Sweet Little Lies
Fake Topologies for Flexible Routing

Nina Zinsli
Motivation

Goal: Send data packet from source to destination
Outline

Common Solutions for Network Routing
  ▪ Link-state Routing
  ▪ Software Defined Networks

Fibbing
  ▪ Using fake topologies for Network Routing
  ▪ Benefits & problems

Evaluation
Common solutions for Network Routing
Common solution

Link-state routing protocols
  ▪ widely used to steer network traffic
Link-state routing protocols

- every node has a map of the whole network
- compute forwarding path for every destination (only needs to know next hop)
Constructing network map

Constructing the map from router A’s point of view:

1. determine neighbours and cost of connection
Constructing network map

Constructing the map from router A’s point of view:

1. determine neighbours and cost of connection

link-state packet:

<table>
<thead>
<tr>
<th></th>
<th>dist. from A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Constructing network map

Constructing the map from router A’s point of view:

2. Flood link state packet through network

link-state packet:

<table>
<thead>
<tr>
<th></th>
<th>dist. from A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Constructing network map

Constructing the map from router A’s point of view:

2. Flood link state packet through network

link-state packet:

<table>
<thead>
<tr>
<th>dist. from A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Constructing network map

Constructing the map from router A’s point of view:

2. Flood link state packet through network

link-state packet:

<table>
<thead>
<tr>
<th>dist. from A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>
Constructing network map

Constructing the map from router A’s point of view:

3. Receive link state packet from other routers
Constructing network map

Constructing the map from router A’s point of view:

3. Receive link state packet from other routers

link-state packets:

<table>
<thead>
<tr>
<th>dist. from B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dist. from C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dist. from D</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>
Constructing network map

Constructing the map from router A’s point of view:

4. Construct network map from link-state packets
Constructing network map

Constructing the map from router A’s point of view:

4. Construct network map from link-state packets

⇒ now A knows the whole topology
Example OSPF (Open Shortest Path First)

- Widely used link-state protocol
- Routers learn about topology like shown before
- Find shortest path
Which path will OSPF choose?
OSPF example

Which path will OSPF choose?
We know the solution

A

12

B

C

10

3

D

4

1

source

destination
Advantages of this approach

- implementations are robust and widely-deployed
- deterministic algorithm
- behaviour well-understood (no surprises!)
- messages are standardized (standard protocol)
We are highly dependent on the red link
Problems with OSPF

3 examples where OSPF is not ideal:

- link failure
- DDoS
- load balancing
What if the link from C to D fails?
Link Failure

A — X — B

C — X — D

source — destination
Link failure

We want to have a backup plan to react fast and redirect the data:

![Diagram showing network links and failure path]

source  destination
DDoS attack

Distributed Denial of Service:
- attacker attempt to make an online service unavailable
- overwhelm it with traffic from multiple sources
- congest links
DDoS

- A
- B
- C
- D

Source: C
Destination: D

Red arrows represent DDoS traffic.
DDoS

Link between C and D congested!

A --- B

C --- D

source            destination
DDoS

What we want:

![Diagram of DDoS network showing nodes A, B, C, and D with arrows indicating the flow of traffic from source to destination.]
Load balancing

Huge amount of traffic from two sources
⇒ we want to split it on two different paths
Load balancing

OSPF solution

source 1

0.25

source 2

destination 1

0.75

destination 2
Load balancing

What we want:

source 1 → destination 1
0.25

source 2 → destination 2
0.75

source 2 → source 1
0.75

source 2 → destination 2
Better solution, maybe?

SDN (Software Defined Networks)

- can also be used to steer network traffic
- central controller chooses path for all traffic
- used by Google, Microsoft, ...
Better solution, maybe?

SDN (Software Defined Networks)

- can also be used to steer network traffic
- central controller chooses path for all traffic
- used by Google, Microsoft, ... 
- does not scale to big networks
- cannot be used with most current routers (e.g. Cisco)
Better solution?

We want a solution which combines the benefits of both OSPF and SDN!
Better solution?

What we want:

- scales to big networks
- no central controller
- routers calculate the paths
- more flexible than OSPF
- works on existing routers (no large deviations from OSPF)
Fibbing
Fibbing

to fib: to lie about something minor or unimportant
Solution: Fibbing

New way to make network routing more flexible.
Solution: Fibbing

New way to make network routing more flexible.

⇒ Shortest-Path-Violations
Solution: Fibbing

Idea: Show the routers a **fake** topology.
Solution: Fibbing

Idea: Show the routers a **fake** topology.

- add fake nodes to real topology (not physically)
  \[ \Rightarrow \text{Router sees a different topology} \]
Solution: Fibbing

Idea: Show the routers a *fake* topology.

- add fake nodes to real topology (not physically)
  \[ \Rightarrow \] Router sees a different topology

How the network looks like:
Solution: Fibbing

Idea: Show the routers a **fake** topology.

› add fake nodes to real topology (not physically)
  ⇒ Router sees a different topology

How the network looks like:

How R thinks the network looks like:
Solution: Fibbing

Idea: Show the routers a **fake** topology.

- add fake nodes to real topology (not physically)
  ⇒ Router sees a different topology

How the network looks like:

```
  B  5  1
     ↓
  A
```

How R thinks the network looks like:

```
  B  5  1
     ↓
  A
```

Router R computes shortest path on the second network
Fibbing

This allows us to make Router R choose a path which is not the shortest.
(if a path with a fake node is shorter)
Fibbing

This allows us to make Router R choose a path which is not the shortest.
(if a path with a fake node is shorter)

But a data packet cannot be sent over a fake node 😞
Fibbing

Assume R wants to send a packet to B:

Shortest path in real network:

Fake network:
Fibbing

Assume R wants so send a packet to B:

Shortest path in real network:

![Real Network Diagram]

Shortest path in fake network:

![Fake Network Diagram]
Fibbing redirects data over existing link:

Shortest path in real network:

Shortest path in fake network:
The Fibbing controller

- announces fake node to routers
  - local (seen by single router)
  - global (seen by all routers)
- chooses them such that routers send traffic over desired path
The Fibbing controller

- just used to insert fake nodes!
  does not compute paths
- mostly only a few Shortest-Path-Violations
- multiple controllers can be used for different subnets
The Fibbing controller

- just used to insert fake nodes!
  does not compute paths
- mostly only a few Shortest-Path-Violations
- multiple controllers can be used for different subnets

⇒ Fibbing controller can be used in big networks
Input
Physical topology + desired path
Output

Physical topology

+ desired path

Topology with fake nodes

⇒
How fibbing solves all three problems
Load Balancing Example

(a) Topology

(b) Throughput evolution
Fibbing is expressive

Good news:

**Theorem** Any set of desired paths can be enforced by Fibbing.
How Fibbing works

2 Algorithms:
1. Simple
2. Merge
Simple
Simple

- is used if we want to react fast
- local fake node for every shortest-path violation
Simple

- is used if we want to react fast
- local fake node for every shortest-path violation
- might introduce a lot of new fake nodes!
Merge

- is used to reduce the number of fake nodes
- can be used to compute backup plans
- can be used after Simple to clean up
Merge

**Goal:** Merge local fake nodes to global fake nodes whenever possible to reduce number of fake nodes
Goal: Merge local fake nodes to global fake nodes whenever possible to reduce number of fake nodes

- for every local fake node, save the minimum and maximum weight
- take two nodes together if possible
Problems with implementation
Problems with implementation

- with current routers not possible to lie about direct neighbour
- if desired path differs from shortest path in the first hop, we can not achieve it
Problems with implementation

- with current routers not possible to lie about direct neighbour
- if desired path differs from shortest path in the first hop, we can not achieve it
- with small changes in routers it should be possible
Evaluation
Evaluation

- test how number of desired Shortest-Path-Violations affects number of fake nodes
Evaluation

- ○: Median # of nodes
- □: Median # of edges
- solid bar: 95th percentile
- dashed bar: 5th percentile
Evaluation

- ○: Median # of nodes
- □: Median # of edges
- solid bar: 95th percentile
- dashed bar: 5th percentile
- real network (AS 6461, 141 nodes, 748 edges)
- random desired paths
Evaluation

- not many fake components needed, max # nodes: 5, edges: 26
- not strictly increasing
Memory and Time

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>RIB memory (MB)</th>
<th>OSPF memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>0.09</td>
<td>0.56</td>
</tr>
<tr>
<td>5,000</td>
<td>1.58</td>
<td>5.19</td>
</tr>
<tr>
<td>10,000</td>
<td>3.56</td>
<td>10.96</td>
</tr>
<tr>
<td>50,000</td>
<td>19.67</td>
<td>56.37</td>
</tr>
<tr>
<td>100,000</td>
<td>39.78</td>
<td>113.17</td>
</tr>
</tbody>
</table>

small memory and CPU overhead
Conclusion

- Fibbing achieves what we want: more flexible routing with few overhead
Conclusion

- Fibbing achieves what we want: more flexible routing with few overhead
- tests on small networks seem to work
Problems

- Fibbing controller takes desired path as an input, does not find an alternative path itself
- We know that Fibbing always works, but there are no guarantees for speed and number of fake nodes
Questions

?