Modified Flood

deactivated = Set()
on recv(pkt, ifin):
    if ifin in deactivated:
        return
    for i in interfaces:
        if (i != ifin and
            i not in deactivated):
            send(pkt, i)

Consider
alice P→ bob

Consider
bob R→ alice

What nodes process each packet?

What nodes need to process each packet?
This lecture: have fewer nodes process each packet.

Why?

- Scalability

  - Each link has a limit on the rate at which it can carry packets

Challenge?

How to get switches to deliver without flooding?

Learning
def on_recv(p, iface):
    table = dict()
    if p.dst in table:
        send(p, table[p.dst])
    else:
        flood(p, iface)

A. Table

<table>
<thead>
<tr>
<th>A</th>
<th>Bob</th>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/2</td>
<td>3</td>
</tr>
</tbody>
</table>

Learning
How to handle hosts moving?

Now for something different.
- Number of nodes between $h_2$ & $h_3$?
- Capacity when $h_2$ sends to $h_3$ and $h_0$ sends to $h_3$?

Can we forward packets better?

**Generalized Forwarding**

```python
table = dict()
on recv(pkt, ifin):
    if pkt.dst in table:
        send(pkt, table[pkt.dst])
    else:
        # Drop the packet
```

**Problem**: How to populate `table`?
Q: What paths to choose? Shortest?

How to compute shortest paths?

Two options:

* Use some extension to the spanning tree protocol. — Lab 2, go over this later

* Get all nodes to learn the entire graph and use Dijkstra's. — Go over this first
Protocol: learn the graph

* Learn neighbors

* Learn neighbors neighbors

Packet:

on boot:

\[ \text{blast}(\langle \text{id}, \Sigma \rangle) \]

on recv:

A: on boot

\[ \langle \text{id}=A, \text{nbrs}=[C] \rangle \]

B: recv (\langle \text{id}=A, \text{nbrs}=[C], 1 \rangle)
\[ \text{adj} = \{3\}. \]

```
onboot
\text{bos}(P(id, \text{adj}))
on recv(p)
\text{adj}[id]. \text{add}(p, \text{id}) <
merge(\text{adj}, p.\text{adj}) <
```
if adj has changed
boost((id, adx))