Communication Networks
Spring 2019

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
March 18 2019

Materials inspired from Scott Shenker & Jennifer Rexford
Please register your group for the routing project: https://bit.ly/2UBsnWw
Last week on Communication Networks
How do local computers communicate?
Communication Networks
Part 2: The Link Layer

#1 What is a link?

#2 How do we identify link adapters?

#3 How do we share a network medium?

#4 What is Ethernet?

#5 How do we interconnect segments at the link layer?
What is a link?

How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
MAC addresses...

identify the sender & receiver adapters
used within a link

are uniquely assigned
hard-coded into the adapter when built

use a flat space of 48 bits
allocated hierarchically
Why don’t we simply use IP addresses?

Links can support any protocol (not just IP) different addresses on different kind of links

Adapters may move to different locations cannot assign static IP address, it has to change

Adapters must be identified during bootstrap need to talk to an adapter to give it an IP address
You need to solve two problems when you bootstrap an adapter

Who am I?  How do I acquire an IP address?
MAC-to-IP binding

Who are you?  Given an IP address reachable on a link,
IP-to-MAC binding  How do I find out what MAC to use?
Who am I?  How do I acquire an IP address?
MAC-to-IP binding  Dynamic Host Configuration Protocol

Who are you?  Given an IP address reachable on a link, How do I find out what MAC to use?
IP-to-MAC binding  Address Resolution Protocol
What is a link?

How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Essentially, there are three techniques to deal with Multiple Access Control (MAC)

- **Divide the channel into pieces**
  either in time or in frequency

- **Take turns**
  pass a token for the right to transmit

- **Random access**
  allow collisions, detect them and then recover
This week on
Communication Networks
Link Layer

Network Layer

The End
The Local Area Networks we have considered so far define **single broadcast domains**

If one user broadcast a frame, every other user receives it.
As the network scales, network operators like to segment their LANs

Why?

- **Improves security**
  - smaller attack surface (visibility & injection)

- **Improves performance**
  - limit the overhead of broadcast traffic (e.g. ARP)

- **Improves logistics**
  - separates traffic by role (e.g. staff, students, visitors)
Organizational changes are too frequent to segment networks purely **physically**—rewiring is a major pain.

What about doing this in software though?
Enters “Virtual Local Area Networks” (VLANs)

Definition

A VLAN logically identifies a set of ports attached to one (or more) Ethernet switches, forming one broadcast domain.
A VLAN identifies a set of ports attached to one or more Ethernet switches.
Switches need configuration tables telling them which VLANs are accessible via which interfaces
Switches need configuration tables telling them which VLANs are accessible via which interfaces.

VLAN Staff: port A, port C
VLAN Student: port B
Consider that A sends a broadcast frame say, an ARP request

- **src_mac**: A
- **dst_mac**: ff:ff:ff:ff:ff:ff
- **payload**: Who has 10.0.1.1?
That frame should be received by all staff members: i.e. C and F, and *only* them

src_mac: A
dst_mac: ff:ff:ff:ff:ff:ff
payload: Who has 10.0.1.1?
src_mac: A
dst_mac: ff:ff:ff:ff:ff:ff
payload: Who has 10.0.1.1?

How does this switch know where to flood it next?
e.g. pointless to send it right
How does this switch know where to flood it next?

Here, *only* to F

**src_mac:** A  
**dst_mac:** ff:ff:ff:ff:ff:ff  
**payload:** Who has 10.0.1.1?
To identify VLAN, switches add new header when forwarding traffic to another switch.

**Without VLAN**

<table>
<thead>
<tr>
<th>preamble</th>
<th>dest address</th>
<th>src address</th>
<th>type</th>
<th>CRC</th>
</tr>
</thead>
</table>

**With VLAN**

802.1q Header (4 bytes)
(4 bits missing)

<table>
<thead>
<tr>
<th>preamble</th>
<th>dest address</th>
<th>src address</th>
<th>TPID</th>
<th>VID</th>
<th>type</th>
<th>data</th>
<th>CRC</th>
</tr>
</thead>
</table>

VLAN IDentifier
With VLANs, Ethernet links are divided in two sets: access and trunks (inter switches) links.
Access links belong to one VLAN
they do not carry 802.1q headers
Trunk links carry traffic for more than one VLAN and as such carry 801.1q tagged frames.
Each switch runs one MAC learning algorithm for each VLAN

When a switch receives a frame with an unknown or a broadcast destination,
it forwards it over all the ports that belong to the same VLAN

When a switch learns a source address on a port
it associates it to the VLAN of this port and only uses it when forwarding frames on this VLAN
src_mac: A
dst_mac: ff:ff:ff:ff:ff:ff
VID: 
payload: Who has 10.0.1.1?

VLAN “Staff”
port 1: A, C  port 2: F

VLAN “Student”
port 1: B  port 2: D,G  port 3: H, I, J
Switches can also compute per-VLAN spanning-tree allowing a distinct SPT for each VLAN allow the operators to use more of their links
root switch for all VLANS

switch 1

switch 2

switch 3

switch 4

switch 5

switch 6

switch 7

deactivated link for all VLANS
Any communication between the red hosts on switch 5 and 6 need to go via switch 1…
root switch for VLAN

switch 1

switch 2

switch 3

switch 4

switch 5

switch 6

switch 7

deactivated link for
switch 1

switch 3

switch 4

switch 2

switch 5

switch 6

switch 7

deactivated link for

root switch for VLAN
Now any communication between the red hosts on switch 5 and 6 go via the direct link.
The Beginning
Moving on to IP and the network layer
Internet Protocol and Forwarding

1. IP addresses
   use, structure, allocation

2. IP forwarding
   longest prefix match rule

3. IP header
   IPv4 and IPv6, wire format

source: Boardwatch Magazine
Internet Protocol and Forwarding

1. IP addresses
   use, structure, allocation

2. IP forwarding
   longest prefix match rule

3. IP header
   IPv4 and IPv6, wire format
IPv4 addresses are unique 32-bits number associated to a network interface (on a host, a router, ...)

IP addresses are usually written using dotted-quad notation

```
01010010 10000010 01100110 00001010
```

```
82.130.102.10
```

```
01010010 10000010 01100110 00001010
```
Routers forwards IP packets based on their destination IP address
If IP addresses were assigned arbitrarily, routers would require forwarding entries for all of them.

LAN 1: 1.2.3.4, 5.6.7.8, 2.4.6.8

LAN 2: 1.2.3.5, 5.6.7.9, 2.4.6.9

WAN 1: IP router

WAN 2: IP router

Forwarding table:

- 1.2.3.4
- 1.2.3.5
- ...

LAN: Local Area Network

WAN: Wide Area Network
18 billion

estimated* # of Internet connected devices in 2017

* Cisco Visual Networking Index 2017—2022
28.5 billion

estimated* # of Internet connected devices in 2022

* Cisco Visual Networking Index 2017—2022
Two universal tricks you can apply
to any computer sciences problem

When you need... more flexibility,
you add... a layer of indirection

When you need... more scalability,
you add... a hierarchical structure
When you need... more scalability,
you add... a hierarchical structure
IP addresses are hierarchically allocated, similarly to the postal service.

Address

Zip 8092
Street Gloriastrasse
Building 35 (ETZ)
Location G 90
in building

Name Laurent Vanbever
Nobody in the Swiss mail system knows where every single house or building is.

Routing tables are separated at each level of the hierarchy, each one with a manageable scale.
Forwarding in the Swiss mail
in 4 steps

1. Deliver the letter to the post office responsible for the zip code
2. Assign letter to the mail person covering the street
3. Drop letter into the mailbox attached to the building
4. Hand in the letter to the appropriate person
IP addressing is hierarchical, composed of a prefix (network address) and a suffix (host address)
Each prefix has a given length, usually written using a “slash notation”

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>82.130.102.0 /24</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix length (in bits)</td>
<td></td>
</tr>
</tbody>
</table>
Here, a /24 means that we have 8 bits left to address hosts address, enough for 256 hosts.

<table>
<thead>
<tr>
<th>Prefix part</th>
<th>Host part</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010010.10000010.01100110.</td>
<td>00000000</td>
<td>82.130.102.0</td>
</tr>
<tr>
<td>01010010.10000010.01100110.</td>
<td>00000001</td>
<td>82.130.102.1</td>
</tr>
<tr>
<td>01010010.10000010.01100110.</td>
<td>00000010</td>
<td>82.130.102.2</td>
</tr>
<tr>
<td>01010010.10000010.01100110.</td>
<td>11111110</td>
<td>82.130.102.254</td>
</tr>
<tr>
<td>01010010.10000010.01100110.</td>
<td>11111111</td>
<td>82.130.102.255</td>
</tr>
</tbody>
</table>
In practice, the first and last IP address of a prefix are not usable

<table>
<thead>
<tr>
<th>prefix part</th>
<th>host part</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010010.10000010.01100110.</td>
<td>00000000</td>
<td>82.130.102.0</td>
</tr>
<tr>
<td>01010010.10000010.01100110.</td>
<td>11111111</td>
<td>82.130.102.255</td>
</tr>
</tbody>
</table>
The address with the host part being all 0s identifies the network itself

<table>
<thead>
<tr>
<th>prefix part</th>
<th>host part</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010010.10000010.01100110.</td>
<td>00000000</td>
<td>82.130.102.0</td>
</tr>
</tbody>
</table>
The address with the host part being all 1s identifies the broadcast address

<table>
<thead>
<tr>
<th>prefix part</th>
<th>host part</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010010.10000010.01100110.</td>
<td>11111111</td>
<td>82.130.102.255</td>
</tr>
</tbody>
</table>
A /24 has therefore only **254 addresses** that can be allocated to hosts
Prefixes are also sometimes specified using an address and a mask

<table>
<thead>
<tr>
<th>Address</th>
<th>82.130.102.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01010010.10000010.01100110.00000000</td>
</tr>
<tr>
<td></td>
<td>11111111.11111111.11111111.00000000</td>
</tr>
<tr>
<td>Mask</td>
<td>255.255.255.0</td>
</tr>
</tbody>
</table>
ANDing the address and the mask gives you the prefix

<table>
<thead>
<tr>
<th>Address</th>
<th>82.130.102.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01010010.10000010.01100110.00000000</td>
</tr>
<tr>
<td>Mask</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td></td>
<td>11111111.11111111.11111111.00000000</td>
</tr>
</tbody>
</table>
Given this IP prefix 82.130.0.0/17

Compute

# of addressable hosts

the prefix mask

network address

1st host address

last host address

broadcast address
Routers forward packet to their destination according to the network part, \textit{not} the host part
Doing so enables to scale the forwarding tables.
Hierarchical addressing enables to add new hosts without changing or adding forwarding rules.
Originally, there were only 5 fixed allocation sizes, (or classes)—known as classful networking

<table>
<thead>
<tr>
<th>Class</th>
<th>Leading bits</th>
<th>Prefix length</th>
<th># Hosts</th>
<th>Start address</th>
<th>End address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>8</td>
<td>$2^{24}$</td>
<td>0.0.0.0</td>
<td>127.255.255.255</td>
</tr>
<tr>
<td>Class B</td>
<td>10</td>
<td>16</td>
<td>$2^{16}$</td>
<td>128.0.0.0</td>
<td>191.255.255.255</td>
</tr>
<tr>
<td>Class C</td>
<td>110</td>
<td>24</td>
<td>$2^{8}$</td>
<td>192.0.0.0</td>
<td>223.255.255.255</td>
</tr>
<tr>
<td>Class D</td>
<td>1110</td>
<td></td>
<td></td>
<td>224.0.0.0</td>
<td>239.255.255.255</td>
</tr>
<tr>
<td>Multicast</td>
<td>1110</td>
<td></td>
<td></td>
<td>224.0.0.0</td>
<td>239.255.255.255</td>
</tr>
<tr>
<td>Class E</td>
<td>1111</td>
<td></td>
<td></td>
<td>240.0.0.0</td>
<td>255.255.255.255</td>
</tr>
</tbody>
</table>
Classful networking was quite wasteful leading to IP address exhaustion.

**Problem**
Class C was too small, so everybody requested class B which where:

1. too big and
2. too few (wasted space)

**Solution**
Classless Inter-Domain Routing (CIDR) introduced in 1993.
CIDR enabled flexible division between network and hosts addresses

CIDR must specify both the address and the mask
classful was communicating this in the first address bits

Masks are carried by the routing algorithms
it is *not* implicitly carried in the address
Say that an organization needs 500 addresses...

with... it gets a... leading to a waste of...

classful class B (/16) 99%

CIDR /23 (=2 class C’s) 2%

With CIDR, the max. waste is bounded to 50% (why?)
Today, addresses are allocated in contiguous chunks
As of now, the Internet has around 710,000 IPv4 prefixes

source http://www.cidr-report.org/
The allocation process of IP address is also hierarchical.
The root is held by Internet Corporation for Assigned Names and Numbers, aka ICANN
ICANN allocates large prefixes blocks to Regional Internet Registries (RIRs)
RIRs allocate parts of these prefixes blocks to Internet Service Providers (ISPs) and large institutions.
ISPs and large institutions may, in turn, allocate even smaller prefixes to their own customers
<table>
<thead>
<tr>
<th>Organization</th>
<th>Prefix</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICANN</td>
<td>82.0.0.0/8</td>
<td>01010010</td>
</tr>
<tr>
<td>RIPE</td>
<td>82.130.64.0/18</td>
<td>010100101000001001</td>
</tr>
<tr>
<td>ETHZ</td>
<td>82.130.102.0/23</td>
<td>01010010100000100110011011</td>
</tr>
<tr>
<td>ITET/TIK</td>
<td>82.130.102.254</td>
<td>0101001010000010011001101110111110</td>
</tr>
<tr>
<td></td>
<td>IP Address</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>--------------------</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>82.130.64.0/18</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>129.132.0.0/16</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>148.187.192.0/19</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>195.176.96.0/19</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>192.33.87.0/24</td>
<td>10</td>
</tr>
</tbody>
</table>
Internet Protocol and Forwarding

IP addresses
use, structure, allocation

2

IP forwarding
longest prefix match rule

IP header
IPv4 and IPv6, wire format
What’s inside an IP router?
Linecards (input)  

Route/Control Processor  

Interconnect (Switching) Fabric  

Linecards (output)  

Processes packets on their way in  

Processes packets before they leave  

Transfers packets from input to output ports  

Input and Output for the same port are on one physical linecard
(1) Implement IGP and BGP protocols; compute routing tables

(2) Push forwarding tables to the line cards
Route/Control Processor

Constitutes the control plane

Linecards (input)

1
2
N

Interconnect (Switching) Fabric

Constitutes the data plane

Linecards (output)

1
2
N
Routers maintain forwarding entries for each Internet prefix
Provider 2’s Forwarding table

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.1.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.2.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
</tbody>
</table>
Let’s say a packet for 129.0.1.1 arrives at Provider 2.

Provider 2’s Forwarding table:

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.1.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.2.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
</tbody>
</table>
When a router receives an IP packet, it performs an IP lookup to find the matching prefix.
Let’s say a packet for **129.0.1.1** arrives at Provider 2

> Provider 2 forwards it to IF#2
CIDR makes forwarding harder though, as one packet can match many IP prefixes
Let’s say a packet for 129.133.0.1 arrives at Provider 2.

Provider 2’s Forwarding table:

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.1.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.2.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
</tbody>
</table>
Let's say a packet for **129.133.0.1** arrives at Provider 2

**We have two matches!**
To resolve ambiguity, forwarding is done along the *most specific* prefix (*i.e.*, the longer one)
Let's say a packet for **129.133.0.1** arrives at Provider 2

> Provider 2 forwards it to IF#3
Could we do something better than maintaining one entry per prefix? **Yep!**
A child prefix can be filtered from the table whenever it shares the same output interface as its parent.

Routing Table

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.1.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.2.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Routing Table

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.1.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.132.2.0/24</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Routing Table

<table>
<thead>
<tr>
<th>IP prefix</th>
<th>Output Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>129.0.0.0/8</td>
<td>IF#2</td>
</tr>
<tr>
<td>129.133.0.0/16</td>
<td>IF#3</td>
</tr>
</tbody>
</table>

Exactly the same forwarding as before
Check out www.route-aggregation.net, to see how filtering can be done automatically.
Internet Protocol and Forwarding

IP addresses
use, structure, allocation

IP forwarding
longest prefix match rule

3
IP header
IPv4 and IPv6, wire format
Here is what an IPv4 packet look like on a wire
<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>4</td>
</tr>
<tr>
<td>header length</td>
<td>4</td>
</tr>
<tr>
<td>Type of Service</td>
<td>8</td>
</tr>
<tr>
<td>Total Length</td>
<td>16</td>
</tr>
<tr>
<td>Identification</td>
<td>32</td>
</tr>
<tr>
<td>Flags</td>
<td>3</td>
</tr>
<tr>
<td>Fragment offset</td>
<td>13</td>
</tr>
<tr>
<td>Time To Live</td>
<td>32</td>
</tr>
<tr>
<td>Protocol</td>
<td>32</td>
</tr>
<tr>
<td>Header checksum</td>
<td>32</td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
The version number tells us what other fields to expect, typically it is set to “4” for IPv4, or “6” for IPv6.

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td>Destination IP address</td>
<td>Options (if any)</td>
</tr>
</tbody>
</table>
The header length denotes the number of 32-bits word in the header, typically set to 5 (20 bytes header)

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment offset</td>
<td></td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The ToS allows different packets to be treated differently, e.g., low delay for voice, high bandwidth for video

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Type of Service</td>
</tr>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment offset</td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The total length denotes the # of bytes in the entire packet, with a maximum of 65 535 bytes.

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th><strong>Total Length</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flags 3</td>
<td>Fragment offset 13</td>
</tr>
<tr>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
</tr>
<tr>
<td></td>
<td>Source IP address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination IP address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options (if any)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The next three fields are used when packets get fragmented.

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
Every link in the Internet has a Maximum Transmission Unit (MTU)

MTU is the max. # of bytes a link can carry as one unit

*e.g.*, 1500 bytes for normal Ethernet

A router can fragment a packet if the outgoing link MTU is smaller than the total packet size

Fragmented packets are recomposed at the destination

why not in the network?
Assume Alice is sending 4000B packets to Bob, who is connected to a 1500B MTU link.
Because the packet is larger than the MTU, router B will split the packet into fragments.
The Identification header uniquely identify the fragments of a particular packet

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fragment offset 13</td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td></td>
<td>Header checksum</td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The fragment offset is used to put back the fragments in the right order in case of reordering

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Flags</td>
<td>Fragment offset</td>
</tr>
<tr>
<td>Identification</td>
<td>3</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
</tr>
</tbody>
</table>

- Source IP address
- Destination IP address
- Options (if any)
- Payload
The flags is used to tell whether there are more fragments coming or not

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Flags 3</td>
<td>Fragment offset 13</td>
</tr>
<tr>
<td>Identification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
</tr>
</tbody>
</table>

Source IP address

Destination IP address

Options (if any)

Payload
The TTL is used to identify packets trapped in a loop, and eventually discard them.

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Fragment offset 13</td>
</tr>
<tr>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
TTL is decremented by 1 at each router, the packet is discarded if it reaches 0

default TTL values

*nix (Linux/Mac) 64
Windows 128 (used for OS fingerprinting)
The protocol field identifies the higher level protocol carried in the packet, “6” for TCP, “17” for UDP

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fragment offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Header checksum</td>
</tr>
<tr>
<td>Source IP address</td>
<td>Protocol</td>
<td>Destination IP address</td>
<td>Options (if any)</td>
</tr>
<tr>
<td>Payload</td>
<td>Protocol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The protocol field identifies the higher level protocol carried in the packet, “6” for TCP, “17” for UDP
The checksum is the sum of all the 16 bits words in the header (does not protect the payload)

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags</td>
<td>Fragment offset</td>
<td></td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td><strong>Header checksum</strong></td>
<td></td>
</tr>
<tr>
<td>Source IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The source and destination IP uniquely identifies the source and destination host

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Flags 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time To Live</td>
<td>Protocol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination IP address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
Options were initially put to provide additional flexibility. For security reasons, there are often deactivated.

<table>
<thead>
<tr>
<th>version</th>
<th>header length</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Flags 3</td>
<td>Fragment offset 13</td>
<td></td>
</tr>
<tr>
<td>Time To Live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
</tr>
</tbody>
</table>

Source IP address

Destination IP address

Options (if any)

Payload
IP options

- Record route
- Strict source route
- Loose source route
- Timestamp
- Traceroute
- Router alert

see http://www.networksorcery.com/enp/protocol/ip.htm#Options for a full list
While there are no new IPv4 available, IPv4 still accounts for most of the Internet traffic (for now)
With respect to IPv4, IPv6 is simpler

IPv6 was motivated by address exhaustion
IPv6 addresses are 128 bits long, that’s plenty!

IPv6 got rid of anything that wasn’t necessary
spring cleaning

Result is an elegant, if unambitious, protocol
With respect to IPv4, IPv6 is simpler

<table>
<thead>
<tr>
<th>IPv6</th>
<th>removed</th>
<th>reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>fragmentation</td>
<td></td>
<td>leave problems to the end host</td>
</tr>
<tr>
<td>checksum</td>
<td></td>
<td>simplify handling</td>
</tr>
<tr>
<td>header length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

added...

| new options mechanism | simplify handling |
| expanded addresses    |                     |
| flow label            | flexibility         |
IPv4 vs IPv6

IPv4 Header vs IPv6 Header

Legend:
- Field’s name kept from IPv4 to IPv6
- Field not kept in IPv6
- Name and position changed in IPv6
- New field in IPv6

source  http://bit.ly/1HXc2BS
IPv6 enables to insert arbitrary options in the packet
see RFC 2460

source  http://bit.ly/1HXc2BS
The problem with IPv4 options is that all of them must be processed by each router, which is slow.
In IPv6, only one type of optional header must be processed by each router.
Internet Protocol and Forwarding

- IP addresses
  - use, structure, allocation

- IP forwarding
  - longest prefix match rule

- IP header
  - IPv4 and IPv6, wire format
Next week on

Communication Networks

Internet routing!