How do we route in really big networks?
Classic shortest-path routing

$\Omega(n)$ memory per node

- store next hop to each destination

$\Omega(n)$ messages per node per unit time

- assuming each node moves once per unit time
- also must recompute routes each of these times

if $n = 1,000,000,000$ and “unit time” = one day,

- $\approx 100$–$1000x$ more fast path memory than routers today
- 11,600 updates per second
- 4.4 Mbit/sec if updates are 50 bytes

How can we scale better than $\Omega(n)$ per node?
Routing in Manhattan

(8, 23) to (3, 46)
Recipe for scaling

1. Convert **name** to **address**
   - **name**: arbitrary
   - **address**: hint about location
   - conversion uses distributed database (e.g., DNS)

2. Nodes have **partial view** of network

3. To route, combine partial view with address

Challenge: **how do we summarize the network effectively** in the partial view and address?
   - and what does “effectively” mean?
Key goals

Addresses are small

Node state is small

Routes are short

- stretch = route length / shortest path length

How does Manhattan routing do?

- Assume square grid of $n$ nodes ($\sqrt{n} \times \sqrt{n}$)
- Address is (street, avenue); nodes store neighbors’ addr.
- Address size: $2 \log_2(\sqrt{n}) = \log_2 n$
- Node state: $\approx 4 \log_2 n$
- Route length: shortest (stretch 1) if we know address!
Scalable routing in **structured** networks

- Manhattan routing
- Greedy routing
- NIRA

Scalable routing in **arbitrary** networks

- Hierarchy
- Compact routing
Structured networks
Grid

(8, 23) → (3, 46)
Torus

3D Torus
[press shot via hexus.net]

2D Torus
A plethora of structured graphs!

Hypercube
Supercomputers,
distributed hash tables

Fat tree
Supercomputers,
proposed data centers

Small world
distributed hash tables
Greedy routing

Technique common in many structured networks

Scheme:

- Each node knows addresses of itself & neighbors
- Given two addresses, can estimate “distance” between them: dist(s,t)
- Forwarding at node v: send to neighbor w with lowest distance to destination d (minimize dist(w,d))

What structure does this require?

- Compact addresses that can “summarize” location
- Good estimate of distance dist(s,t) given two addresses
  - No local minima in dist()! (Q: Why could there be?)
Greedy Perimiter Stateless Routing

[Karp, Kung, MobiCom ’00]

Address is physical location, e.g., from GPS

Distance estimate is Euclidean distance

Can get stuck

• No neighbor is closer to $x$!
• Solution: planarize graph and traverse perimeter of the void
“Small world” effect demonstrated by Milgram ['67]

Kleinberg’s model: n x n lattice, plus long range edges

Result: greedy routing finds short (O(log^2 n)) paths with high probability if and only if r = 2
Assumes a treelike graph (or, graph with a “core”)

- routes go up (provider links), over (peering links), and down (customer links)
- i.e., valley-free
Address is effectively a subgraph, not just a number!

- here “address” means “destination-specific location info”

Up-graphs are small

Union of source and dest subgraphs is all we need

- exploits Internet’s current structure to find good paths

Q: How well does NIRA satisfy our goals?

- small address
- small node state
- low stretch
But what if our network does not have a "special" structure?
Technique in practice: Hierarchy

No structure? Make one!

- 2-level hierarchy: nodes in clusters
- each node knows how to reach one node of each cluster and all nodes in its own cluster

Problems:

- Some paths very long
- Location-dependent addresses (as in earlier techniques)
Compact routing theory

Given arbitrary graph, scheme must:

- Construct state (routing tables) at each router
- Specify algorithm to create and forward packets given packet header and state at current router

Goals:

- Minimize maximum state at each router
- Minimize maximum stretch:

\[
\max_{s,t} \frac{s \leadsto t \text{ route length}}{s \leadsto t \text{ shortest path length}}
\]

- Reasonably small packet headers (e.g., \(O(\log n))\)
Compact routing theory


Name-dependent Addresses assigned by routing protocol
Name-independent Arbitrary (“flat”) names
Algorithm != Protocol

Not distributed

Not dynamic

- Somewhat complex algorithms, e.g.
- multiple passes over graph to find landmarks
- routing through intermediate nodes chosen using non-local information

Many practical issues still unaddressed

- congestion, policy, deployment...

Next: overview of name-dependent stretch 3 compact routing
Everyone knows shortest path to landmarks.
Enable approximately shortest paths.

```
addr(t) = (L(t),b,t)
```

route length = \( \text{dist. to landmark} + \text{dist. to } t \)
\[ \leq d(s,t) + d(t,L(t)) + d(L(t),t) \]

\text{triangle inequality}
Case 1: $d(s,t) \geq d(t,L(t))$: further than landmark

- route length $\leq d(s,t) + d(t,L(t)) + d(L(t),t) \leq 3d(s,t)$

Case 2: $d(s,t) < d(t,L(t))$: closer than landmark

- Trouble!
- Idea: in Case 2, just remember the shortest path.
Vicinities

\[ V(s) = \text{nodes } t \text{ s.t. } d(s,t) < d(t,L(t)) \]

\[ V(s) = \text{nodes } t \text{ s.t. } d(s,t) < d(s,L(s)) \]

Requires “handshaking”, but convenient to implement

How big are \( V(t) \)?

Need a landmark in my vicinity.

\( \tilde{\Theta}(\sqrt{n}) \) random landmarks: \( \tilde{\Theta}(\sqrt{n}) \)-size vicinities
Tool: Chernoff bound

“The sum of many small independent random variables is almost always close to its expected value.”

\[ X_i = m \text{ independent } (0,1) \text{ random variables} \]

\[ X = \sum X_i, \ E[X] = \mu \]

For any \( 0 \leq \delta \leq 2e - 1 \),

\[ \Pr[X < (1 - \delta)\mu] < e^{-\mu\delta^2/2} \]

\[ \Pr[X > (1 + \delta)\mu] < e^{-\mu\delta^2/4} \]

See, e.g., Motwani & Raghavan, Theorems 4.1 - 4.3
How many landmarks are enough?

Show that any node $v$ always has $\sim \ln n$ landmarks in its vicinity if we use about $\sqrt{c \cdot n \ln n}$ landmarks.

$X_i = 1$ if $i$th closest node to $v$ is landmark, else $X_i = 0$

$$
\Pr[X_i] = \frac{\sqrt{c \cdot n \ln n}}{n}
$$

$$
E[X] = (\text{Number of nodes in vicinity}) \cdot \Pr[X_i]
$$

$$
E[X] = \sqrt{c \cdot n \ln n} \cdot \frac{\sqrt{c \cdot n \ln n}}{n} = c \ln n
$$

$$
\Pr \left[ X < \frac{1}{2} c \ln n \right] < e^{-(c \ln n) \cdot \frac{1}{4} \cdot \frac{1}{2}} = e^{\ln n^{-c/8}} = n^{-c/8}
$$

Increase $c$ to make this arbitrarily small.
Analysis

Stretch

• \( \leq 3 \) if outside vicinity (after “handshake”)
• \( = 1 \) if inside vicinity

State (data plane)

• Routes to landmarks: \( O(\sqrt{n \log n \cdot \log n}) = \tilde{\Theta}(\sqrt{n}) \)
• Routes within vicinity: \( O(\sqrt{n \log n \cdot \log n}) = \tilde{\Theta}(\sqrt{n}) \)

Address size

• This simple implementation: depends on path length, but very short in practice
• More complicated/clever storage of route from landmark to destination: \( \Theta(\log n) \)
To build state,

- Estimate $n$
- Pick $\sim n^{1/2}$ landmarks (independent random choices)
- Learn routes to vicinities & landmarks using standard path vector protocol

To route from $s$ to $t$ given $t$’s address,

- Check $V(s)$. If $t$ isn’t there, then...
- Route to landmark, and from there to $t$ along path in $t$’s address
- $t$ checks $V(t)$. If $s$ is there, reply with shorter path
Routing on **flat names** with low stretch and state

- we assumed source knows destination address

Other points state-stretch tradeoff space

- we saw state $\sim n^{1/2}$, stretch 3

**Why you cannot do better than this**

- ...in the general case (dense graphs)

**How to handle interdomain routing policies**

- no one knows!
<table>
<thead>
<tr>
<th>Number of Nodes (in thousands)</th>
<th>State at a Node (in thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disco</td>
</tr>
<tr>
<td>0.2</td>
<td>ND-Disco</td>
</tr>
<tr>
<td>0.4</td>
<td>S4 [Mao et al '07]</td>
</tr>
</tbody>
</table>

**Mean State at a Node**

- **Disco**
- **ND-Disco**
- **S4 [Mao et al '07]**

**Shortest path routing**

- **AS-level Internet**
- **Router-level Internet**

**Geometric random graphs**
16,000-node Geometric random graph

Router-level Internet topology
Disco summary

Builds on past algorithmic work for a distributed solution to routing on flat names, compactly and efficiently.

Only began to address how one would architect a compact routing solution in the Internet.
1. How many levels of routing hierarchy appear in Internet?

2. Could a protocol like Disco, that provides scalable routing on flat names, be integrated with the current Internet?
   
   1. What would it improve?
   2. What would be the challenges?
Announcements

Piazza working?

Next time: Reliability