IP on Ethernet

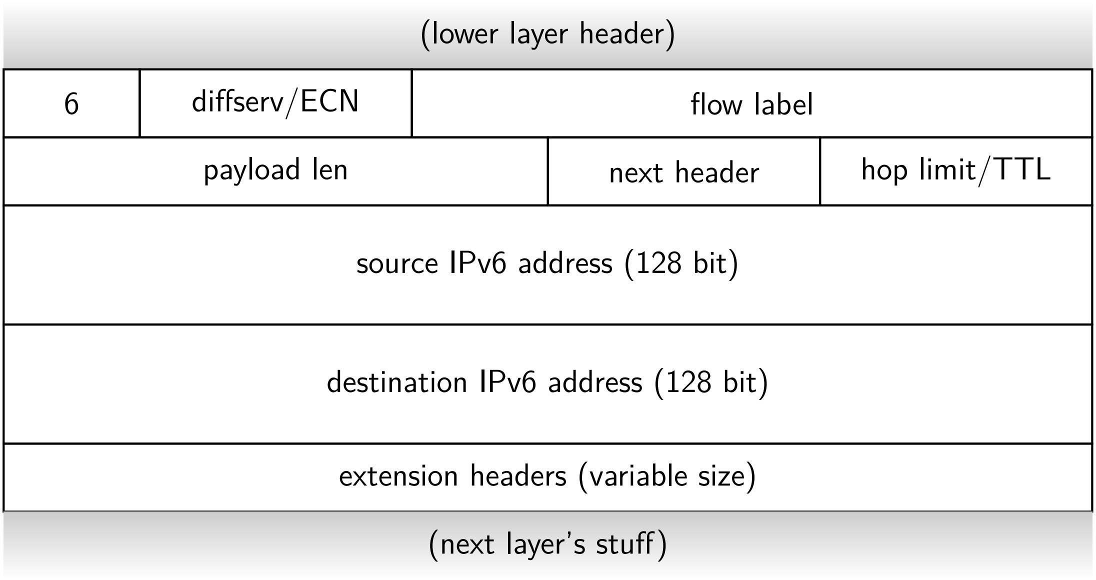
talked about how Ethernet works

but most Ethernet frames going to contain IP packets

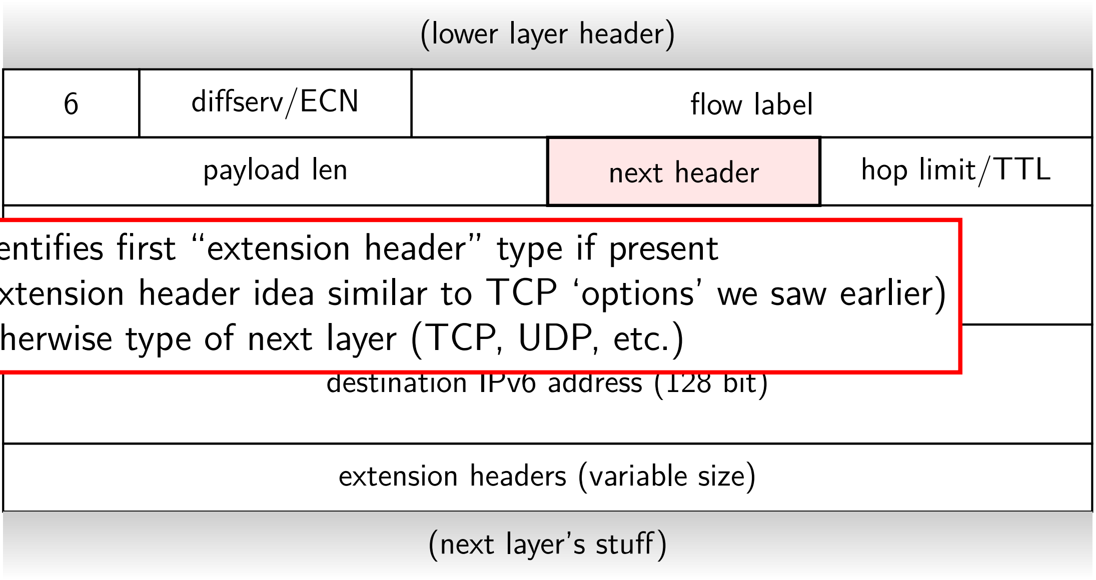
IP has its own idea of addresses

- used alongside MAC addresses

IPv6 packet format

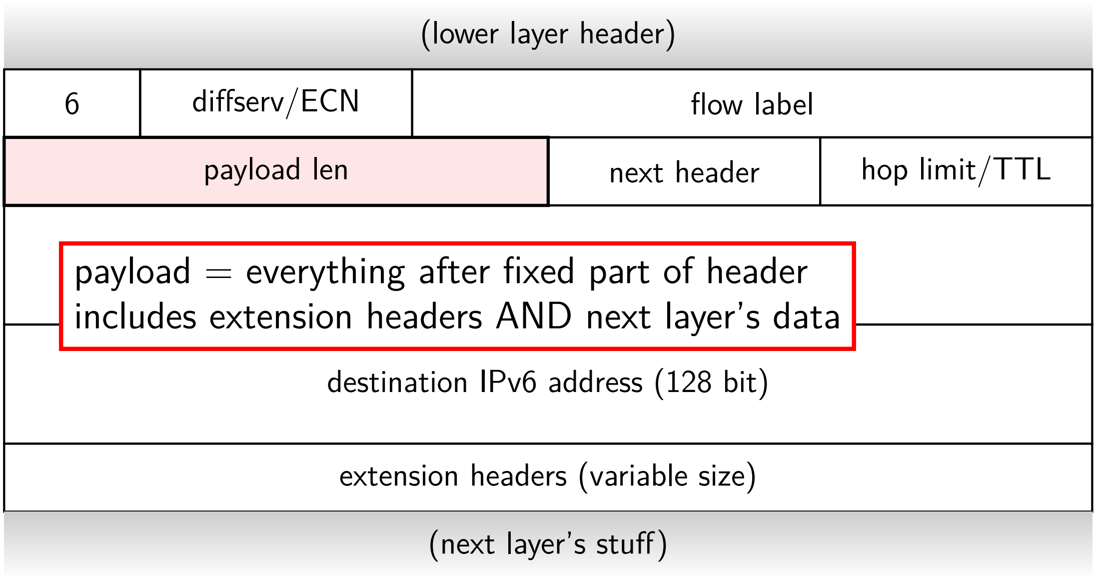


IPv6 packet format

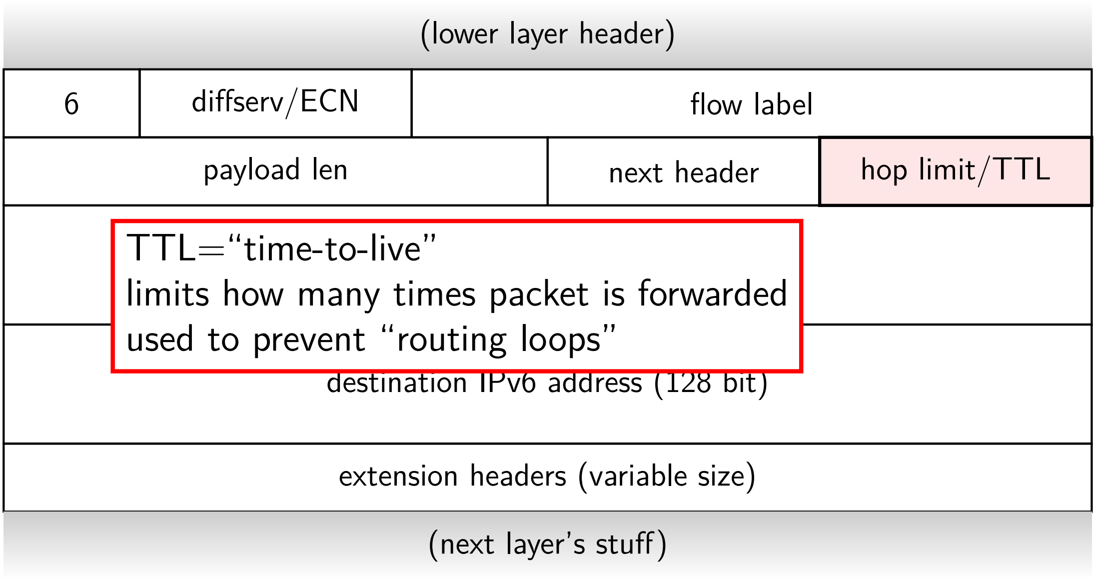


identifies first "extension header" type if present
(extension header idea similar to TCP 'options' we saw earlier)
otherwise type of next layer (TCP, UDP, etc.)

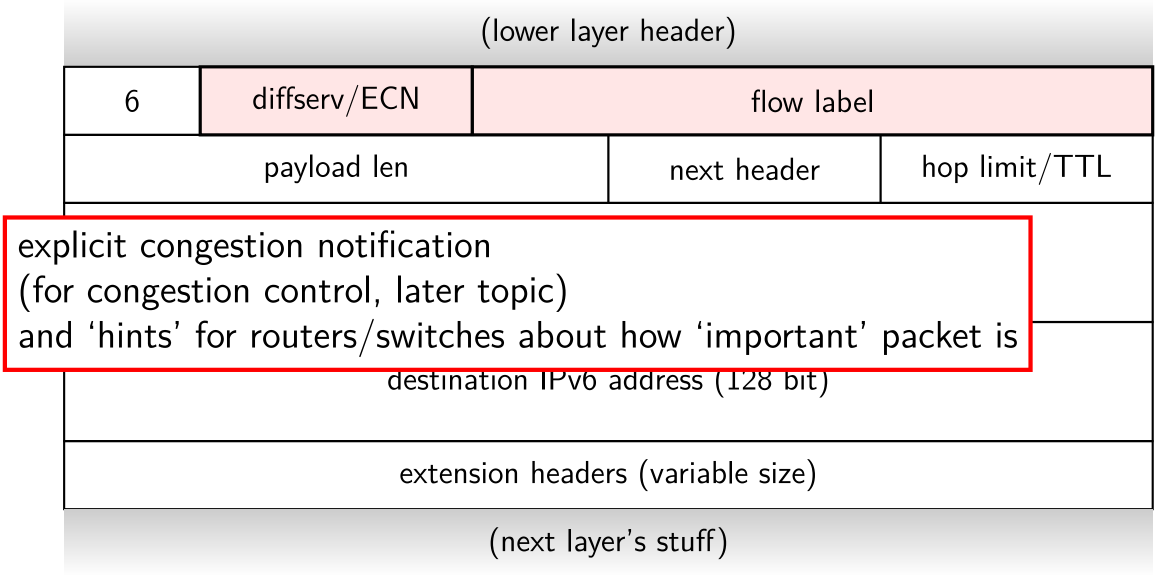
IPv6 packet format



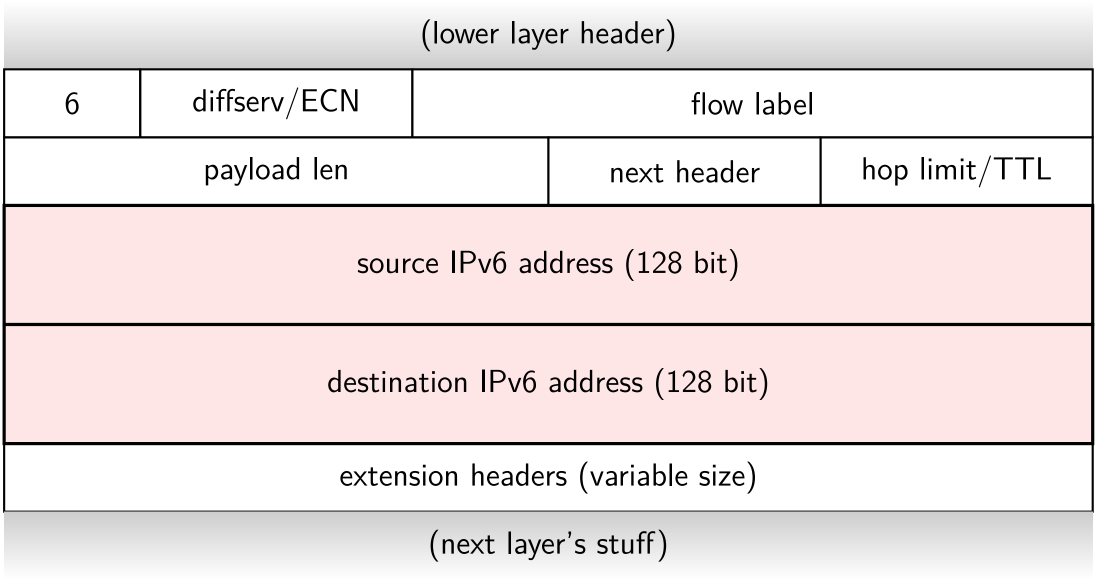
IPv6 packet format



IPv6 packet format



IPv6 packet format



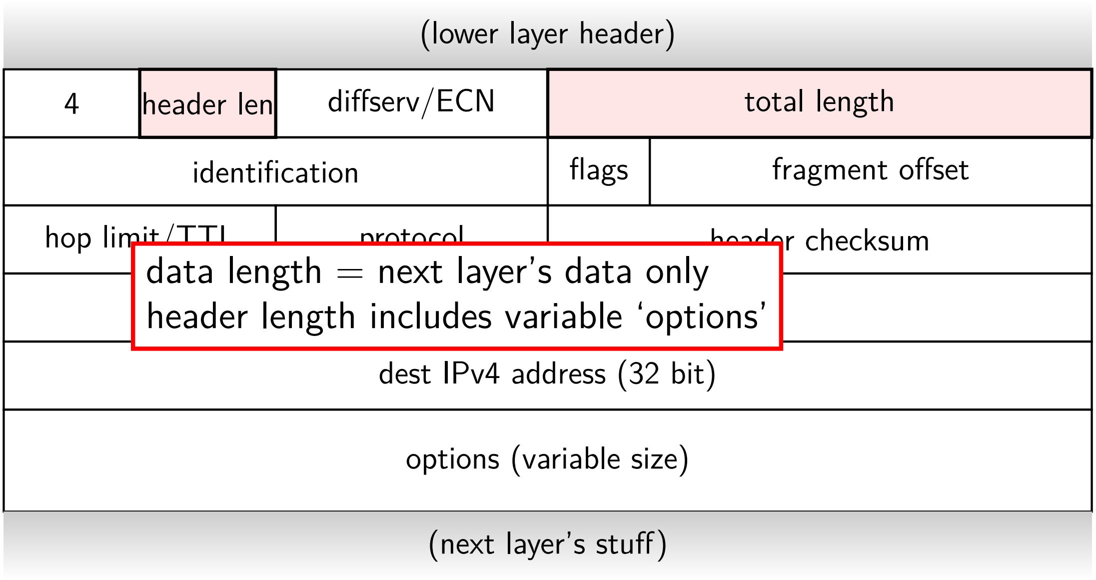
IPv4 packet format

(lower layer header)

<table>
<tr><td>4</td><td>header len</td><td>diffserv/ECN</td><td colspan="2">total length</td></tr>
<tr><td colspan="3">identification</td><td>flags</td><td>fragment offset</td></tr>
<tr><td colspan="2">hop limit/TTL</td><td>protocol</td><td colspan="2">header checksum</td></tr>
<tr><td colspan="5">source IPv4 address (32 bit)</td></tr>
<tr><td colspan="5">dest IPv4 address (32 bit)</td></tr>
<tr><td colspan="5">options (variable size)</td></tr>
</table>

(next layer's stuff)

IPv4 packet format



IPv4 packet format

(lower layer header)

4	header len	diffserv/ECN	total length	
identification			flags	fragment offset

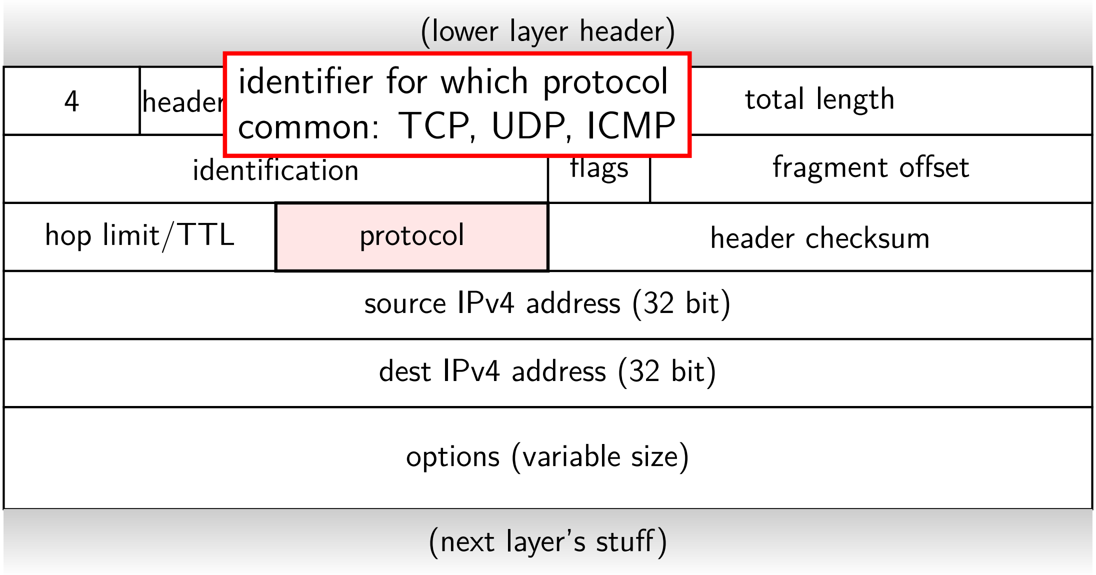
these fields part of support for “fragments”
where packet sent in multiple pieces
also exists in IPv6, but IPv6 uses extension headers for it
we’ll probably revisit this later

n

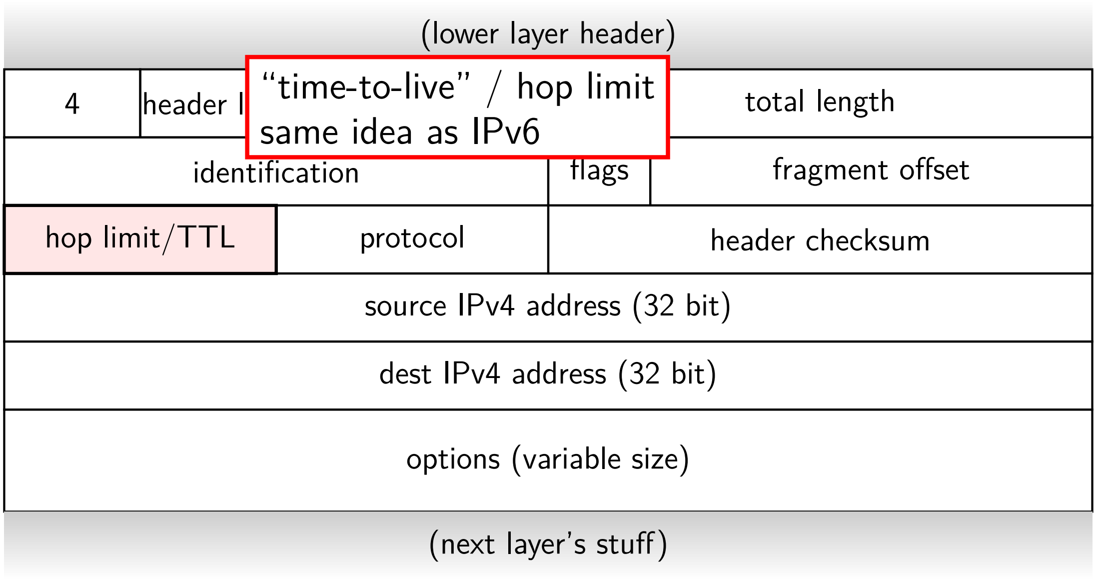
options (variable size)

(next layer’s stuff)

IPv4 packet format



IPv4 packet format



IPv4 packet format

(lower layer header)

checksum of header only
(TCP has own checksum because header isn't enough)
IPv6 got rid of IP-level checksums entirely

offset

hop limit/TTL	protocol	header checksum

source IPv4 address (32 bit)

dest IPv4 address (32 bit)

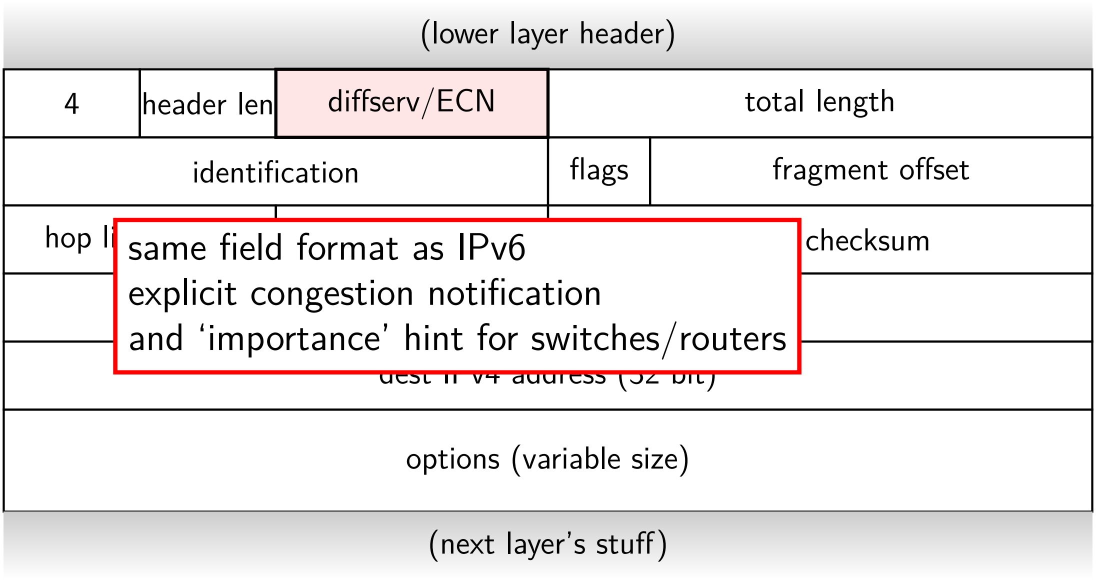
options (variable size)

(next layer's stuff)

IPv4 packet format

<table>
<tr><td colspan="8">(lower layer header)</td></tr>
<tr><td>4</td><td>header len</td><td colspan="2">diffserv/ECN</td><td colspan="4">total length</td></tr>
<tr><td colspan="4">identification</td><td>flags</td><td colspan="3">fragment offset</td></tr>
<tr><td colspan="2">hop limit/TTL</td><td colspan="2">protocol</td><td colspan="4">header checksum</td></tr>
<tr><td colspan="8">source IPv4 address (32 bit)</td></tr>
<tr><td colspan="8">dest IPv4 address (32 bit)</td></tr>
<tr><td colspan="8">options (variable size)</td></tr>
<tr><td colspan="8">(next layer's stuff)</td></tr>
</table>

IPv4 packet format



IPv4 addresses

32-bit numbers

typically written like 128.143.67.11

- four 8-bit decimal values separated by dots
- first part is most significant
- same as $128 \cdot 256^3 + 143 \cdot 256^2 + 67 \cdot 256 + 11 = 2\,156\,782\,459$

IPv4 address blocks

often will want to talk about group of IPv4 addresses

example: 128.143.67.64—128.143.67.127 (inclusive)

IPv4 address blocks

often will want to talk about group of IPv4 addresses

example: 128.143.67.64—128.143.67.127 (inclusive)

`10000000 10001111 01000011 00100000`

`10000000 10001111 01000011 00111111`

IPv4 address blocks

often will want to talk about group of IPv4 addresses

example: 128.143.67.64—128.143.67.127 (inclusive)

`10000000 10001111 01000011 00100000`

`10000000 10001111 01000011 00111111`

first 27 bits always same; anything for last bits

IPv4 address blocks

often will want to talk about group of IPv4 addresses

example: 128.143.67.64—128.143.67.127 (inclusive)

`10000000 10001111 01000011 00100000`

`10000000 10001111 01000011 00111111`

first 27 bits always same; anything for last bits

IPv4 address blocks

often will want to talk about group of IPv4 addresses

example: 128.143.67.64—128.143.67.127 (inclusive)

`10000000 10001111 01000011 00100000`

`10000000 10001111 01000011 00111111`

first 27 bits always same; anything for last bits

more convenient representation: 128.143.67.64/27

called "CIDR notation"

- CIDR = classless inter-domain routing (will come up when we discuss routing)

CIDR notation examples

5.7.3.3/14 = 5.4.0.0/14 = 5.4.0.0—5.7.255.255

128.143.0.0/16 = 128.143.0.0—128.143.255.255

192.168.0.0/24 = 192.168.0.0—192.168.0.255

10.0.0.0/8 = 10.0.0.0–10.255.255.255

CIDR notation examples

5.7.3.3/14 = 5.4.0.0/14 = 5.4.0.0—5.7.255.255

also written 5.4/14

128.143.0.0/16 = 128.143.0.0—128.143.255.255

also written 128.143/16

192.168.0.0/24 = 192.168.0.0—192.168.0.255

10.0.0.0/8 = 10.0.0.0–10.255.255.255

also written 10/8

alternate notation: netmasks

instead of writing 128.143.67.64/27 might say

128.143.67.64 and "network mask" of 255.255.255.224

255.255.255.224 = 27 1's

alternate notation: netmasks

instead of writing 128.143.67.64/27 might say

128.143.67.64 and "network mask" of 255.255.255.224

255.255.255.224 = 27 1's

if some-address bitwise-AND netmask = 128.143.67.64
bitwise-AND netmask,
then some-address is in the range

IPv6 addresses

IPv6 like IPv4, but with 128-bit numbers

written in hex, 16-bit parts, seperated by colons (`:`)

strings of 0s represented by double-colons (`::`)

typically given to users in blocks of 2^{80} or 2^{64} addresses

`2607:f8b0:400d:c00::6a` =

`2607:f8b0:400d:0c00:0000:0000:0000:006a`

`2607f8b0400d0c00000000000000006a`$_{\text{SIXTEEN}}$

IPv6 CIDR notation examples

2607:fb80:400d:0c00::/64 =

`2607:fb80:400d:0c00:0000:0000:0000:0000`—
`2607:fb80:400d:0c00:ffff:ffff:ffff:ffff`

2607:fb80::/30 =

`2607:fb80:0000:0000:0000:0000:0000:0000`—
`2607:fb83:ffff:ffff:ffff:ffff:ffff:ffff`

Internet Assigned Numbers Authority

IANA IPv4 Address Space Registry

Last Updated

2023-12-18

Registration Procedure(s)

Allocations to RIRs are made in line with the Global Policy published at [http://www.icann.org/en/resources/policy/global-addressing]. All other assignments require IETF Review.

Description

The allocation of Internet Protocol version 4 (IPv4) address space to various registries is listed here. Originally, all the IPv4 address spaces was managed directly by the IANA. Later parts of the address space were allocated to various other registries to manage for particular purposes or regional areas of the world. RFC 1466 [RFC1466] documents most of these allocations.

Reference

[RFC7249]

Available Formats

CSV XML HTML Plain text

Prefix	Designation	Date	WHOIS	RDAP	Status [1]	Note
000/8	IANA - Local Identification	1981-09			RESERVED	[2][3]
001/8	APNIC	2010-01	whois.apnic.net	https://rdap.apnic.net/	ALLOCATED	
002/8	RIPE NCC	2009-09	whois.ripe.net	https://rdap.db.ripe.net/	ALLOCATED	
003/8	Administered by ARIN	1994-05	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	

Prefix	Designation	Date	WHOIS	RDAP	Status [1]	Note
000/8	IANA - Local Identification	1981-09			RESERVED	[2][3]
001/8	APNIC	2010-01	whois.apnic.net	https://rdap.apnic.net/	ALLOCATED	
002/8	RIPE NCC	2009-09	whois.ripe.net	https://rdap.db.ripe.net/	ALLOCATED	
003/8	Administered by ARIN	1994-05	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
004/8	Administered by ARIN	1992-12	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
005/8	RIPE NCC	2010-11	whois.ripe.net	https://rdap.db.ripe.net/	ALLOCATED	
006/8	Army Information Systems Center	1994-02	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
007/8	Administered by ARIN	1995-04	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
008/8	Administered by ARIN	1992-12	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
009/8	Administered by ARIN	1992-08	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
010/8	IANA - Private Use	1995-06			RESERVED	[4]
011/8	DoD Intel Information Systems	1993-05	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
012/8	AT&T Bell Laboratories	1995-06	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	
013/8	Administered by ARIN	1991-09	whois.arin.net	https://rdap.arin.net/registry http://rdap.arin.net/registry	LEGACY	

(and 241 more)

Internet Assigned Numbers Authority

Internet Protocol Version 6 Address Space

Last Updated

2019-09-13

Note

The IPv6 address management function was formally delegated to IANA in December 1995 [RFC1881]. The registration procedure was confirmed with the IETF Chair in March 2010

...

IPv6 Prefix	Allocation	Reference	Notes
0000::/8	Reserved by IETF	[RFC3513] [RFC4291]	[1] [2] [3] [4] [5] [6]
0100::/8	Reserved by IETF	[RFC3513] [RFC4291]	0100::/64 reserved for Discard-Only Address Block [RFC6666]. Complete
0200::/7	Reserved by IETF	[RFC4048]	Deprecated as of December 2004 [RFC4048]. Formerly an OSI NSAP-map
0400::/6	Reserved by IETF	[RFC3513] [RFC4291]	
0800::/5	Reserved by IETF	[RFC3513] [RFC4291]	
1000::/4	Reserved by IETF	[RFC3513] [RFC4291]	
2000::/3	Global Unicast	[RFC3513] [RFC4291]	The IPv6 Unicast space encompasses the entire IPv6 address range with are currently limited to the IPv6 unicast address range of 2000::/3. IANA *address-assignments*]. [7] [8] [9] [10] [11] [12] [13] [14] [15]
4000::/3	Reserved by IETF	[RFC3513] [RFC4291]	

IPv6 Global Unicast Address Assignments

Last Updated

2024-07-23

Registration Procedure(s)

Allocations to RIRs are made in line with the Global Policy published at [http://www.icann.org/en/resources/policy/global-addressing]. All other assignments require IETF Review.

Description

The allocation of Internet Protocol version 6 (IPv6) unicast address space is here. References to the various other registries detailing the use of the IPv space can be found in the [IPv6 Address Space registry].

Reference

[RFC7249]

Note

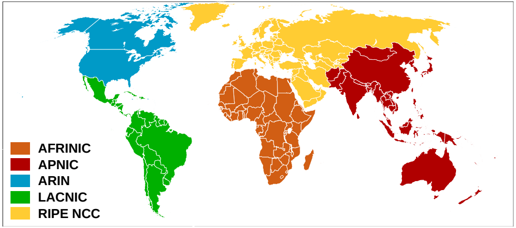
The assignable Global Unicast Address space is defined in [RFC3513] as the ad defined by the prefix 2000::/3. [RFC3513] was later obsoleted by [RFC4291]. A space in this block not listed in the table below is reserved by IANA for fut allocation.

Available Formats

CSV XML HTML Plain text

Prefix	Designation	Date	WHOIS	RDAP
2001:0000::/23	IANA	1999-07-01	whois.iana.org	
2001:0200::/23	APNIC	1999-07-01	whois.apnic.net	https://rdap.apnic.net/
2001:0400::/23	ARIN	1999-07-01	whois.arin.net	https://rdap.arin.net/regist http://rdap.arin.net/regist
2001:0600::/23	RIPE NCC	1999-07-01	whois.ripe.net	https://rdap.db.ripe.net/

regional internet registries (RIRs)



map from Wikimedia Commons,
users Dork, Canuckguy et al, Sémhur, CC-BY-SA 3.0

most useful addresses managed by RIRs

- African Network Information Centre (AFRINIC)
- American Registry for Internet Numbers (ARIN)
- Asia Pacific Network Information Centre (APNIC)
- Latin American and Carribean Network Information Centre (LACNIC)
- Réseaux IP Européens Network Coordination Centre (RIPE NCC)

RIR suballocations

ARIN
American Registry for Internet Numbers

SEARCH Whois*RWS*
all requests subject to *terms of use* *advanced search*

ARIN Online
enter

WHOIS-RWS

You searched for: **128.143.67.88**

Network	
Net Range	128.143.0.0 - 128.143.255.255
CIDR	128.143.0.0/16
Name	UNIVERSITY-OF-VIRGINIA
Handle	NET-128-143-0-0-1
Parent	NET128 (NET-128-0-0-0-0)
Net Type	Direct Allocation
Origin AS	AS225
Organization	University of Virginia (UVA)
Registration Date	1987-12-14
Last Updated	2021-12-14
Comments	University of Virginia

RELEVANT LINKS

- ARIN Whois/Whois-RWS Terms of Service
- Report Whois Inaccuracy
- Search ARIN Whois with RDAP

special IPv4 addresses

Internet Assigned Numbers Authority

IANA IPv4 Special-Purpose Address Registry

Created

2009-08-19

Last Updated

2021-02-04

Available Formats

special IPv4 addresses

Address Block	Name	RFC	Allocation Date	Termination Date	Source	Destination	For
0.0.0.0/8	"This network"	[RFC791], Section 3.2	1981-09	N/A	True	False	Fals
0.0.0.0/32	"This host on this network"	[RFC1122], Section 3.2.1.3	1981-09	N/A	True	False	Fals
10.0.0.0/8	Private-Use	[RFC1918]	1996-02	N/A	True	True	Tru
100.64.0.0/10	Shared Address Space	[RFC6598]	2012-04	N/A	True	True	Tru
127.0.0.0/8	Loopback	[RFC1122], Section 3.2.1.3	1981-09	N/A	False [1]	False [1]	Fals
169.254.0.0/16	Link Local	[RFC3927]	2005-05	N/A	True	True	Fals
172.16.0.0/12	Private-Use	[RFC1918]	1996-02	N/A	True	True	Tru
192.0.0.0/24 [2]	IETF Protocol Assignments	[RFC6890], Section 2.1	2010-01	N/A	False	False	Fals
192.0.0.0/29	IPv4 Service Continuity Prefix	[RFC7335]	2011-06	N/A	True	True	Tru
192.0.0.8/32	IPv4 dummy address	[RFC7600]	2015-03	N/A	True	False	Fals
192.0.0.9/32	Port Control Protocol Anycast	[RFC7723]	2015-10	N/A	True	True	Tru
192.0.0.10/32	Traversal Using Relays around NAT Anycast	[RFC8155]	2017-02	N/A	True	True	Tru
192.0.0.170/32, 192.0.0.171/32	NAT64/DNS64 Discovery	[RFC8880] [RFC7050], Section 2.2	2013-02	N/A	False	False	Fals
192.0.2.0/24	Documentation (TEST-	[RFC5737]	2010-01	N/A	False	False	Fals

selected special IP addresses

loopback (current machine) — `127/8` (v4), `::1/128` (v6)

link-local (current network only) —
    `169.254/16` (v4), `ff80::/10` (v6)

private use (non-public networks only) —
    `192.168/16`, `172.16/12`, `10/8` (v4), (kinda) `fc00::/7` (v6)

multicast groups and related — `224/4` (v4), `ff00::/8` (v6)
    multiple nodes can be part of a single "multicast group"

broadcast (all on current network) —
    `255.255.255.255`, `ff01::1`

"future use" —
    rest of `240/4` (v4), `4000::`—`efff::` (v6)

which link local?

“link local”: `169.254/16`, `fe80::/10`

specific to each local network

`fe80::17` on network A != `fe80::17` on network B

problem: machine can be connected to two networks

which link local?

"link local": `169.254/16`, `fe80::/10`

specific to each local network

`fe80::17` on network A != `fe80::17` on network B

problem: machine can be connected to two networks

solution: `fe80::17%A` versus `fe80::17%B`

which link local?

"link local": `169.254/16`, `fe80::/10`

specific to each local network

`fe80::17` on network A != `fe80::17` on network B

problem: machine can be connected to two networks

solution: `fe80::17%A` versus `fe80::17%B`

what about IPv4? uh... too bad?

> "There is no standard or obvious solution to this problem...must be done explicitly through other means. The specification does not stipulate those means." — RFC 3927, section 3.2

switch v router: tables

switch (‘bridge’) table

MAC address	port
00:11:22:33:44:55	1
00:33:00:01:02:aa	2
00:44:00:01:02:bb	3
…	…
default	(all)

routing table

IP addresses	gateway	iface
2001:0db8:40::/48	---	int
3fff:1000:19::/48	---	ext
…	…	…
default	fe80::17	ext

switch v router: tables

switch ('bridge') table

MAC address	port
00:11:22:33:44:55	1
00:33:00:01:02:aa	2
00:44:00:01:02:bb	3
…	…
default	(all)

routing table

IP addresses	gateway	iface
2001:0db8:40::/48	---	int
3fff:1000:19::/48	---	ext
…	…	…
default	fe80::17	ext

one logical device with multiple ports

not in table: always broadcast

switch v router: tables

switch ('bridge') table

MAC address	port
00:11:22:33:44:55	1
00:33:00:01:02:aa	2
00:44:00:01:02:bb	3
…	…
default	(all)

routing table

IP addresses	gateway	iface
2001:0db8:40::/48	---	int
3fff:1000:19::/48	---	ext
…	…	…
default	fe80::17	ext

'interface' = which network

one interface might have multiple ports that are 'bridged' together

switch v router: tables

switch ('bridge') table

MAC address	port
00:11:22:33:44:55	1
00:33:00:01:02:aa	2
00:44:00:01:02:bb	3
…	…
default	(all)

routing table

IP addresses	gateway	iface
2001:0db8:40::/48	---	int
3fff:1000:19::/48	---	ext
…	…	…
default	fe80::17	ext

gateway = who to send to next

no gateway = 'direct' to destination

need to have specific destination
to send to on interface

trivial tables

let's say we're connected to ONE interface with ONE port

trivial tables

let's say we're connected to ONE interface with ONE port

tables are really trivial:

switch ('bridge') table

MAC address	port
default	the port

routing table (IPv6)

IP addresses	gateway	iface
`2001:0db8:40::/48`	`---`	`the interface`
default	`fe80::17`	`the interface`

routing table (IPv4)

IP addresses	gateway	iface
`192.0.2.0/24`	`---`	`the interface`
default	`192.0.2.1`	`the interface`

trivial tables

let's say we're connected to ONE interface with ONE port

tables are really trivial:

switch ('bridge') table

MAC address	port
default	the port

routing table (IPv6)

IP addresses	gateway	iface
`2001:0db8:40::/48`	`---`	`the interface`
default	`fe80::17`	`the interface`

routing table (IPv4)

IP addresses	gateway	iface
`192.0.2.0/24`	`---`	`the interface`
default	`192.0.2.1`	`the interface`

trivial tables

let's say we're connected to ONE interface with ONE port

tables are really trivial:

switch ('bridge') table

MAC address	port
default	the port

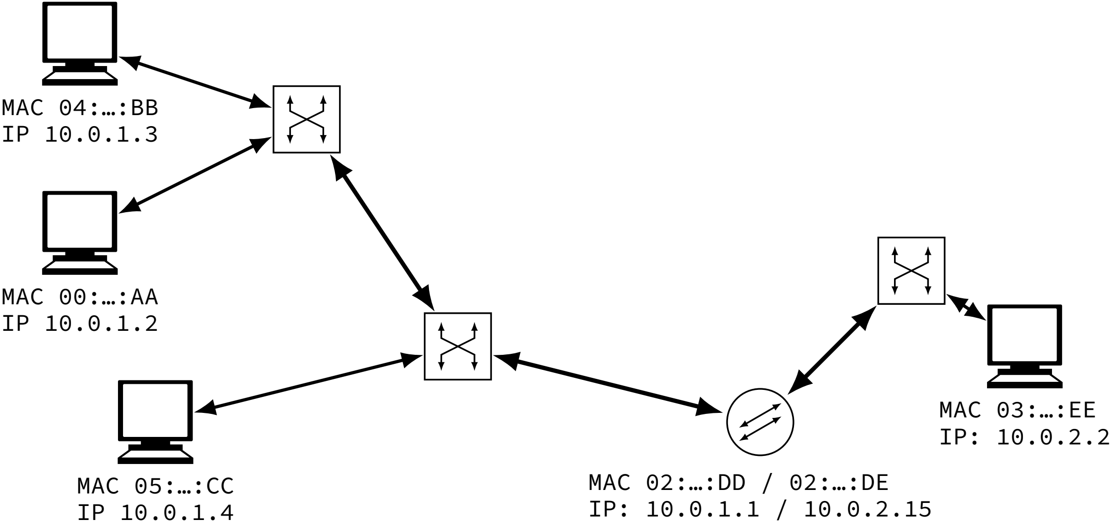
routing table (IPv6)

IP addresses	gateway	iface
`2001:0db8:40::/48`	`---`	`the interface`
default	`fe80::17`	`the interface`

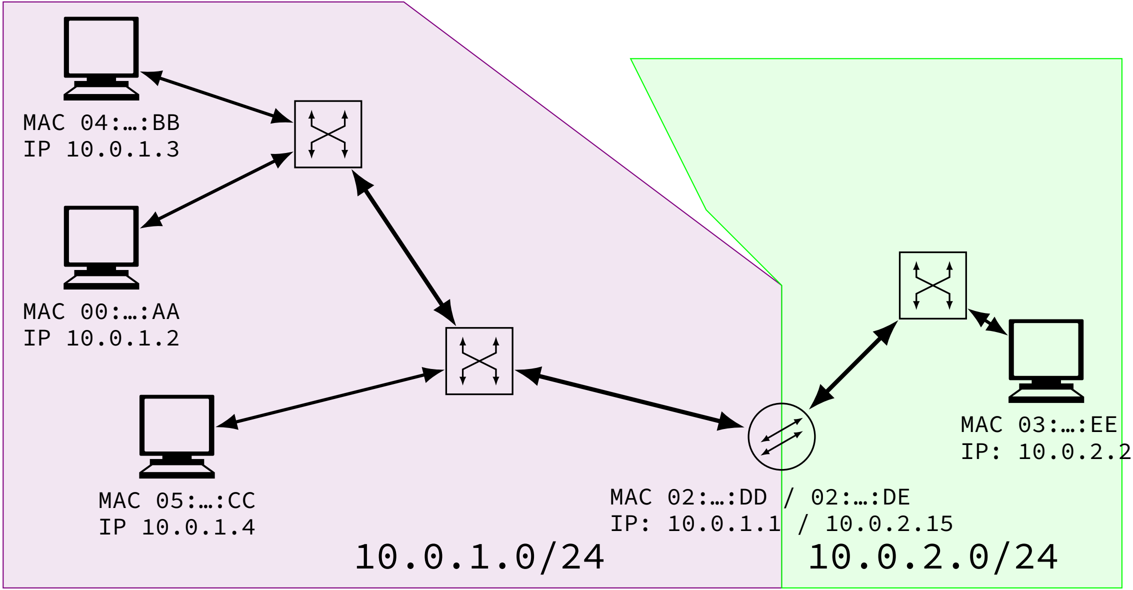
routing table (IPv4)

IP addresses	gateway	iface
`192.0.2.0/24`	`---`	`the interface`
default	`192.0.2.1`	`the interface`

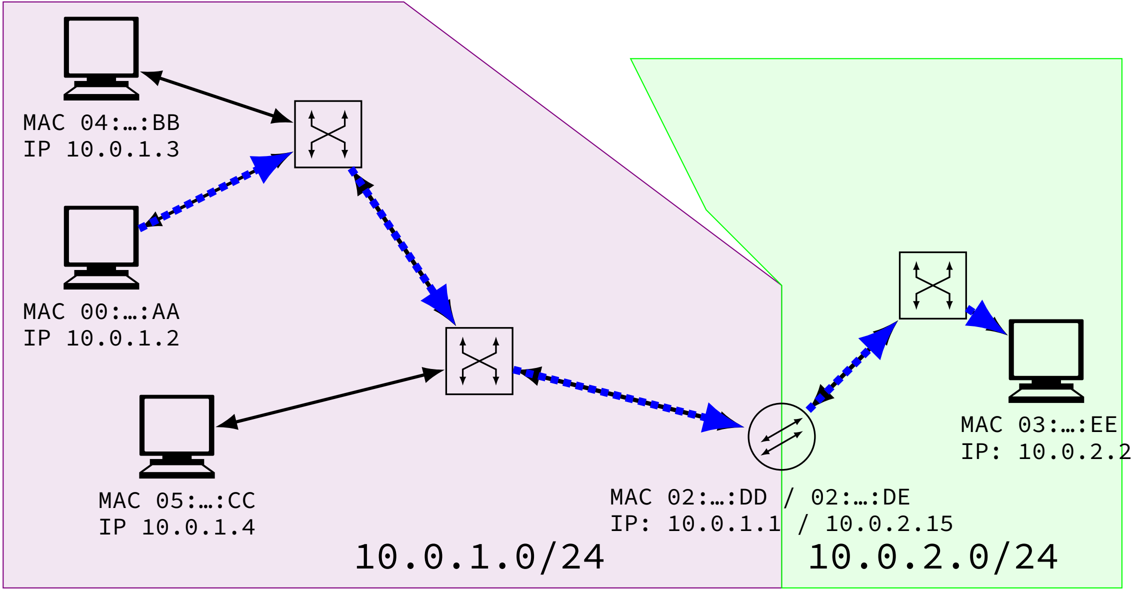
switch v router: on the wires



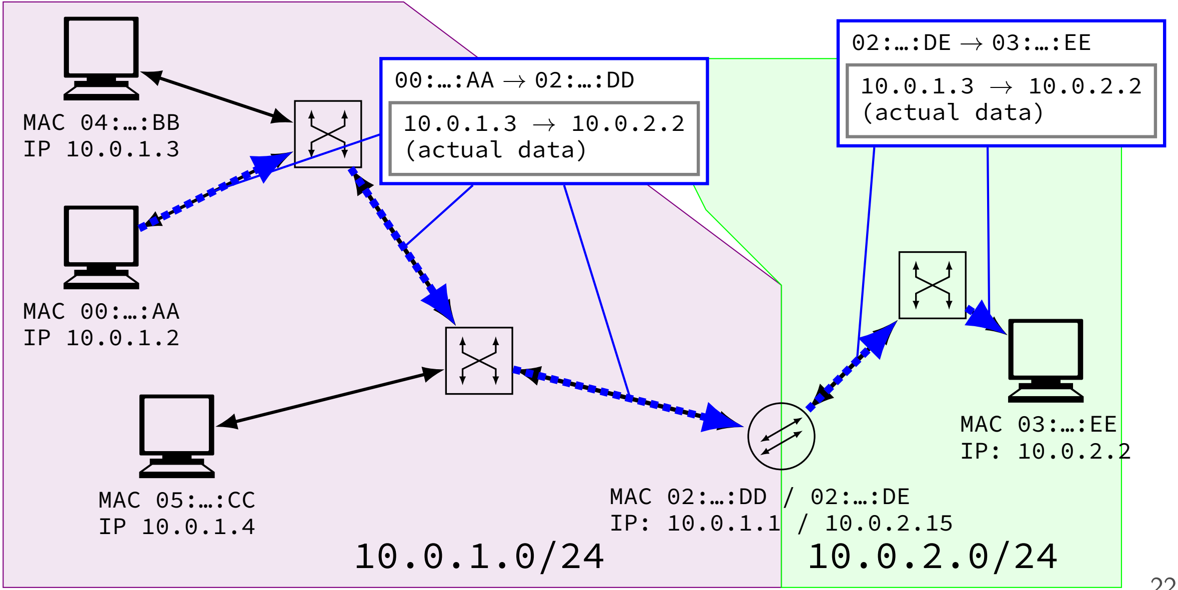
switch v router: on the wires



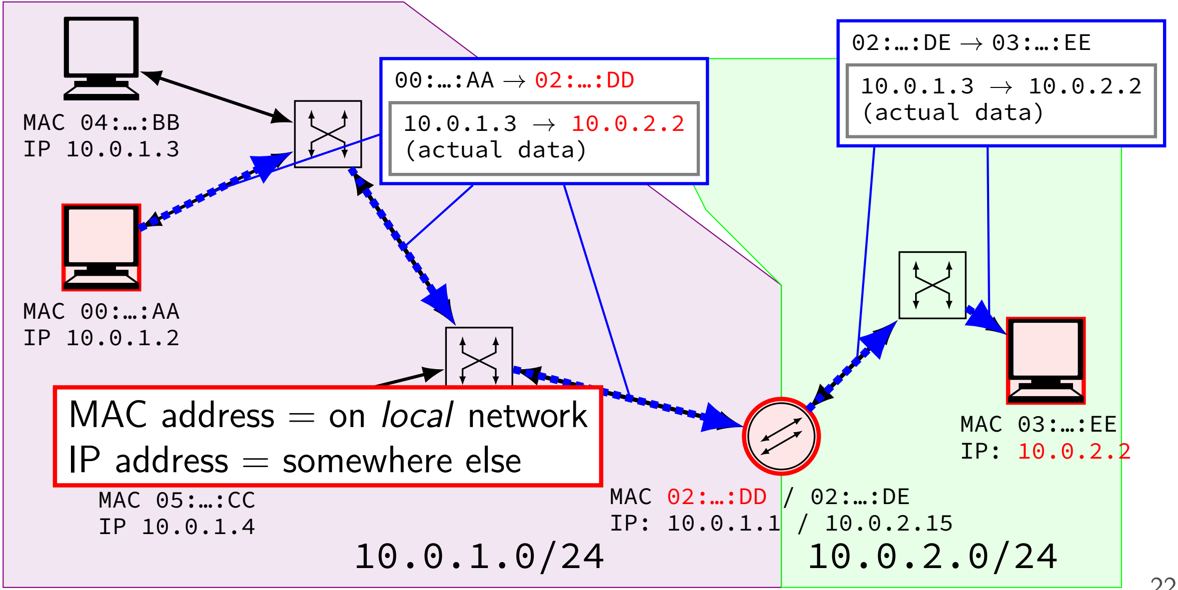
switch v router: on the wires



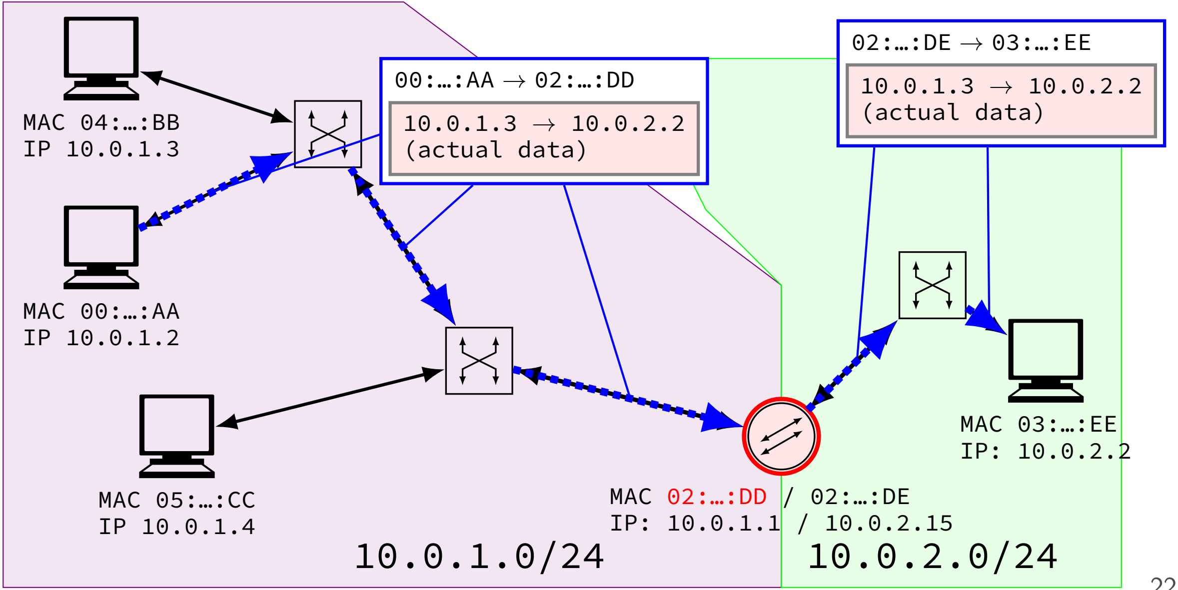
switch v router: on the wires



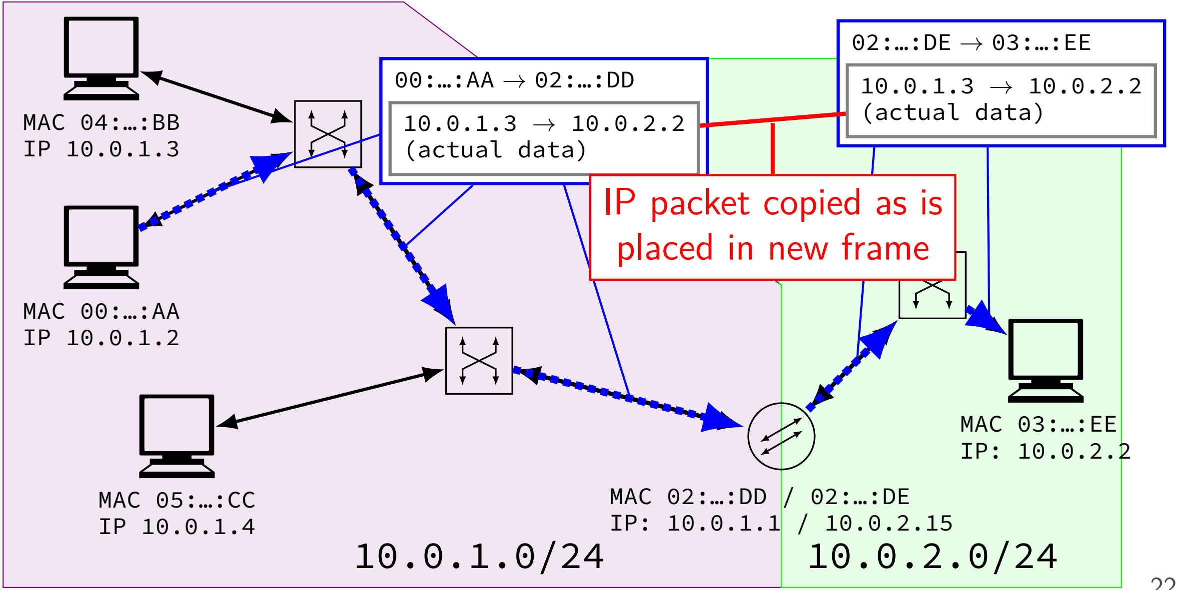
switch v router: on the wires



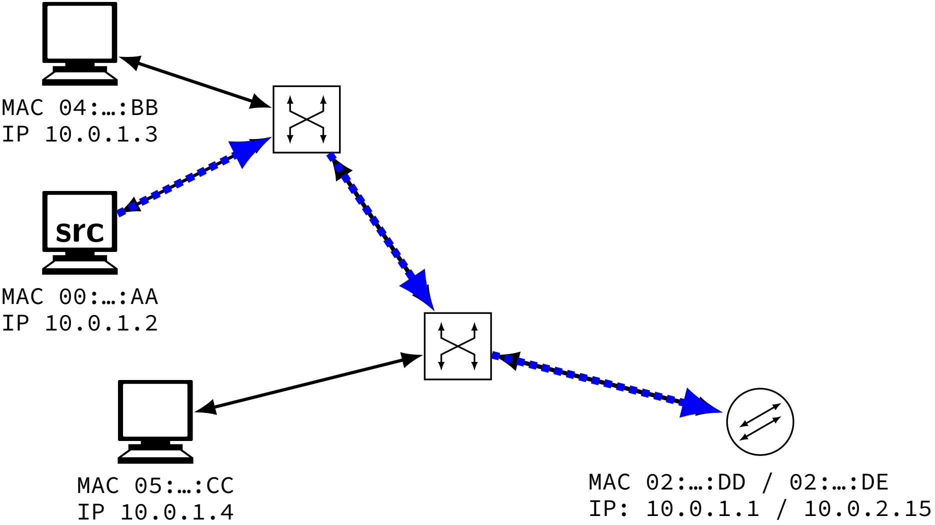
switch v router: on the wires



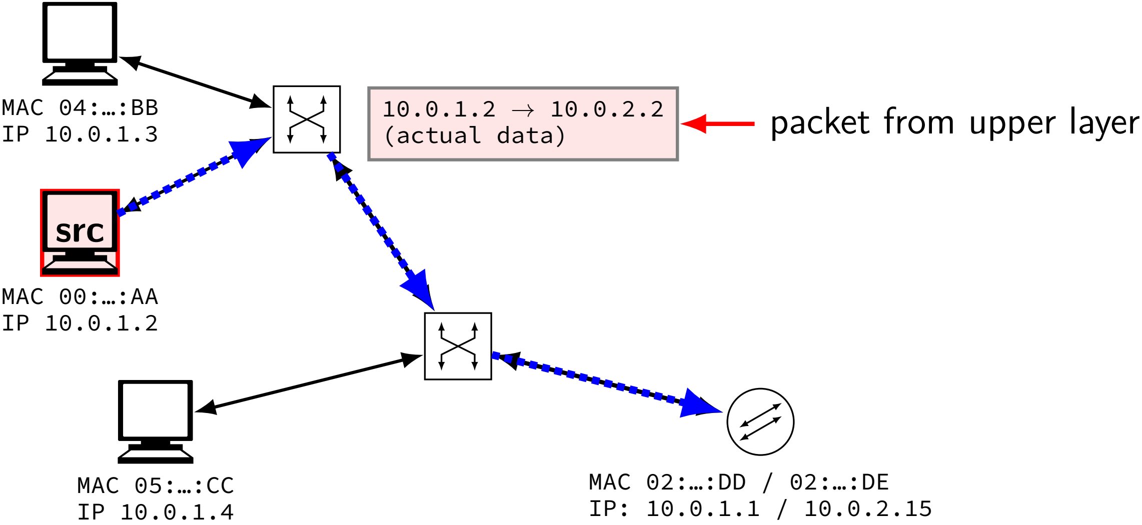
switch v router: on the wires



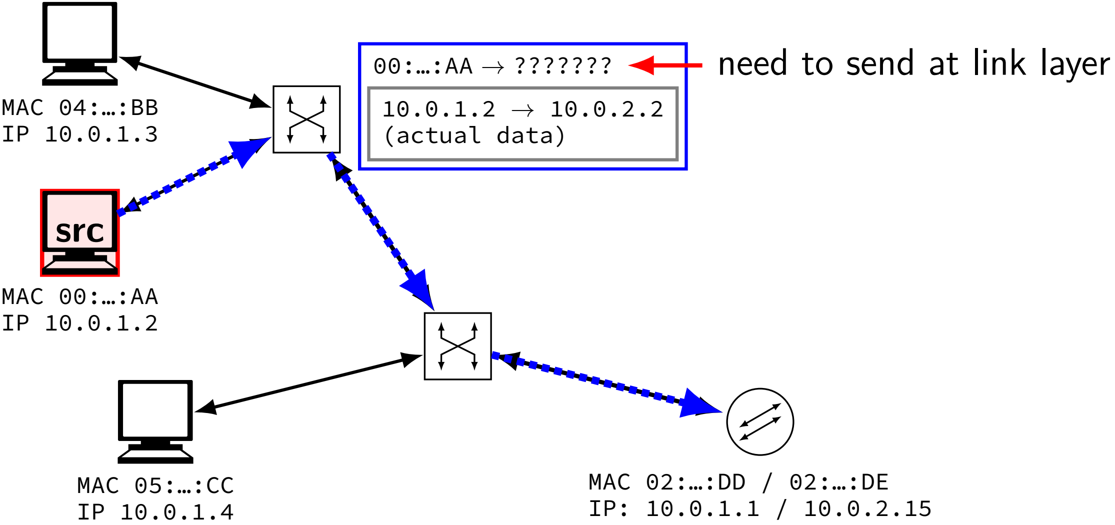
steps at the sender



steps at the sender

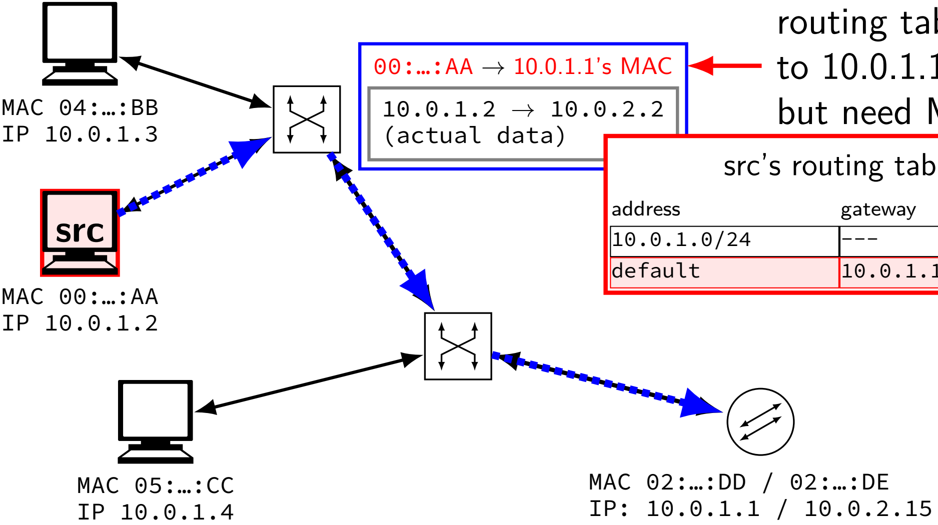


steps at the sender



steps at the sender

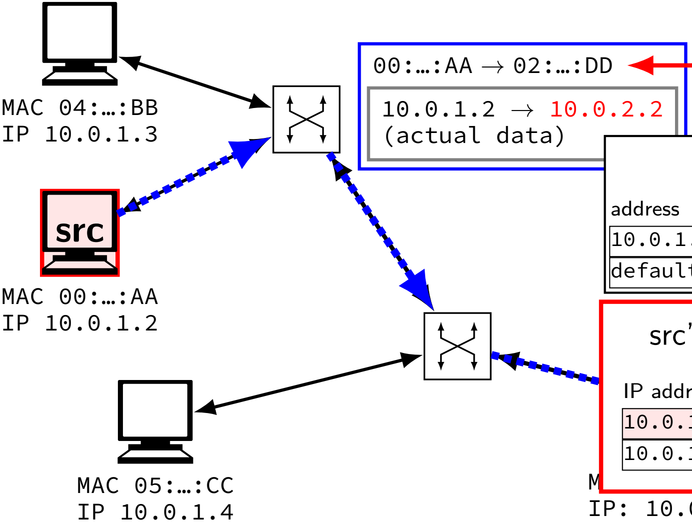
routing table says:
to 10.0.1.1
but need MAC address



src's routing table

address	gateway	iface
10.0.1.0/24	---	wired
default	10.0.1.1	wired

steps at the sender



need IP:MAC address table
called neighbor table
or ARP table

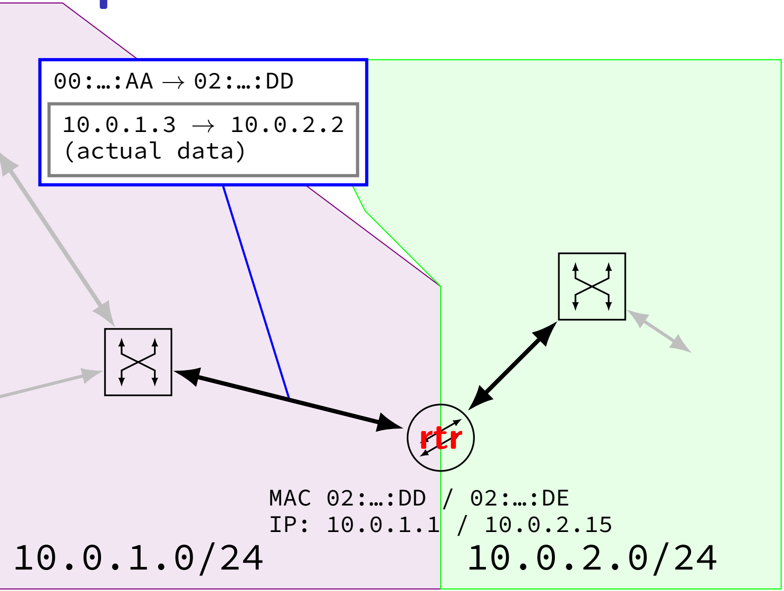
src's routing table

address	gateway	iface
10.0.1.0/24	---	wired
default	10.0.1.1	wired

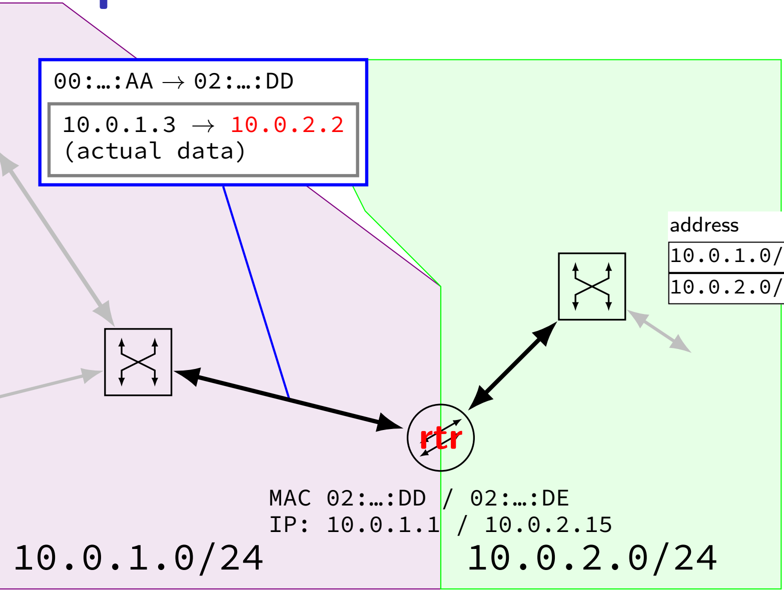
src's neighbor table

IP address	MAC addresss
10.0.1.1	05:…:DD
10.0.1.4	05:…:CC

steps at the router



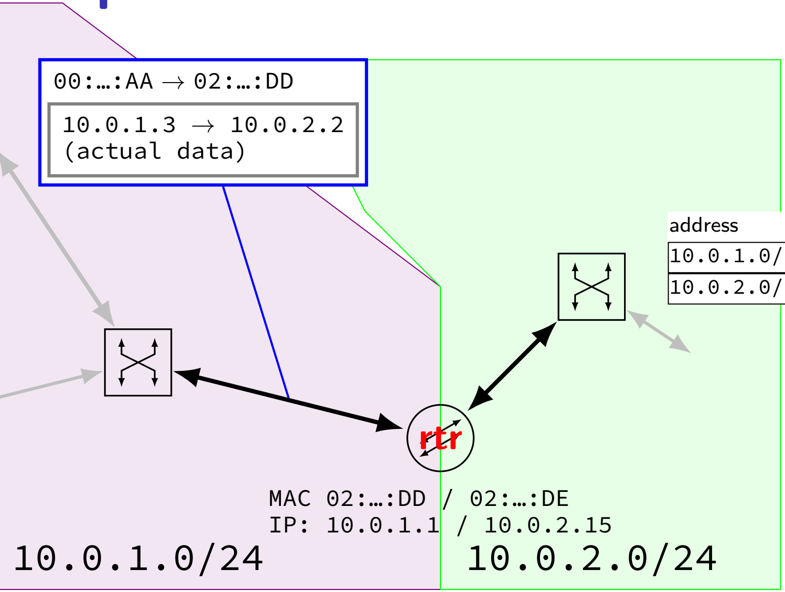
steps at the router



rtr's routing table

address	gateway	iface
10.0.1.0/24	---	left
10.0.2.0/24	---	right

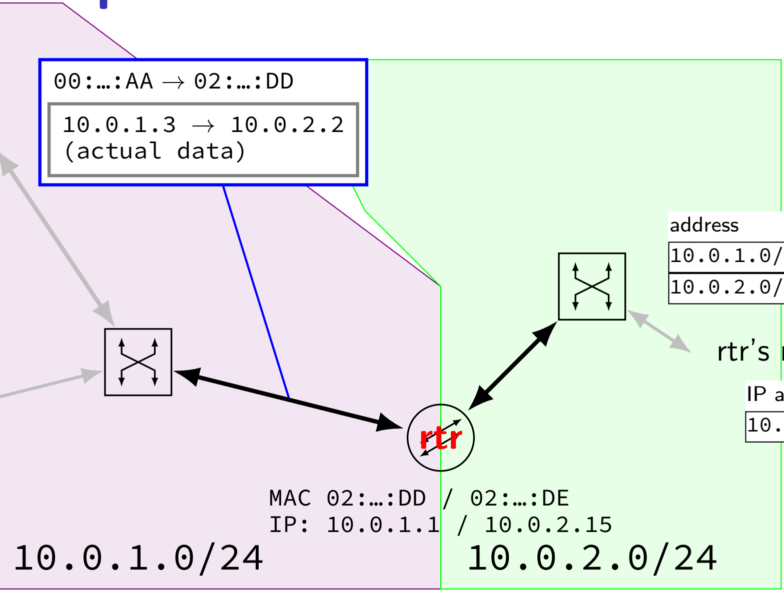
steps at the router



rtr's routing table

address	gateway	iface
10.0.1.0/24	---	left
10.0.2.0/24	---	right

steps at the router



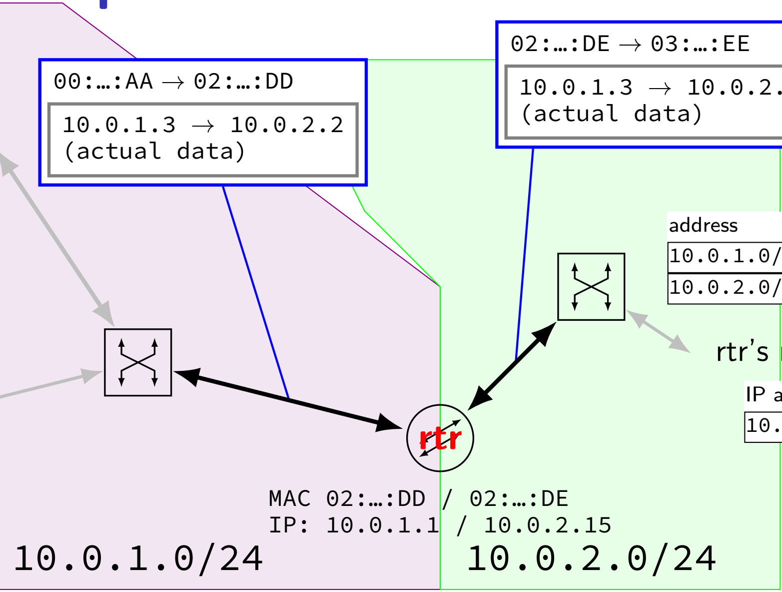
rtr's routing table

address	gateway	iface
10.0.1.0/24	---	left
10.0.2.0/24	---	right

rtr's neighbor table for right

IP address	MAC addresss
10.0.2.2	03:…:EE

steps at the router



rtr's routing table

address	gateway	iface
10.0.1.0/24	---	left
10.0.2.0/24	---	right

rtr's neighbor table for right

IP address	MAC addresss
10.0.2.2	03:…:EE

making neighbor tables

need neighbor table to use IP addresses on network

some options:

- system administrator manually configures
- discover dynamically

manual neighbor tables

on Linux, can run some commands

```
ip niegh add 10.0.2.2 dev right lladdr
03:05:…:EE permanent
```

(newer interface, also supports IPv6)

```
arp -i right -s 10.0.2.2 03:05:…:EE
```

IPv4 only; does not allow setting validity duration

ARP/ND protocols

filling in tables dynamically?

key idea: ask *everyone* on network

entity with that IP address responds

IPv4: Address Resolution Protocol (ARP)

IPv6: ICMPv6 Neighbor Discovery (ND)

ICMP = Internet Control Message Protocol

ARP messages

suppose router IP address 10.0.2.15 and MAC address 02:...:DE
needs to find out that 10.0.2.2 uses 03:...:EE

02:...:DE→FF:FF:FF:FF:FF:FF: request 10.0.2.2
tell 10.0.2.15 (=02:...:DE)

FF:FF:FF:FF:FF:FF = broadcast (send to all on network)

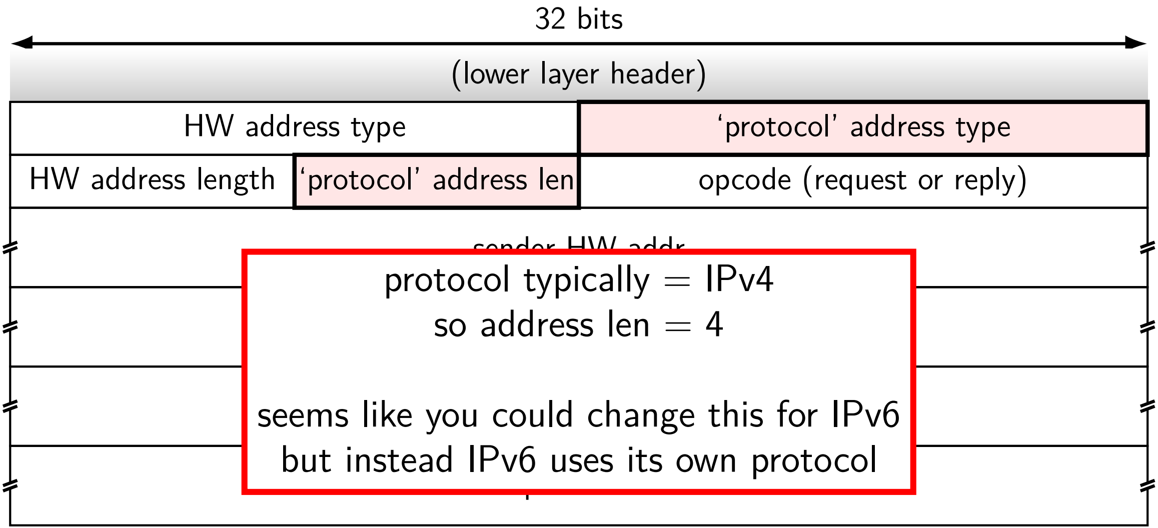
03:...:EE→02:...:DE: reply 10.0.2.2=03:...:EE
tell 10.0.2.15=02:...:DE

ARP message format

32 bits

(lower layer header)			
HW address type		‘protocol’ address type	
HW address length	‘protocol’ address len	opcode (request or reply)	
sender HW addr			
sender protocol addr			
dest HW addr			
dest protocol addr			

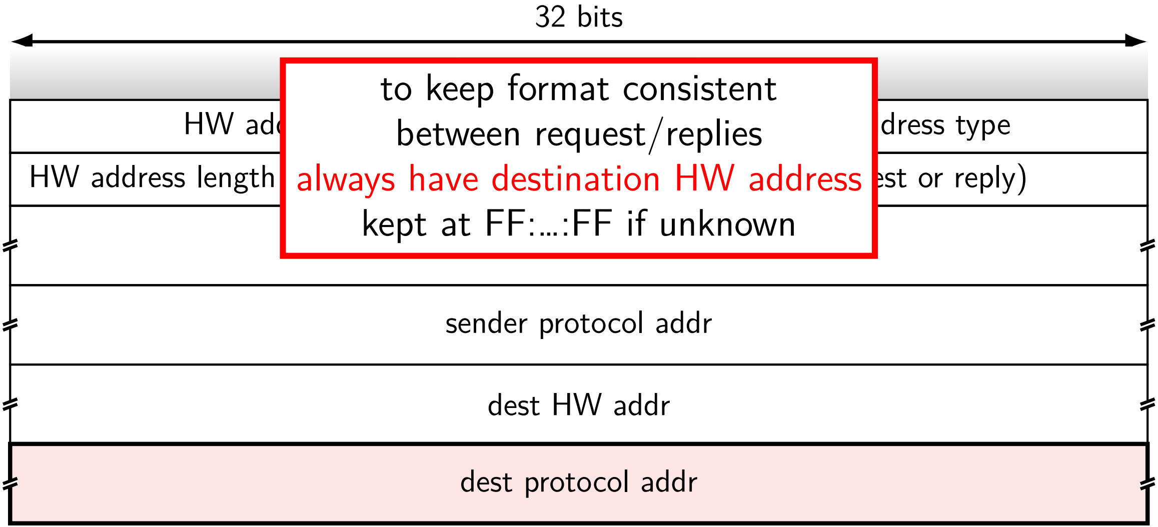
ARP message format



protocol typically = IPv4
so address len = 4

seems like you could change this for IPv6
but instead IPv6 uses its own protocol

ARP message format



ARP messages (revisited)

suppose router IP address 10.0.2.15 and MAC address 02:...:DE
needs to find out that 10.0.2.2 uses 03:...:EE

02:...:DE→FF:FF:FF:FF:FF:FF: request 10.0.2.2
tell 10.0.2.15 (=02:...:DE)

everyone who receives this can add 10.0.2.15=02:...:DE to neighbor table

03:...:EE→02:...:DE: reply 10.0.2.2=03:...:EE
tell 10.0.2.15=02:...:DE

everyone who receives this can add 10.0.2.2=03:...:EE to neighbor table

ICMPv6 ND

IPv6 uses different protocol for this

...but mostly works the same

differences:

sent as IPv6 packets

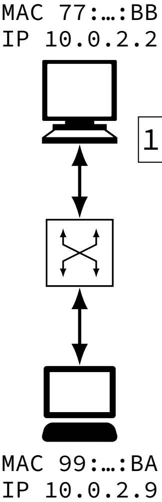
requests sent to special multicast address

goal: allow nodes to easily ignore irrelevant requests

different names:

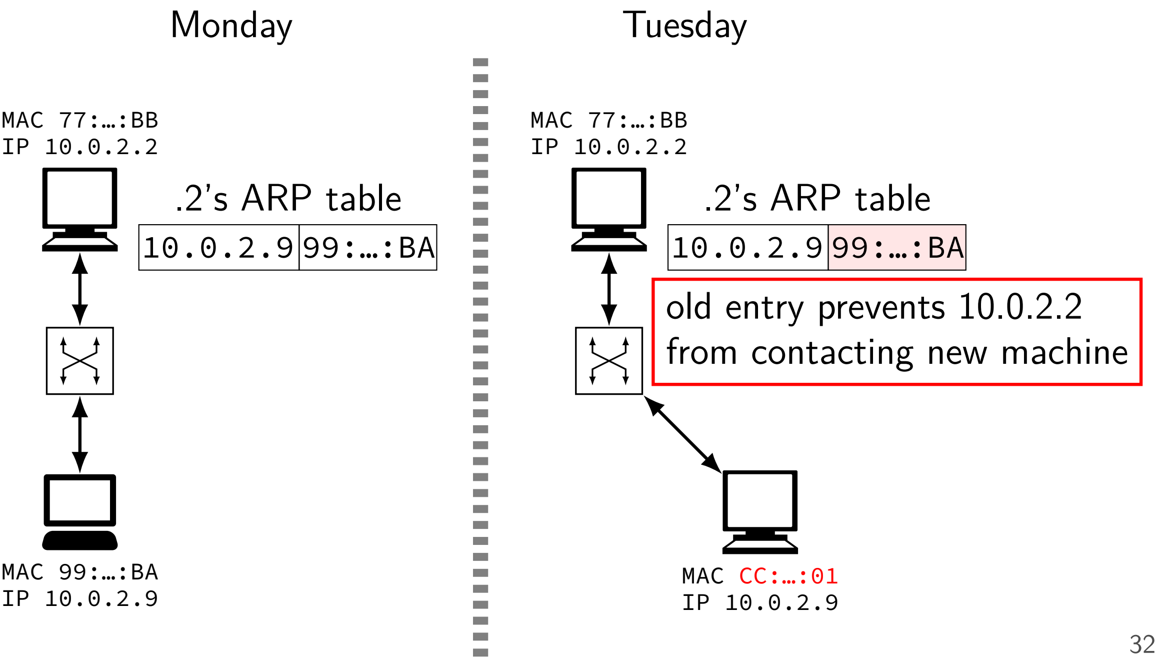
request = solicitiation

reply = advertisement



.2's ARP table

10.0.2.9	99:…:BA


Monday
Tuesday
MAC 77:…:BB
IP 10.0.2.2
.2's ARP table
10.0.2.9 99:…:BA
MAC 99:…:BA
IP 10.0.2.9
MAC 77:…:BB
IP 10.0.2.2
.2's ARP table
10.0.2.9 99:…:BA
old entry prevents 10.0.2.2
from contacting new machine
MAC CC:…:01
IP 10.0.2.9

gratituous ARP requests

solution: send *unsolicited* ARP messages

CC:…:01→FF:…:FF: request: who has 10.0.2.9, tell 10.0.2.9=CC:…:01

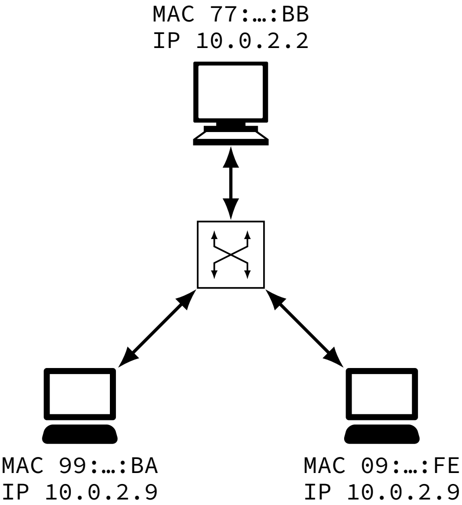
gratuitous ARP requests

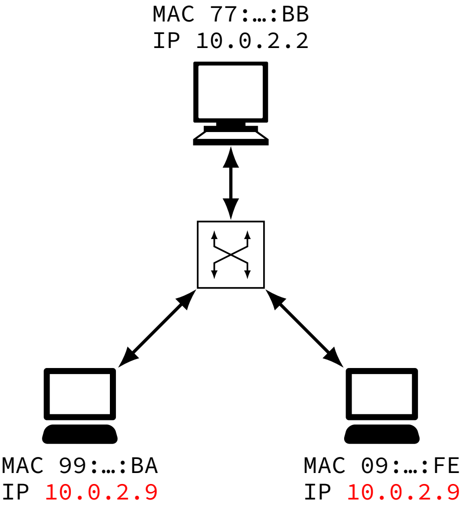
solution: send *unsolicited* ARP messages

CC:...:01→FF:...:FF: request: who has 10.0.2.9, tell 10.0.2.9=CC:...:01

request not reply b/c concerns about old/broken implementations

ICMPv6 ND fixes this:
message is ‘advertisement’ (∼ reply), not ‘solicitation’ (∼ request)


MAC 77:…:BB
IP 10.0.2.2
MAC 99:…:BA
IP 10.0.2.9
MAC 09:…:FE
IP 10.0.2.9


MAC 77:…:BB
IP 10.0.2.2
MAC 99:…:BA
IP 10.0.2.9
MAC 09:…:FE
IP 10.0.2.9

duplicate addresses

recommendations in RFC 5227 "IPv4 Address Conflict Detection"

probe for IP address before using it

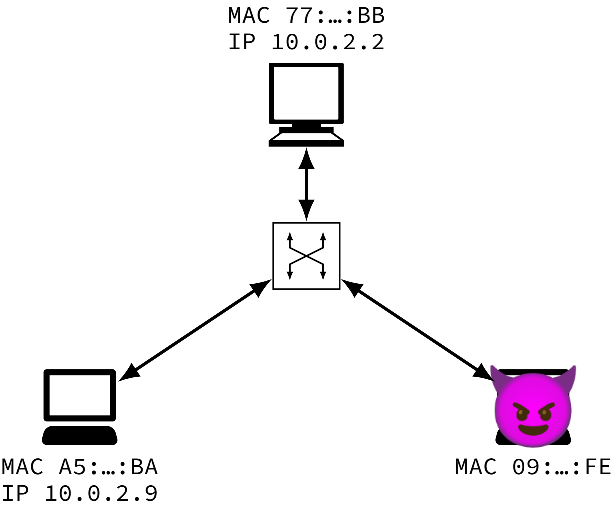
- make sure to broadcast when starting to use address
- probably give up on address if conflict found

watch out for ARP messages indicating address in use

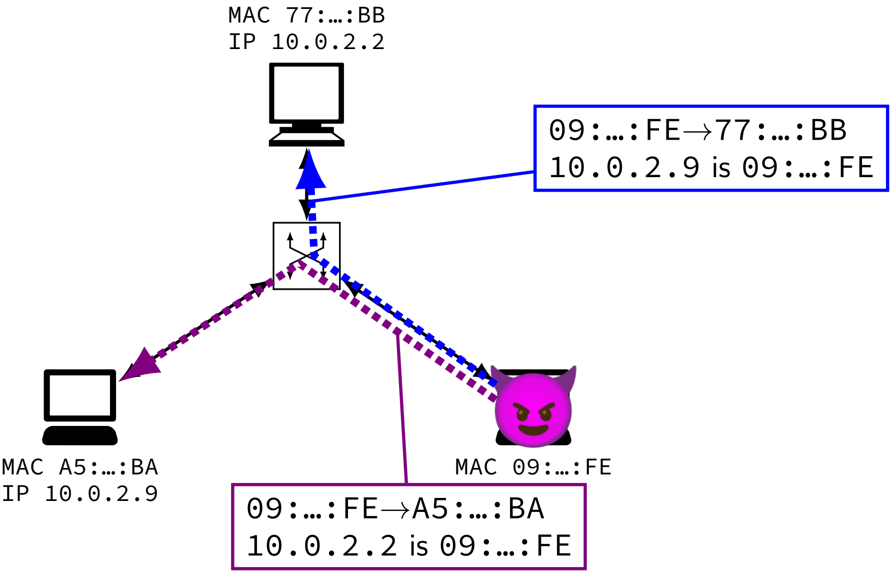
on detecting conflict choose between:

- 'defend' address with more gratituous requests
- give up address

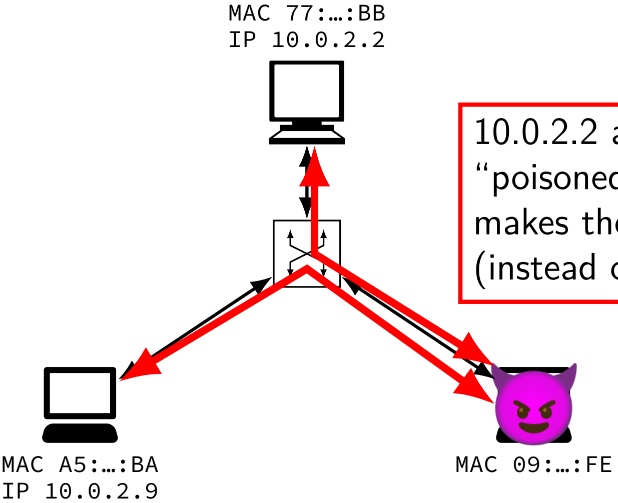
ARP hijacking



ARP hijacking



ARP hijacking

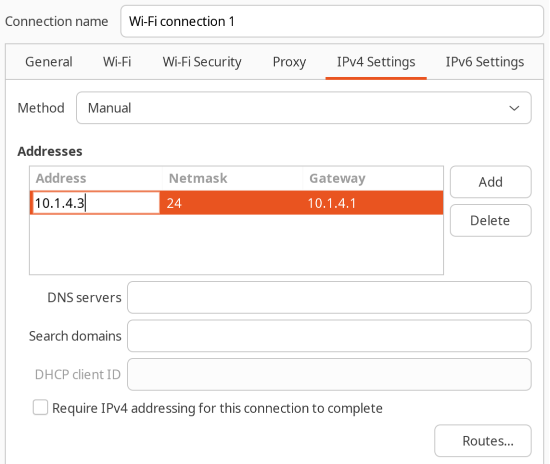


10.0.2.2 and 10.0.2.9 have "poisoned" ARP tables
makes them send everything to attacker
(instead of each other)

autoconfiguration

how do hosts get address + default routing table?

one answer: set manually



simple network config

IP address: 10.0.2.45

(sub)net mask: /25 (aka 255.255.255.128)

varies which format is input

(default) gateway: 10.0.2.102

simple network config

IP address: 10.0.2.45

(sub)net mask: /25 (aka 255.255.255.128)

varies which format is input

(default) gateway: 10.0.2.102

addresses	next hop	device
10.2.0.0/25	(direct)	out
default	10.2.0.102	out

DHCP messages (1)

protocol looks weird in packet traces because of history

built on top of UDP + IP

built as extension to older BOOTP (bootstrap protocol)

common message format for different “operations”

DHCP messages (2)

from client (looking to configure itself):

DISCOVER (look for configuration server)
REQUEST (get configuration from server)

from server (offering configurations):

OFFER ('I am a configuration server')
ACK (here's a configuration)

DHCP request example

```
▸ Frame 66: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
▾ Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x1fbef68f
    Seconds elapsed: 1
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (53) DHCP Message Type (Request)
  ▸ Option: (61) Client identifier
  ▸ Option: (55) Parameter Request List
  ▸ Option: (57) Maximum DHCP Message Size
  ▸ Option: (50) Requested IP Address (172.25.143.52)
  ▸ Option: (12) Host Name
  ▸ Option: (255) End
```

DHCP request example

```
Frame 66: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
User Datagram Protocol, Src Port: 68, Dst Port: 67
Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Ha
    Ha
    Ho
    Tr
    Se
    Bo
    Cl
    Yo
    Ne
    Relay agent IP address: 0.0.0.0
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
    Option: (53) DHCP Message Type (Request)
    Option: (61) Client identifier
    Option: (55) Parameter Request List
    Option: (57) Maximum DHCP Message Size
    Option: (50) Requested IP Address (172.25.143.52)
    Option: (12) Host Name
    Option: (255) End
```

built on IP+UDP rather than special protocol like ARP

sending to broadcast ethernet/IP address (all 1 bits)
placeholder source IP of 0.0.0.0

DHCP request example

```
▸ Frame 66: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
▾ Dynamic Host Configuration Protocol (Request)
     Message type: Boot Request (1)
     Hardware type: Ethernet (0x01)
     Hardware address length:
     Hops: 0
     Transaction ID: 0x1fbef6
     Seconds elapsed: 1
   ▸ Bootp flags: 0x0000 (Uni
     Client IP address: 0.0.0.0
     Your (client) IP address: 0.0.0.0
     Next server IP address: 0.0.0.0
     Relay agent IP address: 0.0.0.0
     Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
     Client hardware address padding: 00000000000000000000
     Server host name not given
     Boot file name not given
     Magic cookie: DHCP
   ▸ Option: (53) DHCP Message Type (Request)
   ▸ Option: (61) Client identifier
   ▸ Option: (55) Parameter Request List
   ▸ Option: (57) Maximum DHCP Message Size
   ▸ Option: (50) Requested IP Address (172.25.143.52)
   ▸ Option: (12) Host Name
   ▸ Option: (255) End
```

'boot' probably because derived from bootstrap protocol (BOOTP)

DHCP request example

```
▸ Frame 66: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
▾ Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x1fbef68f
    Seconds elapsed: 1
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file n
    Magic cooki
  ▸ Option: (53
  ▸ Option: (61
  ▸ Option: (55
  ▸ Option: (57) Maximum DHCP Message Size
  ▸ Option: (50) Requested IP Address (172.25.143.52)
  ▸ Option: (12) Host Name
  ▸ Option: (255) End
```

message format same in both directions, so fields here intended for use in response

DHCP ACK example

```
▾ Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 1
    Transaction ID: 0x1fbef68f
    Seconds elapsed: 1
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 172.25.143.52
    Next server IP address: 0.0.0.0
    Relay agent IP address: 172.25.128.3
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (53) DHCP Message Type (ACK)
  ▸ Option: (54) DHCP Server Identifier (128.143.107.118)
  ▾ Option: (51) IP Address Lease Time
      Length: 4
      IP Address Lease Time: (1800s) 30 minutes
  ▾ Option: (1) Subnet Mask (255.255.254.0)
      Length: 4
      Subnet Mask: 255.255.254.0
  ▸ Option: (6) Domain Name Server
  ▾ Option: (3) Router
      Length: 4
      Router: 172.25.142.1
  ▸ Option: (255) End
```

DHCP ACK example

```
▾ Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 1
    Transaction ID: 0x1fbef68f
    Seconds elapsed: 1
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 172.25.143.52
    Next server IP address: 0.0.0.0
    Relay agent IP address: 172.25.128.3
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (53) DHCP Message Type (ACK)
  ▸ Option: (54) DHCP Server Identifier (128.143.107.118)
  ▾ Option: (51) IP Address Lease Time
      Length: 4
      IP Address Lease Time: (1800s) 30 minutes
  ▾ Option: (1) Subnet Mask (255.255.254.0)
      Length: 4
      Subnet Mask: 255.255.254.0
  ▸ Option: (6) Domain Name Server
  ▾ Option: (3) Router
      Length: 4
      Router: 172.25.142.1
  ▸ Option: (255) End
```

response (ACK) has address fields filled in

DHCP ACK example

```
▾ Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 1
    Transaction ID: 0x1fbef68f
    Seconds elapsed: 1
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0
    Your (client) IP address: 172.25.143.52
    Next server IP address: 0.0.0.0
    Relay agent IP address: 172.25.128.3
    Client MAC address: f4:6d:3f:d3:64:59 (f4:6d:3f:d3:64:59)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (53) DHCP Message Type (ACK)
  ▸ Option: (54) DHCP Server Identifier (128.143.107.118)
  ▾ Option: (51) IP Address Lease Time
      Length: 4
      IP Address Lease Time: (1800s) 30 minutes
  ▾ Option: (1) Subnet Mask (255.255.254.0)
      Length: 4
      Subnet Mask: 255.255.254.0
  ▸ Option: (6) Domain Name Server
  ▾ Option: (3) Router
      Length: 4
      Router: 172.25.142.1
  ▸ Option: (255) End
```

/23 = 255.255.254.0 mask

172.25.142.0 = 172.25.143.52 b

DHCP leases

DHCP ACKs specify a time limit

- (example from prior slide (UVa eduroam): 30 minutes)

need to be renewed (new REQUEST + ACK)

- REQUESTs can contain 'desired address' (= current address when renewing)

how many DHCP servers?

DHCP assumption: broadcast to local network and there's the server

conflicting goals:

- want broadcasts not to go too many machines
- want to have few DHCP servers

how many DHCP servers?

DHCP assumption: broadcast to local network and there's the server

conflicting goals:

- want broadcasts not to go too many machines
- want to have few DHCP servers

solution: DHCP *relays*

DHCP relays

IPv6 autoconfiguration

in IPv4, autoconfiguration “bolted on”

- one protocol to be assigned address (DHCP)
- one protoocl to communicate IP address to other nodes on network (ARP)

IPv6 was designed later, so they thought about it early

big network address assignment

IPv6 local networks are typically /64s

2^{64} address available for local network

why so big? allow easy address assignment

StateLess Address Auto Configuration (SLAAC)

MAC-address based address assignment

let's say my local network is 2001:db8:4999:3333::/64

MAC address	IPv6 address
`11:22:33:44:55:66`	`2001:db8:4999:3333:1122:33ff:fe44:5566`
`01:A0:B3:CC:DD:FF`	`2001:db8:4999:3333:01a0:b3ff:fecc:ddff`
...	...

MAC-address based address assignment

let's say my local network is 2001:db8:4999:3333::/64

MAC address	IPv6 address
`11:22:33:44:55:66`	`2001:db8:4999:3333:1122:33ff:fe44:5566`
`01:A0:B3:CC:DD:FF`	`2001:db8:4999:3333:01a0:b3ff:fecc:ddff`
...	...

Network Working Group T. Narten
Request for Comments: 3041 IBM
Category: Standards Track R. Draves
Microsoft Research

...

Abstract

.... Use of the extension causes nodes to generate global-scope addresses from interface identifiers that change over time, even in cases where the interface contains an embedded IEEE identifier. Changing the interface identifier (and the global-scope addresses generated from it) over time makes it more difficult for eavesdroppers and other information collectors to identify when different addresses used in different transactions actually correspond to the same node.

late timeline

privacy extensions weren't default until

- MacOS X 10.7 (2011)
- Windows Vista (2007)
- Ubuntu 12.04 (2012)

SLAAC

uses as ICMPv6 (same as for neighbor discovery)

two modes:

- when there's a router (get *global* addresses)
- when there's only a local network (get *link-local* addresses)

router adverisements

nodes send a ICMPv6 *Router Solicitation* message

receive back a ICMPv6 *Router Advertisement* which can have:

prefix information
- nodes choose address starting with prefix
- check for duplicates using neighbor discovery

DNS information

“managed configuration” flag
- nodes use DHCPv6 to get configuration

“other configuration” flag
- nodes choose address using prefix, get additional information from DHCPv6

router adverisements

nodes send a ICMPv6 *Router Solicitation* message

receive back a ICMPv6 *Router Advertisement* which can have:

prefix information

- nodes choose address starting with prefix
- check for duplicates using neighbor discovery

DNS information

"managed configuration" flag

- nodes use DHCPv6 to get configuration

"other configuration" flag

- nodes choose address using prefix, get additional information from DHCPv6

router adverisements

nodes send a ICMPv6 *Router Solicitation* message

receive back a ICMPv6 *Router Advertisement* which can have:

prefix information

- nodes choose address starting with prefix
- check for duplicates using neighbor discovery

DNS information

"managed configuration" flag

- nodes use DHCPv6 to get configuration

"other configuration" flag

- nodes choose address using prefix, get additional information from DHCPv6

router adverisements

nodes send a ICMPv6 *Router Solicitation* message

receive back a ICMPv6 *Router Advertisement* which can have:

prefix information

- nodes choose address starting with prefix
- check for duplicates using neighbor discovery

DNS information

"managed configuration" flag

- nodes use DHCPv6 to get configuration

"other configuration" flag

- nodes choose address using prefix, get additional information from DHCPv6

router adverisements

nodes send a ICMPv6 *Router Solicitation* message

receive back a ICMPv6 *Router Advertisement* which can have:

prefix information

- nodes choose address starting with prefix
- check for duplicates using neighbor discovery

DNS information

"managed configuration" flag

- nodes use DHCPv6 to get configuration

"other configuration" flag

- nodes choose address using prefix, get additional information from DHCPv6

backup slides