Link-State Protocols

Lecture 6, CS 168, Spring 2026

Routing protocols can be classified by:

- *Where* they operate. (Intra-domain or inter-domain.)
- *How* they operate. (Distance-vector, link-state, or path-vector.)

Today, we'll look at **link-state** protocols.

- Very common as an Interior Gateway Protocol.
- Major examples:
  - IS-IS (Intermediate System to Intermediate System).
  - OSPF (Open Shortest Path First).

	Intra-domain (Interior Gateway Protocol)	Inter-domain (Exterior Gateway Protocol)
Distance-vector	RIP	–
Link-state	**IS-IS, OSPF**	–
Path-vector	–	BGP

Link-State Protocols: Definition

Link-state protocols:

1. Every router learns the full network graph. ← How do routers learn this?
2. Then, each router runs a shortest-path algorithm on the graph to populate the forwarding table. ← What algorithm do we run?

Distance-vector:

- **Local data**: Each node only knows about part of the network.
- **Global (distributed) computation**: Each node computes part of the solution, working with other nodes.

Link-state:

- **Global data**: Each node knows about the full network graph.
- **Local computation**: Each node computes the full solution by itself.

Global Data
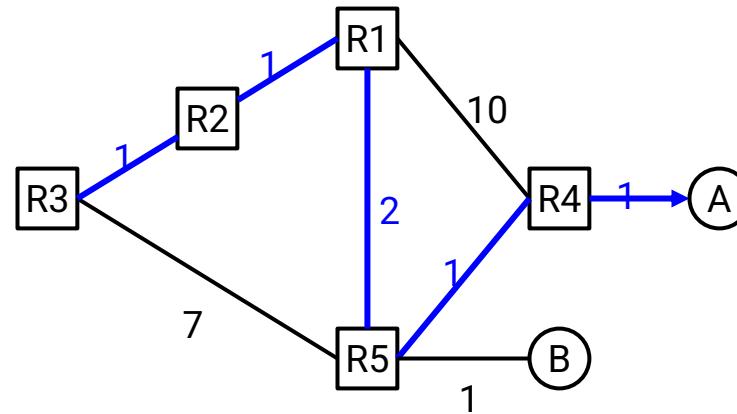
Suppose R3 has learned the full network graph. ← How do routers learn this?

- We know all the links, their status (up or down?), and their costs.
- We know about all the destinations. ← What algorithm do we run?

R3 can run a graph algorithm to compute paths.

- We can populate the forwarding table with the next-hop (in the computed path).

R3's Table	
Destination	Next Hop
A	R2
...	...

Computing Paths

Lecture 6, CS 168, Spring 2026

Link-State Protocols

Link-State Algorithms

What graph algorithm do we run to compute paths?

Any single source shortest-path algorithm will work.

- Need to find shortest paths from a single source (the router) to every host.

Some possible choices:
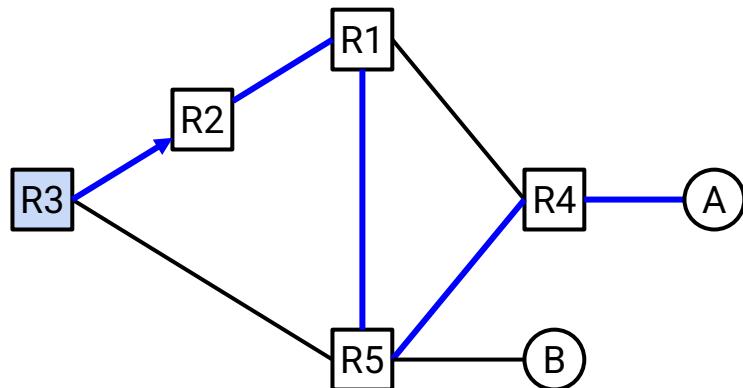
- Bellman-Ford (the original serial version).
- Dijkstra's algorithm.
- Breadth-first search.
- Dynamic shortest path.
- Approximate shortest path.
- Parallel single-source shortest path.

Ensuring Consistency
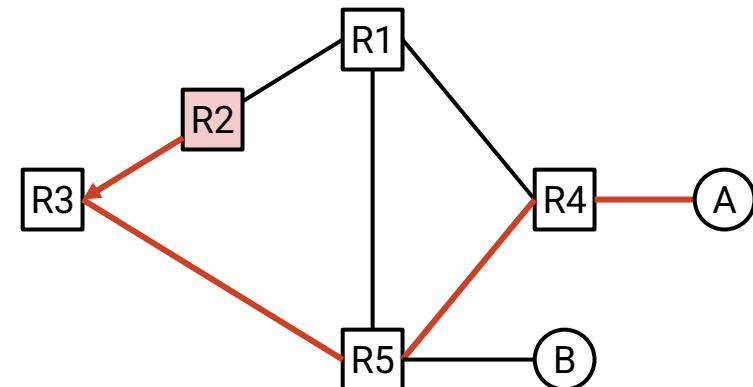
Each router can only influence its next hop.

No guarantee that routers compute the same paths!

We have to ensure that every router is using a "compatible" approach.

R3 forwards to R2.

R2 forwards to R3.

Requirements for routers to produce valid, compatible decisions:

1. Everyone agrees on the network topology.
2. Everyone is minimizing the same cost metric.
3. All costs are positive.
4. All routers use the same tie-breaking rules.

Routers don't necessarily need to use the same shortest-path algorithm, as long as they follow these rules.

- In practice, easier to have everyone use the same algorithm.

Learning Graph Topology

Lecture 6, CS 168, Spring 2026

Link-State Protocols

Link-state protocols:
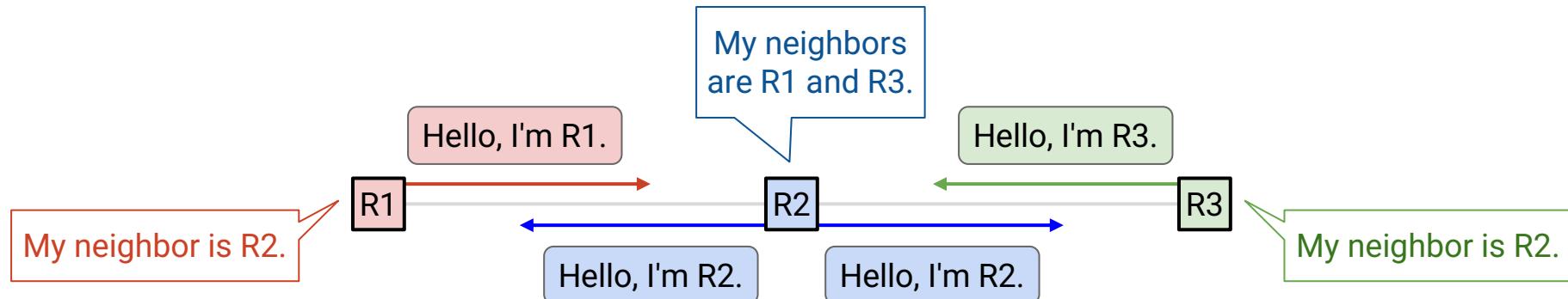
1. Every router learns the full network graph.   ← How do routers learn this?
   - Step 1A: Discover my neighbors.
   - Step 1B: Tell everybody about my neighbors.
2. Then, each router runs a shortest-path algorithm on the graph to populate the forwarding table.

How do we discover who is adjacent to us and their identity? Say hello!
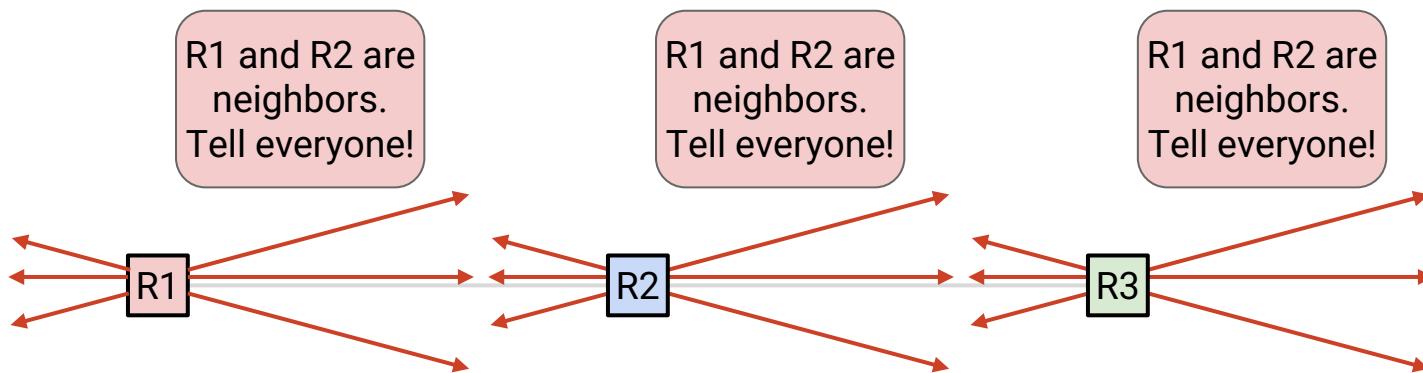
- Routers periodically send *hello* messages to their neighbors.
- If they stop saying hello, assume that they disappeared.
- This helps us learn about our direct neighbors, but not the whole network.
  - R1 does not know about the rest of the network (e.g. R3).

How do we learn about the rest of the network, beyond our neighbors?
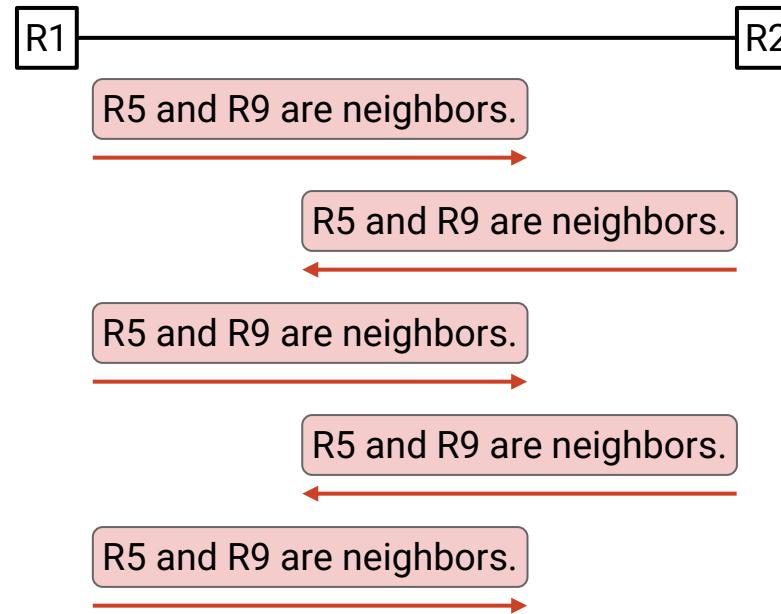
- Solution: **Flood** information across the network.
- When local information changes, send it to everyone.
- When you receive information from your neighbor, send it to everyone.

Avoiding Infinite Flooding
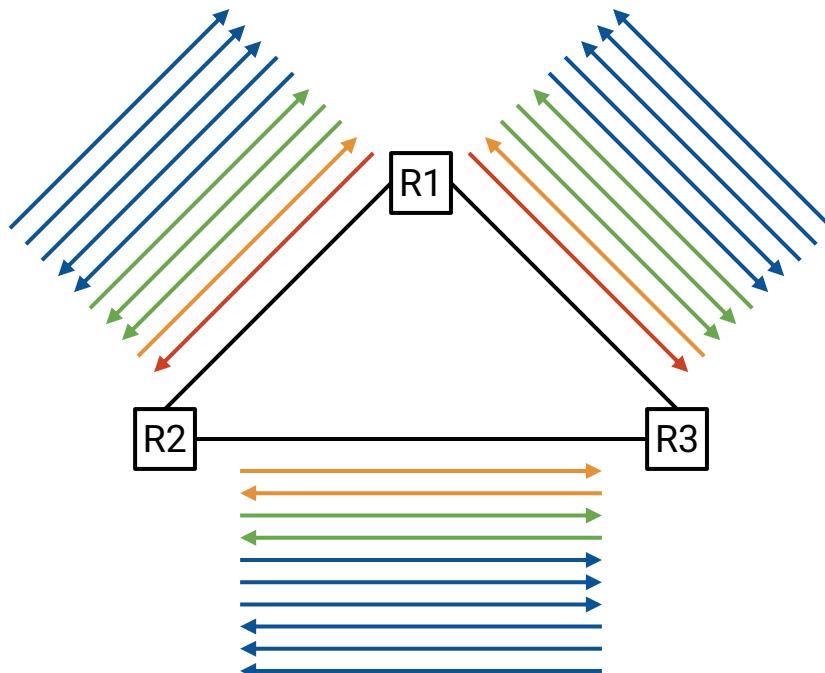
Flooding: When you receive an update, send it to everybody.

We have to be careful of the same update being sent repeatedly.

Amplification: When there's a loop, copies of the same message get multiplied.
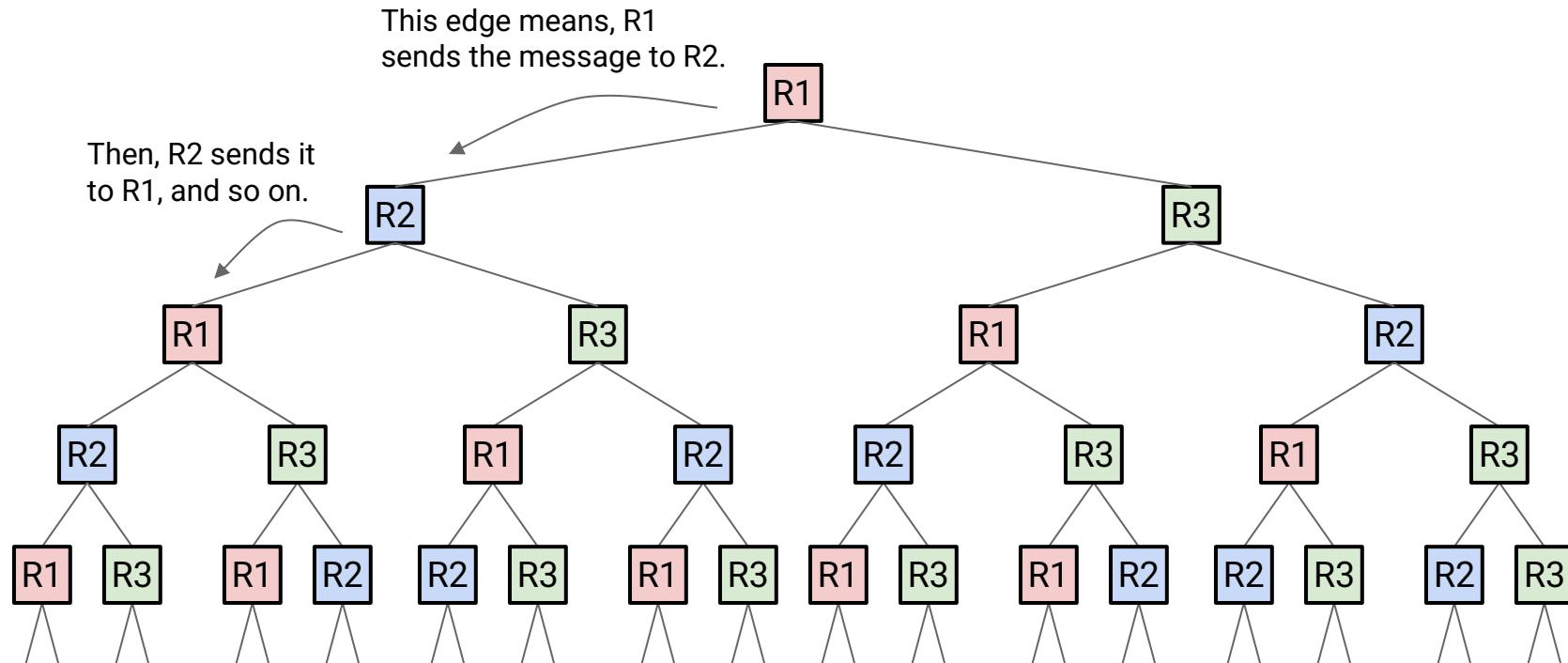
# Avoiding Infinite Flooding

Amplification: When there's a loop, copies of the same message get multiplied.

This edge means, R1 sends the message to R2.

Then, R2 sends it to R1, and so on.

Avoiding Infinite Flooding
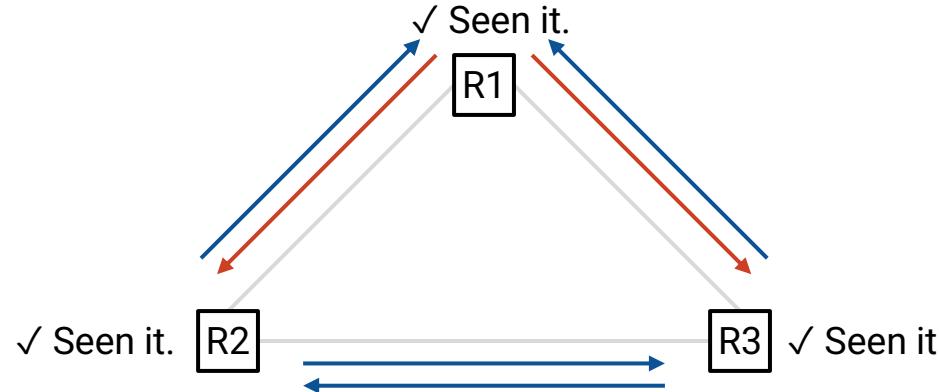
Problem: Naive solution (send to all neighbors) causes amplification.

Solution:

- When local information (about yourself) changes, send it to all neighbors.
- When you receive a packet from a neighbor, send it to all neighbors...
- ...unless you've already seen the packet before.

To identify packets you've seen before, add a timestamp.

- Or some other unique identifier, e.g. strictly increasing sequence numbers.

✓ Seen it.

R1

✓ Seen it. R2

R3 ✓ Seen it.

Ensuring Reliability

The network is still best-effort. Updates could get dropped.

- Recall: We need all routers to agree on topology, or else paths might be invalid.

Solution: Periodically re-send the update.
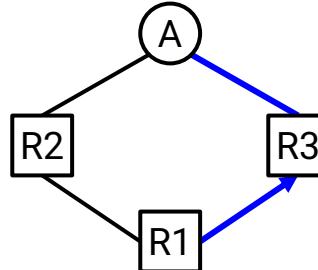
Ensuring Convergence

The network could change. What happens if a link goes down?

- Wait for routers to detect the failure.
- Wait to flood new information.
- Wait to recompute paths.
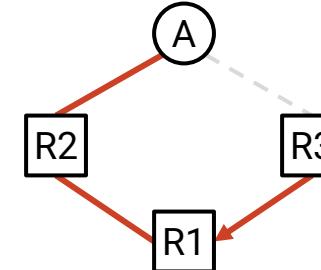
While waiting for convergence, the routing state might be invalid.

- Dead ends and loops. Packets using longer routes.

Link is down, but R1 doesn't know!

R1 forwards to R3.

R3 knows about the link failure!

R3 forwards to R1.

Link-State vs. Distance-Vector

Link-state is relatively simple. All the complexity is in the details!

- Everyone floods link/destination information.
- Everyone has a global map of the network.
- Everyone independently computes next-hops.

Why might we want to use link-state over distance-vector?

- Link-state gives routers more control over the path they choose.
  - Distance-vector: We have to trust what our neighbor says, and we don't know the path they're using.
- Link-state might be better at converging.
  - Distance-vector: Wait for neighbor to recompute and re-advertise path.
  - Link-state: Flood information before recomputing.

Real networks often use a combination of path/distance-vector and link-state.

Hierarchical Addressing

Lecture 6, CS 168, Spring 2026

Scaling Routing

Can we really scale routing and forwarding to every host on the Internet?

- Distance-vector: How many routing calculations does each router do?
- Link-state: Can we really store the entire network graph in every router?
- How long does it take to re-converge?

The secret to scaling routing is how we do addressing.

The trick: Use more informative names than A, B, C, D for destinations.

R3's Table	
Destination	Port
A	0
B	1
C	1
D	2
...	...

One entry for every possible destination.
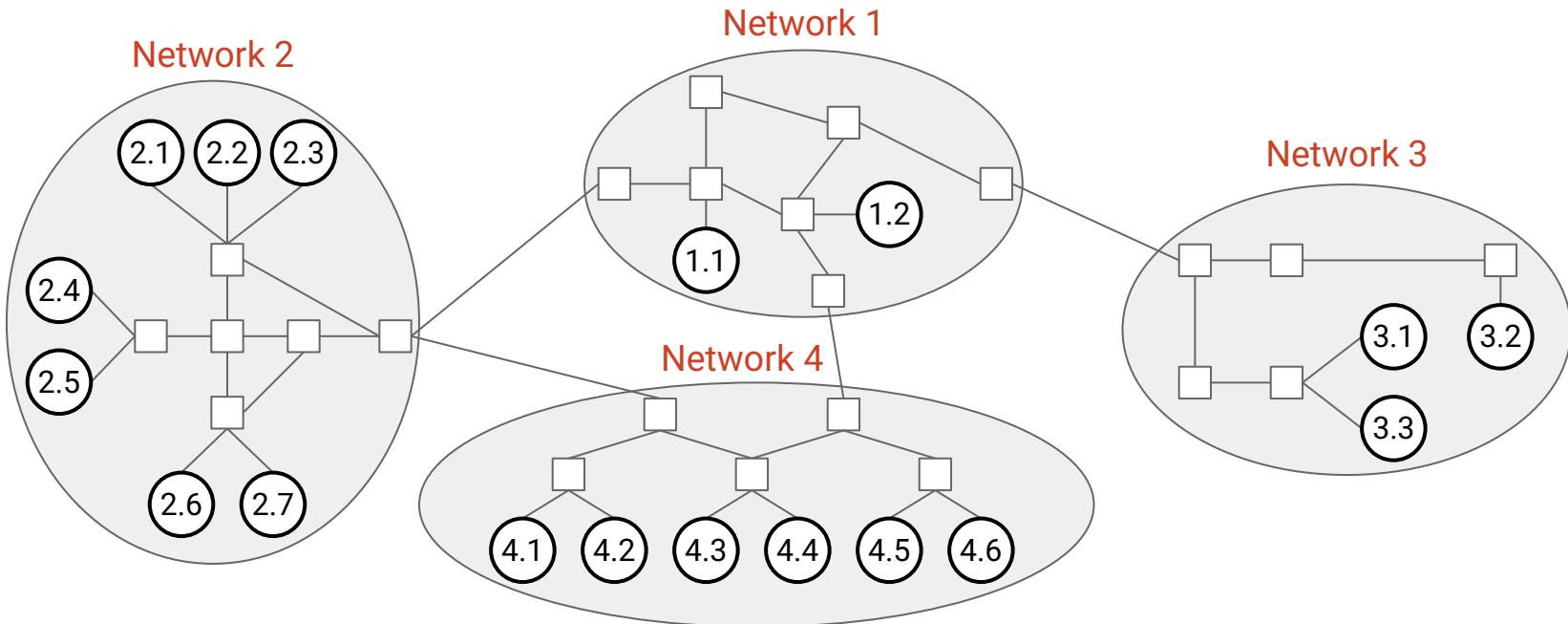
Disclaimer: Today is about IP (Layer 3) addresses.
Other layers use different addressing systems.

Recall: The Internet is a network of networks.

- Leads naturally to a hierarchy of addresses.
- Each network gets a number. Then, each host gets a number inside the network.
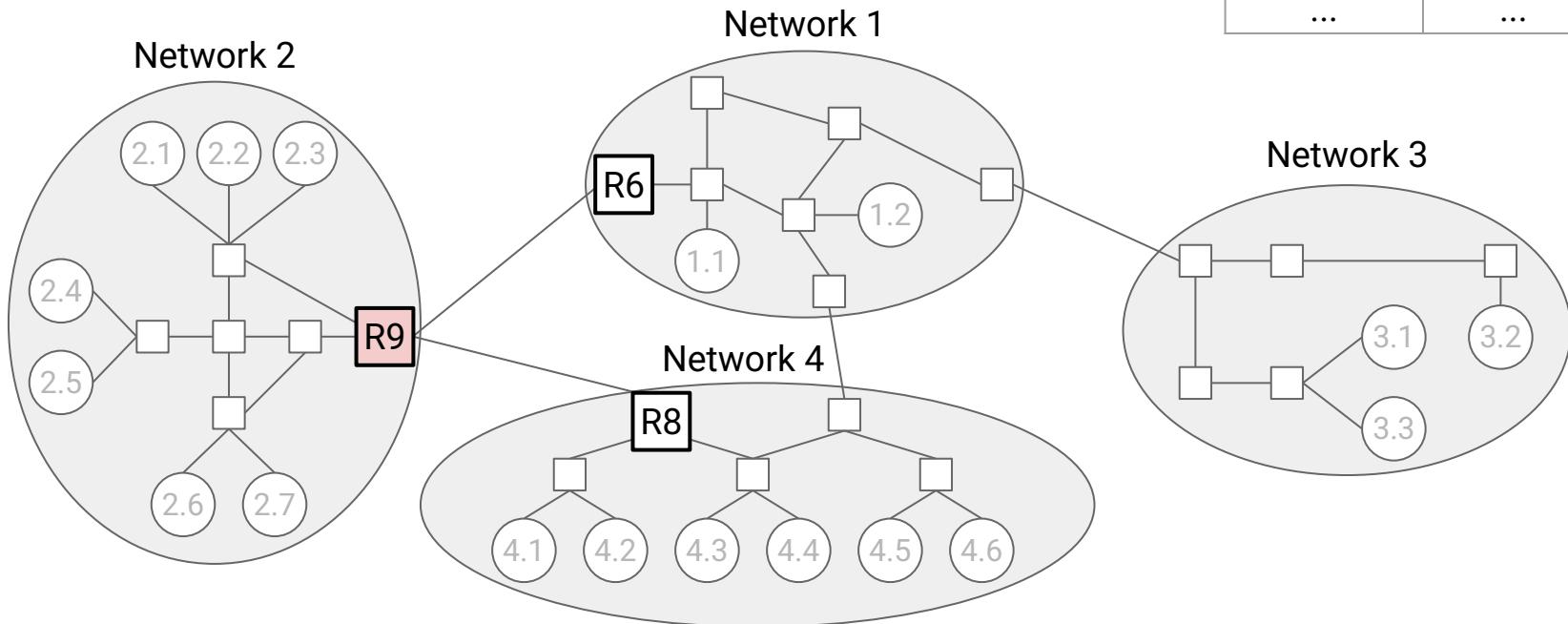
Hierarchical Addressing − Conceptual

R9 can summarize all hosts in another network with a single table entry.

Huge scaling improvement! Tables are smaller now.
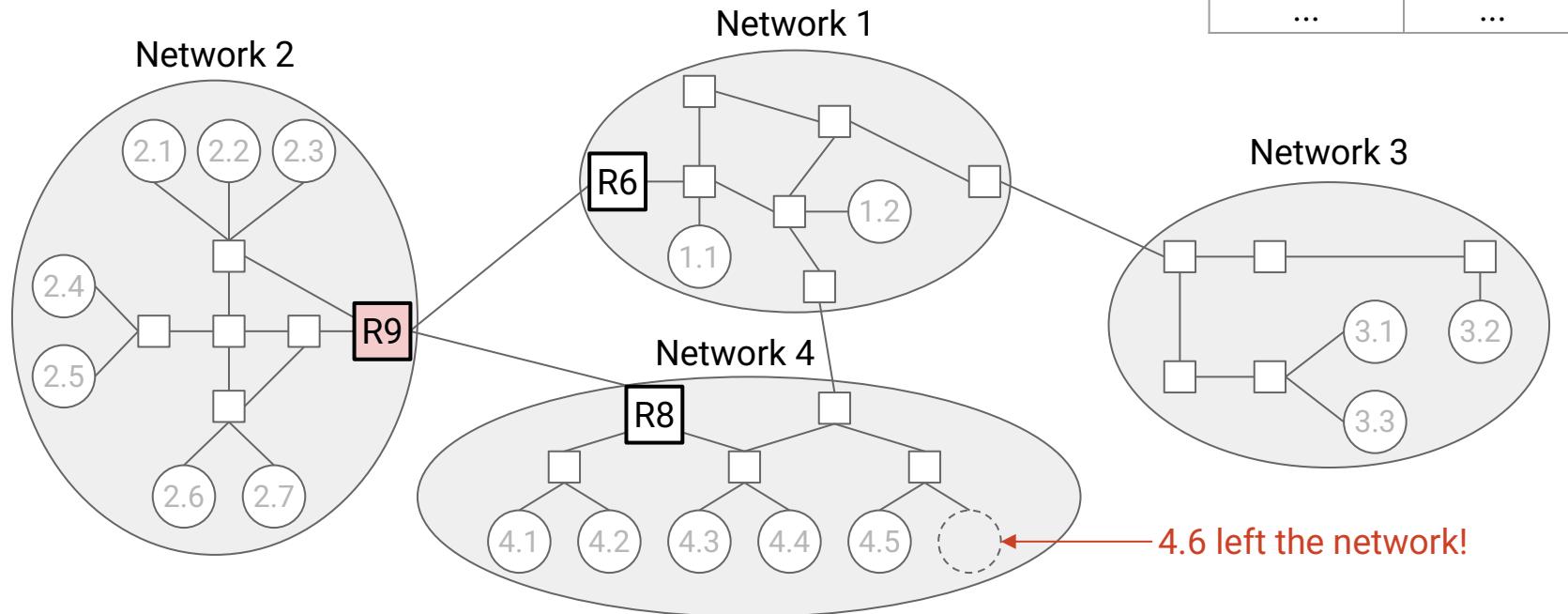
R9's Table	
Destination	Next Hop
1.*	R6
3.*	R6
4.*	R8
...	...

Hierarchical Addressing − Conceptual

Hierarchical addressing limits *table churn*.
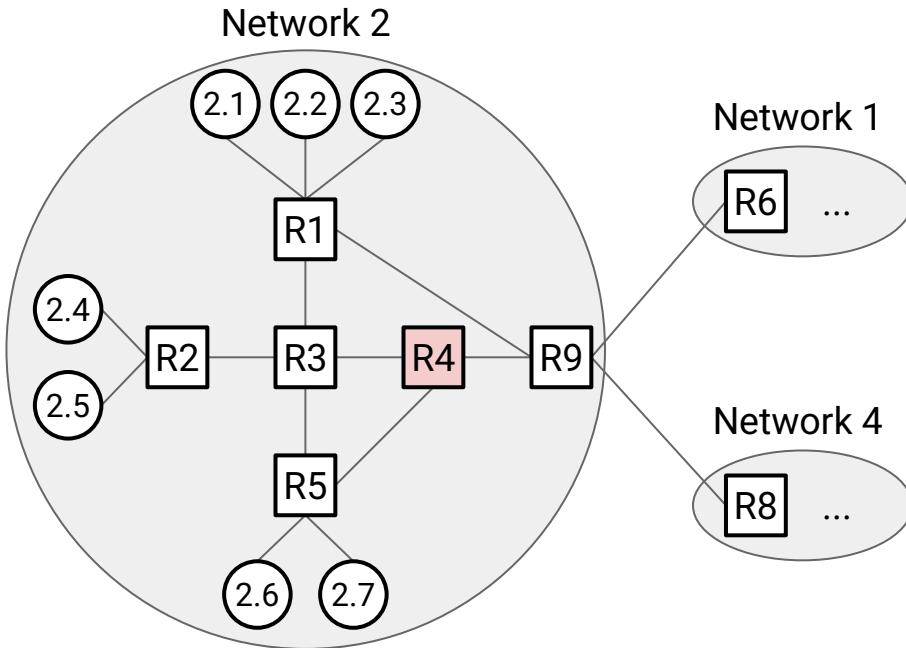
Changes inside a network don't affect tables in other networks.

R9's Table	
Destination	Next Hop
1.*	R6
3.*	R6
4.*	R8
...	...

No change in table entry! →

Network 2

Network 1

Network 3

Network 4

4.6 left the network!

Hierarchical Addressing – Conceptual

Size of the forwarding table scales with number of hosts in the current network, plus number of other networks.

- Big scaling improvement! Much better than listing every single host.

R4's Table	
Destination	Next Hop
2.1	R3
2.2	R3
2.3	R3
2.4	R3
2.5	R3
2.6	R5
2.7	R5
1.*	R9
3.*	R9
4.*	R9

Internal destinations.

External destinations.

Implications of Hierarchical Addressing

Inter-domain routing computes routes between networks.

- No need to think about churn inside networks.
- Just think about network 1, network 2, etc.

Intra-domain routing computes routes inside a network.

- No need to think about churn in other networks.
- Just think about 2.1, 2.2, etc.

R4's Table	
Destination	Next Hop
2.1	R3
2.2	R3
2.3	R3
2.4	R3
2.5	R3
2.6	R5
2.7	R5
1.*	R9
3.*	R9
4.*	R9
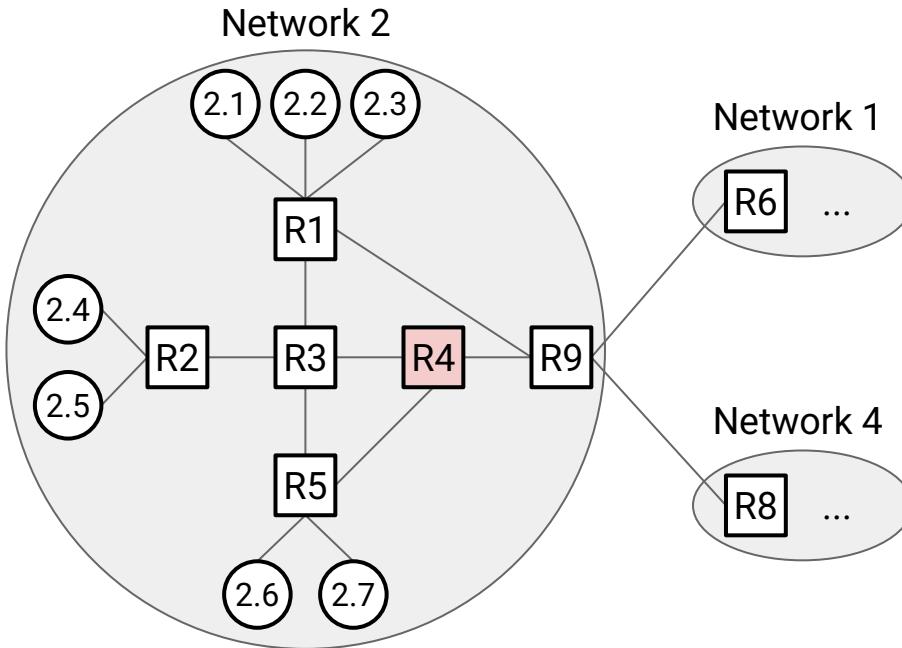
Internal destinations.

External destinations.

Aggregation – Conceptual

Sometimes, we can *aggregate* several rows into a single row.

- From R4, any packet to an external network has a next-hop of R9.
- *.* wildcard says: For any destination not in the table, forward to R9.
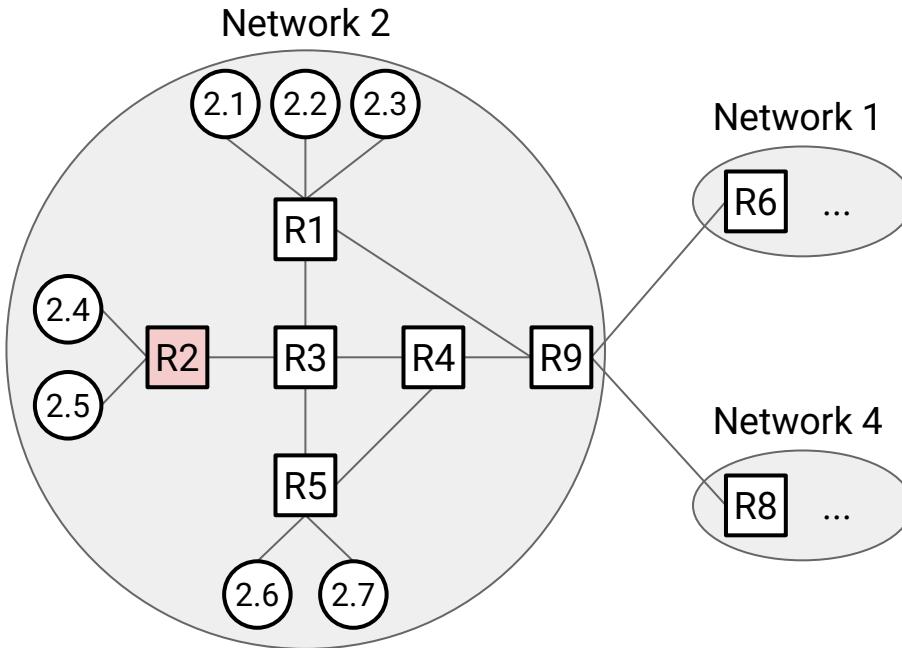
R4's Table	
Destination	Next Hop
2.1	R3
2.2	R3
2.3	R3
2.4	R3
2.5	R3
2.6	R5
2.7	R5
1.*	R9
3.*	R9
4.*	R9

R4's Table	
Destination	Next Hop
2.1	R3
2.2	R3
2.3	R3
2.4	R3
2.5	R3
2.6	R5
2.7	R5
.	R9

Aggregation – Conceptual

From R2, everything is reached through R3, so we can aggregate entries.

- The *.* wildcard (everything not in the table) is the **default route**.
- Most hosts only have the default route!

R2's Table	
Destination	Next Hop
2.4	Direct
2.5	Direct
2.1	R3
2.2	R3
2.3	R3
2.6	R3
2.7	R3
1.*	R3
3.*	R3
4.*	R3

R2's Table	
Destination	Next Hop
2.4	Direct
2.5	Direct
.	R3

Assigning Addresses

Lecture 6, CS 168, Spring 2026

Assigning Addresses

Hierarchical addressing makes routing scalable.

- Hosts that are "close to each other" (in some sense) share part of their address.
- Analogy: All third-floor room numbers start with the digit 3.

When a host joins the network, it's assigned an IP address based on its location in the network.

- Analogy: When you move to a new house, your address changes.
- How do we assign addresses to hosts?

Addresses are 32 bits long.

- Top 8 bits = network ID.
- Bottom 24 bits = host ID.

Each organization gets a unique network ID.

- AT&T = ID 12.
- Apple = ID 17.
- Ford = ID 19.
- US Department of Defense = IDs 6, 7, 11, 21, 22, 26, 28, 29, 30, 33, 55, 214, 215.

Bob's address: 11010110 10000100 00111010 01101110

Joe's address: 11010110 10010001 00000000 01001101

Same network ID, so they must be in the same network.

Network ID ◄——————— Host ID ———————►

Addresses are 32 bits long.

- Top 8 bits = network ID.
- Bottom 24 bits = host ID.

Problems:

- Only 2^8 = 256 different network IDs.
  - And we already gave the US Department of Defense 13 of them.
- Imagine a tiny network (Joe's Tire Shop) with 10 hosts.
  - If we give them an ID, we're giving them 2^{24} = 16,777,216 addresses!
- We're going to run out!

Idea: Allocate different network sizes based on need.

- The top bits (`0`, `10`, or `110`) tell us how to split up the rest of the bits.

Class A:

`0`	Network (7 bits)	Host (24 bits)

~128 networks   ~16m hosts per network

Class B:

`10`	Network (14 bits)	Host (16 bits)

~16k networks   ~65k hosts per network

Class C:

`110`	Network (21 bits)	Host (8 bits)

~2m networks   ~256 hosts per network

Assigning Addresses, Attempt 2/3 − Classful Addressing

The top bits indicate this is Class B...

...so read the next 14 bits as the network ID...

...and read the last 16 bits as the host ID.

Bob's address: 10010110 10000100 00111010 01101110

Joe's address: 10010110 10000100 00000000 01001101

←——— Network ID ———→ ←——— Host ID ———→

Class A:	0	Network (7 bits)	Host (24 bits)	
Class B:	10	Network (14 bits)		Host (16 bits)
Class C:	110	Network (21 bits)		Host (8 bits)

- Class A: ~128 networks.   ~16m hosts per network.
- Class B: ~16k networks.   ~65k hosts per network.
- Class C: ~2m networks.   ~256 hosts per network.

Problems:

- Class A is way too big for most organizations.
- Class C is way too small for most organizations.
- Class B is the best option for many.
  - ~65k hosts is still too big for most organizations.
  - ~16k is still not enough networks. We're running out again!

Number of inter-domain routes by year (approximate)

With classful addressing, we tried to use convenient 8-bit boundaries.

- Class A: 8 bits for class/network.   24 bits for host.
- Class B: 16 bits for class/network.   16 bits for host.
- Class C: 24 bits for class/network.   8 bits for host.

What if we could assign network IDs of any length?

- Leads to **CIDR (Classless Inter-Domain Routing)**.
- This is what the Internet uses today.

Class A:	0	Network (7 bits)	Host (24 bits)	
Class B:	10	Network (14 bits)	Host (16 bits)	
Class C:	110	Network (21 bits)	Host (8 bits)	

Suppose Joe's Tire Shop has 450 hosts.

Classful addressing:

- Class C gives us ~256 hosts. Not enough!
- Class B gives us ~65k hosts. Too many!
- We have to use Class B (and waste tons of addresses).

Classless (CIDR) addressing:

- 8 host bits = 256 bits. Not enough!
- 9 host bits = 512 bits.
- Still not exactly 450, so still some wasted addresses.
- But much less wasteful!

Network ID (23 bits)	Host ID (9 bits)

Granular Hierarchical Assignment with CIDR

CIDR enables multi-layered hierarchical assignment of addresses.

- ICANN. *(Internet Corporation for Names and Numbers)*
  - Top-level organization that owns all the IP addresses.
  - They allocate blocks to...
- RIRs. *(Regional Internet Registries)*
  - Representing Europe *(RIPE)*, North America *(ARIN)*, Asia/Pacific *(APNIC)*, South America *(LACNIC)*, and Africa *(AFRINIC)*.
  - They give out portions to...
- Large organizations or ISPs.
  - Sometimes called Local Internet Registries *(especially in Europe)*.
  - They give out portions to...
- Small organizations and individuals.
  - Examples: UC Berkeley, Joe's Tire Shop.

Granular Hierarchical Assignment with CIDR

ICANN owns all addresses: ....

ARIN (North America) owns: 1101............................
4 bits fixed, $2^{28} \approx 268m$ addresses.

AT&T (large ISP) owns: 110111001.......................
9 bits fixed, $2^{23} \approx 8m$ addresses.

UC Berkeley owns: 110111001110100010..............
18 bits fixed, $2^{14} \approx 16k$ addresses.

Soda Hall owns: 110111001110100010011010........
24 bits fixed, $2^8 \approx 256$
addresses.

Prof. Ratnasamy owns: 110111001110100010011010001011101
All bits fixed, 1 address.

Writing Addresses

Lecture 6, CS 168, Spring 2026
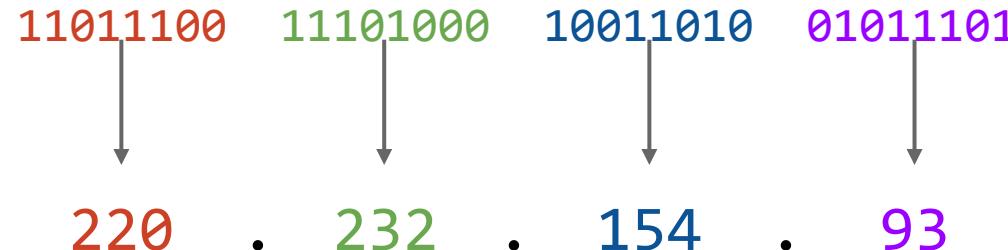
We could write an IP address as a 32-bit number: `11011100111010001001101001011101`

- Nobody wants to read that.

Dotted quad notation:

- Split into groups of 8 bits.
- Write each 8-bit number in decimal, separated by dots.

```
11011100   11101000   10011010   01011101
```

```
  220   .   232   .   154   .   93
```
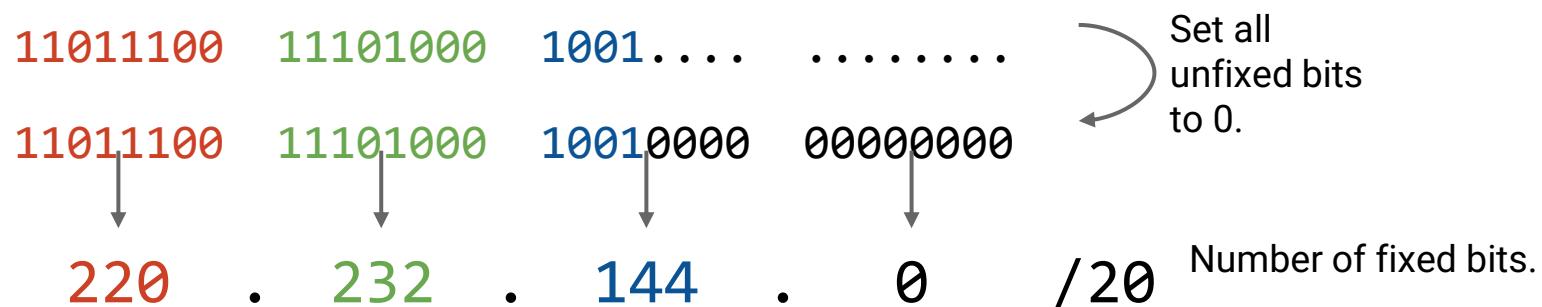
Writing IPv4 Address Ranges

Slash notation for writing ranges of addresses:

- Set all unfixed bits to 0. Write as a dotted quad.
- After the slash, write how many bits are fixed.

Examples:
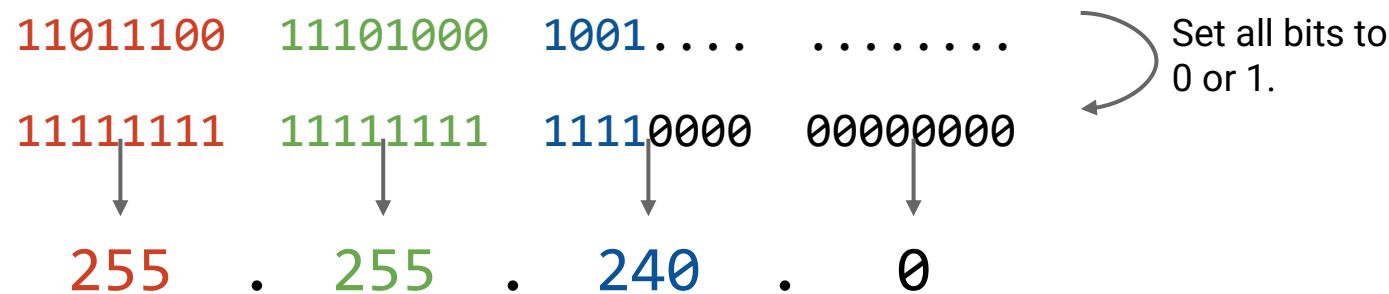
- 192.168.1.0/24 means 24 bits are fixed. Range is 192.168.1.0 – 192.168.1.255.
- 192.168.1.0/29 means 29 bits are fixed. Range is 192.168.1.0 – 192.168.1.7.
- 192.168.1.1/32 means 32 bits are fixed. A single address.
- 0.0.0.0/0 means no bits are fixed. All addresses.

```
11011100   11101000   1001....   ........        Set all
                                                  unfixed bits
11011100   11101000   10010000   00000000         to 0.

  220    .   232    .   144    .     0      /20    Number of fixed bits.
```

Writing IPv4 Address Ranges

Netmask is an alternative to the number after the slash..

- Set all *fixed* bits to 1, and all *unfixed* bits to 0. Write as a dotted quad.
- Useful in code: To extract network ID, bitwise AND the address and netmask.
- Slash notation is more convenient for humans.

```
11011100  11101000  1001....  ........        Set all bits to
                                                0 or 1.
11111111  11111111  11110000  00000000

   255   .   255   .   240   .    0
```

Slash notation: `220.232.144.0/20`

Netmask notation: `220.232.144.0` (netmask `255.255.240.0`)

Aggregating Routes

Lecture 6, CS 168, Spring 2026

Inter-Domain Routing

In inter-domain routing, we're looking for routes to other networks.

- Each destination is a network, represented by an IP prefix.
- Already better than one entry per host...but we can do even better!
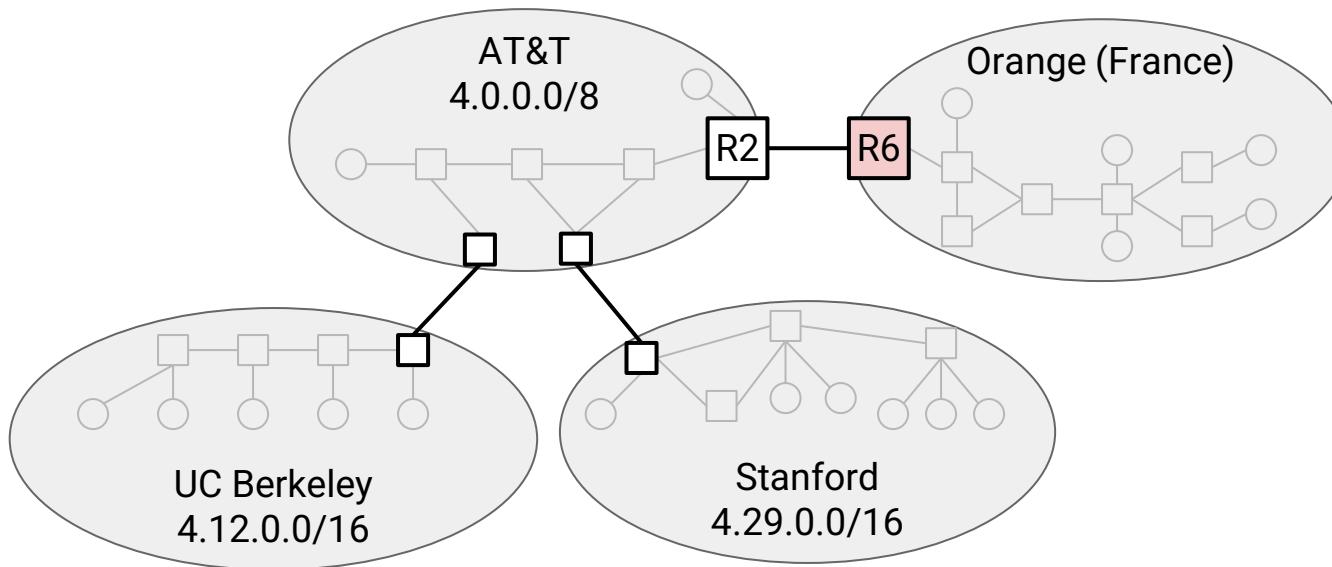
R6's Table	
Destination	Next Hop
4.0.0.0/8	R2
130.0.0.0/8	R2
146.0.0.0/8	R2
...	...

Inter-Domain Routing

With CIDR, AT&T allocates parts of its ranges to UC Berkeley and Stanford.

- We can aggregate many networks into a single entry!
- 4.0.0.0/8 represents AT&T, and all its subordinates.
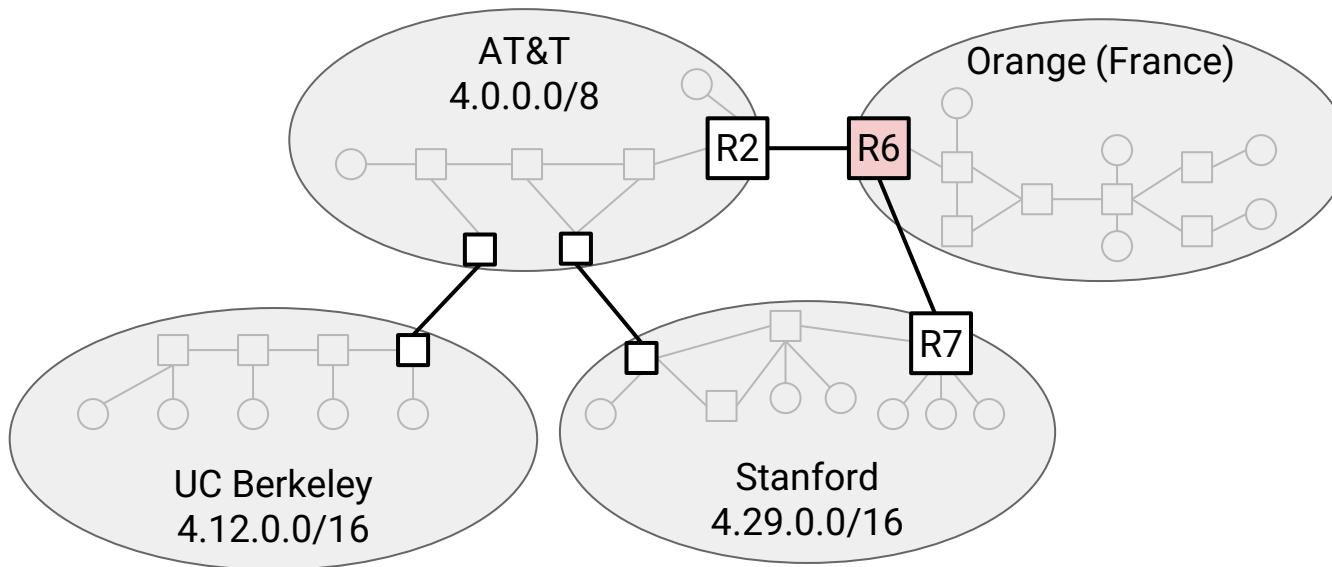
R6's Table	
Destination	Next Hop
4.0.0.0/8	R2
4.12.0.0/16	R2
4.29.0.0/16	R2
...	...

Inter-Domain Routing

Now, Stanford wants to connect to multiple networks. This is called **multi-homing**.

- We need an entry for both AT&T and Stanford (one of its subordinates).
- Multi-homing limits aggregation!
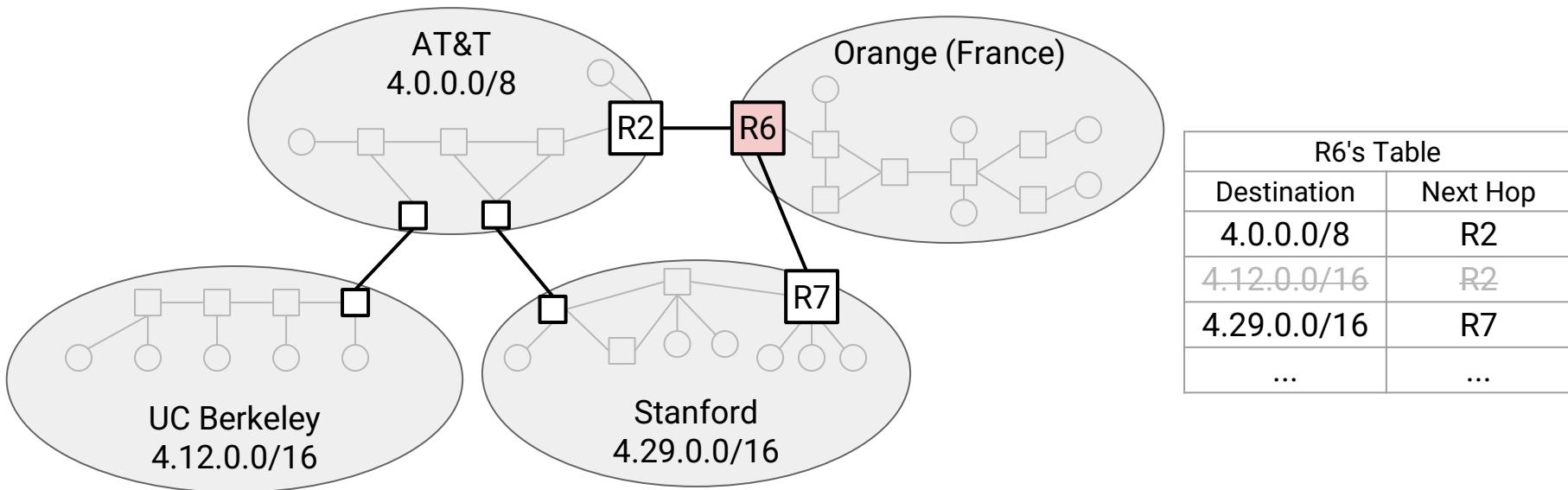
R6's Table	
Destination	Next Hop
4.0.0.0/8	R2
4.12.0.0/16	R2
4.29.0.0/16	R7
...	...

Ranges in the table might overlap.

- **Longest prefix matching**: If a destination matches many ranges, pick the most specific range.
- Example: 4.29.1.2 matches both 4.0.0.0/8 and 4.29.0.0/16 (more specific).

R6's Table	
Destination	Next Hop
4.0.0.0/8	R2
4.12.0.0/16	R2
4.29.0.0/16	R7
...	...

IPv6 Changes

Lecture 6, CS 168, Spring 2026
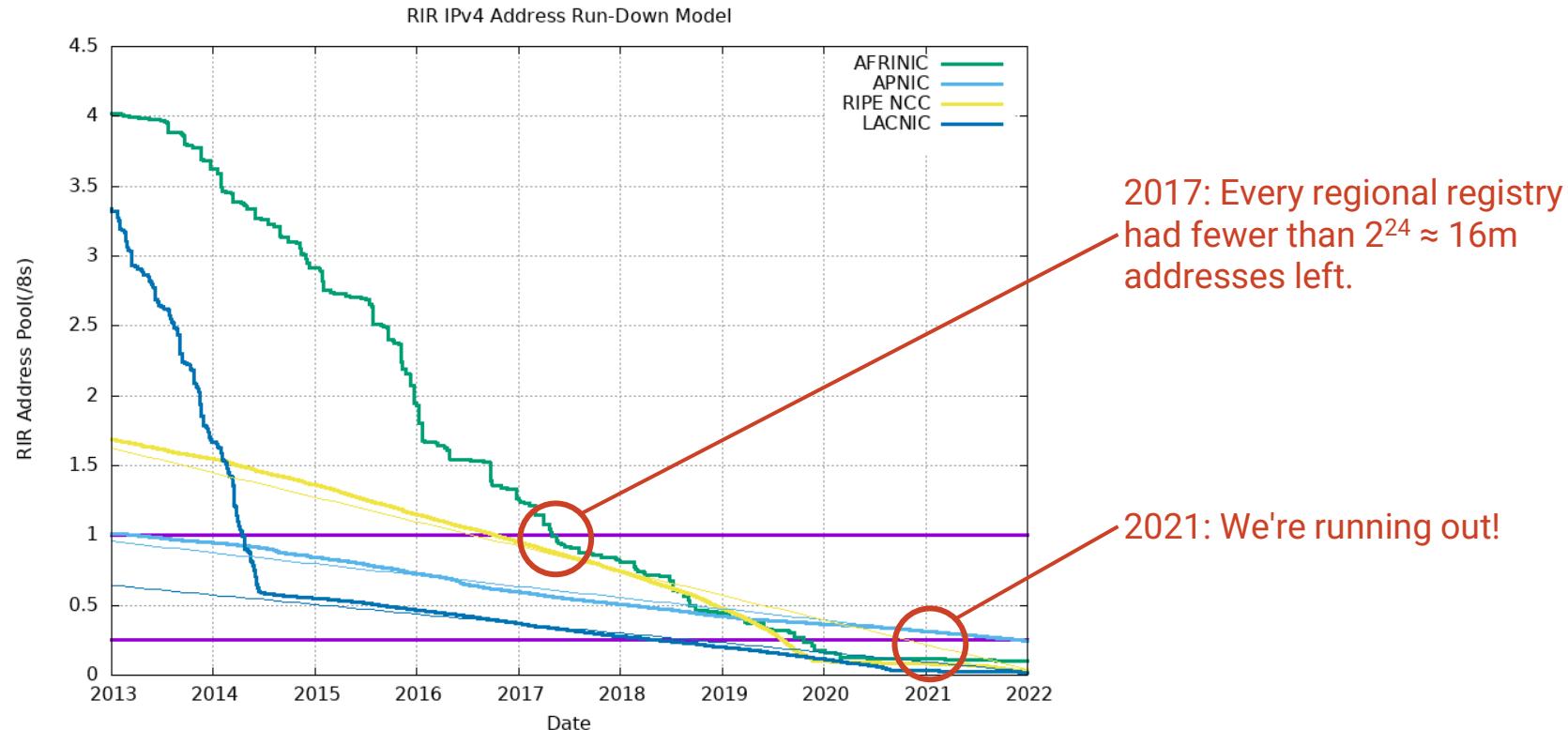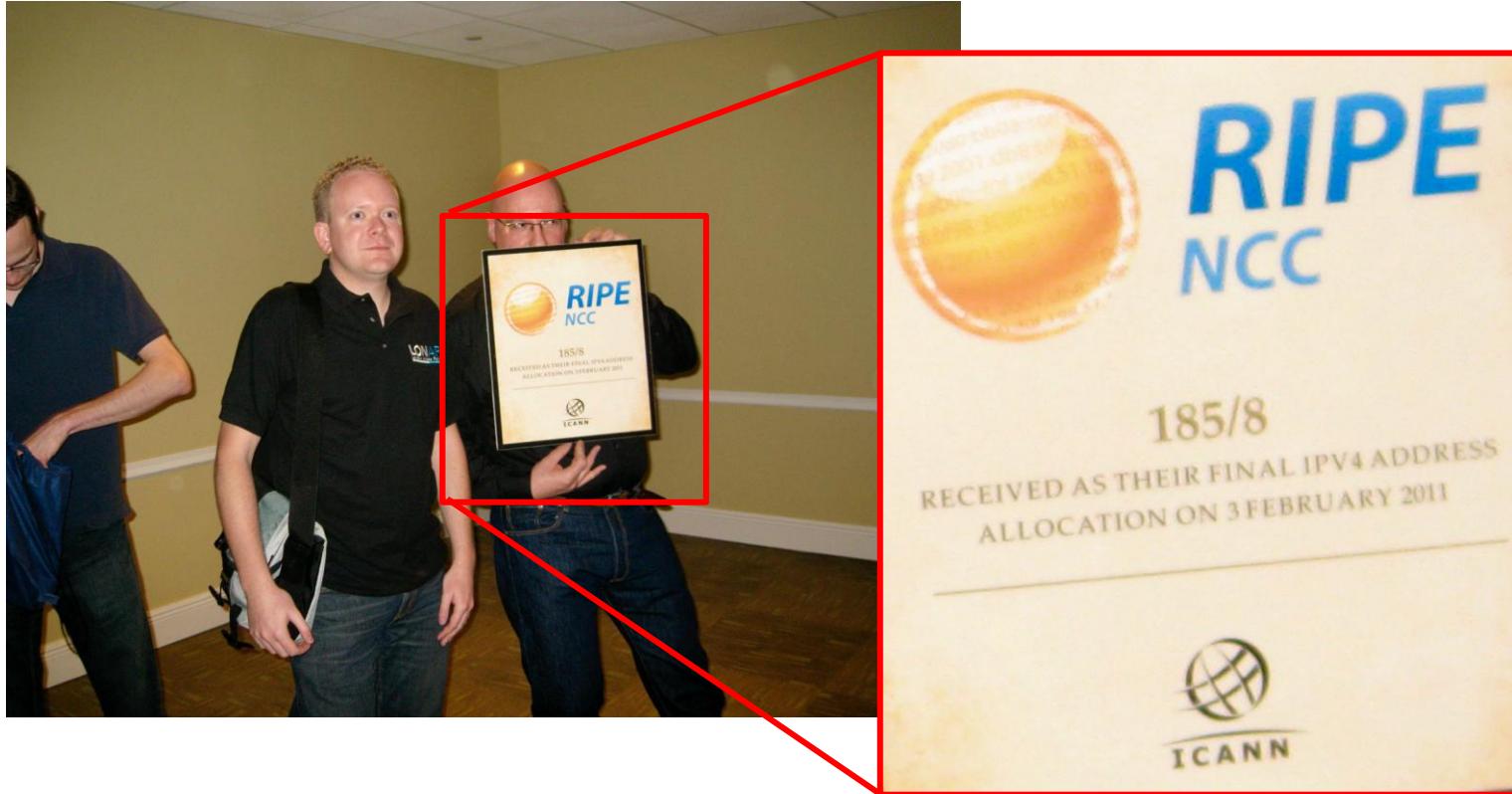
Running Out of IPv4 Addresses

IPv4 addresses are 32 bits long. $2^{32} \approx 4$ billion addresses. Is that enough?

RIR IPv4 Address Run-Down Model

2017: Every regional registry had fewer than $2^{24} \approx$ 16m addresses left.

2021: We're running out!

Running Out of IPv4 Addresses

A ceremony where the last range of IPv4 addresses was allocated. (February 2011)

Introducing IPv6

IPv6 was introduced in 1998 to deal with IPv4 address exhaustion.

- Main difference: Addresses are now 128 bits, instead of 32.
- Fundamentally, same addressing structure as IPv4.
- Some other minor changes (won't discuss here).

Is 128 bits enough?

- $2^{128} \approx 3.4 \times 10^{38}$ possible addresses.
- Assigning an address to every second of the universe's history uses 0.000000000000001% of the addresses.
- We won't run out again.

IPv5 was an experimental protocol from the 1990s. It was never widely implemented.

IPv6 Addresses

IPv6 uses hexadecimal instead of decimal.

- `2001:0DB8:CAFE:BEEF:DEAD:1234:5678:9012`
- Colon between every 4 hex digits (16 bits).

Shorthand:

- Omit leading zeros per block:
  `2001:0DB8:0000:0000:0000:0000:0000:0001` → `2001:DB8:0:0:0:0:0:1`
- Omit a long string of zeros, once per address:
  `2001:DB8:0:0:0:0:0:1` → `2001:DB8::1`

Can still use slash notation for ranges.

- `2001:0DB8::/32` has 32 bits fixed, and 2^{96} addresses.

IPv6 Addresses

IPv6 generally uses the same hierarchical addressing approach as IPv4.

Some changes:
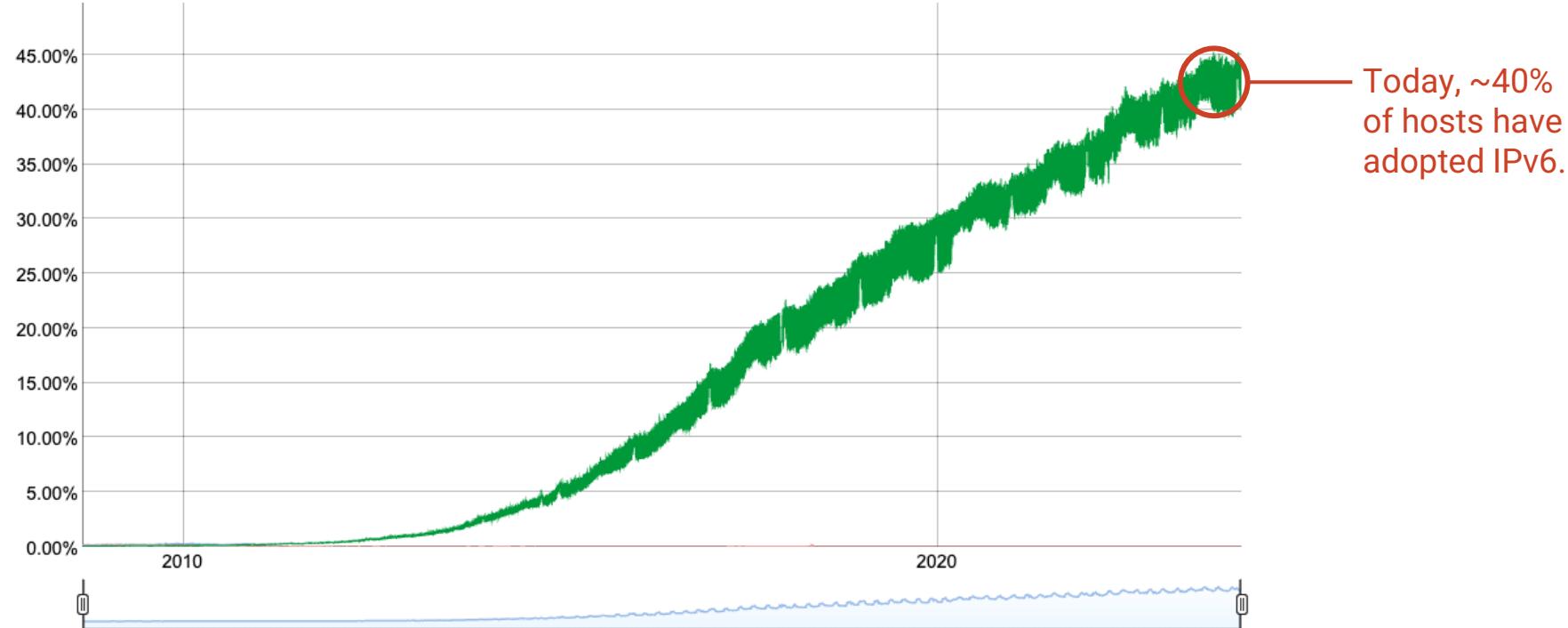
- Instead of someone (e.g. ICANN) giving you a prefix, you can pick your own prefix.
    - Uses a protocol called SLAAC *(Stateless Address Autoconfiguration)*.
    - Pick a random prefix. If someone else is using it, pick another one.
    - Works in IPv6 (not IPv4) because there are so many prefixes.
- In practice, prefixes usually fix at most 64 bits.
    - Even the smallest network has 2^{64} hosts.

IPv6 Adoption

IPv6 introduced in the 1990s, but wide adoption started in the 2010s.
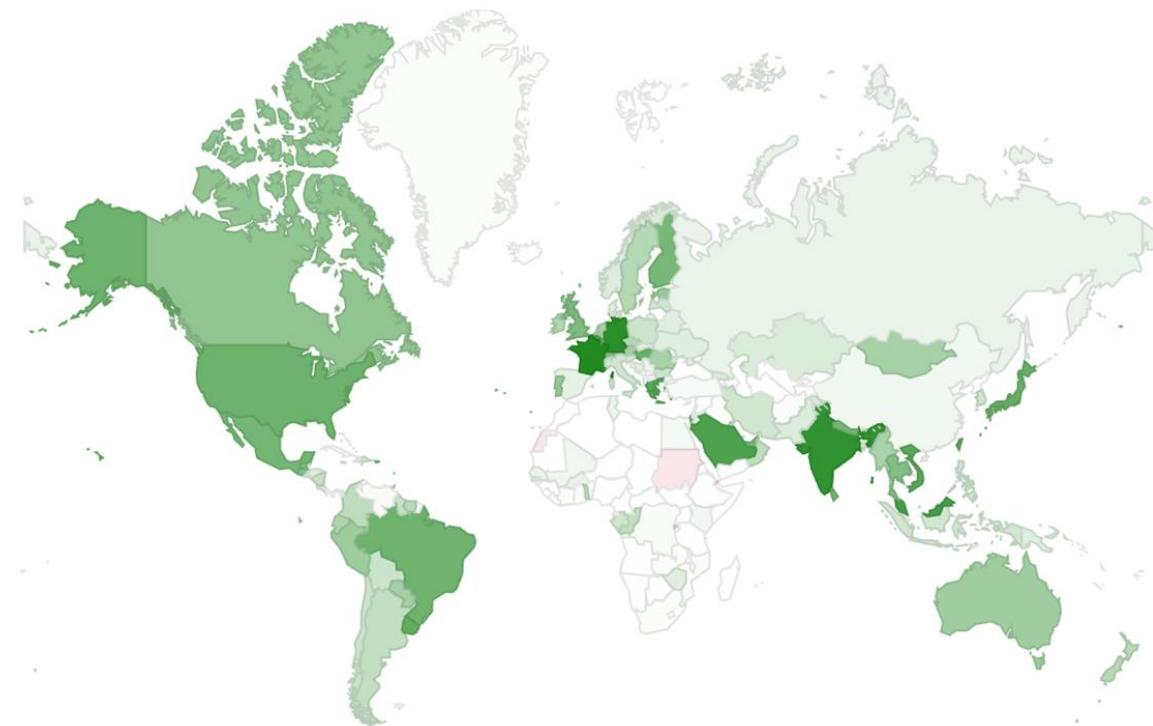
We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.

Native: 41.23% 6to4/Teredo: 0.00% Total IPv6: 41.23% | Jan 15, 2024

Today, ~40% of hosts have adopted IPv6.

IPv6 Adoption

Adoption generally correlated with areas where there are many Internet users.

Why is IPv6 adoption hard?

- Requires software and hardware upgrades, from both hosts and routers.
- IPv4 and IPv6 are not compatible with each other.
  - To support both, you need 2 forwarding tables.
  - No way to convert between IPv4 and IPv6.
- If I support both IPv4 and IPv6, which should I use?
  - IPv6 usually faster, but other factors could affect your choice.

Main driver for IPv6 adoption: We're running out of IPv4 addresses!

Summary: IP Addressing

- Hosts on the Internet have addresses (IPv4, IPv6, or both).
- These addresses are hierarchical.
  - They are assigned in groups to specific organizations.
- Wildcard matching means that this can help our forwarding and routing scale better.