Generalized Forwarding

table = dict()
on recv(pkt, ifin):
    if pkt.dst in table:
        send(pkt, table[pkt.dst])
    else:
        # Drop the packet
At hosts:
on boot?
  bcast(id)

On recv(p,i)
  if !fromrecv on
  bcast(id)

At switches:
on boot
  bcast(id, ady ma)

on recv(c,p)
  p.
  updates ady matrix
  face-map[C(p.send)]
  id
  if changed ady matrix
  .bcast(id,001)

Link state routing
An Aside: Detecting failures

on boot

on recv

outoftime() ≥ 10s

3 = us * 1us
Handing failures in Link State Routing

The Flooding Best Link State Algorithm

SpTree

Root → Tree

Table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>h3</td>
<td>2</td>
</tr>
</tbody>
</table>

Inkstate routing
Problems with Link state

Amount of state in each node? \(O(Ed)\) # Node's degree

How to address?
Distance Vector

Want to move to Eko using less memory per node.

What can we get to?

What we will get to.
Think about starting from the spanning free protocol.

Treat each host as a root?

D: ho: 1: 2
   2: 3
   0: 0

h0: (0,1)

h1

h2

h3

h4

ho: (0,3)

h0: (3,2)

h0: (2,13)

ho: (0,13)

ho: (3,2)