Routing #3 / Addressing

Autumn 2024
cs168.io

Rob Shakir

Thanks to Murphy McCauley for some of the material!

Last Time

- We've talked a lot about *distance-vector* routing protocols.

Plan for today

- Types of routing protocols.
- Another type of protocol: *Link State.*
- Addressing – IPv4 + IPv6.

Link-State Protocols

Link-State Routing

- As mentioned, another major class of routing protocols.
- Very common as an *Interior Gateway Protocol*.

- Major examples:
    - IS-IS (Intermediate System to Intermediate System)
    - OSPF (Open Shortest Path First)

- Very different operation to Distance-Vector!

Distance-Vector vs. Link-State

- Distance-Vector
  - Global computation (distributed across all nodes)
  - Only local data (local node plus whatever our neighbours told us).

Distance-Vector vs. Link-State

- Distance-Vector
  - Global computation (distributed across all nodes)
  - Only local data (local node plus whatever our neighbours told us).

- Link-State
  - Local computation
  - Using global data (from all parts of the network)

Link-State

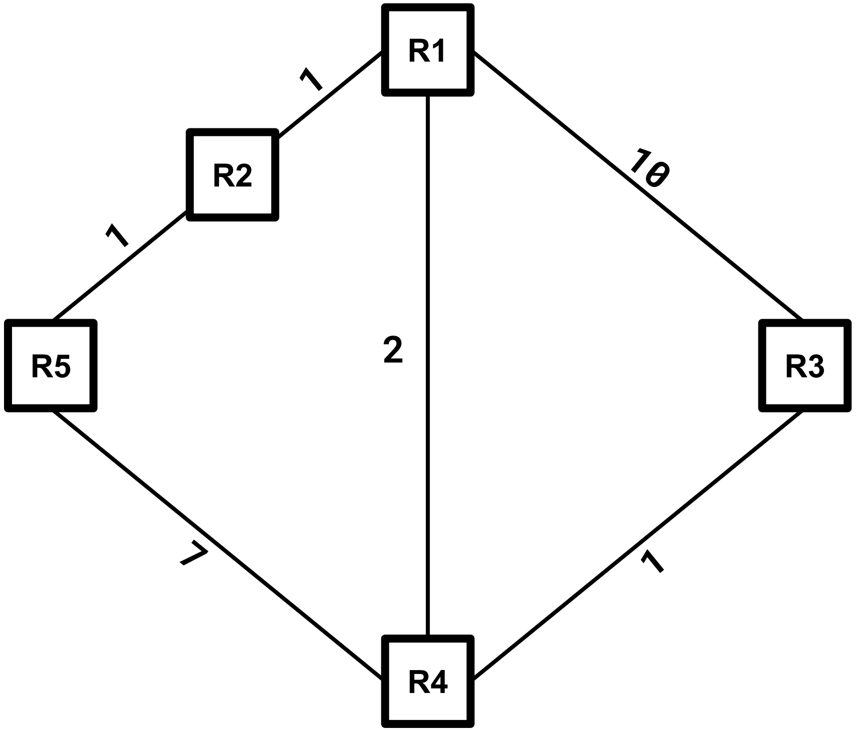
- A router locally computes routing state
- …using “global data” (?!)

- What is “global data”?
  - The state of every link in the network.
  - Is it up or down?
  - What is its cost?

Global Data in Link-State

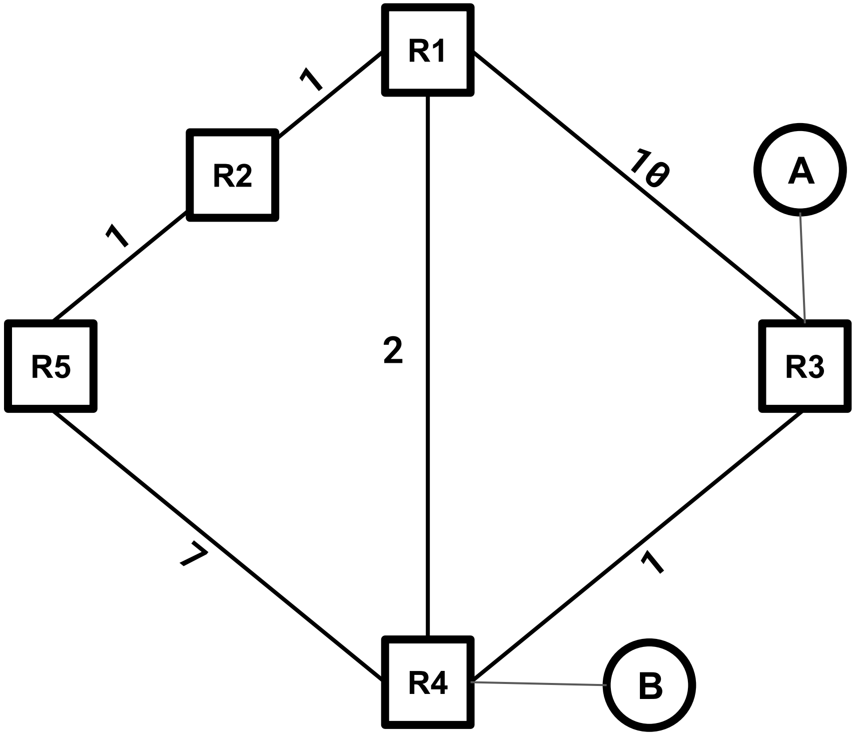
- Going back to our handy topology.

- Information about state of links:
  - R1-R2 exists, and has cost 1
  - R1-R3 exists, and has cost 10
  - R4-5 exists and has cost 7
  - etc.



Global Data in Link-State

- Going back to our handy topology.
- Information about state of links:
  - R1-R2 exists, and has cost 1
  - R1-R3 exists, and has cost 10
  - R4-5 exists and has cost 7
  - Etc.
- Information about destinations:
  - R3 has destination A
  - R4 has destination B



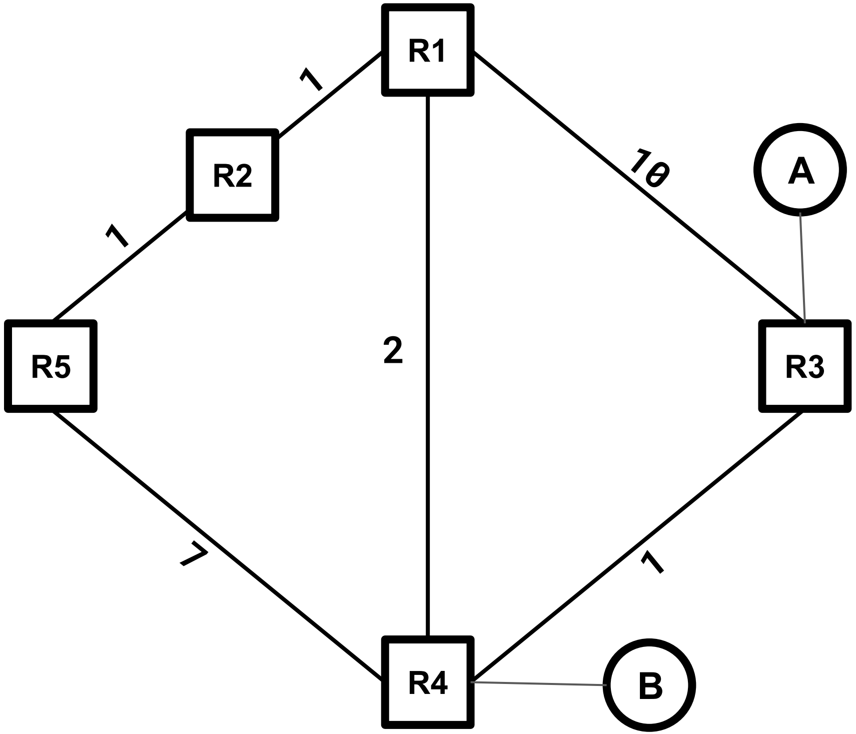
Global Data in Link-State

- Going back to our handy topology.

- Information about state of links:
  - R1-R2 exists, and has cost 1
  - R1-R3 exists, and has cost 10
  - R4-5 exists and has cost 7
  - Etc.

- Information about destinations:
  - R3 has destination A
  - R4 has destination B

- This can be used to build a *global view of* the topology.



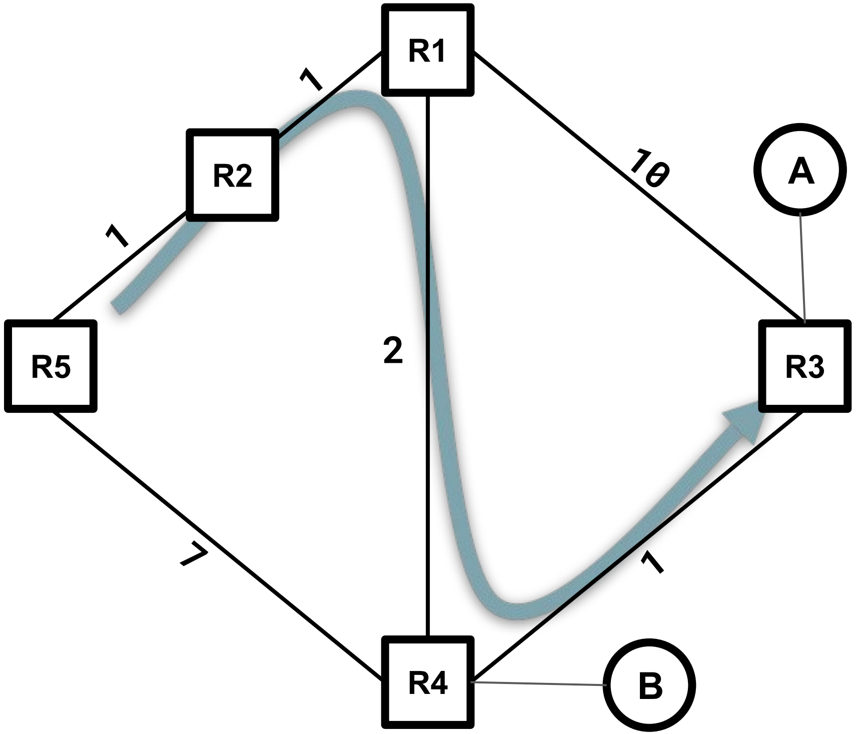
Global Data in Link-State

- With this global view, we can easily compute paths.

- If we're R5 – what's the best path to A?
  - R5, R2, R1, R4, R3, A

- What's useful to R5 for forwarding?
  - Only the next-hop → R2.

Dst	*Nxt*
A	R2



Questions?

Link-State: Overview

- Every router in the topology:
    - Gets the state of all links and the location of all destinations.
    - Uses that information to build a full graph.
    - Finds paths from itself to every destination on the graph.
    - Uses the next-hop (adjacent router) in those paths to populate the forwarding table.

Link-State: Overview

- Every router in the topology:
  - Gets the state of all links and the location of all destinations.
    - **Need some way to distribute this graph!**
  - Uses that information to build a full graph.
    - **Glue together all link/destination information received.**
  - Finds paths from itself to every destination on the graph.
    - **Run some algorithm over the graph.**
  - Uses the next-hop (adjacent router) in those paths to populate the forwarding table.

Link-State: Algorithms

- Since each router has the complete topology - we just need a Single Source Shortest Path algorithm.

- Some obvious choices:
  - Bellman-Ford (serial)
  - Dijkstra's algorithm

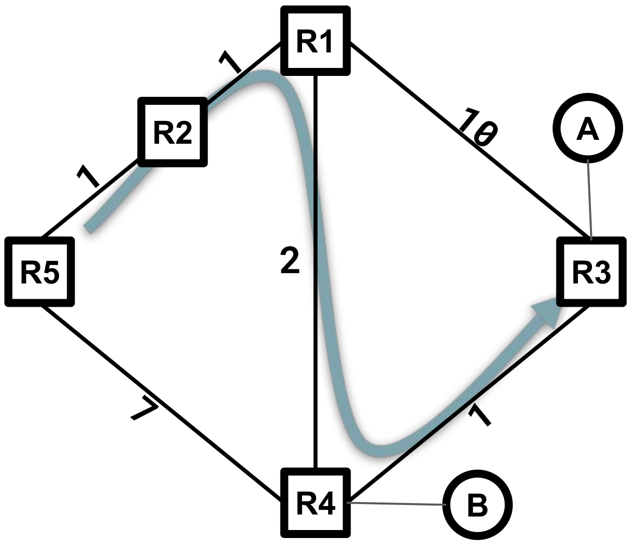
- Can we do better?
  - Breadth-first search
  - Dynamic shortest path
  - Approximate shortest path
  - Parallel SSSP

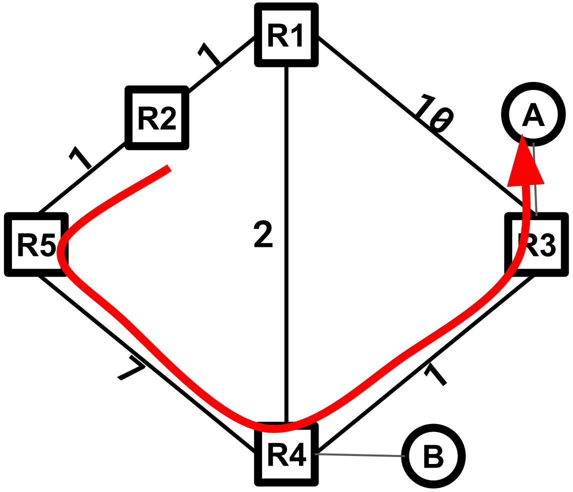
Link-State: Overview

- Every router in the topology:
  - Gets the state of all links and the location of all destinations.
    - **Need some way to distribute this graph!**
  - Uses that information to build a full graph.
    - **Glue together all link/destination information received.**
  - Finds paths from itself to every destination on the graph.
    - **Run some algorithm over the graph.**
  - Uses the next-hop (adjacent router) in those paths to populate the forwarding table.

Link-State: Populating Tables

- Remember: each router can only influence its own next-hop.
- Other routers must be using an approach which is "compatible".





Link-State: Populating Tables

- Remember: each router can only influence its own next-hop.

- Other routers must be using an approach which is “compatible”.

- Simple for least-cost routing if:
  - Minimising the same cost metric.
  - All costs are > 0.

Link-State: Populating Tables

- Remember: each router can only influence its own next-hop.

- Other routers must be using an approach which is “compatible”.

- Simple for least-cost routing if:
  - Minimising the same cost metric.
  - All costs are > 0.
  - **All routers agree on topology**.

- Given these, can have different algorithms (e.g., break ties the same).
  - Since we can guarantee no loops.

L-S: Learning about the topology

- We need to understand information about:
  - All links between all routers.
  - All destinations.

- We need to:
  - Discover who my neighbours are.
  - Tell everyone about my neighbours.
  - Tell everyone about destinations attached to me.

L-S: Learning about the topology

- We need to understand information aout:
  - All links between all routers.
  - All destinations.

- We need to:
  - **Discover who my neighbours are.**
  - Tell everyone about my neighbours.
  - Tell everyone about destinations attached to me.

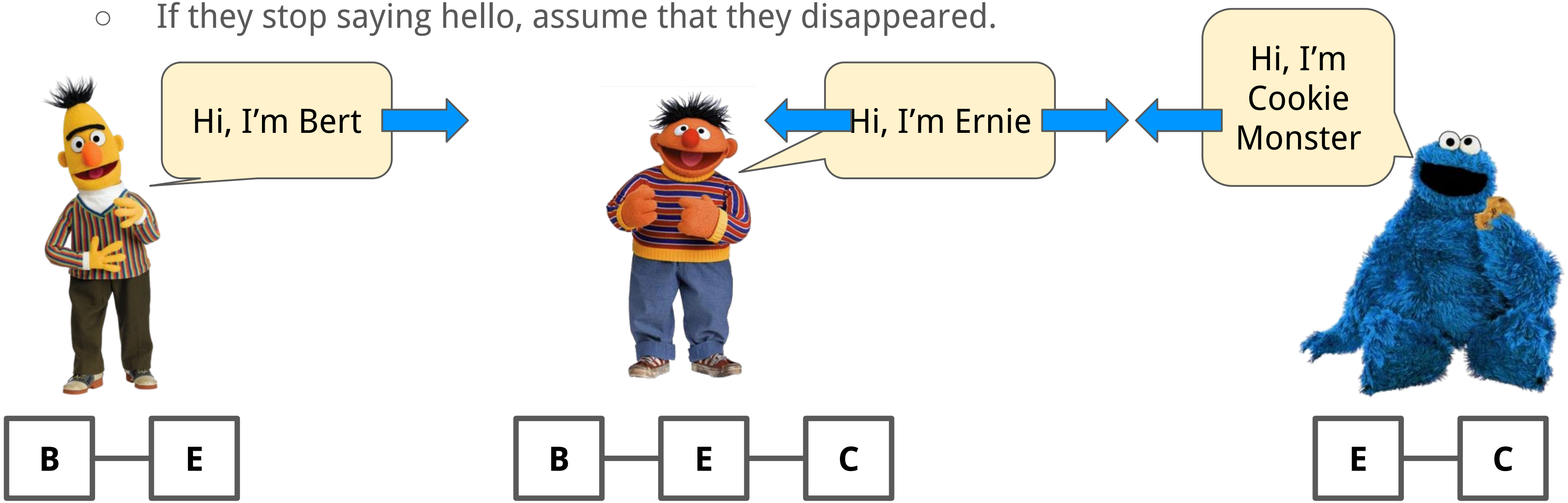
L-S: Hello Messages

- How do we find who is adjacent to us and their identity?
  - Say hello!

L-S: Hello Messages

- How do we find who is adjacent to us and their identity?
  - Say hello!

- Routers periodically send *hello* messages to neighbours.
  - If they stop saying hello, assume that they disappeared.



L-S: Learning about the topology

- We need to understand information about:
  - All links between all routers.
  - All destinations.

- We need to:
  - **Discover who my neighbours are** ***by exchanging hellos.***
  - Tell everyone about my neighbours.
  - Tell everyone about destinations attached to me.

L-S: Flooding

- Exchanging hellos just finds your next-door neighbour.
- But we need to know about *everyone* within the network.

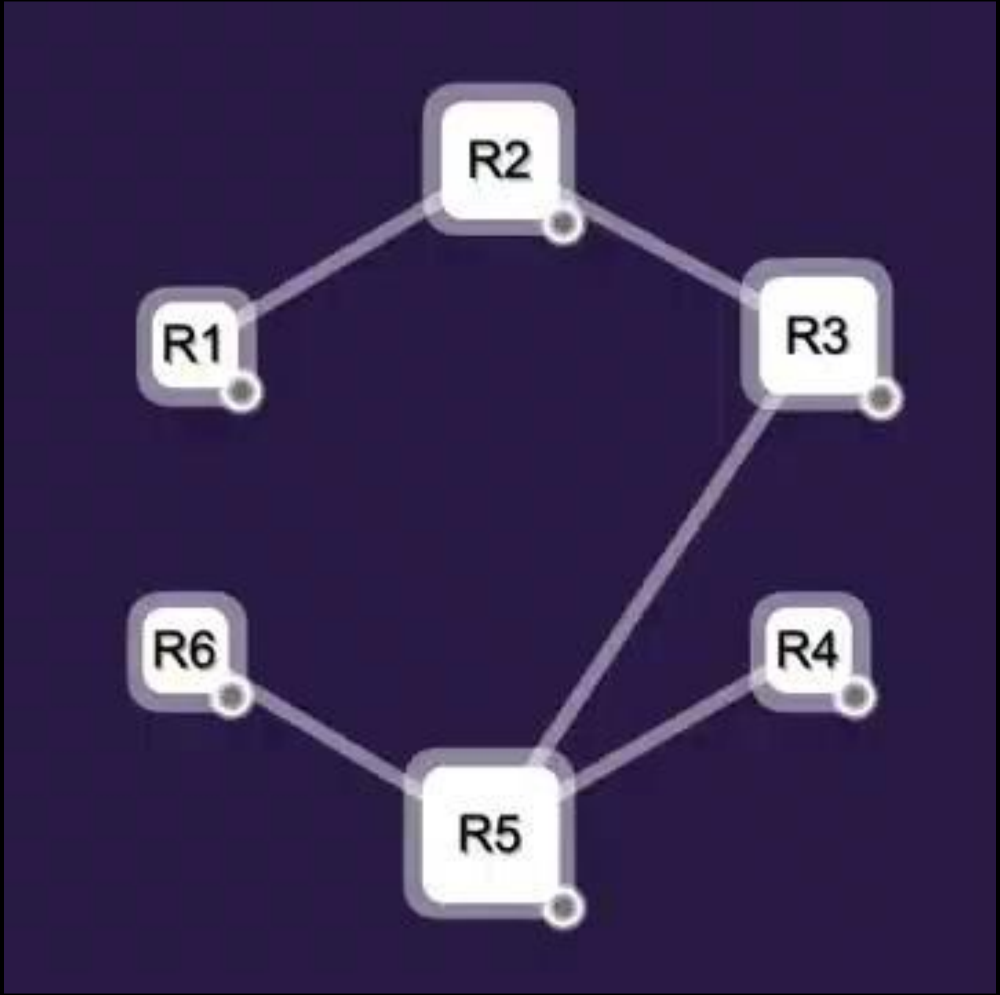
- Solution: **flood information across the network**.

- Straw-person solution:
  - When local information changes – send it to everyone.
  - When you receive information from your neighbour – send it to everyone else.

Bert and Ernie are neighbours, tell everyone!

L-S: Flooding

- Does this always work?


R2
R1
R3
R6
R4
R5

Link State: Flooding

- Naïve solution causes amplification:
  - One-loop – packets get forwarded forever.
  - Multiple loops – packets multiply exponentially.

Link State: Flooding

- Naïve solution causes amplification:
  - One-loop – packets get forwarded forever.
  - Multiple loops – packets multiply exponentially.

- Solution:
  - When local information changes, send to all neighbours.
  - When you receive a packet from a neighbour, send to all other neighbours.
    - Unless you’ve already seen it!

- Identifying packets you have seen can be via a *sequence number* or any other unique identifier.

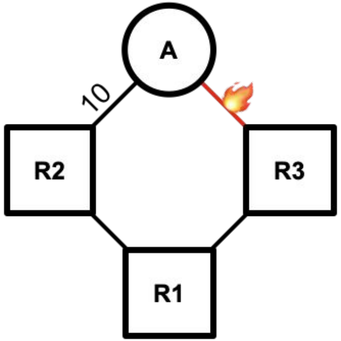
Link State: Flooding Reliability

- We need to make sure that other routers don’t “miss” updates.
  - Remember, we wanted a consistent view of the network!
- Use the same trick as D-V protocols: periodically re-send the packet.
  - IS-IS and OSPF both do these things.
- Generally, this ties in with reliability of message delivery.

Questions?

L-S: Convergence

- When a failure occurs, Dijkstra (or similar) will avoid a looping path.
- However, we can still have loops in link state protocols.
- We only control our own next-hop.
  - If our neighbour doesn't know about a link failure – i.e., has a different topology
  - …they might forward back to us!
- For example:
  - R1, which doesn't know about a failure, forwards to R3
  - R3 sends packet to R1.



L-S: Convergence

- Link-State protocols rely on the graph being consistently understood to converge.

- Sources of delay:
  - Time to detect failure.
  - Time to flood link-state information.
  - Time to recompute paths.

- During convergence.
  - Dead-ends
  - Loops
  - Out of order delivery

L-S: Overview

- Simple concept:
  - Everyone floods link/destination information
  - Everyone has a global map of the network
  - Everyone independently computes next-hops
- All the complexity is in the details!

Why might we use a link-state protocol?

- Aren't Link State protocols just *worse*?

- Distance-Vector hides some details from each node.
  - Must accept what our neighbour told us, and we don't know what the path is.
- Distance-Vector relies on our neighbour recomputing and readvertising their path.

- Link state protocols can:
  - Flood information before recomputing (just tell everyone the state).
  - Make all the topology available to every node (so they know what path they are choosing)

- Generally, we use a path/distance vector and link state protocol in combination in real networks.

Questions?

Addressing

Thus far…

Routing Table

Dst	*Nxt,Cost*	*TTL*
A	Direct,1	---

Forwarding Table

R2's Table	
Dst	*Port*
A	0
B	1
C	1
D	2

Thus far…

Routing Table

Dst	*Nxt,Cost*	*TTL*
A	Direct,1	---

Forwarding Table

R2's Table	
Dst	***Port***
A	0
B	1
C	1
D	2

One entry per destination

Really?

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

- If routing on the Internet is D-V, how long does it take to reconverge and how many routing calculations does each router do?
- If routing on the Internet is L-S, can we really store the entire state of the network including all hosts at each node?

Really?

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

- If routing on the Internet is D-V, how long does it take to reconverge and how many routing calculations does each router do?
- If routing on the Internet is L-S, can we really store the entire state of the network including all hosts at each node?

- **No.**

So…

- We’ve referred to each node just based on some name.
  - e.g., R1, R2, A.
- But is that really the case?
- The “secret” to scaling routing ⇒ how we do addressing!

Addressing at each Layer

- Remember, we talked about our letter example.
  - If I send a letter to Sylvia…
    - FedEx used *Soda Hall's* address.
    - The department used *413 Soda's* address.
    - Inside 413, we used Sylvia's name.

- Each layer had a separate type of address.
  - This is the same on the Internet.

- We already discussed addresses in Ethernet (at Layer 2) – **MAC addresses**.

IP addresses

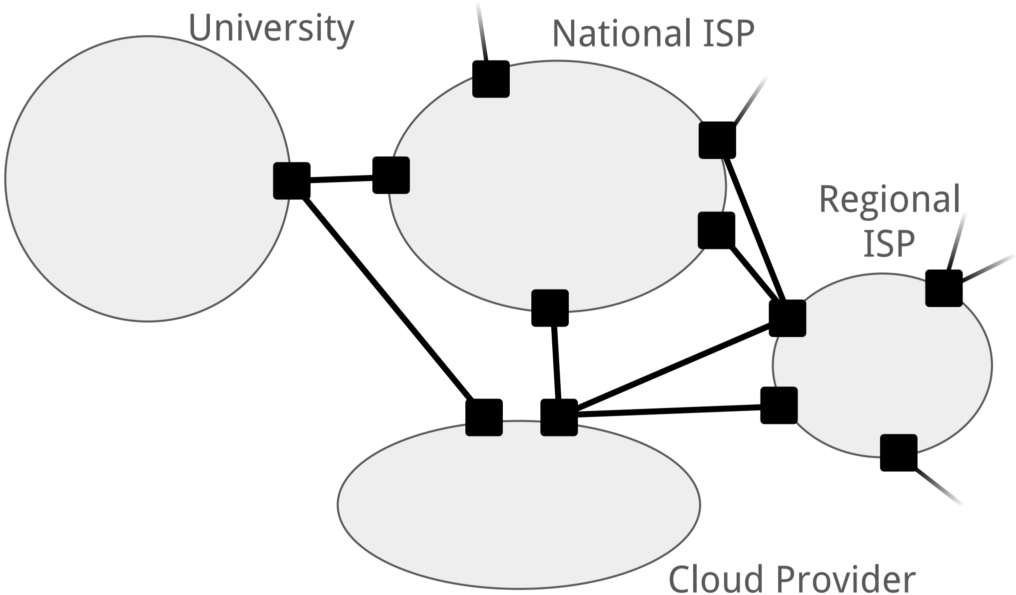
- You'll have seen them when thinking about networking at home.
- But sometimes they are hidden.
  - We'll talk about that later.

- Two flavours: IPv4 and IPv6.
  - The fundamentals for *routing* are similar.
  - We'll use IPv4 mainly in our examples.

- A number assigned to each host on the network.
  - 32bits for IPv4.
  - 128bits for IPv6.

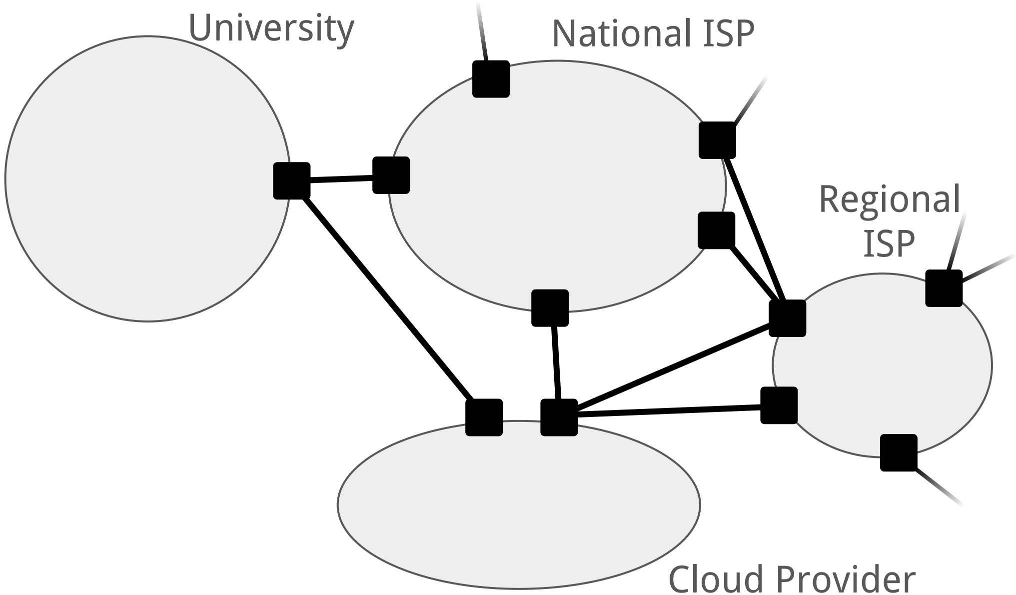
Addressing in the Early Internet

- The Internet is a network of networks.



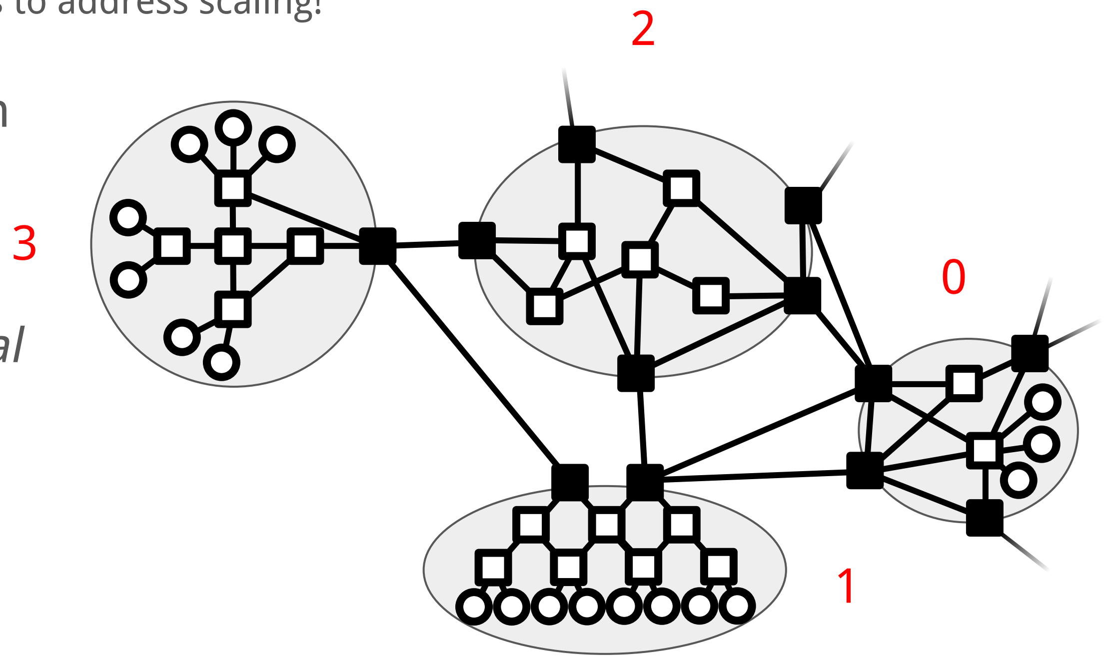
Addressing in the Early Internet

- The Internet is a network of networks.
  - Leads naturally to a hierarchy of addresses.
  - And hierarchy is one of the ways to address scaling!



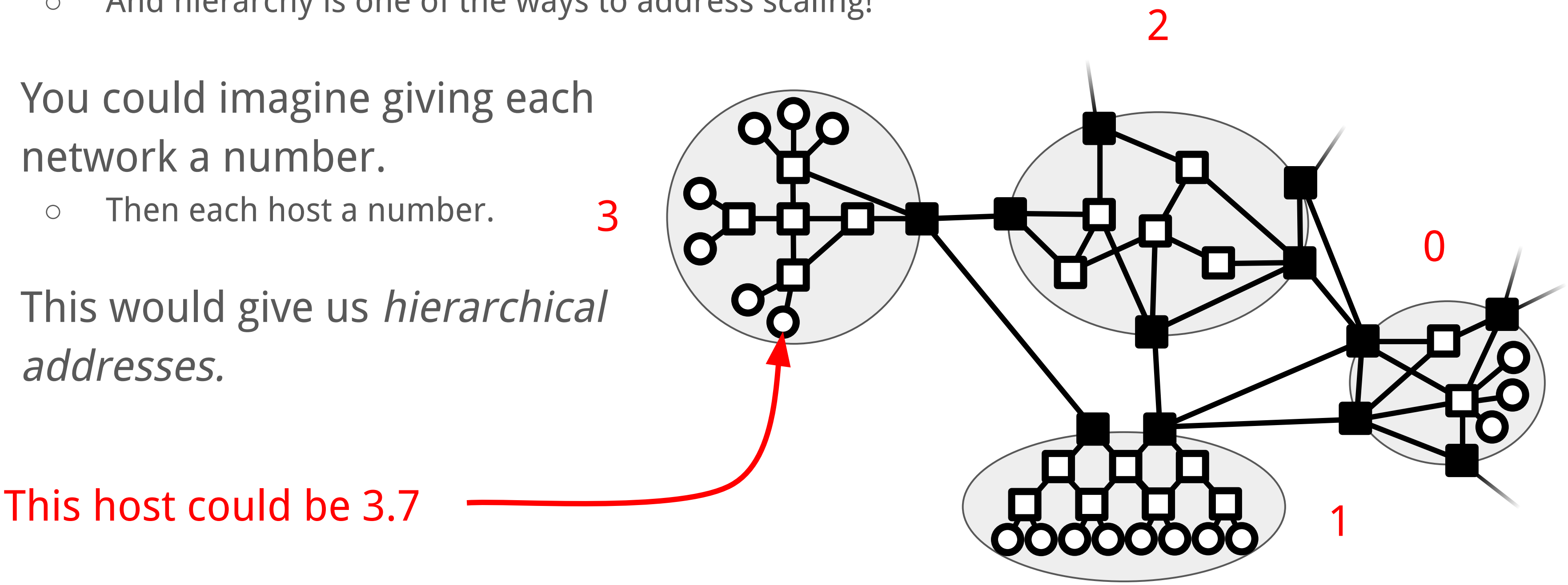
Addressing in the Early Internet

- The Internet is a network of networks.
  - Leads naturally to a hierarchy of addresses.
  - And hierarchy is one of the ways to address scaling!
- You could imagine giving each network a number.
  - Then each host a number.
- This would give us *hierarchical addresses.*



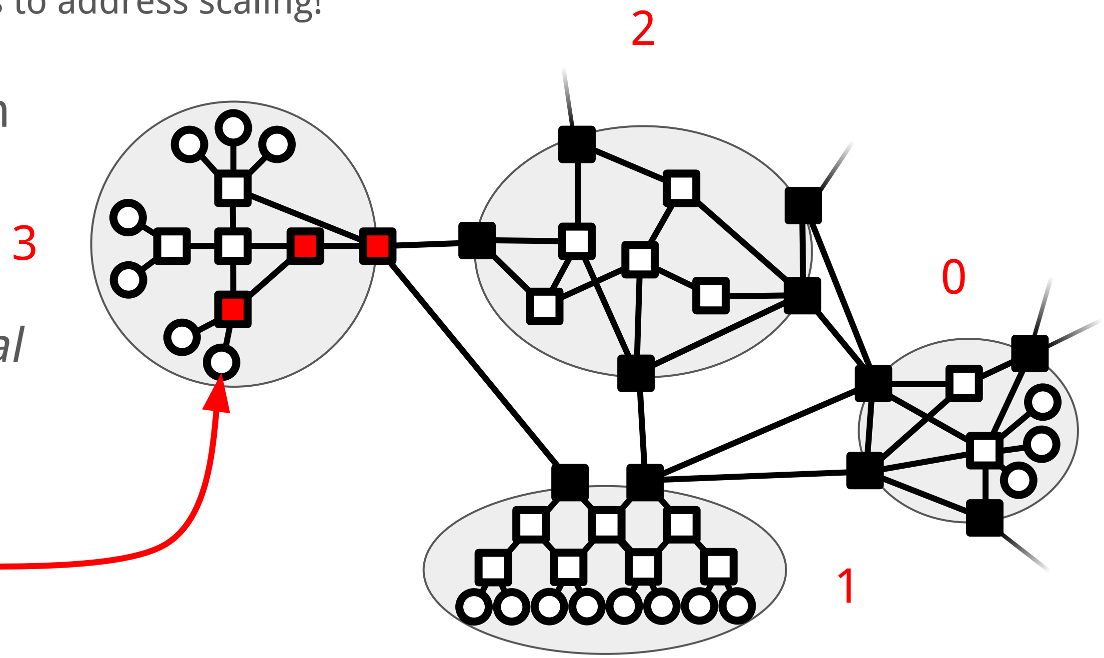
Addressing in the Early Internet

- The Internet is a network of networks.
    - Leads naturally to a hierarchy of addresses.
    - And hierarchy is one of the ways to address scaling!
- You could imagine giving each network a number.
    - Then each host a number.
- This would give us *hierarchical addresses.*



Addressing in the Early Internet

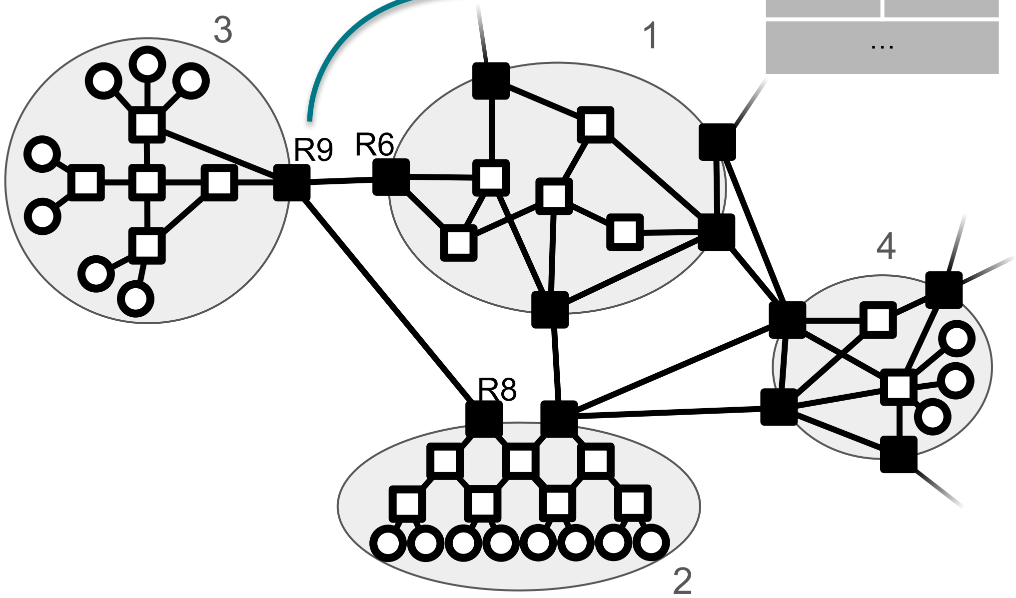
- The Internet is a network of networks.
    - Leads naturally to a hierarchy of addresses.
    - And hierarchy is one of the ways to address scaling!
- You could imagine giving each network a number.
    - Then each host a number.
- This would give us *hierarchical addresses.*



Or it could be 3.42.7.1

Hierarchical Addressing

- Routing between domains can now think only about the network part.
- Inter-domain routing: 4 nodes!
- Limits both:
  - Table size
  - Churn
    - Changes inside domains == no recalculation in other domains.
- Huge scaling improvement.



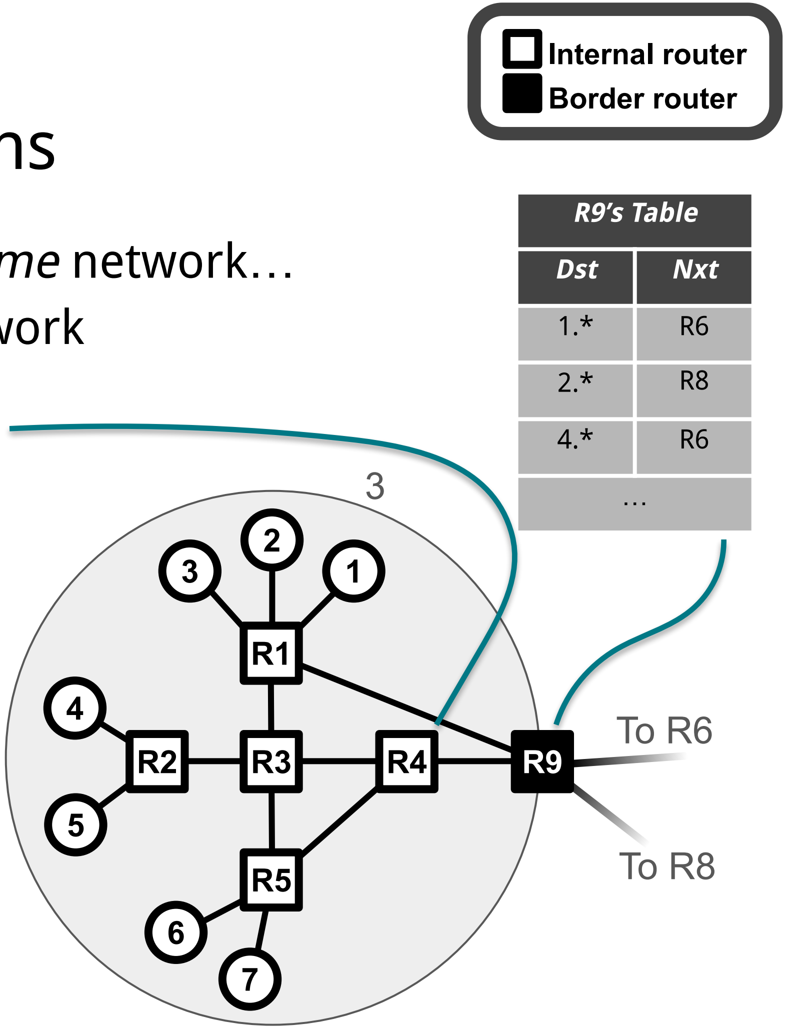
R9's Table	
Dst	**Nxt**
1.*	R6
2.*	R8
4.*	R6
…	

Hierarchical Addressing Implications

- Internal routers need routes for all hosts in *same* network…
  - Scales with number of hosts in single network

R4's Table	
Dst	**Nxt**
3.1	R3
3.2	R3
3.3	R3
3.4	R3
3.5	R3
3.6	R5
3.7	R5

R9's Table	
Dst	**Nxt**
1.*	R6
2.*	R8
4.*	R6
…	



Internal router
Border router

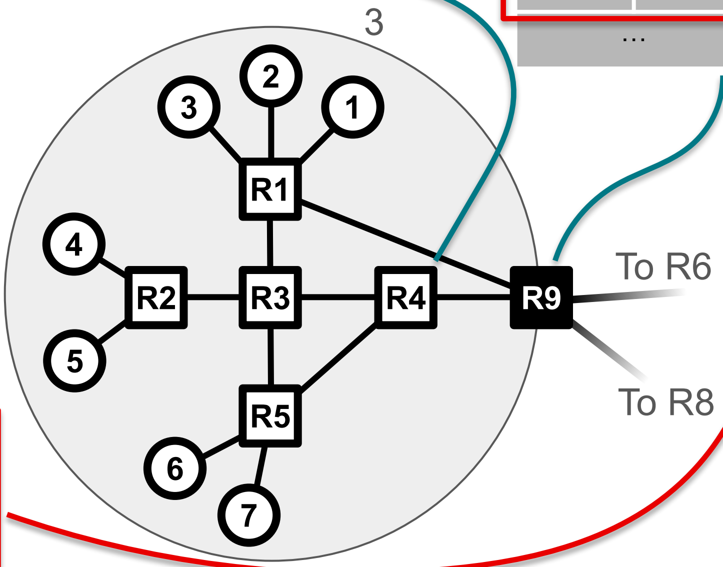
Hierarchical Addressing Implications

- Internal routers need routes for all hosts in *same* network…
  - Scales with number of hosts in single network

- .. *and* routes for other networks

R4's Table	
Dst	**Nxt**
3.1	R3
3.2	R3
3.3	R3
3.4	R3
3.5	R3
3.6	R5
3.7	R5
1.*	**?**
2.*	**?**
4.*	**?**

R9's Table	
Dst	**Nxt**
1.*	R6
2.*	R8
4.*	R6
…	

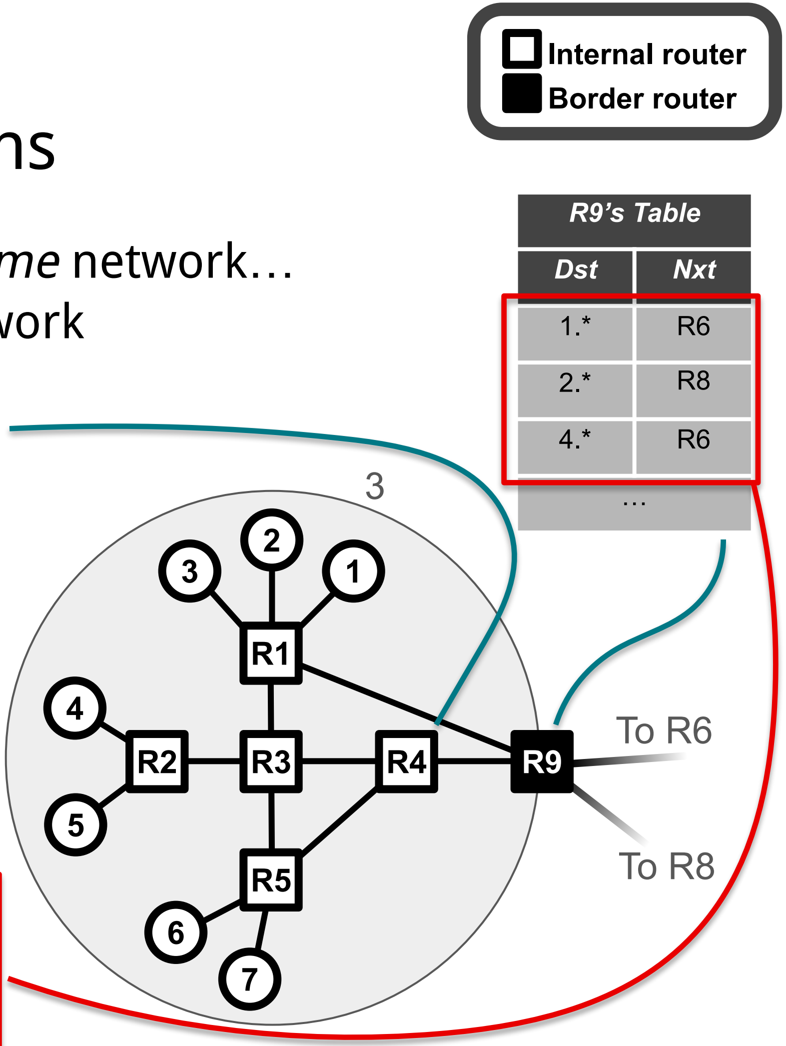


Hierarchical Addressing Implications

- Internal routers need routes for all hosts in *same* network…
  - Scales with number of hosts in single network
- .. *and* routes for other networks
- So total state scales with number of *hosts in this network* plus number of *other networks*
- Again: big scalability improvement assuming many more hosts than networks

R4's Table	
Dst	***Nxt***
3.1	R3
3.2	R3
3.3	R3
3.4	R3
3.5	R3
3.6	R5
3.7	R5
1.*	**?**
2.*	**?**
4.*	**?**

R9's Table	
Dst	***Nxt***
1.*	R6
2.*	R8
4.*	R6
…	



Questions?

Wait…what?

1.* → ?

Wait…what?

$$1.* \rightarrow ?$$

Hierarchy means that we might need our *routing* and *forwarding* to understand some form of wildcards.

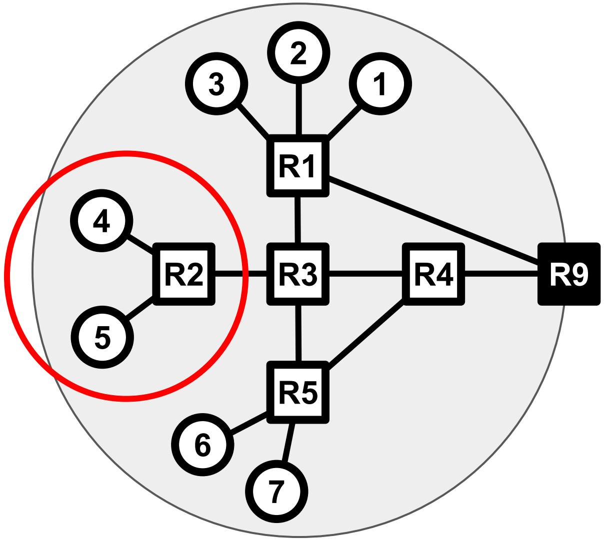
Wait…what?

$1.* \rightarrow ?$

We’ll come back to this when we talk about how routers do matches for forwarding. For routing - we carry this “wildcard” information.

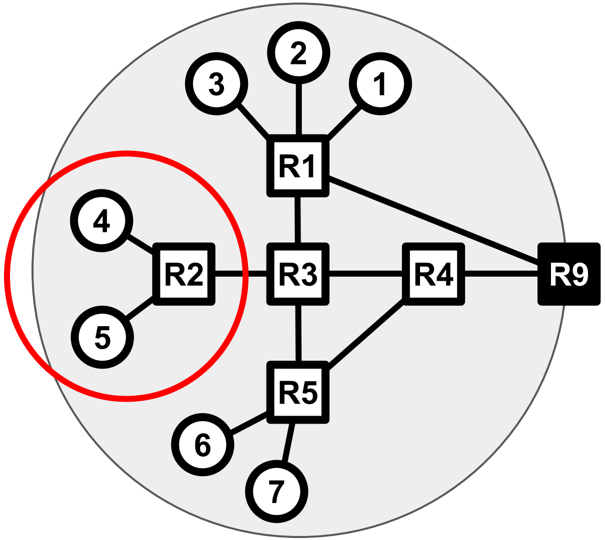
Improving scale with wildcards.

- What routing information does R2 need?



Improving scale with wildcards.

- What routing information does R2 need?
- Everything is reached through R3.
- So a wildcard can be used.
  - *.*
  - Called the *default route*.
- Most hosts just have this route!



Hierarchical Addressing

- Note that addresses aren't assigned randomly!
- Hosts that are "close to each other" (in some sense) share part of their address
- We leverage this structure to make routing (and forwarding) scale better

- We use structured addresses like this all the time!
  - Soda Hall #413 is much easier to work with than if we just numbered every office in the world uniquely…

- This also explains why hosts don't generally participate in routing protocols…
  - A human decided how to divide up the network in a way that makes sense
  - Your computer doesn't have its own IP address wherever it goes…
  - .. it changes it address depending on where it is
  - .. it "moves in" to the network where it's attached (and gets a new address there)

Our letter example

- Inside FedEx for a letter from London (🇬🇧) to Berkeley.

- Hierarchical lookups:
    - USA
    - California
    - Berkeley
    - 2551 Hearst Ave (Soda Hall)
    - 413
    - Sylvia Ratnasamy

Implications of Hierarchical Addressing

- Assuming addresses have two parts: Network.Host
- **Border routers** figure out routes between networks
- **Internal routers** figure out host routes for hosts *in that network*
  .. and *may* propagate the network routes from the EGP (it's one way to do it)
- Scales much better than “flat” routing:
  - Border routers don’t see churn inside networks
  - Internal routers don’t see churn in other networks
  - Routers only need state for:
    - Hosts in *their network*
    - And *other networks* themselves

IP addresses vs. MAC addresses

- For this (IP) hierarchy, addresses are allocated to hosts based on their **position within a network**.

- MAC addresses are assigned based on the hardware manufacturer.
  - And remember, we said they are “burnt in” at manufacture time.

- Creating wildcard routes for MAC addresses would be possible – but…
- …we don’t know where a particular Intel Ethernet card is plugged in.

- In the worst case, we can’t create wildcard routes at all for MAC addresses ⇒ significant scaling challenge.

- This is a key consideration in why we consider IP/L3 to be global, and L2 to be “local”

Questions?

Addressing in the Early Internet

- Not very many organisations.
- Give them all a unique number!

- Maybe lots of hosts inside their organisation with different hierarchy.
    - Our summarisation can be used internally.

Addressing in the Early Internet

- Not very many organisations.
- Give them all a unique number!

- Maybe lots of hosts inside their organisation with different hierarchy.
  - Our summarisation can be used internally.

- So – addresses are 32-bits long (IPv4).
  - Historically:
    - Organisation ID == 8 bits.
    - Host ID == 24 bits.

Addressing in the Early Internet

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

Addressing in the Early Internet

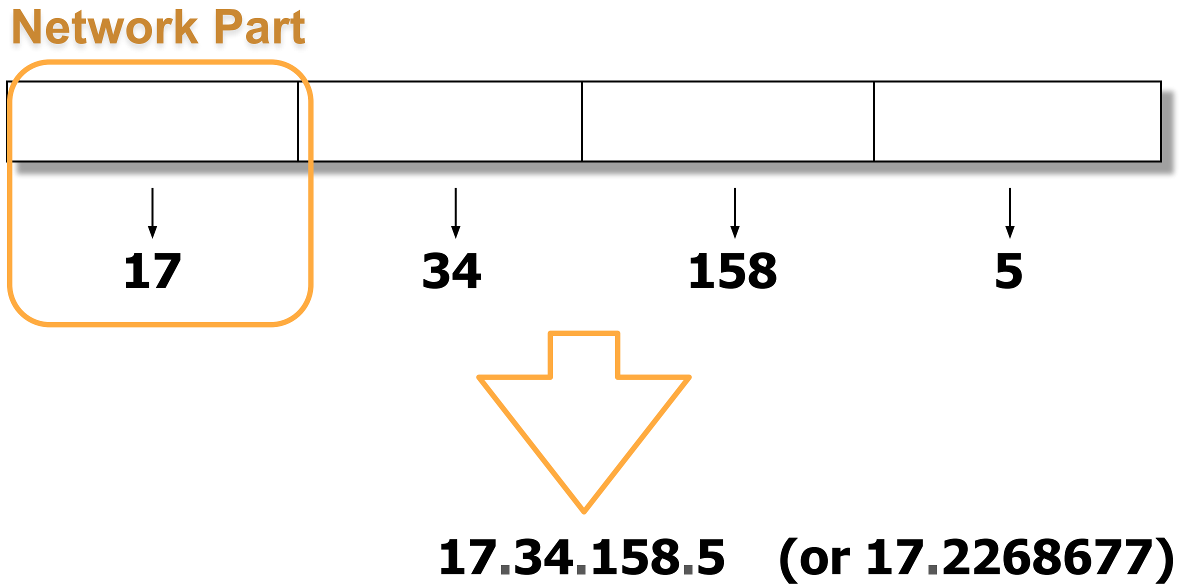
- AT&T: ID = 12
- Apple: ID = 17
- Ford: ID = 19
- Dept. of Defense: ID = 6, 7, 11, 21, 22, 26, 28, 29, 30, 33, 55, 214, 215.

- Wait – 2^8 = 256.
    - DoD = 13/256ths of the address space?

- Let’s come back to this.

Representing IPv4 addresses

- You could just represent an IPv4 address as a single big integer
- But far more common is a *dotted quad* or *dot quad*



Scaling addressing.

- Assigned the first 8-bits to “network ID”.

- Joe’s Tyre Shop: 10 computers but wants to connect to the Internet.
    - ID = 42.
    - 2^24 = 16777216 addresses.

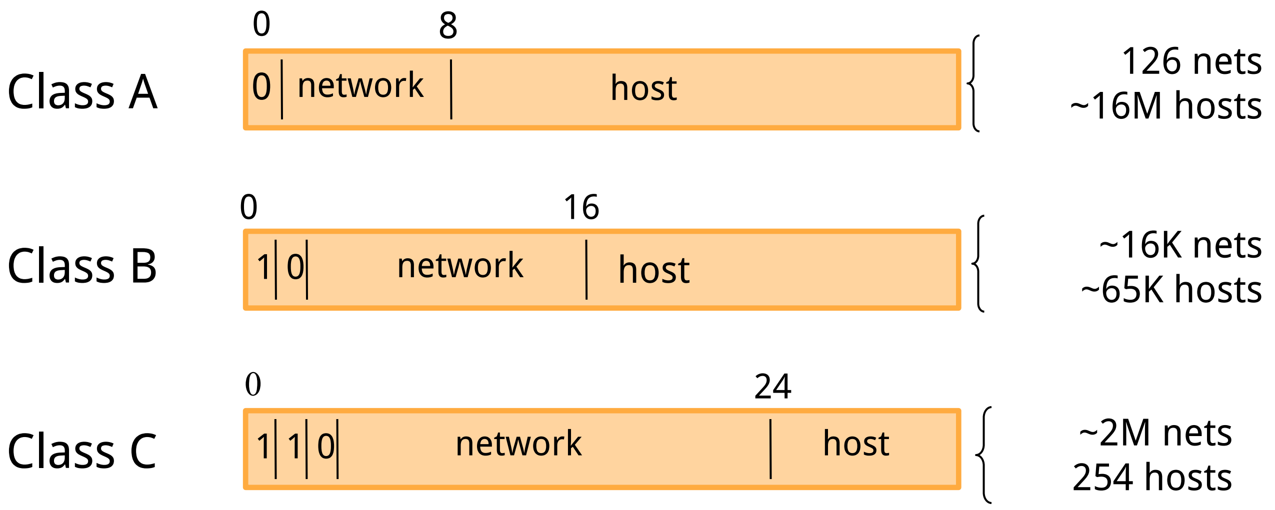
- And we already gave DoD 13 * 2^24 = 218103808 addresses.

Scaling addressing.

- Assigned the first 8-bits to “network ID”.
- Joe’s Tyre Shop: 10 computers but wants to connect to the Internet.
    - ID = 42.
    - 2^24 = 16777216 addresses.
- And we already gave DoD 13 * 2^24 = 218103808 addresses.
- **We’re going to run out!** 😱

“Classful” Addressing

- Allocate different size blocks based on need.



Classful Addressing: Fixing bits.

- What is a “Class B”?

- Fixing 16 bits of the address to be constant, the rest is variable.

Classful Addressing: Fixing bits.

- What is a "Class B"?
- Fixing 16 bits of the address to be constant, the rest is variable.

192	11000000	192	11000000
168	10101000	168	10101000
0	00000000	255	11111111
0	00000000	255	11111111

Classful Addressing

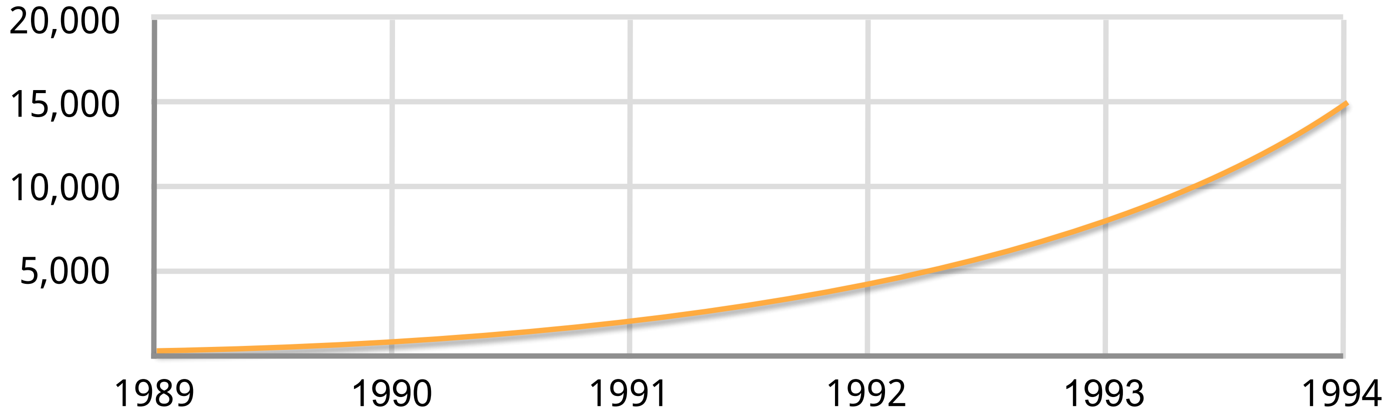
- Ran into problems of its own!

- The sizes of the classes weren't that useful
  - Class A far too big for most organizations!
  - Class C far too small for many organizations!
  - Class B is best option for many
    - Still too big for many organizations
    - Not that many of them!

- Running out of Class B? That's a lot of routes…
  - Number of interdomain routes was going up!

Classful Addressing

- Number of interdomain routes by year (approximate)



CIDR: Classless Inter-Domain Routing

- Our wildcards are arbitrary.
  - 1.*.*.* just means “the first 8 bits are 00000001”.

- Classes are just dividing based on “convenient” 8-bit boundaries.
  - 8 = Class A
  - 16 = Class B
  - 24 = Class C

- What happens if we made the number of *fixed bits* arbitrary?

CIDR: Classless Inter-Domain Routing

- Return to Joe's Tyre Shop.
    - 10 computers.

- Rather than giving them a Class C (2^(32-24) = 256 addresses).
    - Can we give them fewer?

CIDR: Classless Inter-Domain Routing

- Return to Joe's Tyre Shop.
  - 10 computers.

- Rather than giving them a Class C (2^(32-24) = 256 addresses).
  - Can we give them fewer?

- Yes, fix more bits!

CIDR: Classless Inter-Domain Routing

- Return to Joe's Tyre Shop.
  - 10 computers.
- Rather than giving them a Class C (2^(32-24) = 256 addresses).
  - Can we give them fewer?
- Yes, fix more bits!
- Can we give them 10 addresses?

CIDR: Classless Inter-Domain Routing

- Can we give them 10 addresses?
- Fix 28 bits: 2^(32-28) = 16 addresses.
- Fix 29 bits: 2^(32-29) = 8 addresses.
- No…

CIDR: Classless Inter-Domain Routing

- Can we give them 10 addresses?
- Fix 28 bits: 2^(32-28) = 16 addresses.
- Fix 29 bits: 2^(32-29) = 8 addresses.
- No…but at least we didn’t need to give them 256 addresses.

CIDR: Classless Inter-Domain Routing

- A Class B: 2^(32-16) = 65536
- A Class C: 2^(32-24) = 256

- If we can fix only 23 bits for someone that needs 450 addresses we save a **lot** of addresses!

Questions?

Hierarchical Assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Gives out blocks of addresses to….
- Regional Internet Registries (RIRs)...
  - RIPE (EU), ARIN (NA), APNIC (Asia/Pacific), LACNIC (SA), AFRINIC (Africa)
  - Give out portions to…
- Large organisations or ISPs…
  - Called Local Internet Registries (in the RIPE region)
  - Who give out portions to...
- Small organisations and individuals.
  - E.g., UC Berkeley, Rob's startup.

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.

1101
208.0.0.0

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.
- Regional Internet Registries (RIRs)...
  - ARIN allocates 8,000,000 addresses to AT&T.
  - Requires 23 bits (2^23 = 8,388,608) of the address to be variable.
  - Fixes (32-23) = 9 bits

1101
208.0.0.0

110111001
220.128.0.0

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.
- Regional Internet Registries (RIRs)...
  - ARIN allocates 8,000,000 addresses to AT&T.
  - Requires 23 bits (2^23 = 8,388,608) of the address to be variable.
  - Fixes (32-23) = 9 bits
- AT&T
  - Allocates 16,000 addresses to UC Berkeley.
  - Requires 14 bits of the address to be variable (2^14 = 16,384)
  - Fixes (32-14) = 18 bits.

1101
208.0.0.0

110111001
220.128.0.0

110111001110100010
220.232.128.0

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.
- Regional Internet Registries (RIRs)...
  - ARIN allocates 8,000,000 addresses to AT&T.
  - Requires 23 bits (2^23 = 8,388,608) of the address to be variable.
  - Fixes (32-23) = 9 bits
- AT&T
  - Allocates 16,000 addresses to UC Berkeley.
  - Requires 14 bits of the address to be variable (2^14 = 16,384)
  - Fixes (32-14) = 18 bits.
- UCB…
  - Now can determine how it wants to split its addresses.
  - Allocates 200 addresses to Soda Hall.
  - Requires 8 bits of the address to be variable (2^8 = 256).
  - Fixes (32-8) = 24 bits.

1101
208.0.0.0

110111001
220.128.0.0

110111001110100010
220.232.128.0

110111001110100010011010
220.232.154.0

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.
- Regional Internet Registries (RIRs)...
  - ARIN allocates 8,000,000 addresses to AT&T.
  - Requires 23 bits (2^23 = 8,388,608) of the address to be variable.
  - Fixes (32-23) = 9 bits
- AT&T
  - Allocates 16,000 addresses to UC Berkeley.
  - Requires 14 bits of the address to be variable (2^14 = 16,384)
  - Fixes (32-14) = 18 bits.
- UCB…
  - Now can determine how it wants to split its addresses.
  - Allocates 200 addresses to Soda Hall.
  - Requires 8 bits of the address to be variable (2^8 = 256).
  - Fixes (32-8) = 24 bits.
- Prof. Ratnasamy...
  - Allocates 1 address to Rob.
  - Requires 0 bits of the address to be variable (2^0 = 1)
  - Fixes (32-0) = 32 bits.

1101
208.0.0.0

110111001
220.128.0.0

110111001110100010
220.232.128.0

110111001110100010011010
220.232.154.0

11011100111010001001101001011101
220.232.154.93

CIDR allows more granular assignment

- ICANN (Internet Corporation for Names and Numbers)
  - Fixes 4 bits and assigns this to ARIN – 2^(32-4) = 268435456 addresses.
- Regional Internet Registries (RIRs)...
  - ARIN allocates 8,000,000 addresses to AT&T.
  - Requires 23 bits (2^23 = 8,388,608) of the address to be variable.
  - Fixes (32-23) = 9 bits
- AT&T
  - Allocates 16,000 addresses to UC Berkeley.
  - Requires 14 bits of the address to be variable (2^14 = 16,384)
  - Fixes (32-14) = 18 bits.
- UCB…
  - Now can determine how it wants to split its addresses.
  - Allocates 200 addresses to Soda Hall.
  - Requires 8 bits of the address to be variable (2^8 = 256).
  - Fixes (32-8) = 24 bits.
- Prof. Ratnasamy…
  - Allocates 1 address to Rob.
  - Requires 0 bits of the address to be variable (2^0 = 1)
  - Fixes (32-0) = 32 bits.

1101
208.0.0.0

110111001
220.128.0.0

How do we know that 220.128.0.0 is in the allocation?

110111001110100010
220.232.128.0

110111001110100010011010
220.232.154.0

11011100111010001001101001011101
220.232.154.93

CIDR Notation

- We need to show how many bits are fixed in the *network address* in order to know the range.

- Use “slash notation”:
    - 192.168.0.0/16 → 16 bits are fixed.
        - 192.168.0.0 – 192.168.255.255
    - 192.168.1.0/24 → 24 bits are fixed.
        - 192.168.1.0 – 192.168.1.255
    - 192.168.1.0/29 → 29 bits are fixed.
        - 192.168.1.0 – 192.168.1.7
    - 192.168.1.1/32 → 32 bits are fixed.
        - 192.168.1.1

An alternative: netmask notification

- Alternative to slash notation.

- Set a 1 for every bit that is fixed – and represent it as a “dotted quad”.

- 11111111 11111111 11111111 11111111 = 255.255.255.255 (32)
- 11111111 11111111 11111111 11111000 = 255.255.255.248 (29)
- etc.

- Equivalent notations.
    - But slash notation is much more convenient.

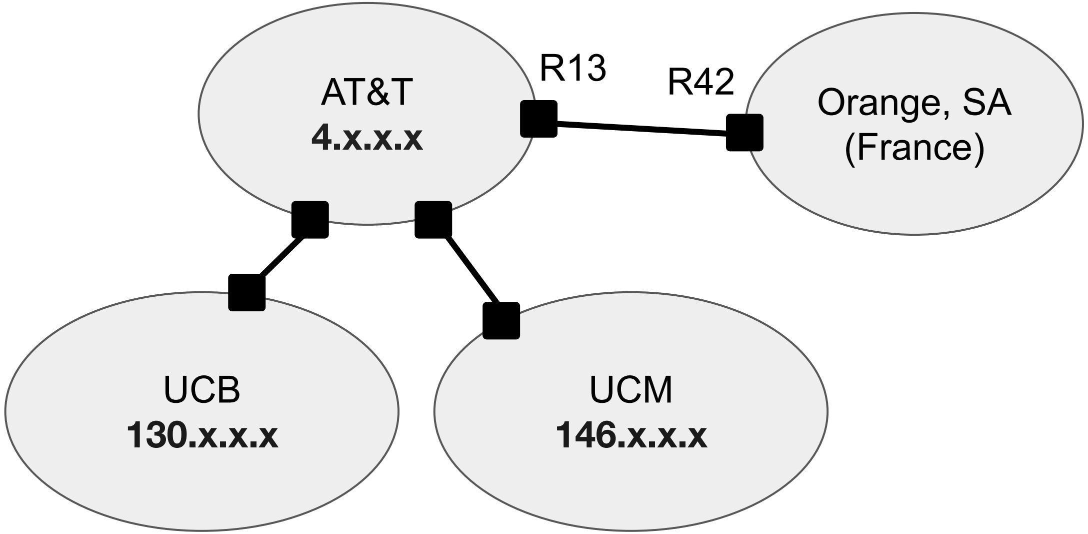
Questions?

CIDR and Route Scaling

- Also solving for the number of Inter-Domain Routes.

Route Aggregation

Classful addressing…

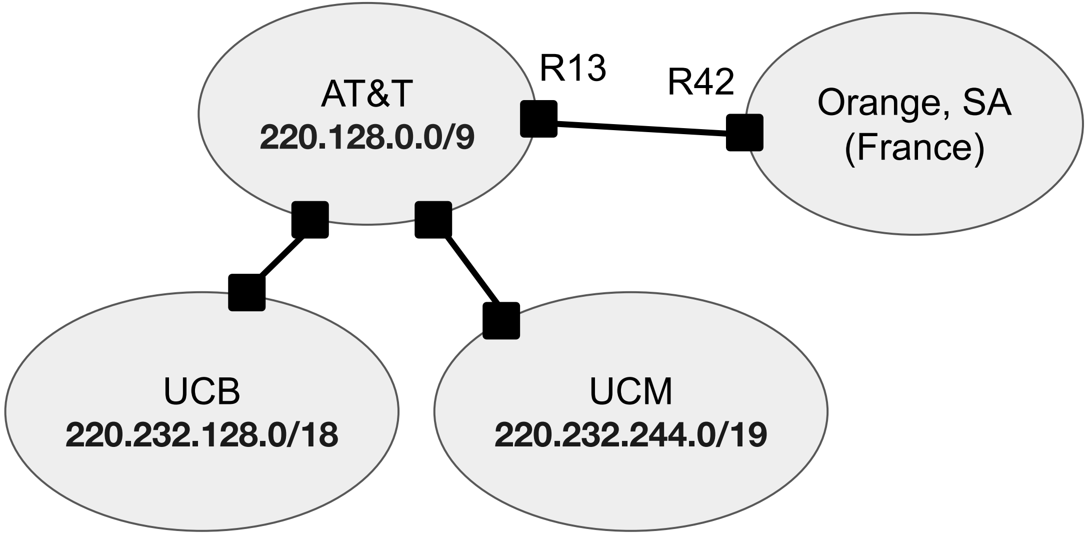


R42's Table	
Dst	***Nxt***
4.x.x.x	R13
130.x.x.x	R13
146.x.x.x	R13
…	

Route Aggregation

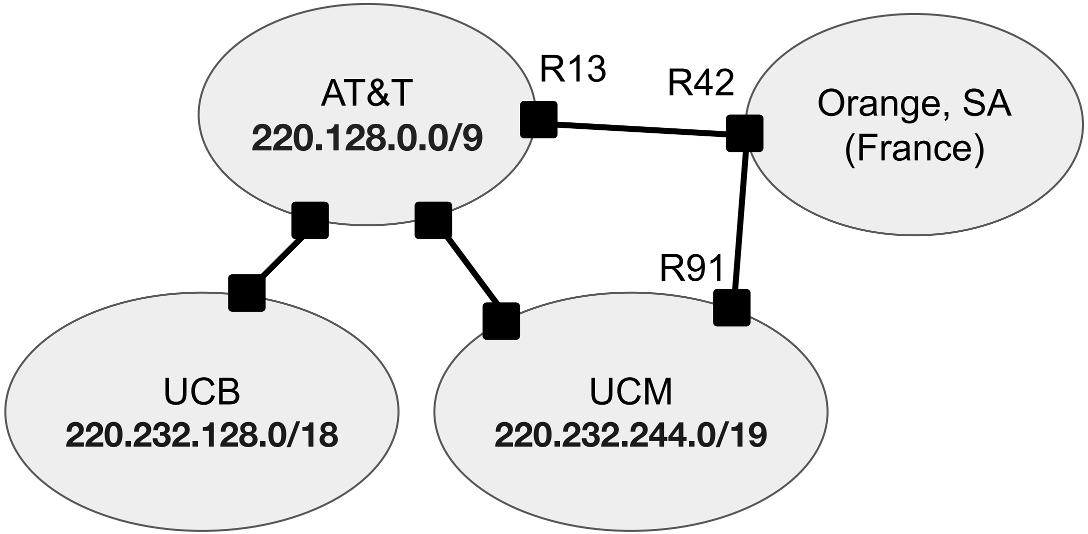
CIDR addressing…

Allows us to *aggregate* routes



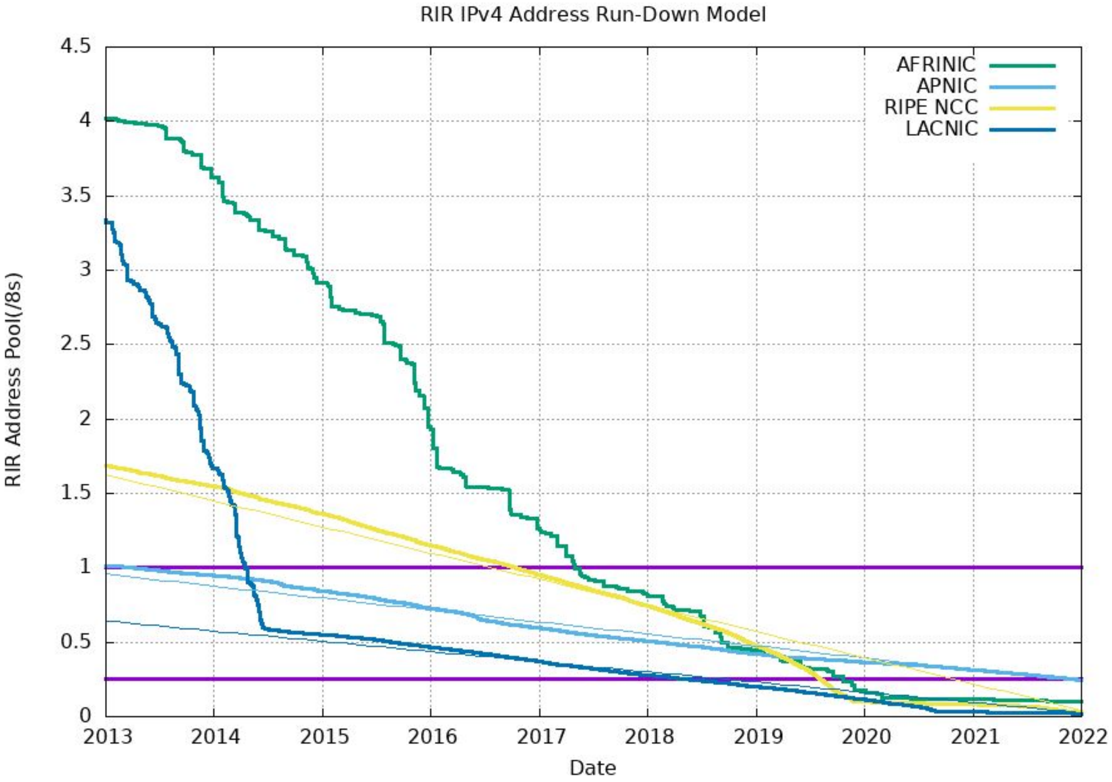
R42's Table	
Dst	***Nxt***
220.128.0.0/9	R13
~~**220.232.128.0/18**~~	~~R13~~
~~**220.232.244.0/19**~~	~~R13~~
…	

Longest Prefix Matching



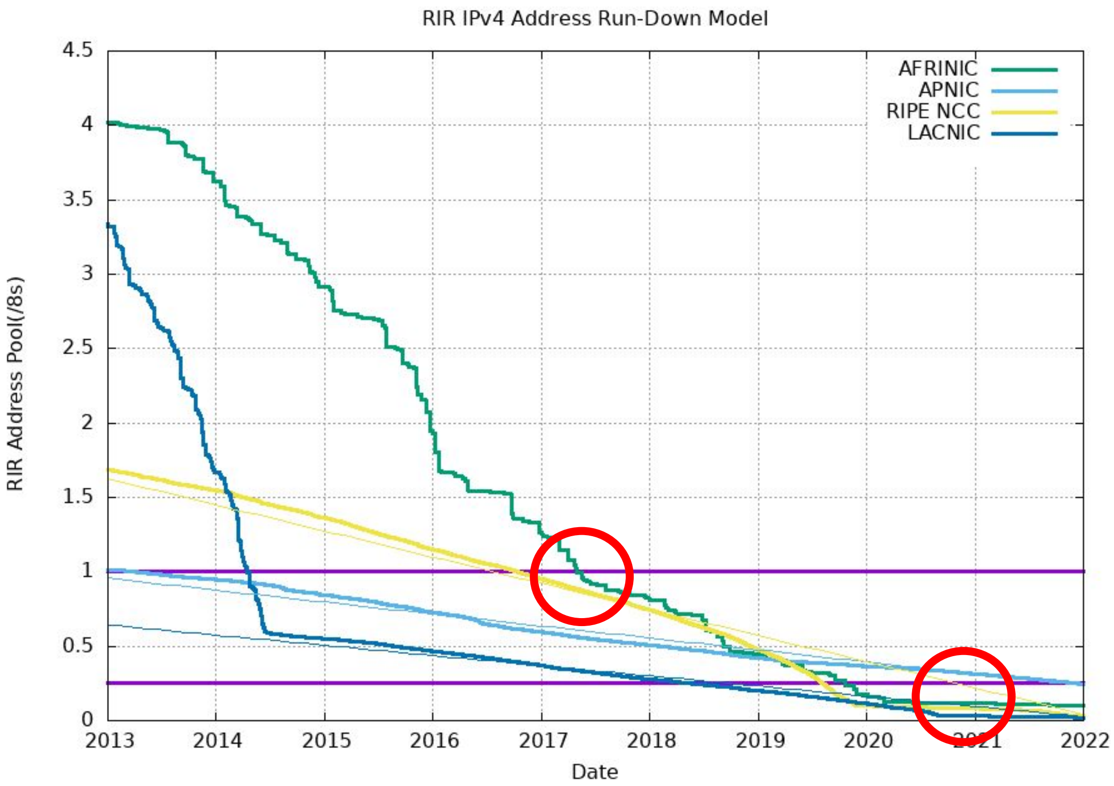
R42's Table	
Dst	***Nxt***
220.128.0.0/9	R13
~~**220.232.128.0/18**~~	~~R13~~
220.232.244.0/19	R91
…	

Was 32 bits enough?



https://ipv4.potaroo.net/

Was 32 bits enough?



https://ipv4.potaroo.net/

NANOG 51 - February 2011

IP version 6

Network Working Group — S. Deering
Request for Comments: 2460 — Cisco
Obsoletes: 1883 — R. Hinden
Category: Standards Track — Nokia
December 1998

**Internet Protocol, Version 6 (IPv6)
Specification**

What happened to version 5?

```
Network Working Group                                   CIP Working Group
Request for Comments: 1190                            C. Topolcic, Editor
Obsoletes: IEN-119                                           October 1990


        Experimental Internet Stream Protocol, Version 2 (ST-II)
```

IPv6

- Fundamentally uses the same addressing structure as IP version 4.

- But with 128-bits of address space.
  - And some new requirements and rules…
  - Not relevant to our discussion.

- Went from 2^32 to 2^128 addresses.

IPv6

2^128 = 3.4028236692209385e+38 addresses available.

IPv6

- Switches to hexadecimal representation rather than longer dotted address.

- 2001:0DB8:CAFE:BEEF:DEAD:1234:5678:9012
- 2001:0DB8:0000:0000:0000:0000:0000:0001

- Can omit leading zeros: 2001:DB8:0:0:0:0:0:1
- Can omit repeated zeros *once per address:* 2001:DB8**::**1

IPv6

- Still uses *slash notation.*

- 128-bits fixed == /128.
- 32-bits fixed == /32.

IPv6

- Some changes!

- We leave the last 64-bits of the address variable to allow for hosts to configure their own addresses.
  - **S**tate**L**ess **A**ddress **A**uto**C**onfiguration (SLAAC).

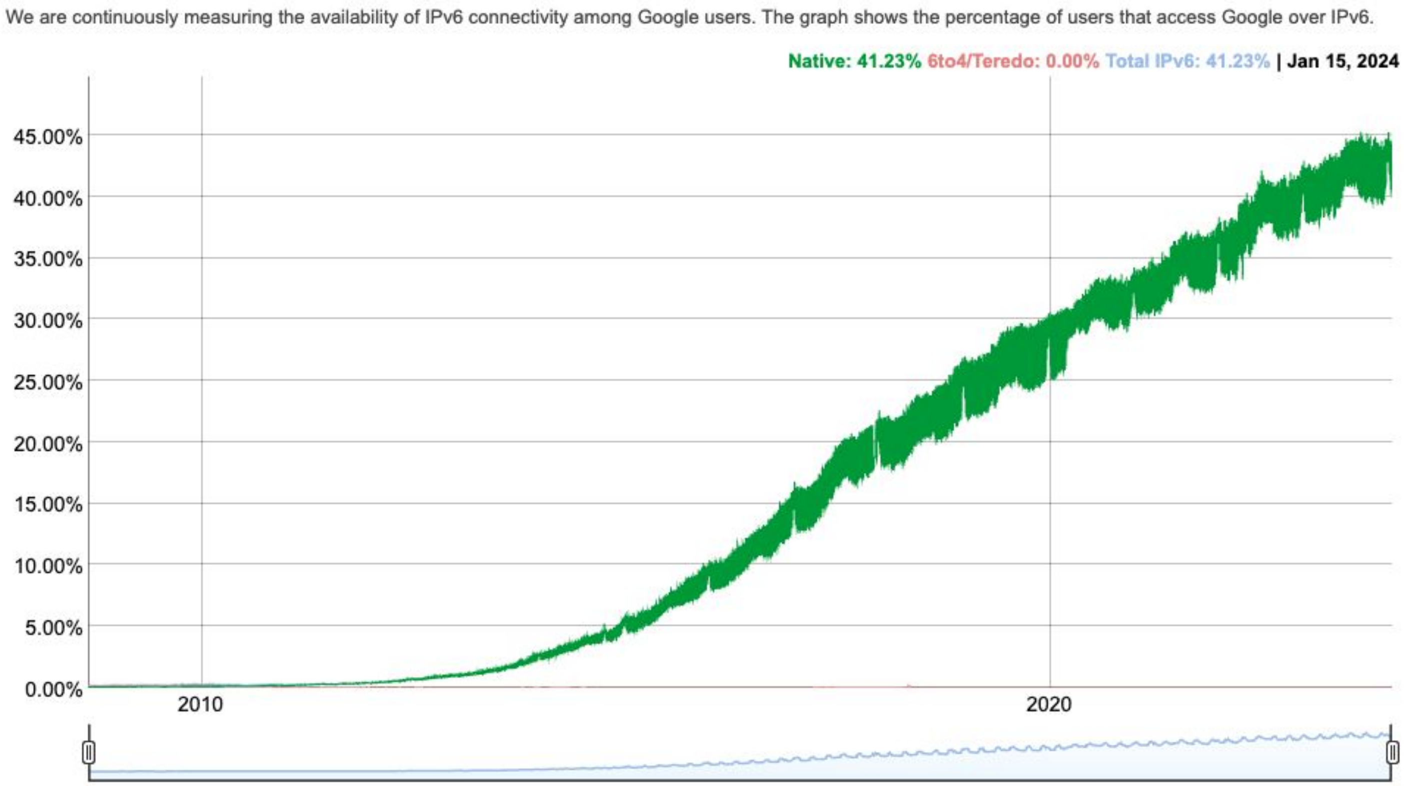
- This means practically, we don't expect to see routes with /64 or *longer* (greater).
  - Although in special cases we might.

IPv6

- The same hierarchical addressing approach is used in IPv6 and IPv4.

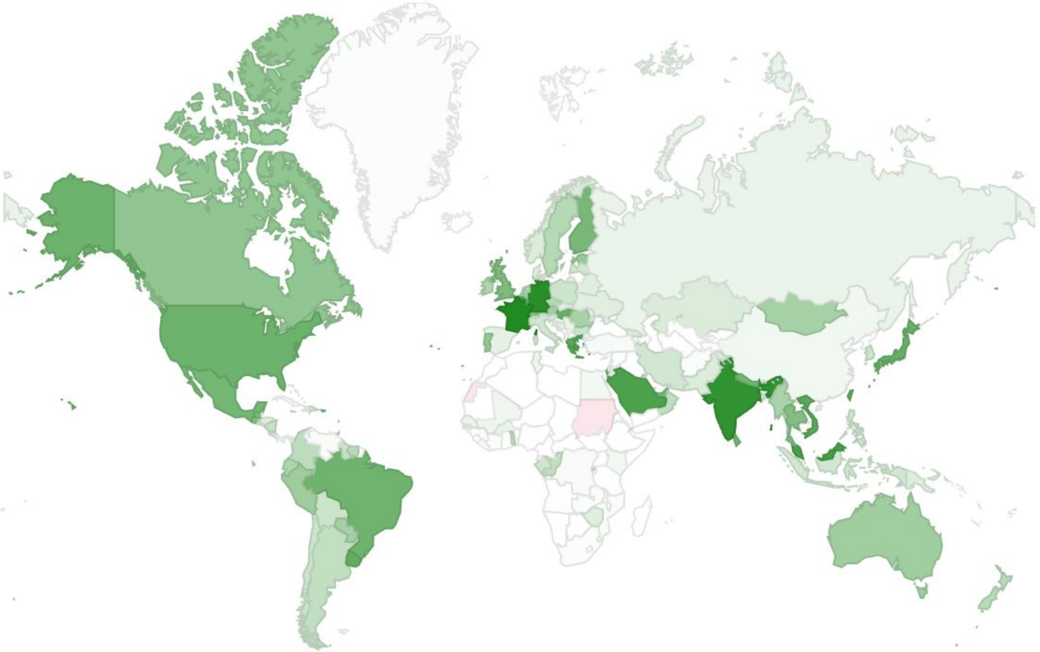
- We tend to use IPv4 for examples.
    - Because long strings of numbers are harder to remember.

IPv6 Adoption



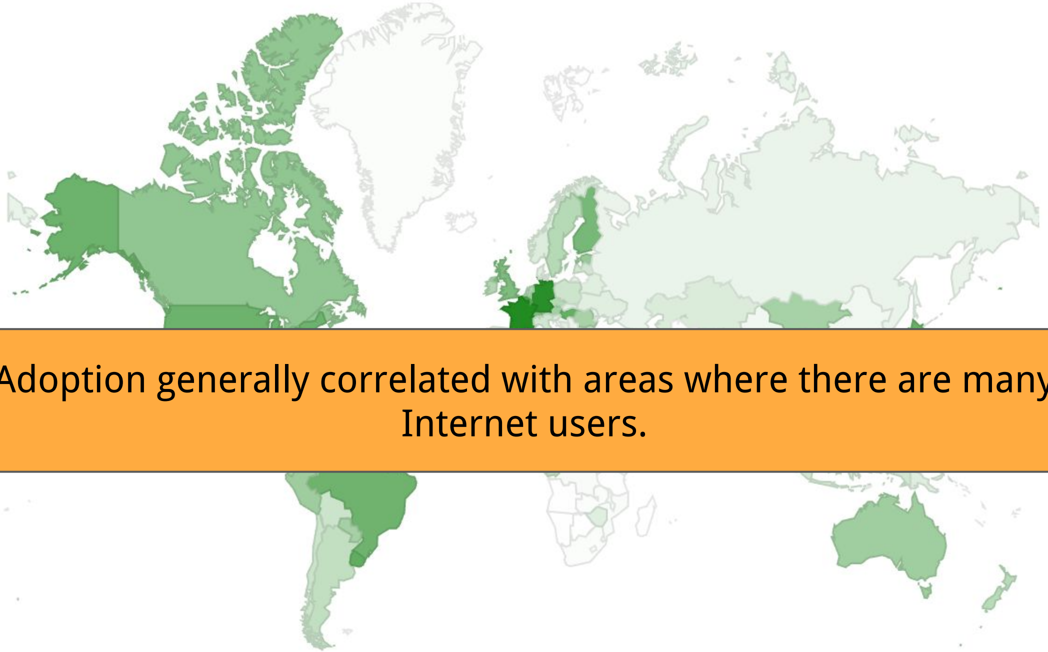
https://www.google.com/intl/en/ipv6/statistics.html

IPv6 Adoption



https://www.google.com/intl/en/ipv6/statistics.html

IPv6 Adoption



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

https://www.google.com/intl/en/ipv6/statistics.html

Challenges for IPv6 Adoption

- No smooth path
  - Hosts and ISPs need both addresses.
- Rebuilding the Internet.
  - Partial coverage where only some things are on IPv6.
- Coexistence.
  - If something is on IPv4 and IPv6 which should I use?
- Main driver for IPv6 adoption
  - We're running out of IPv4 addresses!

Recap

- Hosts on the Internet have addresses – either IPv4 or IPv6 or both.

- These addresses are hierarchical.
  - They are assigned in groups to specific organisations.

- Wildcard matching means that this can help our forwarding and routing scalability.
  - We’ll talk about this more next time!