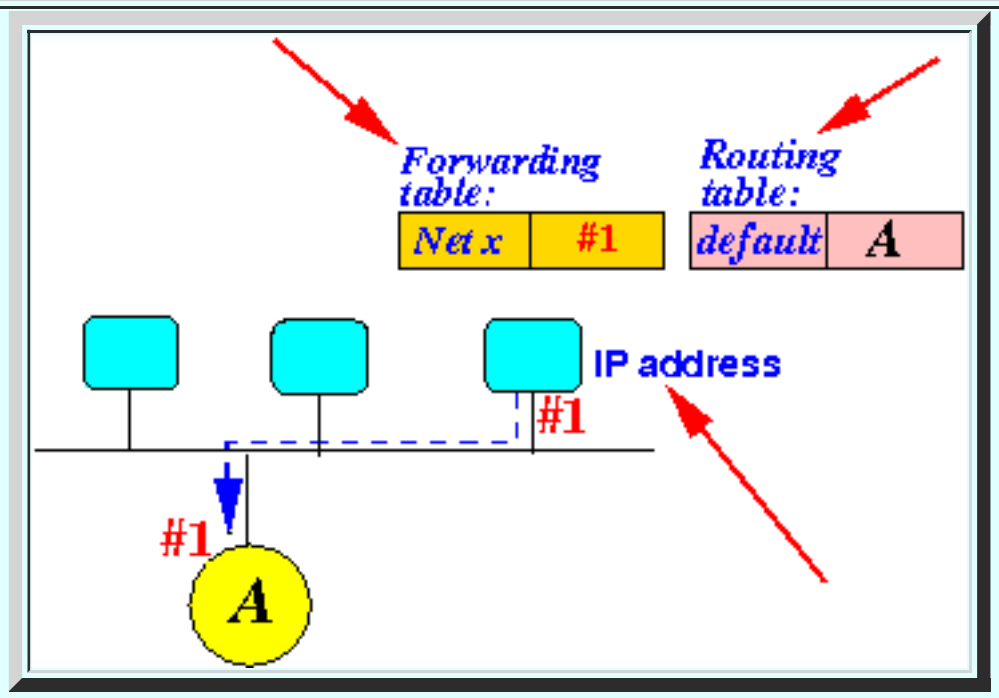
- **Fixing the out-of-date ARP mapping information**
  - **Fixing** the **out-of-date** **ARP cache entry** problem:

    - Each **entry** in the **ARP table** has:
      - a ***expiration*** **("aging") time out** value

    (The **time out value** is usually **several hours**)

Setting up hosts/nodes on the Internet

- **Minimal *information* needed to connect a host/node on the Internet**
  - **Minumum** amount of **information** needed to connect a **host/node (computer)** on the **Internet:**



    - The **host** must be given a (unique) ***IP address***
    - The **host** must have a ***minimum* (smallest possible) routing table** (in order to **forward IP packets** !!!)

- **Pre-lude: information available to a computer at *boot time***
  - **Booting**:

    - **Booting** = **loading/starting** an ***Operating System*** into a **computer**.

      (= **Turn on** a **computer**)

  - **Information available** to a **computer** at **boot time**:

    - The **Ethernet address** of its **network card**

      Because:

      - The **Ethernet address** is **enbedded** inside the **Ethernet card**

      (A **computer can** read the **Ethernet address** from an **Ethernet card**)

    And **nothing else**...

- **Configuring a computer/router on the Internet in general**
  - **In general:**

- **Setting up** a computer **in general** involves **entering** a ***lot*** of **information** into the **routing table**

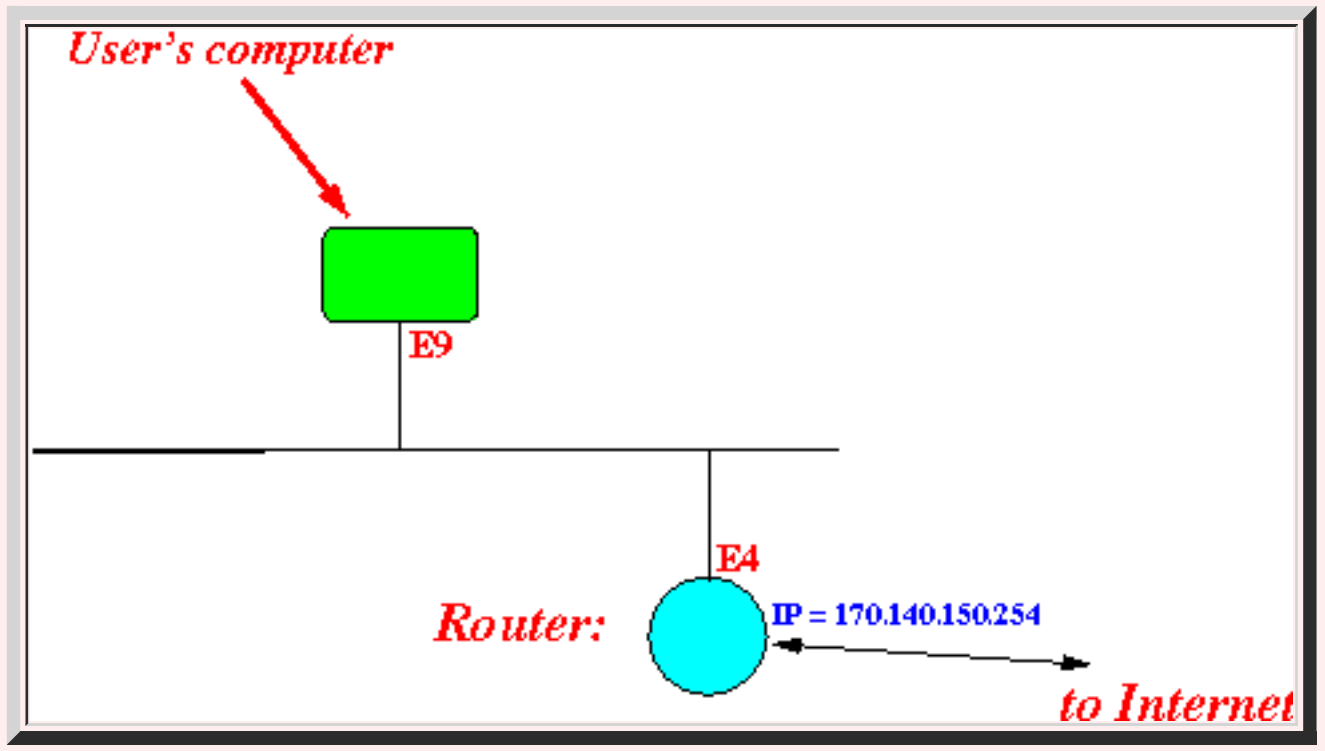
  - **Setting up** a computer on the **Internet** **in general is *not easy***

    This process must be done manually by humans

  (That's why there are **good paying jobs** like **network administrators**)

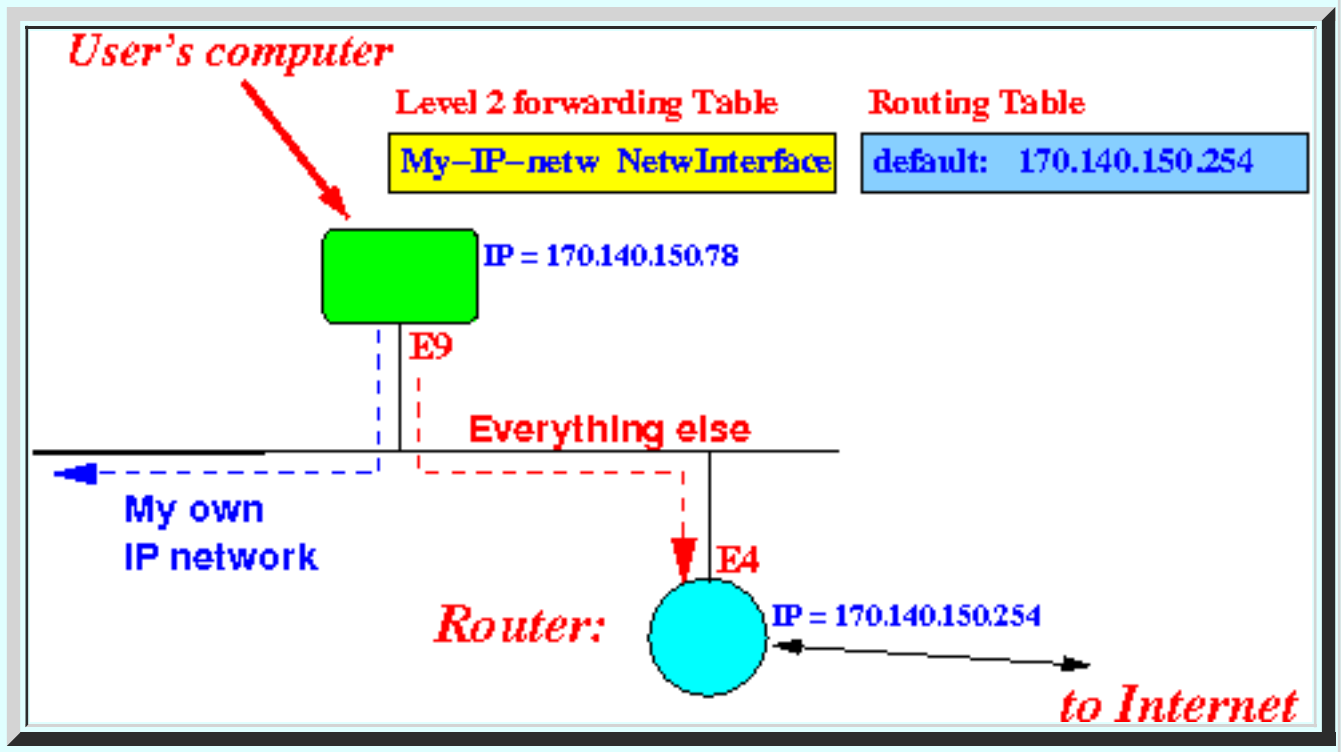
- **The most common IP host configuration**
  - The ***most common*** **configuration** of an **Internet host** is:

    - A **user's desktop workstation/laptop** that is **connected** to ***one*** **physical (Ethernet/Wireless) network**



  - **IP configuration setup** for **such** an **IP host (computer)**:

    - **Level 2 forwarding table** and **level 3 routing table** for **such** a **host/node**:



This **configuration** will:

- Forward **IP packets** with ***same*** **network ID** as **yourself**:

  - Onto your own **IP network *directly*** to the **destination** (**according** to the **<level 2 forwarding table**)

- Forward ***all other*** **IP packets** to:

  - The ***default*** **router**

- **Example:** output of `netstat -r -n` on **W303**

```
cheung@W303> netstat -r -n

Kernel IP routing table

Destination        Gateway         Genmask         Flags   MSS Window  irtt Iface
=============================================================================
170.140.150.0/23   0.0.0.0         255.255.254.0   U         0 0          0 eth0
169.254.0.0/16     0.0.0.0         255.255.0.0     U         0 0          0 eth0
0.0.0.0            170.140.150.254 0.0.0.0         UG        0 0          0 eth0
```

How to read the **information**:

- The **entry**:

  ```
  170.140.150.0/23   0.0.0.0         255.255.254.0   U         0 0          0 eth0
  ```

  functions as an **entry** in the **forwarding table**

  (The **forwarding table** is a ***text book*** **concept** --- **how** it is **implemented** depends on the **software vendor**)

- The **default entry** in **routing table** is:

  ```
  0.0.0.0            170.140.150.254 0.0.0.0         UG        0 0          0 eth0
  ```

- The **169.254.0.0/16** entry is the **"zero configuration" route** put in to the **routing table** by the **start up procedure**

  - **By default**, the **zeroconf route (169.254.0.0)** is **enabled** when the **system boots**.

    See: **click here**

  (Looks like our **System Administrator** left it **on**)

Intro to DHCP

- ***Automated procedure*** **to setup the most common (and simplest) configuration: DHCP**
  - **Fact:**
    - The **most common (simplest) IP host/node configuration** can be ***automated*** by using:
      - The **Dynamic Host Configuration Protocol** (**DHCP**)

  - The **Dynamic Host Configuration Protocol (DHCP)** allows an **IP host/node** to:
    - obtain a ***unique*** **IP-address** at **boot time**

      Note:

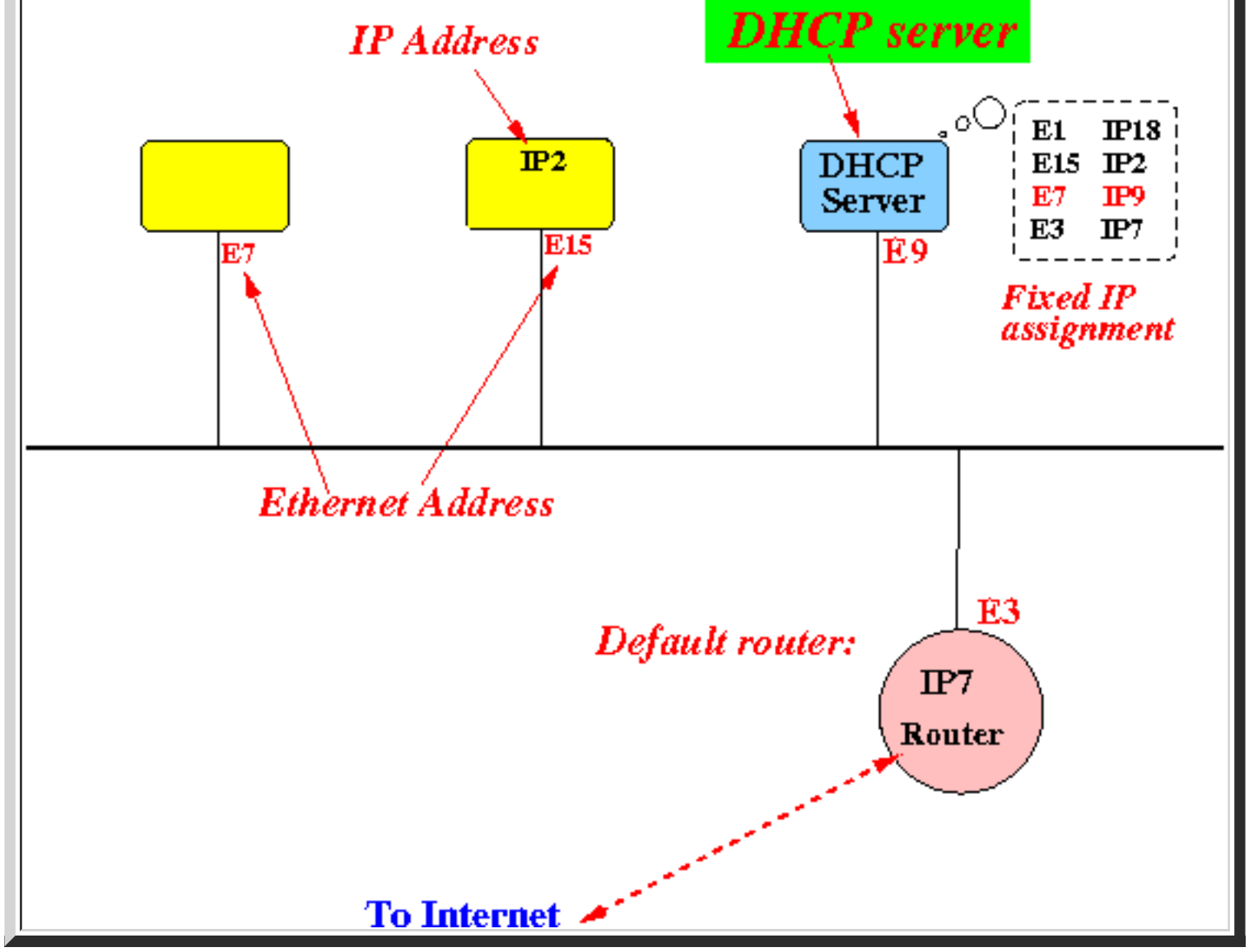
      - The **node** can **determine** its **IP network ID** because the **IP network ID** is a ***part*** of the **IP address** !!!

    - obtain the **IP-address** of the ***default*** **router** on the **network**

  - **Note:**
    - The **DHCP** protocol **works like** a ***reverse*****-ARP** protocol:
      - **ARP** finds the ***physical*** **address** for a **given IP address**
      - **DHCP** finds (allocates) an ***IP*** **address** for a **given physical (Ethernet) address**
    - The **DHCP** protocol was develped from an **older protocol** called **BOOTP** (**Bootstrap Protocol**)

      See: **click here**

- **The DHCP server**
  - The **DHCP protocol** is implemented using a **DHCP "server"**:



- **DHCP server**:

  - **DHCP server** = a **program** that **implements** the **DHCP protocol** that **responds** to **DHCP *requests*** sent by **(DHCP) clients**

  (A **program** that sends **DHCP requests** (to an **DHCP server**) is called a **DHCP *client***)

Note:

- Many **home routers**:

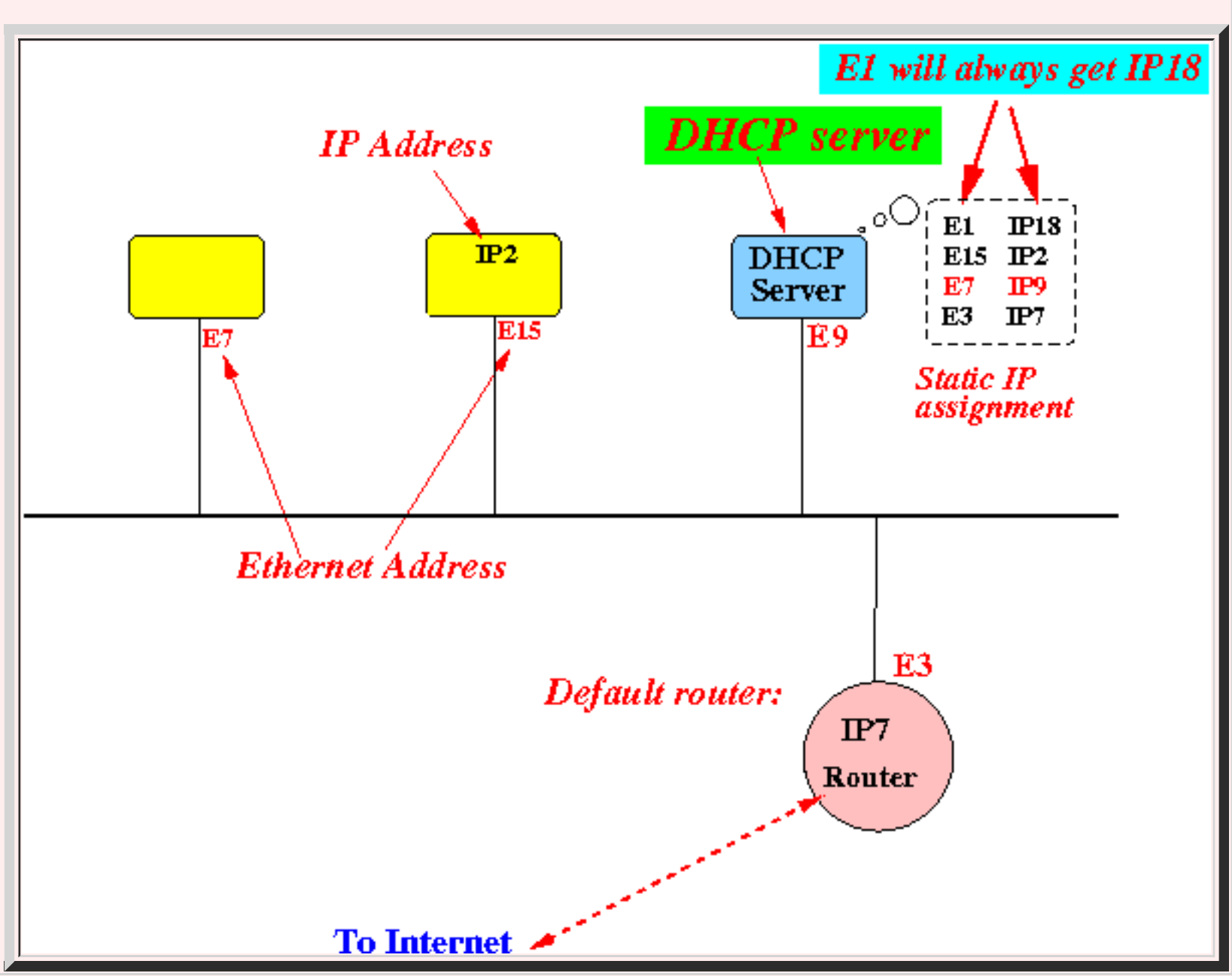
  has a **built-in DHCP server (software !)**

- **Types of IP address assignment schemes**
  - **2 types** of **IP address assignment methods**:
    - **Static IP address assignment**
    - **Dynamic IP address assignment**

- ***Static* IP address assignment**
  - **Static IP address assignment:**
    - ***Static* IP address assignment** = a given **computer** is ***always*** assigned with the ***same* IP address**

  - **Implementation:**
    - In **static IP address assignment**, the **DHCP server** will use:
      - A ***static* IP address table** to **maps** a **fixed set** of **physical addresses** to **IP addresses**

      Example:



  - **Common scenarios** that **require** a ***static* assignment**:

- ***Servers*** (= **computers** that provides a **specific *service***) needs a ***fixed* IP address**

  - **Example: file servers**

  Servers can **only** be **reached *easily*** when its **IP address** does ***not*** **change** !!!

  - **Analogy:** suppose the **Pizza Delivery** keeps ***changing*** its **phone number**.... they'll be **out of business** very soon !!!

- **Computers** that have a **higher level of *trust*** (= privilege) use a ***fixed* IP address**

  - In this case, the **IP address** is used as the **identifier** to **gain access** to **special privileges**.

- ***Dynamic* IP address assignment**

  - ***Dynamic* IP address assignment:**

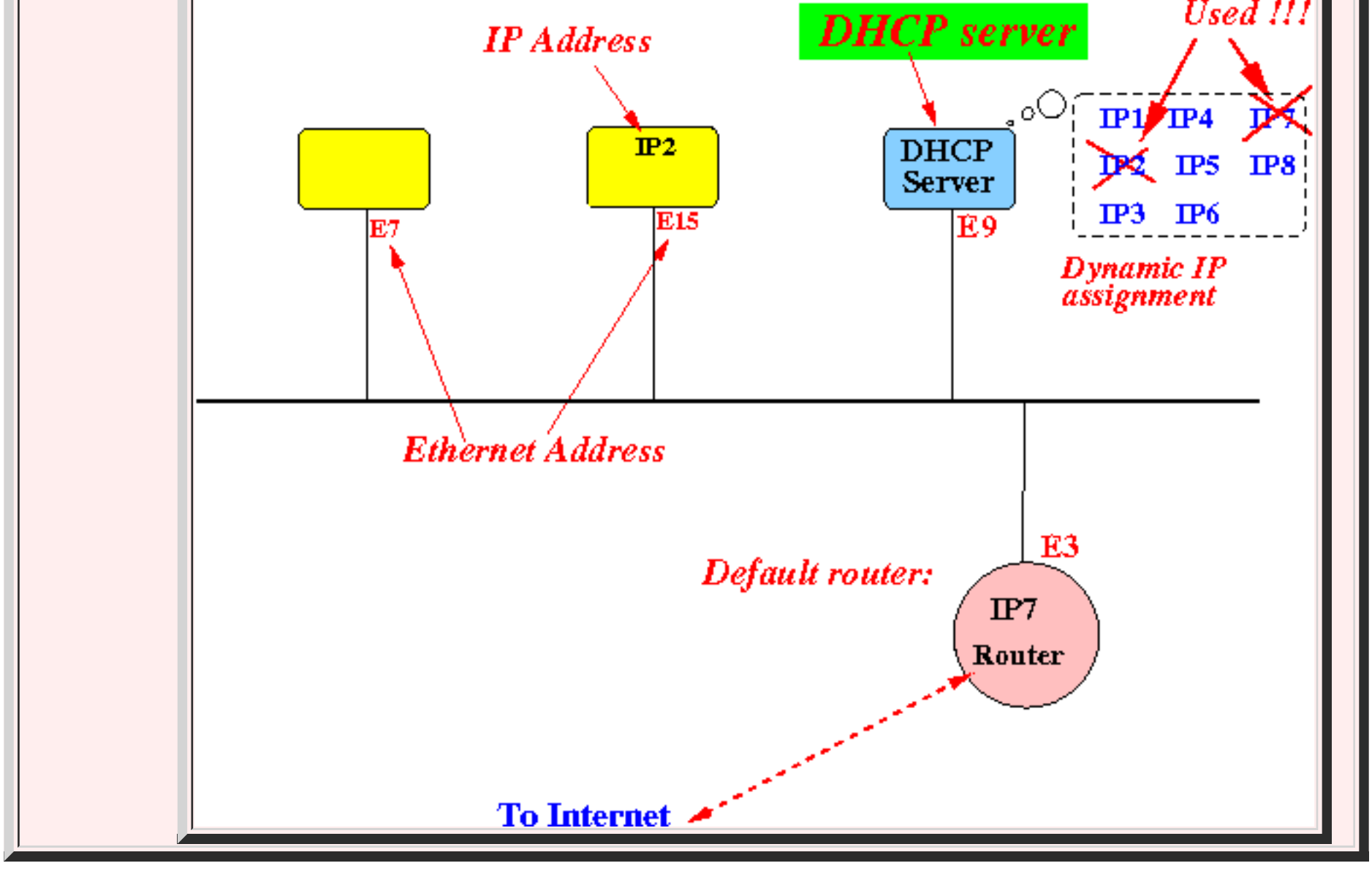
    - ***Dynamic* IP address assignment** = a given **computer** can be assigned with ***different* IP addresses** at ***different* times**

  - **Implementation:**

    - In **dynamic IP address assignment**, the **DHCP server** has a ***pool* (set) of IP address**:

      - The **DHCP** server will ***randomly* select** some ***unassigned* IP address** and assign the **IP address** to a computer

      Example:



- **When** to use ***dynamic* IP address assignment**:

  - ***Dynamic* IP address assignment** is used when a **computer** does **not** provide any ***special* service**

  - The **computer** is as a **user terminal** to run **applications** (such as **web browser**, **editor**, etc....

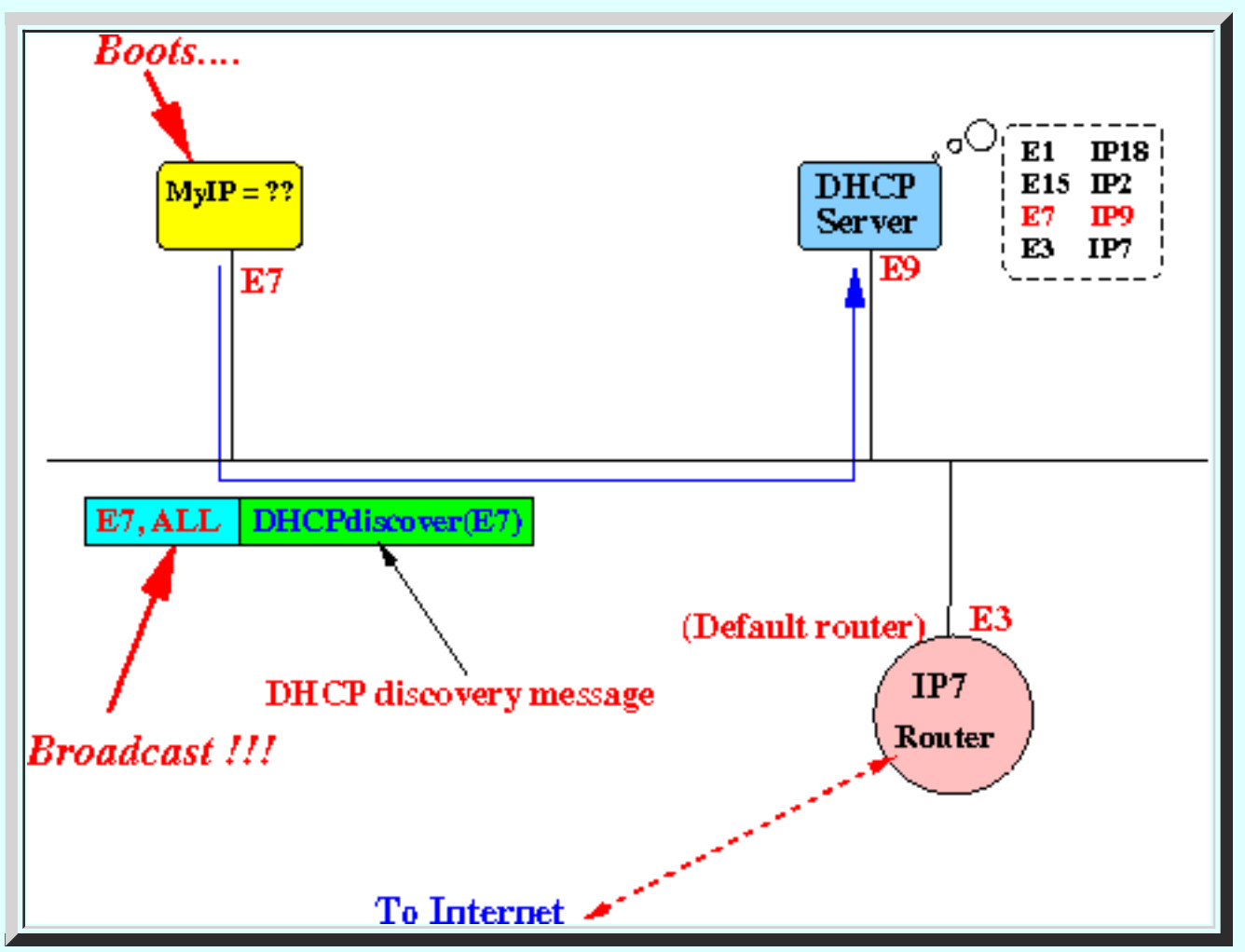
The DHCP protocol

- **The DHCP protocol**
  - **Operation** of the **DHCP protocol:**

1. When a **host computer** **boots (starts)**. the computer sends a **broadcast** **physical (e.g., Ethernet) frame** containing a ***DHCP discovery* message**:

```
DHCP_discover( Physical-address of computer )
```

Example:



---

2. This **message** will be **received** by ***all* nodes** on the network, but:

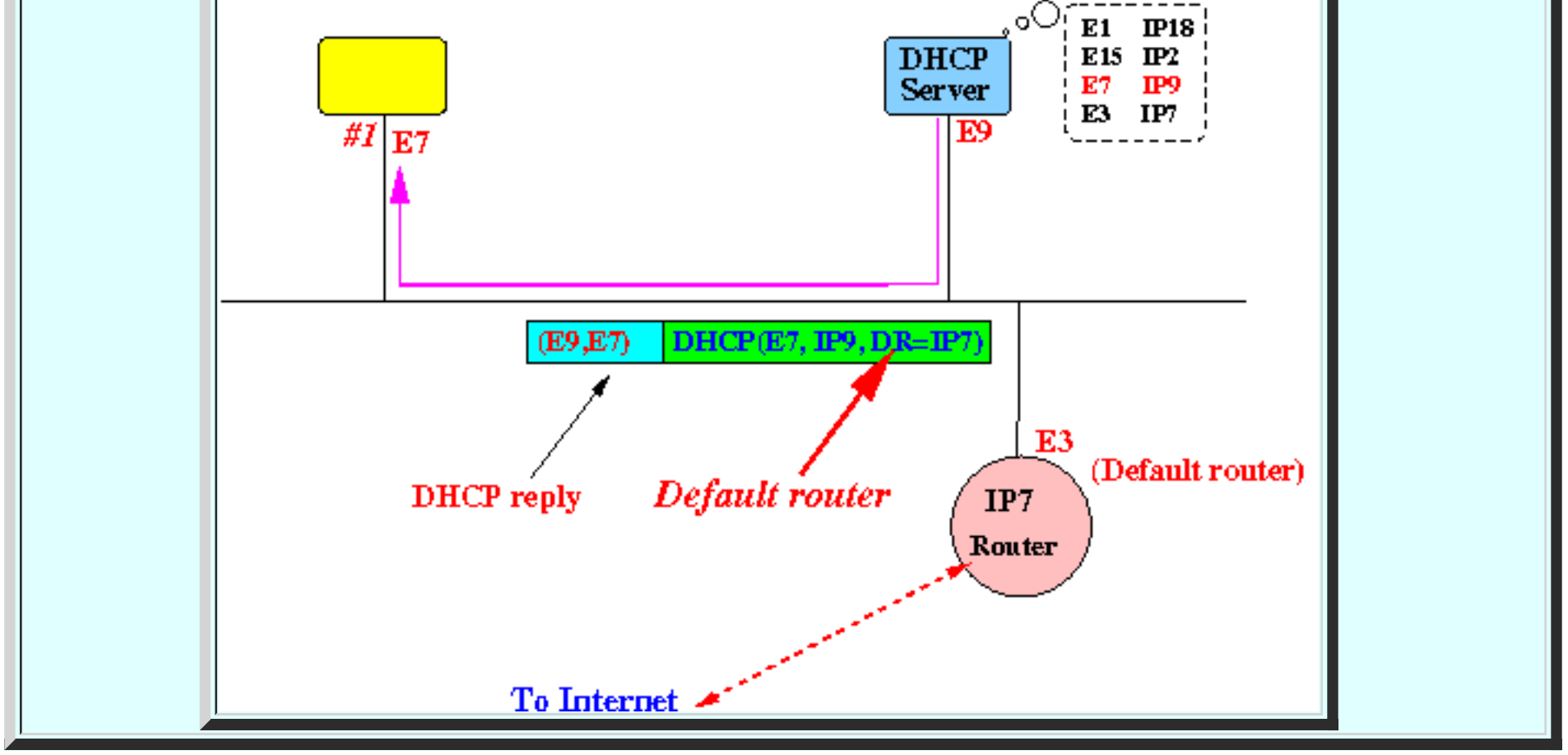
   - ***Only*** the **DHCP server** will **process** the **DHCP_discover** request.

---

3. The **DHCP server** will send back a **DHCP response** containing:

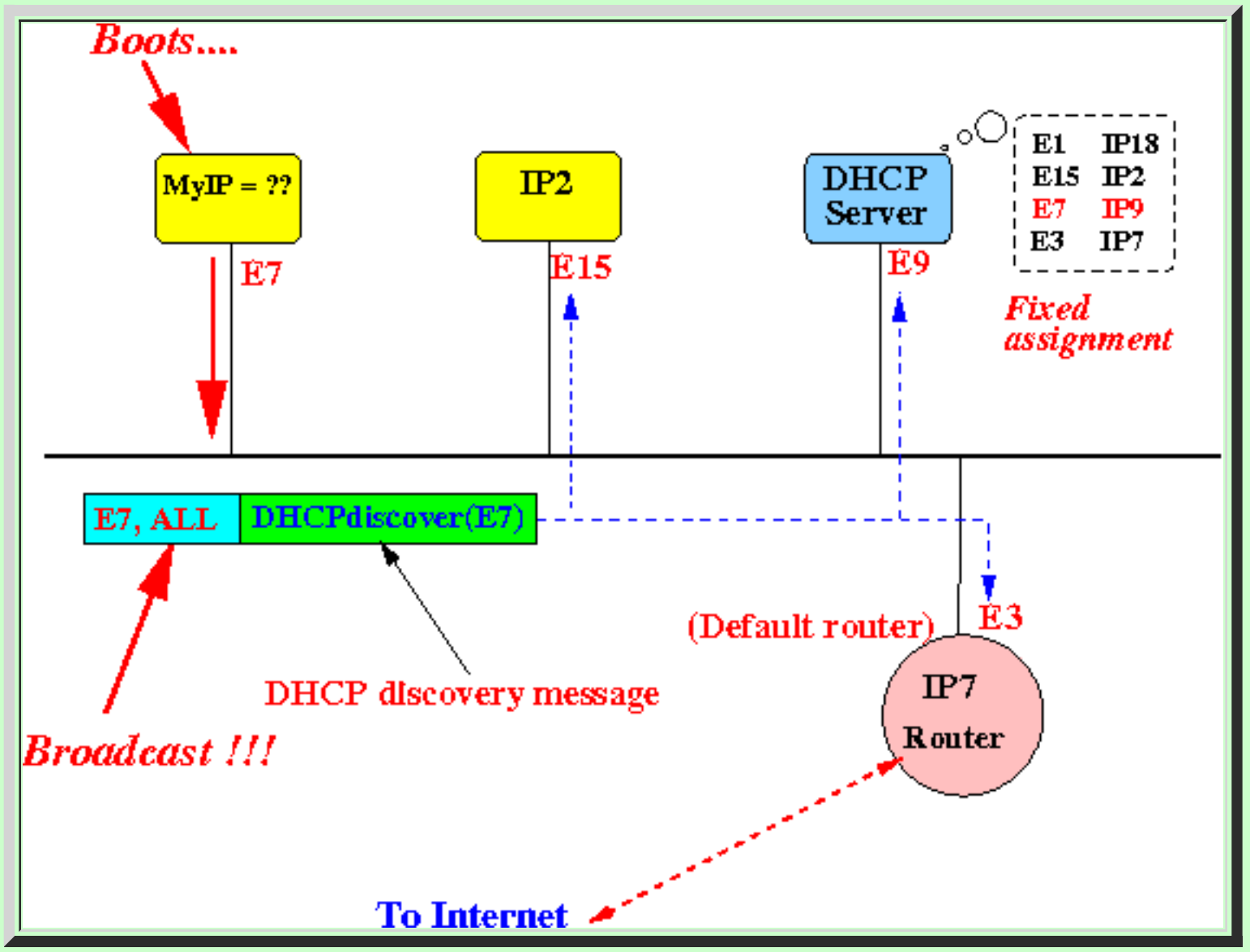
   - the **IP address** for the **requesting host**
   - the **IP address** of the ***default* router**

Example:



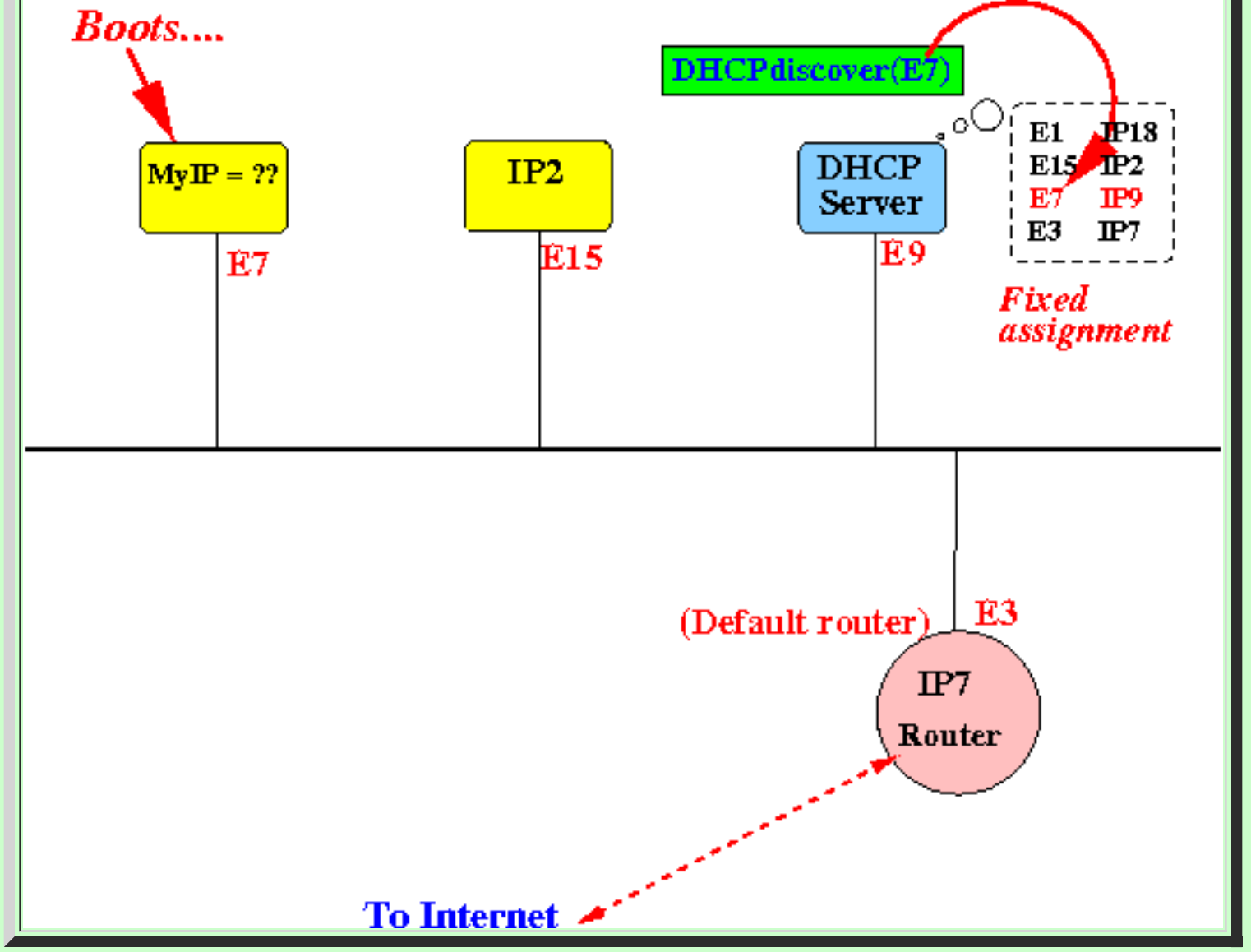
- **DHCP with *static* IP address assignment**
  - **DHCP protocol** using ***static* address assignment**:

1. **Suppose** the **host computer** with **Ethernet Address E_7 boots** and **sends** a ***DHCP discovery* message**:

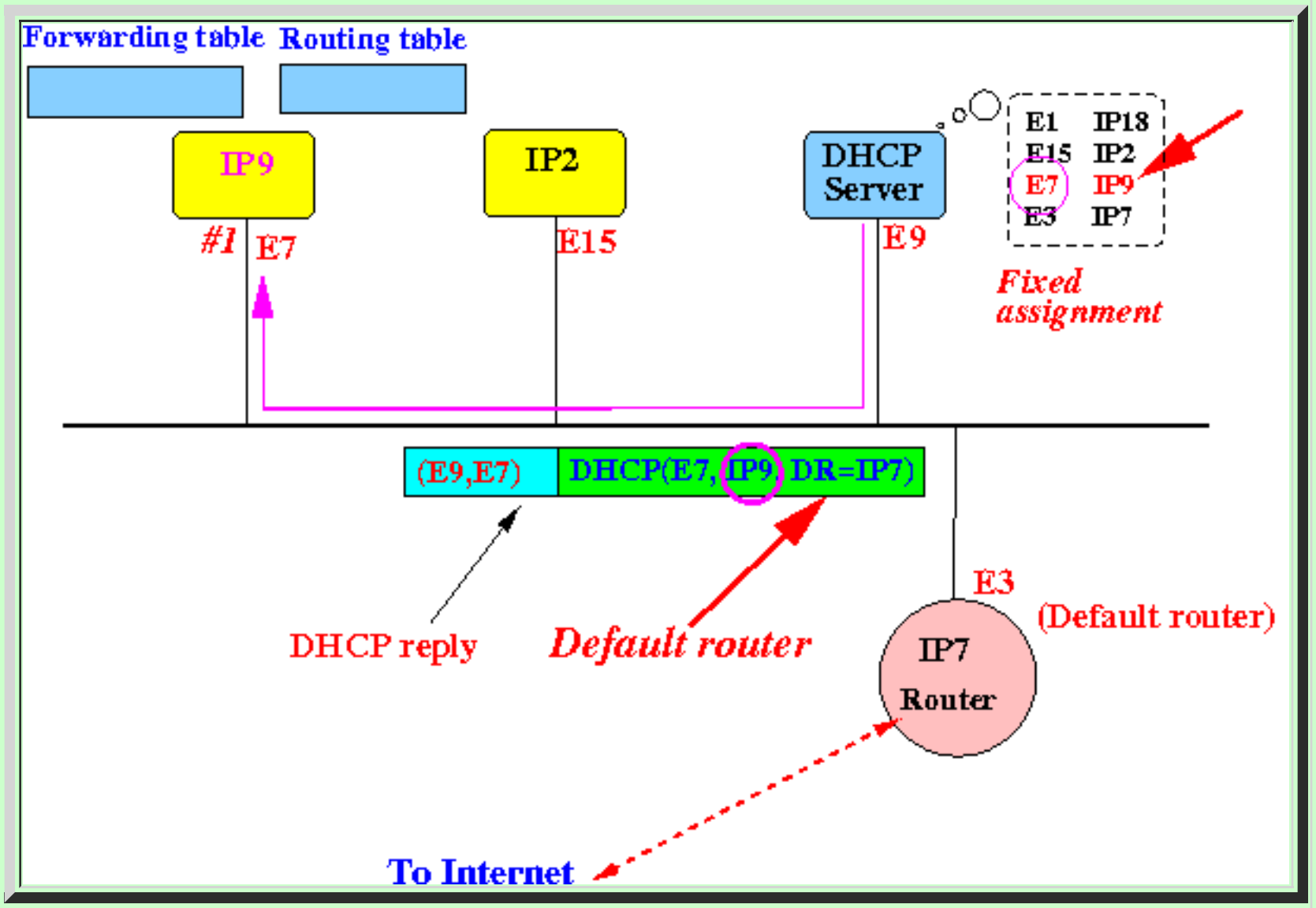


---

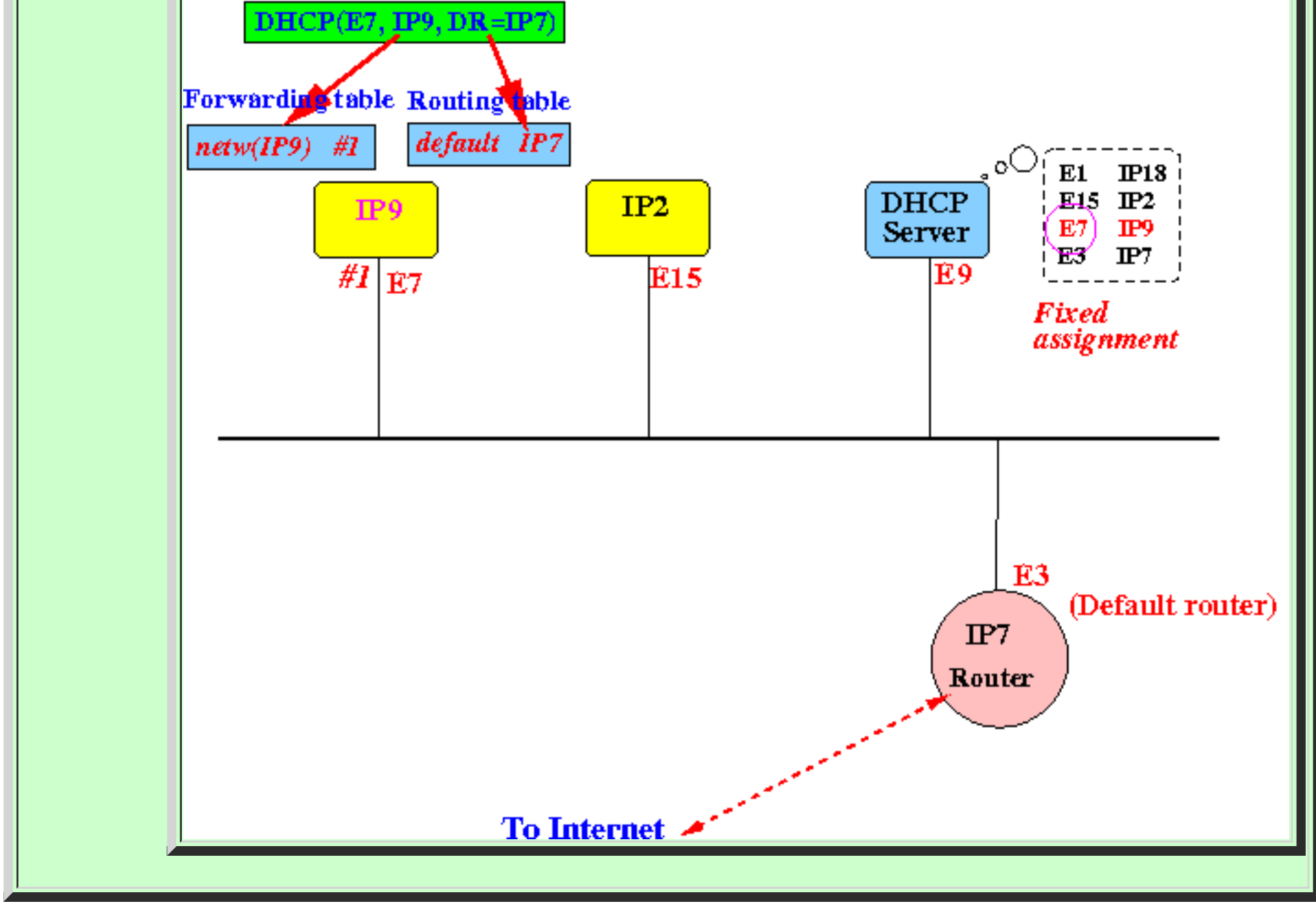
2. ***Only*** the **node** with a **DHCP server** will **process** the **request**:



3. The **DHCP server** sends back a **DHCP reply** containing the **IP address** that corresponds to the given **physical (Ethernet) address**:

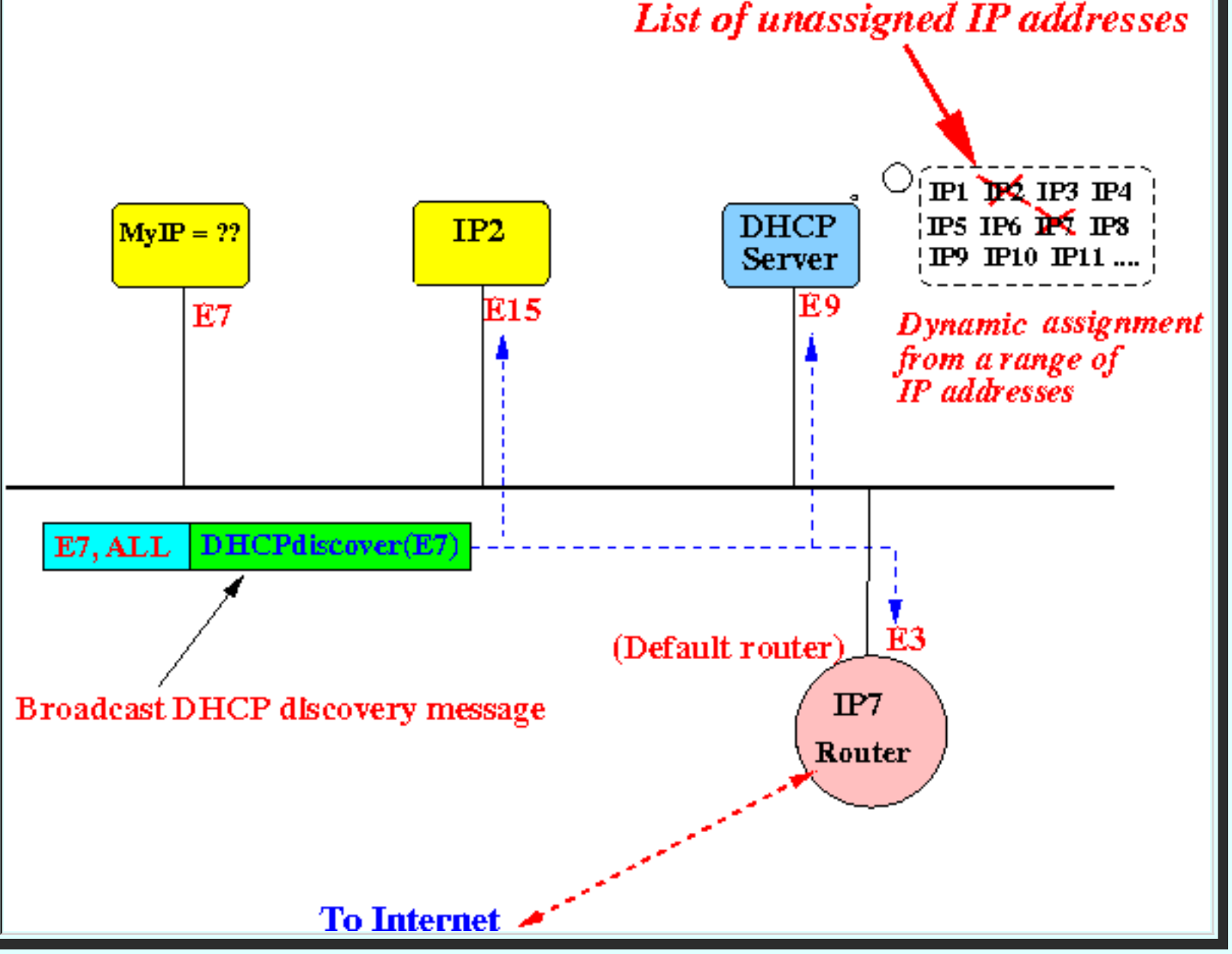


4. The **node** can now **configure** its **forwarding and routing tables**:



- **DHCP with *dynamic* IP address assignment**
  - **DHCP** with ***dynamic* address assignment**:

1. The **DHCP server** maintains a **list** of ***unassigned* IP numbers** :



2. When the **DHCP server** receives a **DHCP discover request**, it selects an ***unused* IP address**, marks it as **unused** and sends a **DHCP reply**.

- **Comments**
  - **Implementation note:**
    - **Most(all ?) home routers** sold today has a **DHCP server** running ***inside*** the **router**.

      E.g.:

    - In the **MathCS department**, the **DHCP server** is running on a **UNIX machine**....

(Easier to manage)

Intro to the Internet Control Message Protocol (ICMP)

- **Managing the Internet**
  - **Errors:**
    - Sometimes, **errors** occur while **transmitting** IP-packets

    Sample errors:

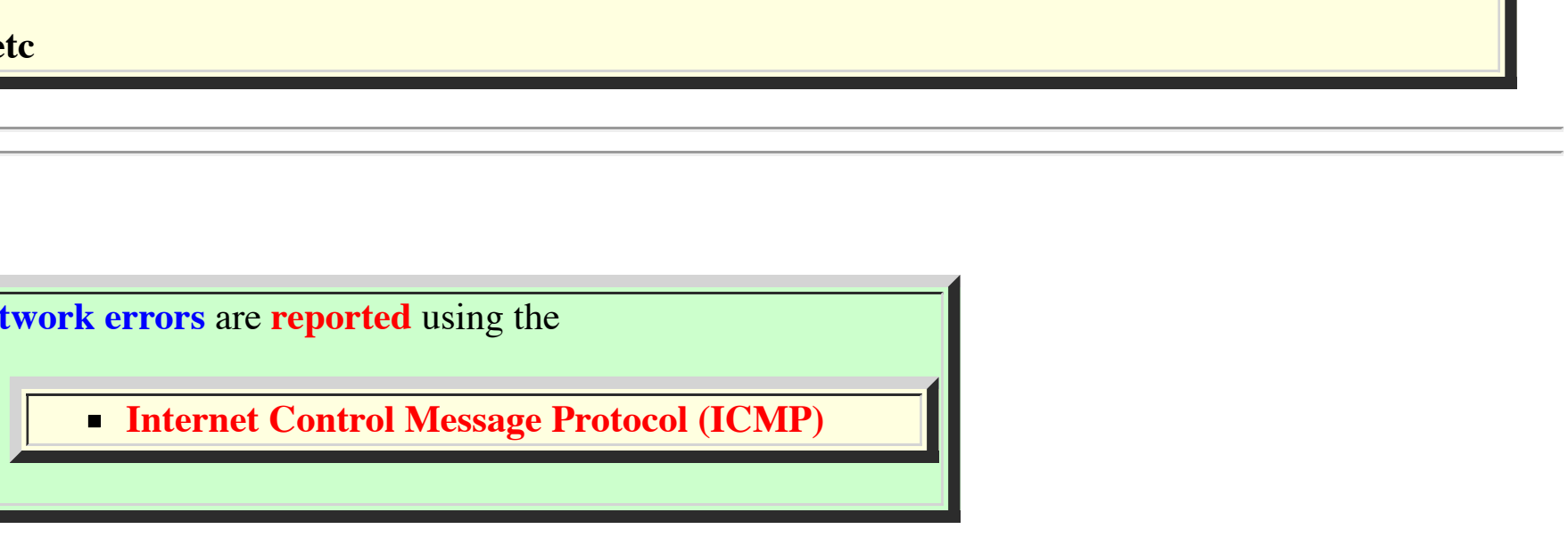
    - The **destination network ID** is ***not*** **found** in the **routing table**
      - Such **router** does **not** have a ***default*** **entry** in the **routing table** !!!!

    - The **TTL field** has counted down to **0** and the **IP packet** is **discarded**
      - The **most common casue** of this **phenonemon** is a **routing loop** in the **routing tables**)

    - **Etc, etc**

  - **Reporting errors:**
    - **IP network errors** are **reported** using the
      - **Internet Control Message Protocol (ICMP)**

- **Transmitting ICMP messages**
  - **How** to **send** an **ICMP message** through the **Internet**:
    - The **ICMP information (packet)** is **transmitted** ***inside*** the **data portion** of an **IP packet**:



- The `protocol` field is **set** to **1 (one)** to **indicate** that the **data** is an **ICMP message**

- **The ICMP data packet format**
  - **Structure** of an **ICMP data packet**:

```
+------------+------------+---------------------------+
|   type     |    code    |       checksum            |
+------------+------------+---------------------------+
|  ICMP data (depending of the type of the ICMP msg)   |
+------------------------------------------------------+
```

Meaning of the **type** code:

Type code	ICMP packet type
0	Echo reply (for echo request)
3	Destination unreachable
4	Source quench
5	Redirect
8	Echo request
9	Router advertisement
10	Router solicitation
11	Time exceeded (**TTL=0**)
etc etc	

- **Network jargon: "out-of-band" communication**
  - **Normal communication** between **2 nodes** on the **Internet**:
    - ***Normally***, **nodes** on the **Internet** transmits **data messages** (such as **email** or **webpages**) to each outer

---

  - **Out-of-band communication:**
    - **Out-of-band communication** = ***abnormal*** **communication**
      - The term **out-of-band** refers to **communications** which **occur *outside*** of a **previously established communication method** or channel.

    - **Out-of-band communications** are **messages** that are ***not*** **part** of the **normal** messages that are **transmitted** between the **nodes**
      - **Out-of-band messages** are ***not*** **generated** by **users**
      - **Out-of-band messages** are the **result** of some ***unexpected*** **event** (like **errors**)

- **Example** of **out-of-band** communication:

  - The **Internet Control Message Protocol (ICMP)** messages

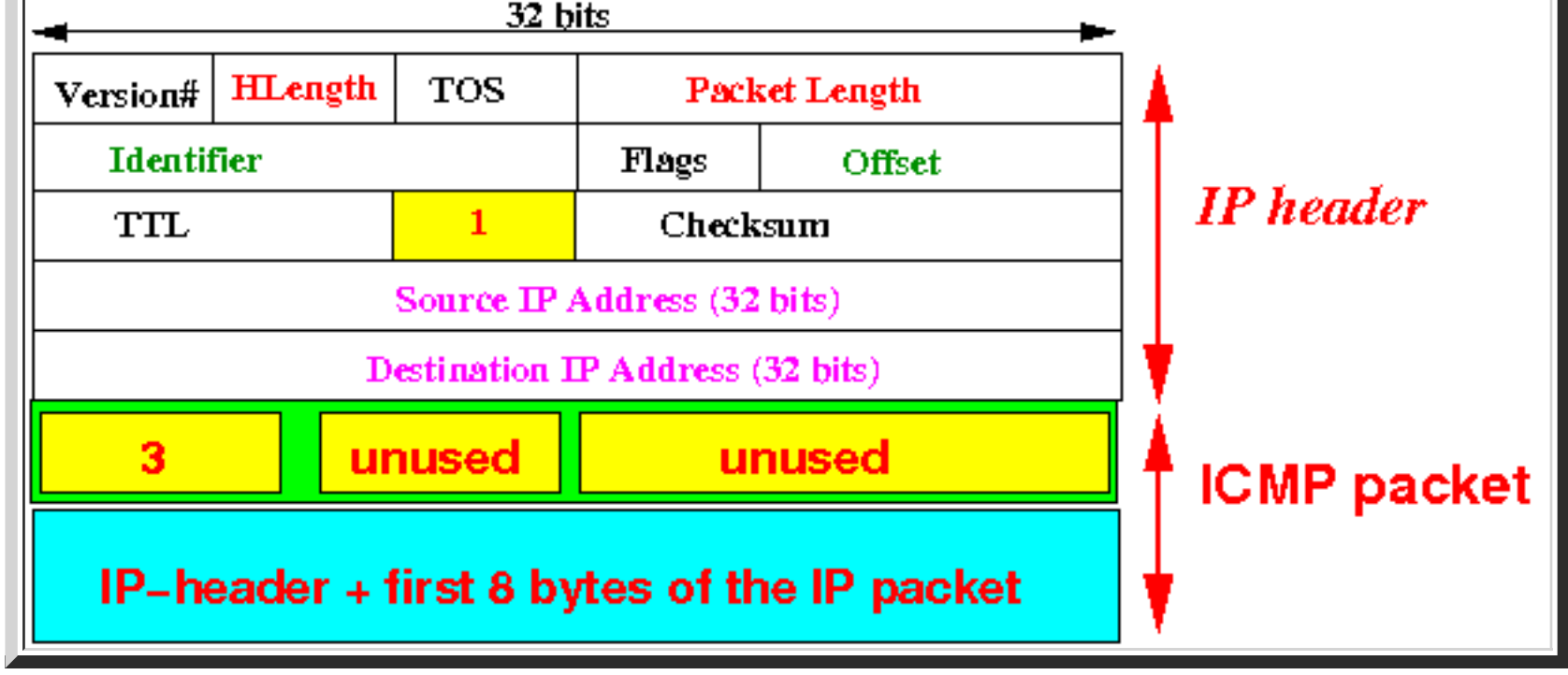
ICMP: reporting routing errors

- **Routing errors**
  - **Routing error:**
    - **Routing error** can **occur** when:
      - Some **intermediate node** or **link** has **failed**
      - A **top-level router** (a **router** in the **Internet backbone**) of the **Internet** do **not** recognize the **destination network ID**
        - (At the **time** that the **routing tables** were **computed**, the **destination network** was ***not*** **operational**)

      Internet backbone: **click here**

  - **Reporting** a **routing error:**
    - Use the **ICMP** `Destination unreachable` **packet**
    - **Packet format:**
      - The **type code** is **equal** to **3**
      - The **ICMP data** contains the ***first*** **8 bytes** of the **IP packet** (for **error reference**)

Graphically:

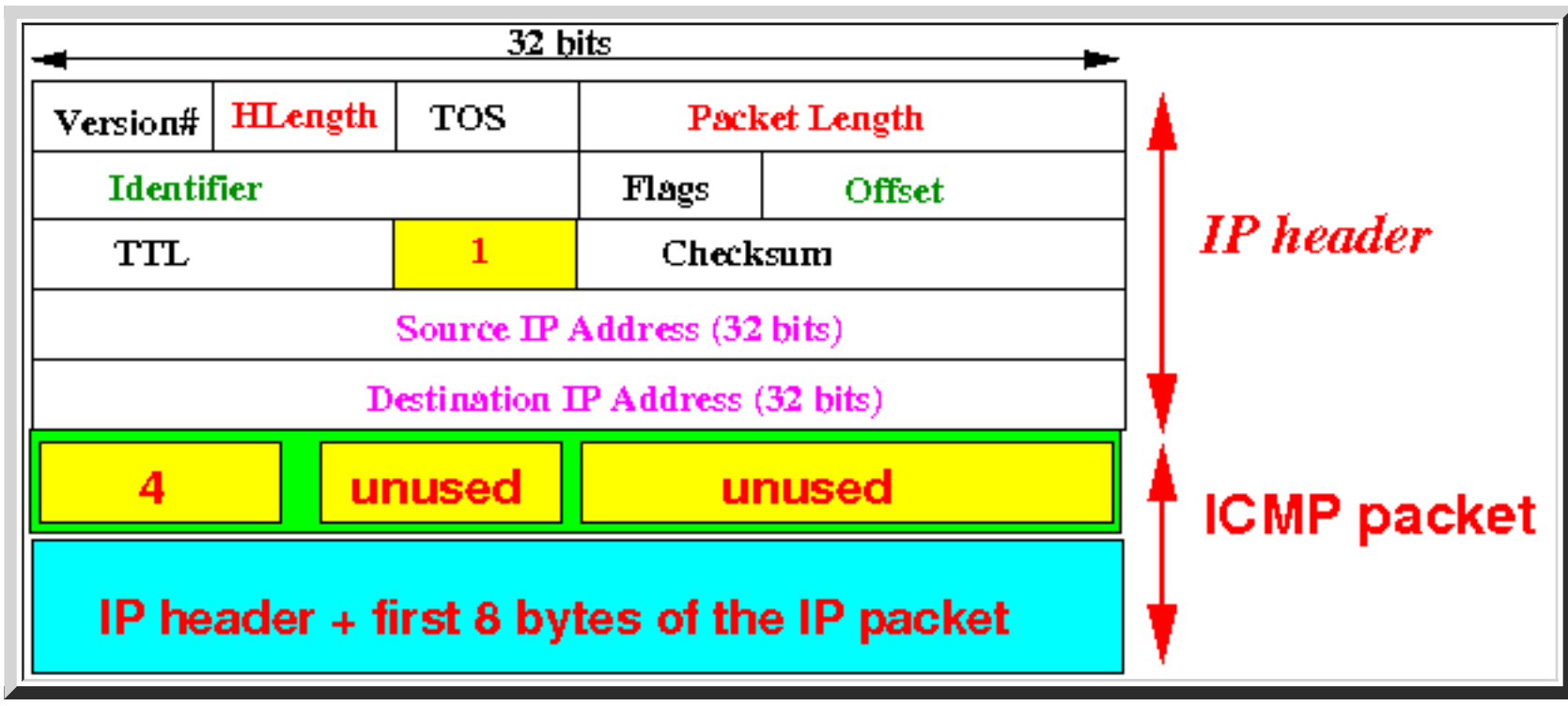

32 bits
Version#
HLength
TOS
Packet Length
Identifier
Flags
Offset
TTL
1
Checksum
Source IP Address (32 bits)
Destination IP Address (32 bits)
IP header
3
unused
unused
ICMP packet
IP–header + first 8 bytes of the IP packet

ICMP: reporting congestion

- **Reporting congestion**
  - The **Source Quench ICMP** control message:

    - An **intermediate router** will transmits a **source quench** (**type = 4**) **ICMP control message** to the **source node** if:
      - The **intermediate router** does **not have** enough **buffer space** to **store** the **IP packet**

    Packet format:



  - The ***desired*** **response**:

    - A **ICMP source quench message** ***should*** cause the **sender** to **reduce** the **packet transmission rate**

    (But it is ***not*** **guaranteed** that the **sender** will **"obey"** the **suggestion** to **slow down**....)

ICMP: assisting in network trouble shooting

- **Checking for *liveliness***
  - **Echo request:**

    - **ICMP code = 8** is **echo request**

    - When a **IP node** receives an **echo request**, it **must**

      - **respond** with an **echo reply (code = 0) ICMP frame**

- **The `ping` utility**
  - The **`ping` utility**:

    - **`ping`** = a **utility** that **check** is a **host** is **alive**

    Example:

```
cheung@aruba> ping -c 3 compute.mathcs.emory.edu

PING compute.mathcs.emory.edu (170.140.150.8) 56(84) bytes of data.
64 bytes from compute.mathcs.emory.edu (170.140.150.8): icmp_seq=1 ttl=255 time=0.363 ms
64 bytes from compute.mathcs.emory.edu (170.140.150.8): icmp_seq=2 ttl=255 time=0.517 ms
64 bytes from compute.mathcs.emory.edu (170.140.150.8): icmp_seq=3 ttl=255 time=0.412 ms

--- compute.mathcs.emory.edu ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.363/0.430/0.517/0.068 ms
```

  - **How** is it **implemented**:

    - The **ping** utility transmits an **ICMP echo request** to the **destination node**...

    Experiment:

    - **Google** the **source code** of the **ping** utility: click here

    - Here is one: click here

      Search for **`sendto`**, you will see:

```
u_char icmp_type = ICMP_ECHO;

....

static void
pinger(void)
{
        struct timeval now;
        struct ip *ip;
        struct icmp *icp;
        int cc, i;
        u_char *packet;

        /* ================================
           Make ICMP packet
           ================================ */
        packet = outpack;
        icp = (struct icmp *)outpack;
        icp->icmp_type = icmp_type;
        icp->icmp_code = 0;
        icp->icmp_cksum = 0;
        icp->icmp_seq = htons(ntransmitted);
        icp->icmp_id = ident;

        ....
        icp->icmp_cksum = in_cksum((u_short *)icp, cc);
        ....

        i = sendto(s, (char *)packet, cc, 0, (struct sockaddr *)&whereto,
            sizeof(whereto));
        ....
}
```

ICMP: reporting packet "time out" errors

- **Packet "time out"**
  - **Packet time out**:

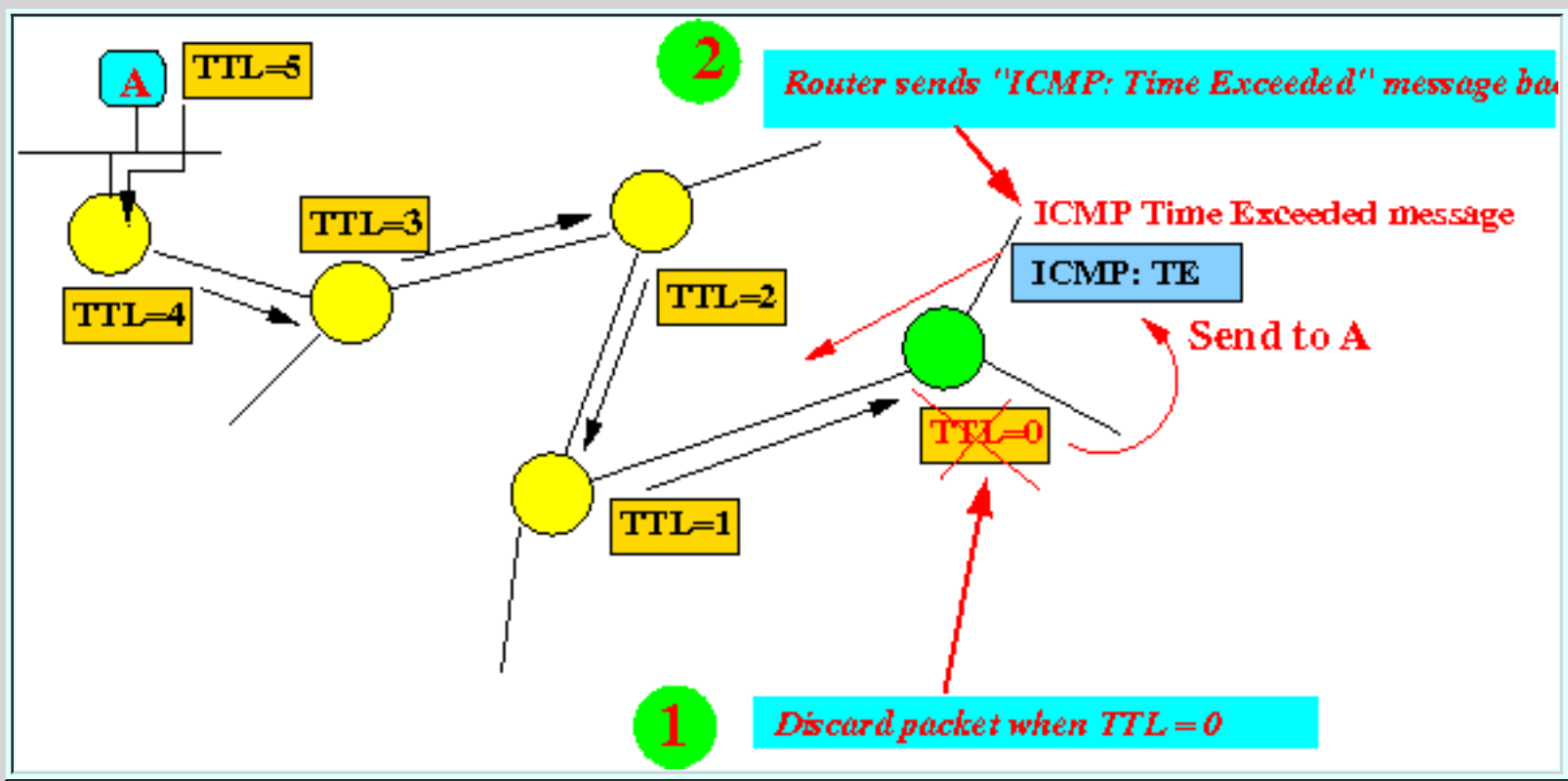
    - **Each time** a **IP packet** is **forwarded** by a **router**:
      - The **TTL header field** is ***decremented*** by **1**

    - When the **TTL (Time To Live)** field in the **IP packet header** becomes **zero**:
      - The **IP packet** is **discarded ("times out")**.

  - **Reporting** the **Packet "timeout" error** :

    - A **router** that **discards** an **IP packet** ***must***:
      - send an ***ICMP Time Exceeded*** **message** back to the **sender** !!!

    Example:



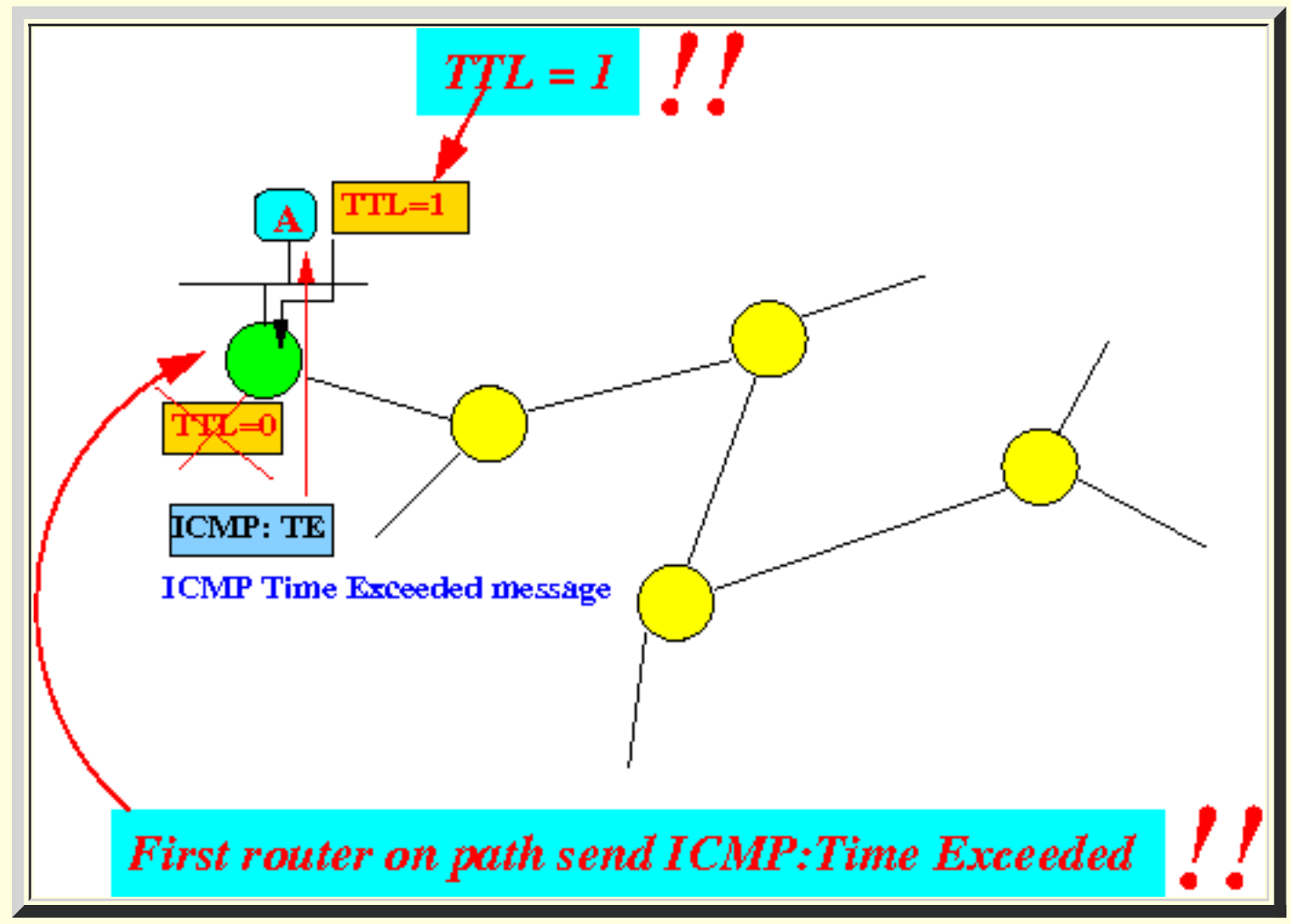
  - **Note:**

    - The **ICMP:** `time exceeded` **error** **packet** contains:
      - **Source IP address** = **IP address** of the **router** that has ***dropped*** the **IP packet** !!!!

- **Creative way to use the Time Exceeded error reports**
  - **Deliberate time outs**:

- **Suppose** we ***deliberately*** set the **TTL value** of the **IP packets** as follows:

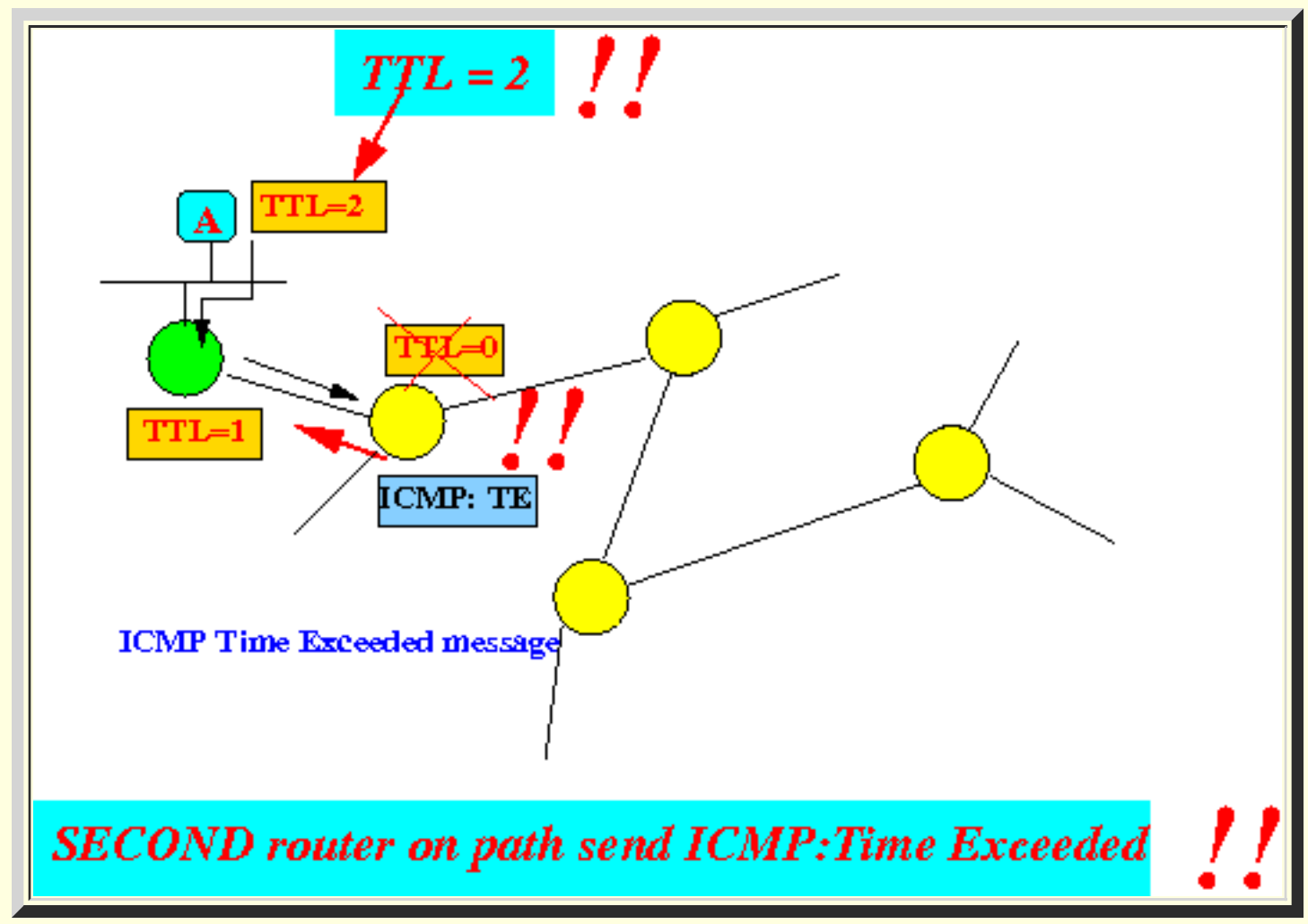
  - **Transmit** the **first IP packet** with **TTL = 1**:



Result:

> - The ***first*** **router** on the path to the **destination** will **send back** a **ICMP time exceeded** message.

- **Next**. transmit ***another*** **(dummy) IP message** with **TTL = 2**



Result:

> - **Now** the **second router** on the path to the **destination** will respond with a **ICMP time exceeded** message.

- **And so on:**

- We can **discover** the **identity** of the **routers** on the **path** to the **destination** !!!!

- The **traceroute** utility:

  - `tracroute` = a **network utility** that **exploits** the **Time Exceeded ICMP mechanism** to find the **routers** on the **path** to a **destination**

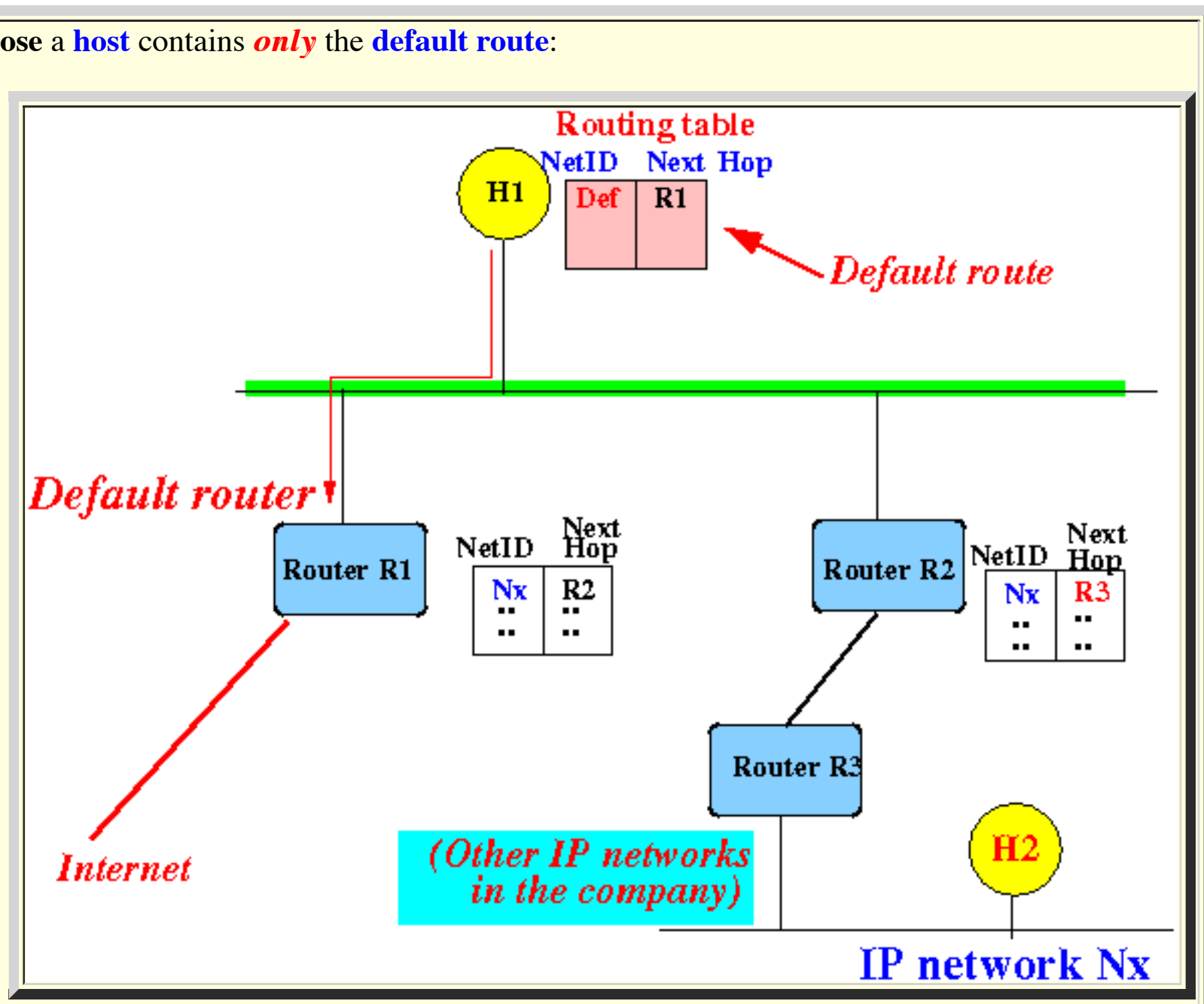
Example: `traceroute www.google.com`

```
cheung@shangchun(1146)> traceroute www.google.com
traceroute: Warning: www.google.com has multiple addresses; using 74.125.45.99

traceroute to www.google.com (74.125.45.99), 30 hops max, 40 byte packets
 1  170.140.14.33 (170.140.14.33)  0.898 ms  0.442 ms  0.327 ms
 2  COX1-academic-ve1162.emory.net (170.140.0.9)  0.549 ms  0.463 ms  0.732 ms
 3  CMONT1-academic-ve1142.emory.net (170.140.0.6)  0.436 ms  0.416 ms  0.329 ms
 4  NDB1-academic-ve1262.emory.net (170.140.0.41)  0.578 ms  0.420 ms  0.463 ms
 5  FWNDB2-academic-ve3934.emory.net (170.140.0.169)  0.451 ms  0.456 ms  0.459 ms
 6  NDB1-academic-iborder-ve3944.emory.net (163.246.200.126)  1.214 ms  0.797 ms  0.714 ms
 7  BNDB1-eborder-ve3369.emory.net (170.140.0.1)  0.828 ms  0.690 ms  0.785 ms
 8  SoX-Internet2.emory.net (163.246.200.86)  1.377 ms  1.278 ms  1.190 ms
 9  74.125.48.33 (74.125.48.33)  1.575 ms  1.471 ms  1.797 ms
10  72.14.239.100 (72.14.239.100)  1.476 ms  1.741 ms  1.574 ms
11  72.14.232.213 (72.14.232.213)  1.943 ms  1.965 ms 209.85.254.241 (209.85.254.241)  2.193 ms
12  209.85.253.133 (209.85.253.133)  5.937 ms 209.85.253.141 (209.85.253.141)  2.355 ms  9.245 ms
13  yx-in-f99.1e100.net (74.125.45.99)  98.039 ms  53.444 ms  44.371 ms
```

ICMP: reporting a better route to the destination

- **Example of a better route**
  - **Consider** the following **scenario**:



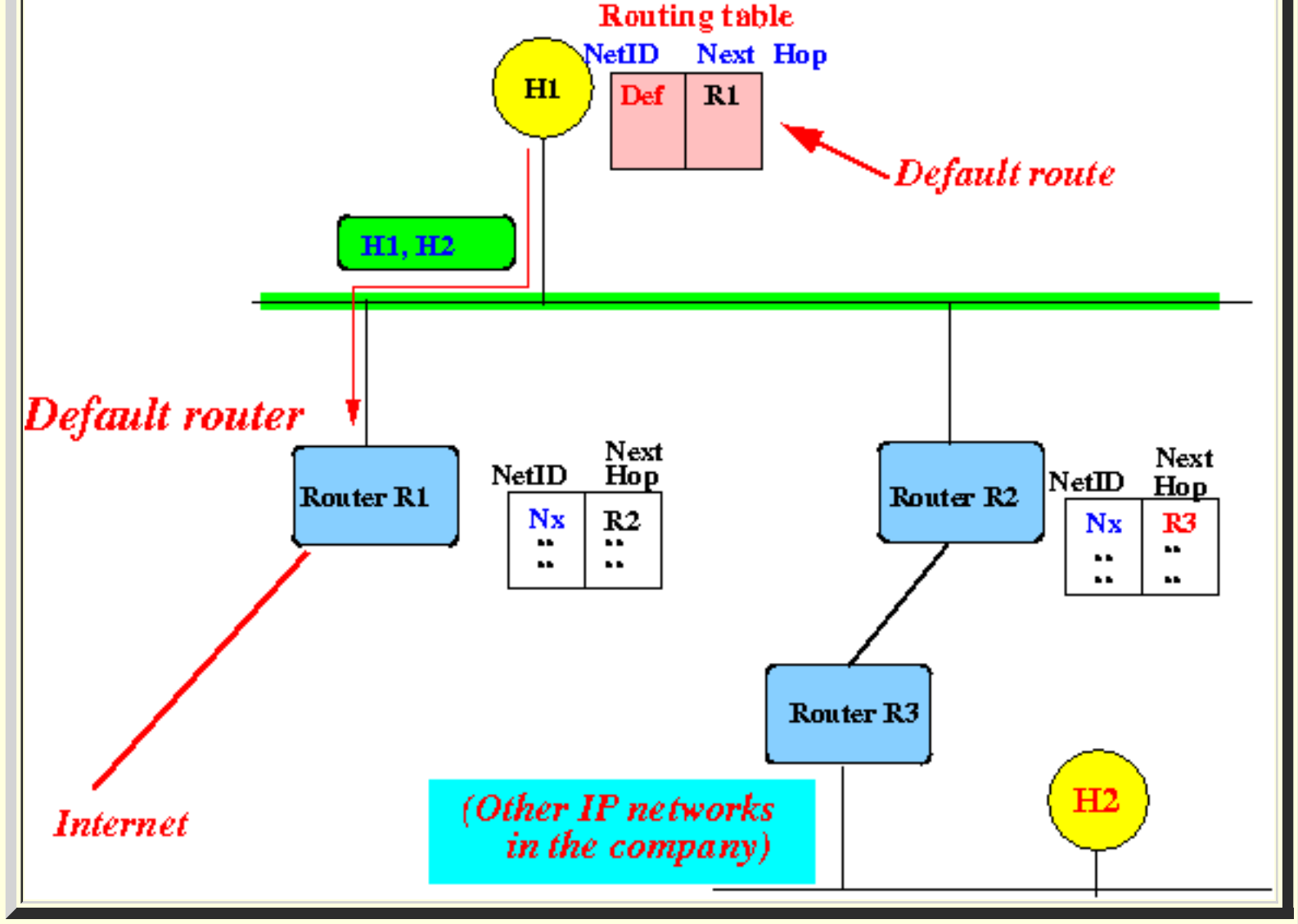
- **Suppose** a **host** contains ***only*** the **default route**:

But there are ***multiple*** **router (R1 and R2)** on the **IP network**

Specifically:

- **Another (internal) IP network** is **connected** to the **second router R2**

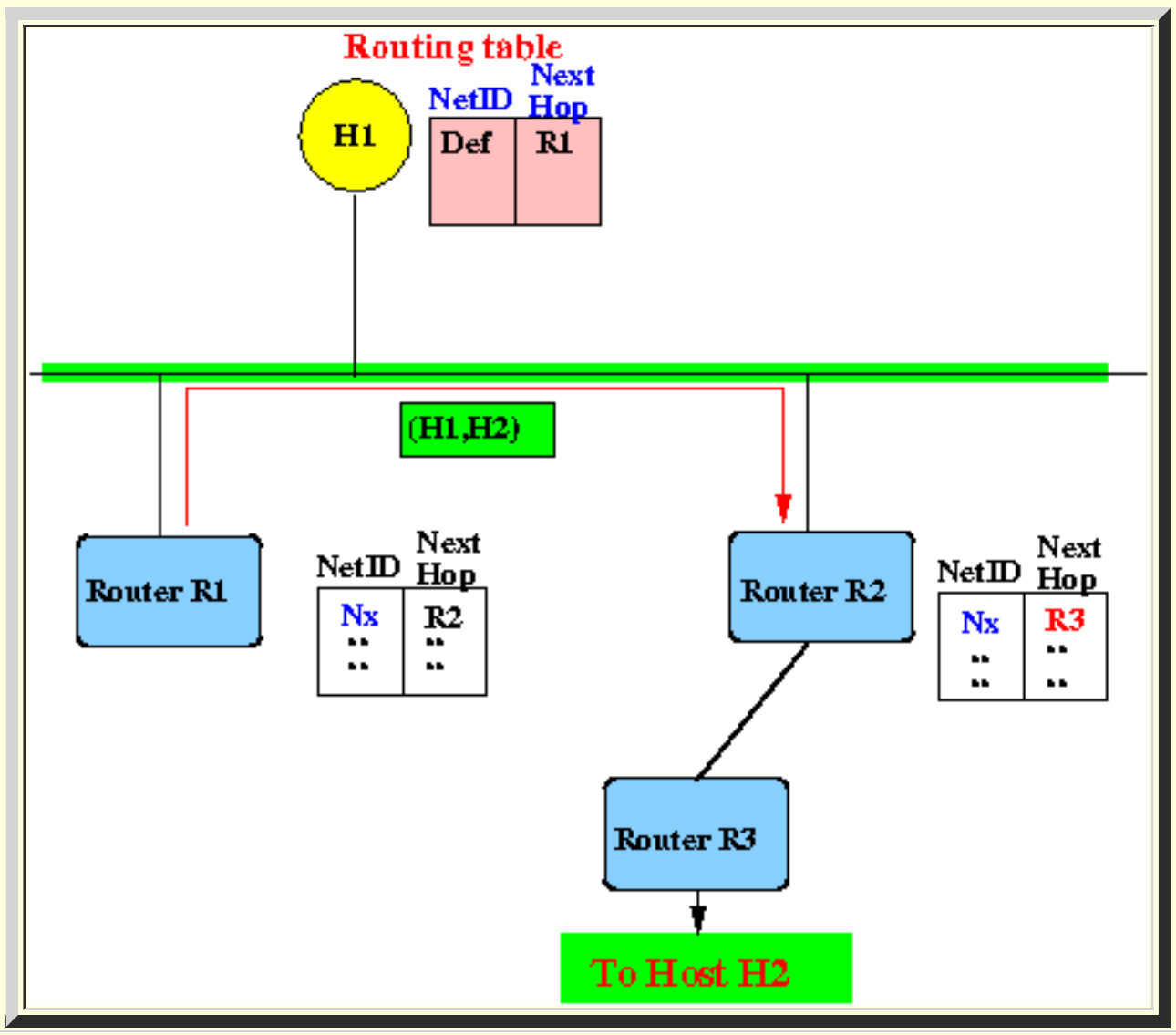
- **Suppose** he **host *H1*** sends a **packet** to the **host *H2*** on the ***other*** **IP network (network Nx)**:



According the **routing table** in **H1**:

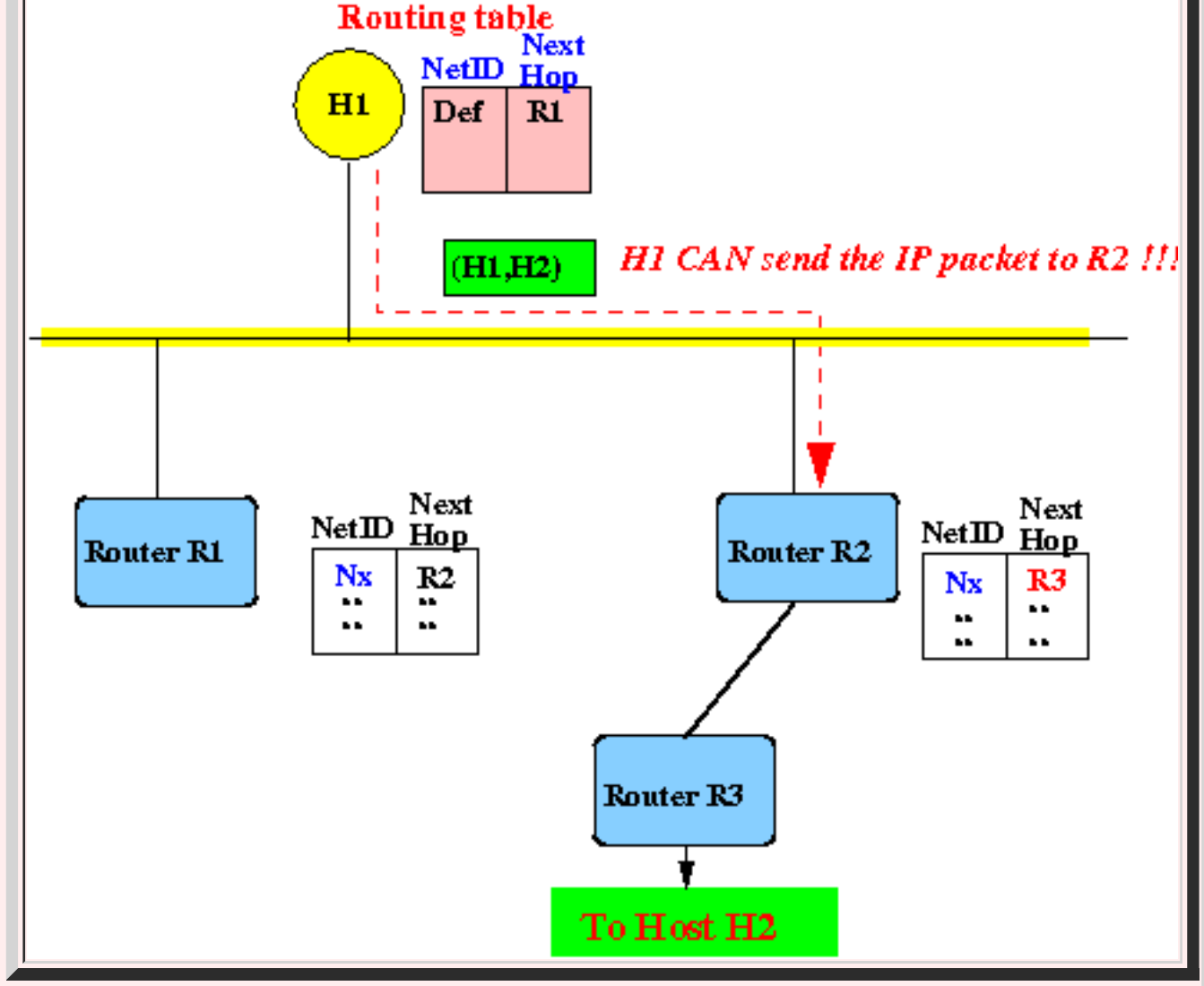
- The **IP packet** will be **transmitted** to **(default) router R1**

- The **router R1** forwards the packet **(H1,H2)** (**H2** is on network **Nx**) to **router R2**:



- **Better route:**

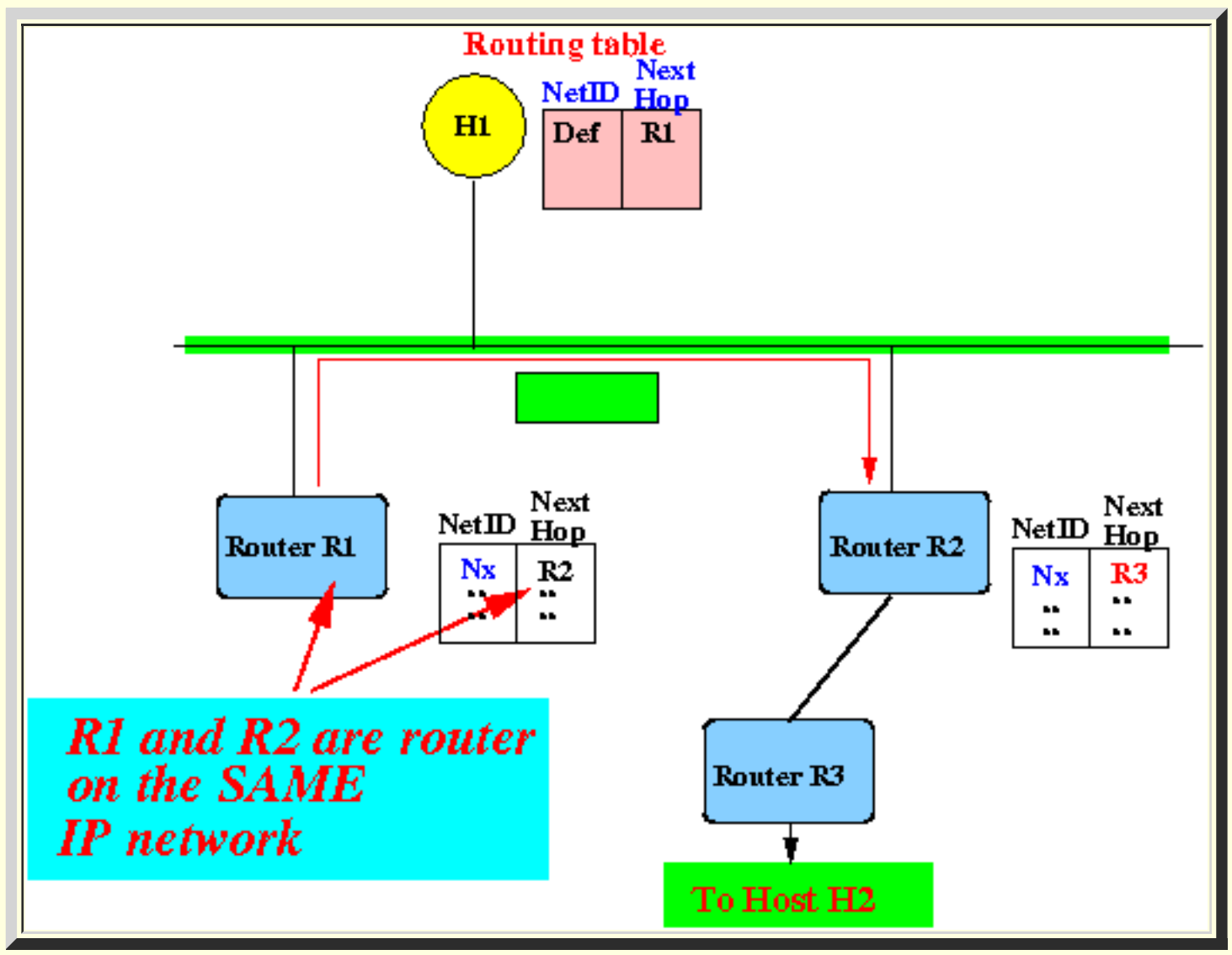
  - The host **H1** can **transmit** the **IP packet** to **Router R2** ***directly***:



- **Detecting a better route**
  - **How** to **detect** a ***better*** **route**:

    - When some **router R1 forwards** an **IP packet** to ***another*** **router** on the ***same*** **IP network**

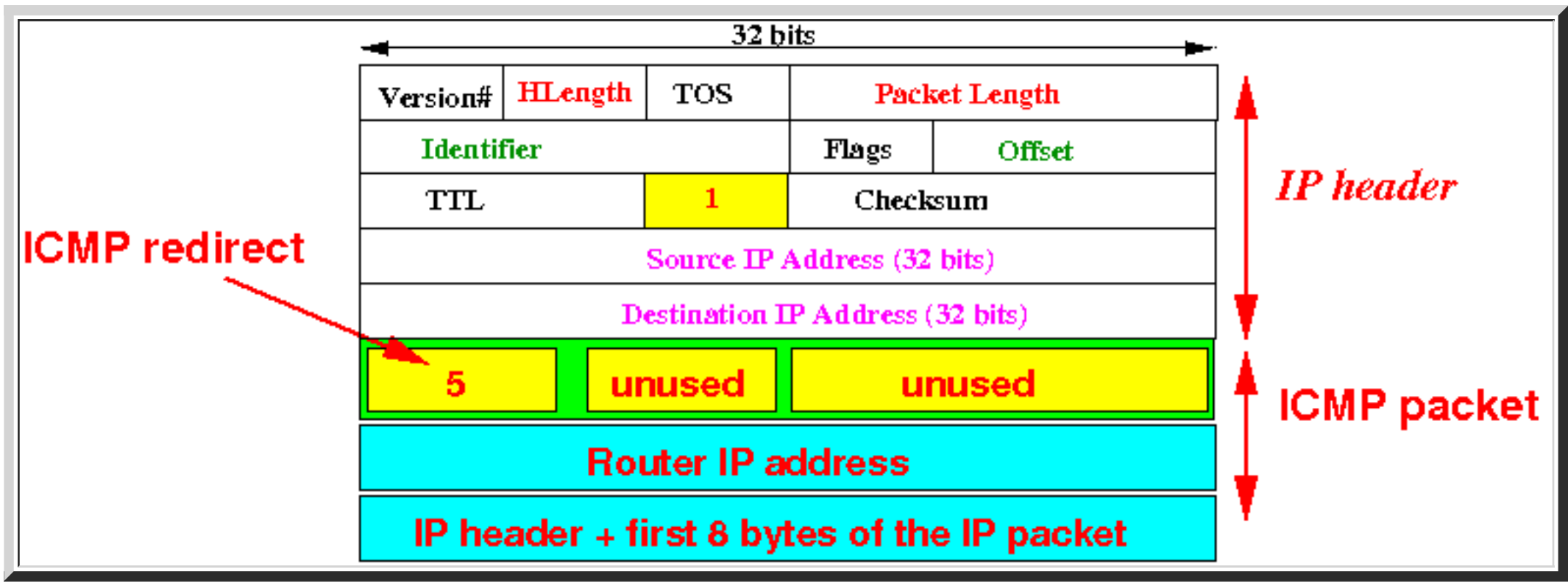
      Example:



(Clearly, the **transmission** should use the ***other*** **router** !!!)

- **Reporting a *better* route**
  - **Reporting** a ***better*** **route**:
    - When a **router** discovers a ***better*** **route**:
      - The **router** transmits an **ICMP *re-direct* control message** to the **source (sender)** of the **IP packet**

  - **Structure** of the **ICMP redirect** control message:



Comment:

- The **ICMP redirect** control message contains:
  - The **IP address** of the **"better" router** for the **IP packet** !!!

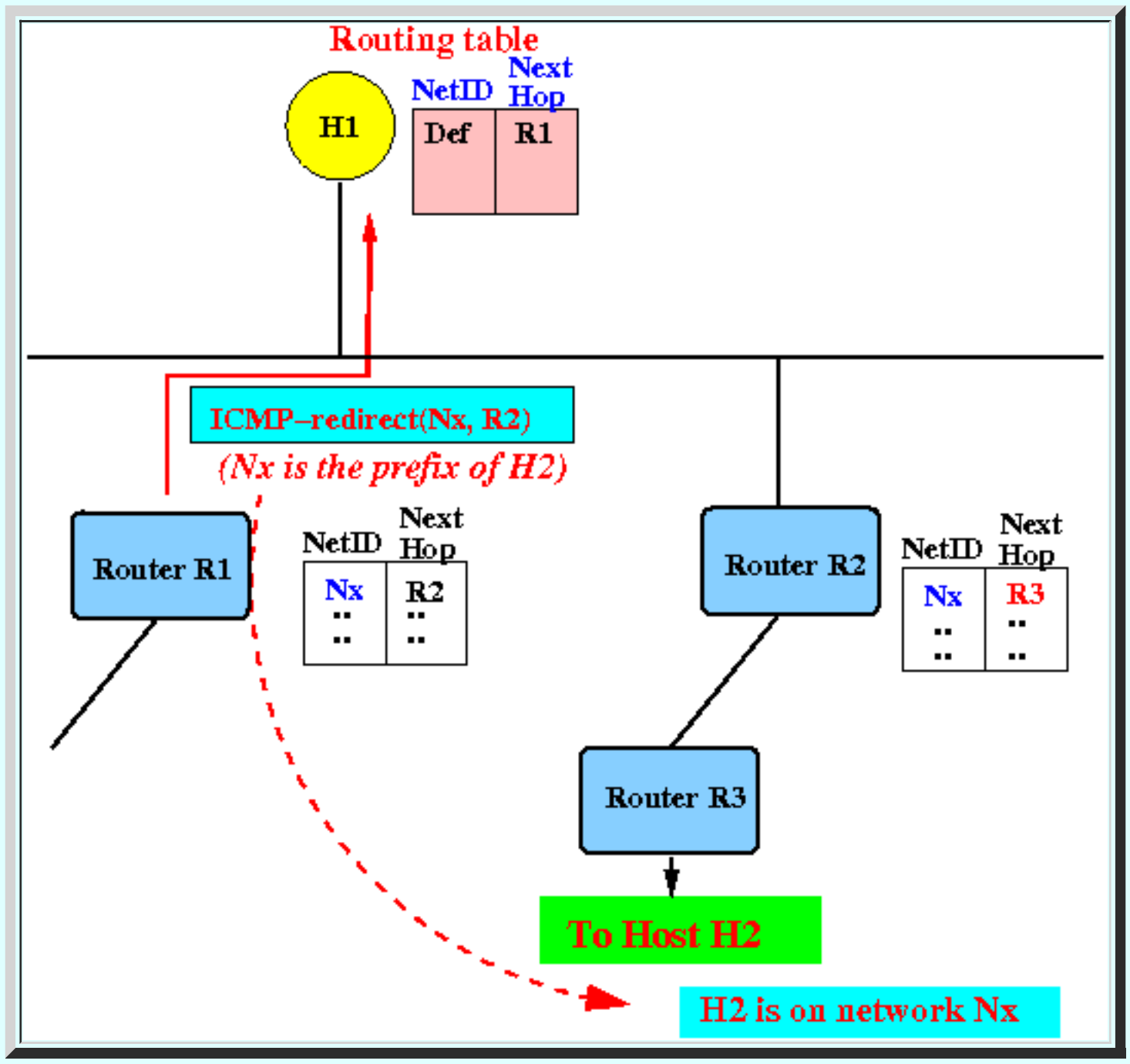
- ***Another*** **piece** of **information** that is **sent back** is:
  - The (**network ID**) of the **IP destination**

    (This **information** is in the `IP header` portion of the **ICMP packet**)

- **Processing an ICMP redirect control message**
  - **Processing** an **ICMP redirect** control message:
    - The **ICMP redirect message** will **insert** the **following** **routing entry** into the **routing table** of the **source**:
      - (**NetID** of the `destination IP dest`, **IP address** of the ***better*** **router**)

  - **Example:**

- The **destination (H1)** receives an **ICMP redirect** control message:



- The host **H1** will **insert** the **routing table** entry (**netID**, **R2**):

- **Result**:

- Now **host H1 now** has a **better route** to the **destination H2**

- ***Security Problems* with ICMP-redirect**
  - **Fact:**
    - The **ICMP redirect** messages pose **major security risks**...
  - But this topic (**security**) is **beyond the scope** of this **course**...

  - If you want to **explore** some **spoofing technqiues**, read this excellent article: **click here**

    (I have a local copy: **click here**)
