POP-ROUTING

OR: HOW TO AUTO-TUNE YOUR PROTOCOL TIMERS AND MAKE YOUR PROTOCOL MORE SCALABLE

LEONARDO.MACCARI
@
UNITN.IT

BM2016, Oporto

Who am I

- Rsearcher at the DISI (University of Trento), working on:
  - wireless mesh protocols
  - Community Networks (CN)
  - . . . lately trying to have a multidisciplinary approach.
- Activist of **Ninux.org**. And i try to mix the two things together :-)

Before we enter Pop-Routing...

Some Good News:

We got a new research project accepted to work on Community Networks.

netCommons: Network Infrastructure as Commons

- Three years: starting Jan 2016
- 6 partners
- Interdisciplinary: computer science, law, sociology
- What we will do:
  - Study and propose network protocols and applications for CNs
  - Study and propose organization models for CNs
  - Understand how the models used by CNs can scale, and what can we learn from them.

Some Good News:

- We will be working with the communities, many of us already have contacts with the communities, or are part of one community.
- There are other people from the project around
- Don't hesitate to ask more

More Good News:

We are hiring:

- In the DISI (university of Trento) we have two open positions, that will close in the next few weeks:
- One Ph.D. position: 3 years starting September/November
- One post-doc position (or equivalent experience): 2+1 years, starting ASAP

If you are motivated to do applied research on CNs, talk to me!

More Info About the project and the positions:

- http://netcommons.eu
- info@netcommons.eu

Back to Pop-Routing: what is this about?

- Every routing protocol uses control messages, for instance, OLSR uses primarily `Hello` and `TC` messages.
- How do you set the generation timers for `Hello` and `TC` (H_t, T_t)?
- RFC says H_t =2s and T_t =5s.
- is it too much? or too small?
- it depends. . . pros and cons:

Cons: increase overhead

Consider a network with:

- N nodes
- L links
- an average of d links per node

- Every node sends 1 `Hello` every H_t on every link:

$$\frac{N * d}{H_t}$$

messages per second.

- A TC must reach every node in the network, so, in the best case, it will pass across each link in the network:

$$\frac{N * L}{T_t}$$

messages per second.

Example:

Consider a network with:

- $N = 100$
- $L = 120$
- $d = 4$

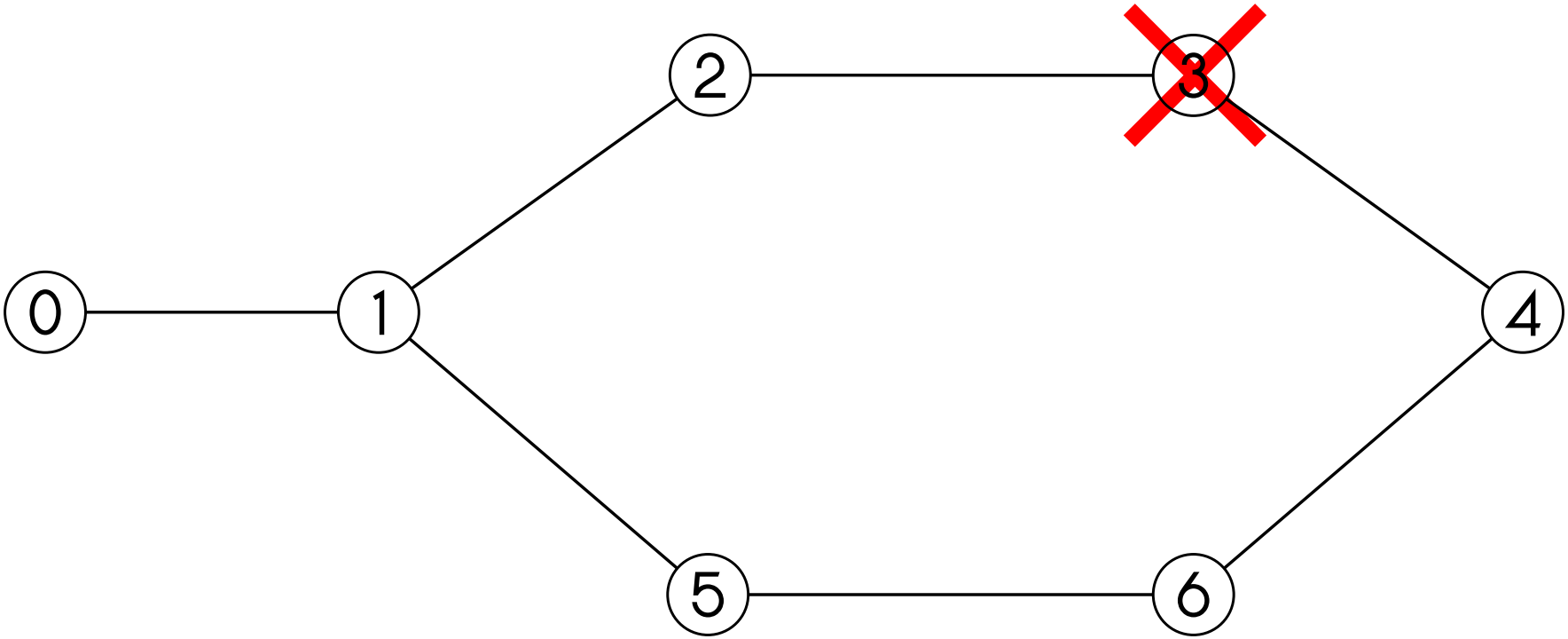
$$\frac{100 * 4}{2} + \frac{100 * 120}{5} = 2600 \text{ messages per second}$$

$$\frac{2600}{120} \simeq 21 \text{ messages per second per link}[1]$$

- Basic rate in 802.11n is 6Mb/s which is tens of times slower than the data rate

[1] No aggregation considered here

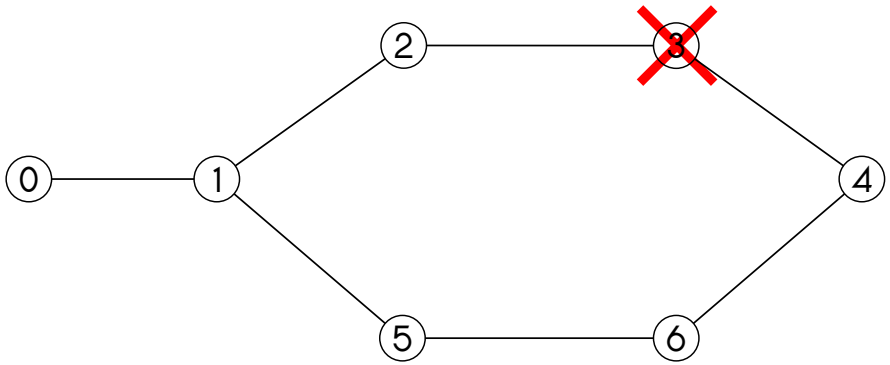
Pros. Consider this network:



Let's say that:

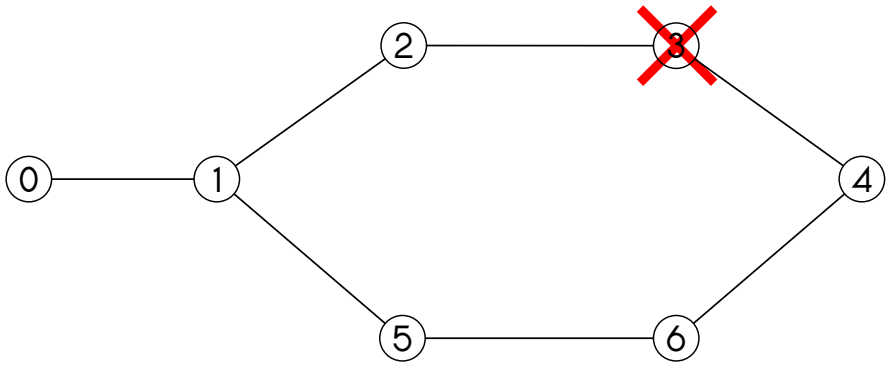
- the shortest path from 0 to 4 is: 1,2,3,4
- node 3 breaks down at time 0

Pros. Link failure detection:



- node 2 will wait for 3 lost `Hello` messages before he considers the link 2-3 dead (T_d)
- After T_d node 2 will decide the best path to node 4 is 1,5,6,4.
- Node 1 does not know yet that node 4 has failed, it still uses path 1,2,3,4.
- A loop is created.

Pros. Link failure propagation[2] :



- At time T_p node 2 generates a `TC`, that says that link 2-3 is not active anymore
- Node 1 will flip its route and pass through 5,6,4

[2] Note there is a simplification here, since i don't consider the fact `Hello` messages contain the whole neighborhood.

Summing up (worst case scenario):

- at time 0 node 4 fails
- it takes $T_d = 3 \times H_t$ to detect the failure
- now node 2 detected the failure but still you have a loop
- at time $T_d + T_t$ node 2 generates a `TC` and the problem is solved.
- with default values, in the worst case scenario the routes remain broken for $2 \times 3 + 5 = 11s$!!

Summing up:

Note that:

- This is the worst case scenario (you may be more lucky than that)
- I omitted the time due to message propagation, which makes everything worse, especially if you have aggregation.
- In any case the total time in which routes are broken is proportional to T_d and to T_t

This is why you want to have high frequency in the generation of `Hello` and `TC` messages.

We need a trade-off

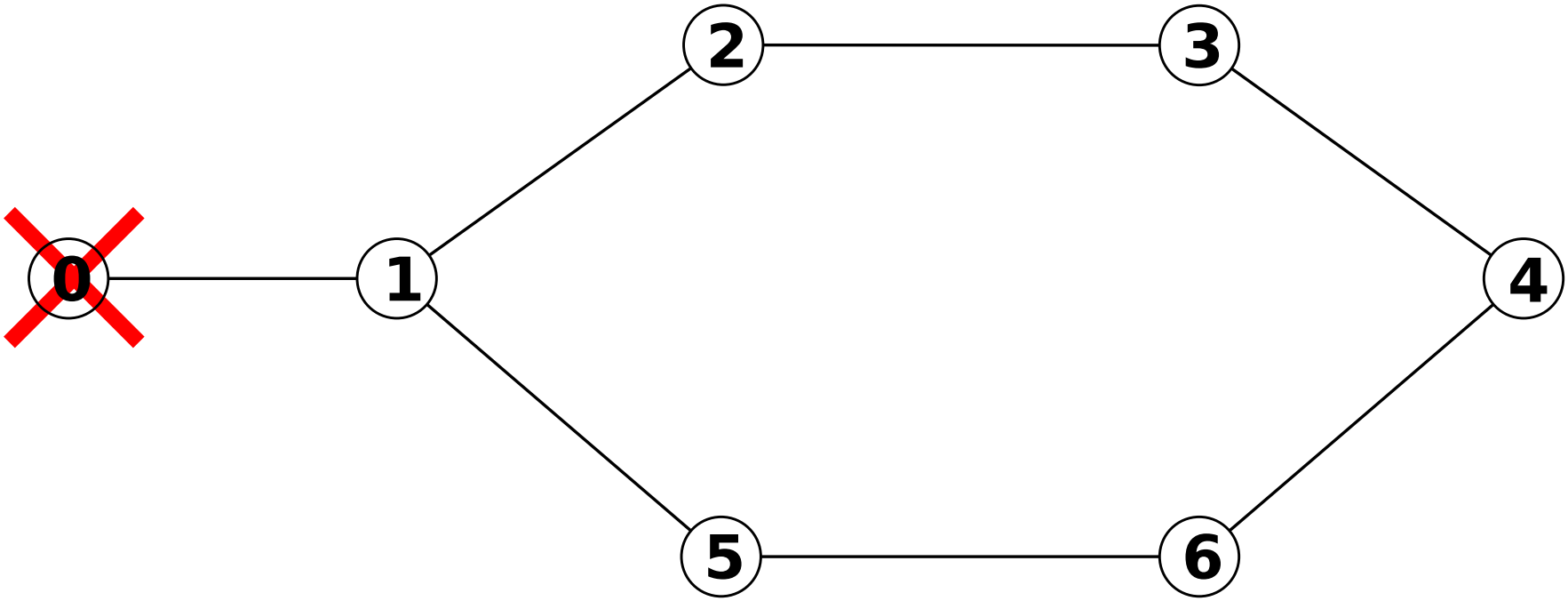
Observation 1:

Timers do not need to be set to the same value for all the nodes:

- `Hello` and `TC` messages include a field that says how often they are generated
- Receiving nodes can estimate when they will receive a new message
- Thus they can detect link failure also with etherogeneous timers

Observation 2:

Not all failures are equal, who cares if node 0 breaks down?



Intution behind Pop-Routing:

- We want important nodes to generate control messages with high frequency
- We want non-important nodes to generate control messages with low frequency
- How do we define an important node?
- An important node is a node across which many shortest path pass.
- The number of shortest paths that pass through a node i, is called *Betweenness Centrality*: B_i

Pop-Routing principles:

- Every node needs to know the full network graph
- It will compute the network centrality of every node in the network
- It will set its timers depending on the values of centrality
- Basically, the nodes in the center of the network will be *loud*, while the nodes in the extremes of the network will not be *loud*.
- This resembles the equalization presets of media players for pop music. From here, Pop-Routing :-)

Pop-Routing: math

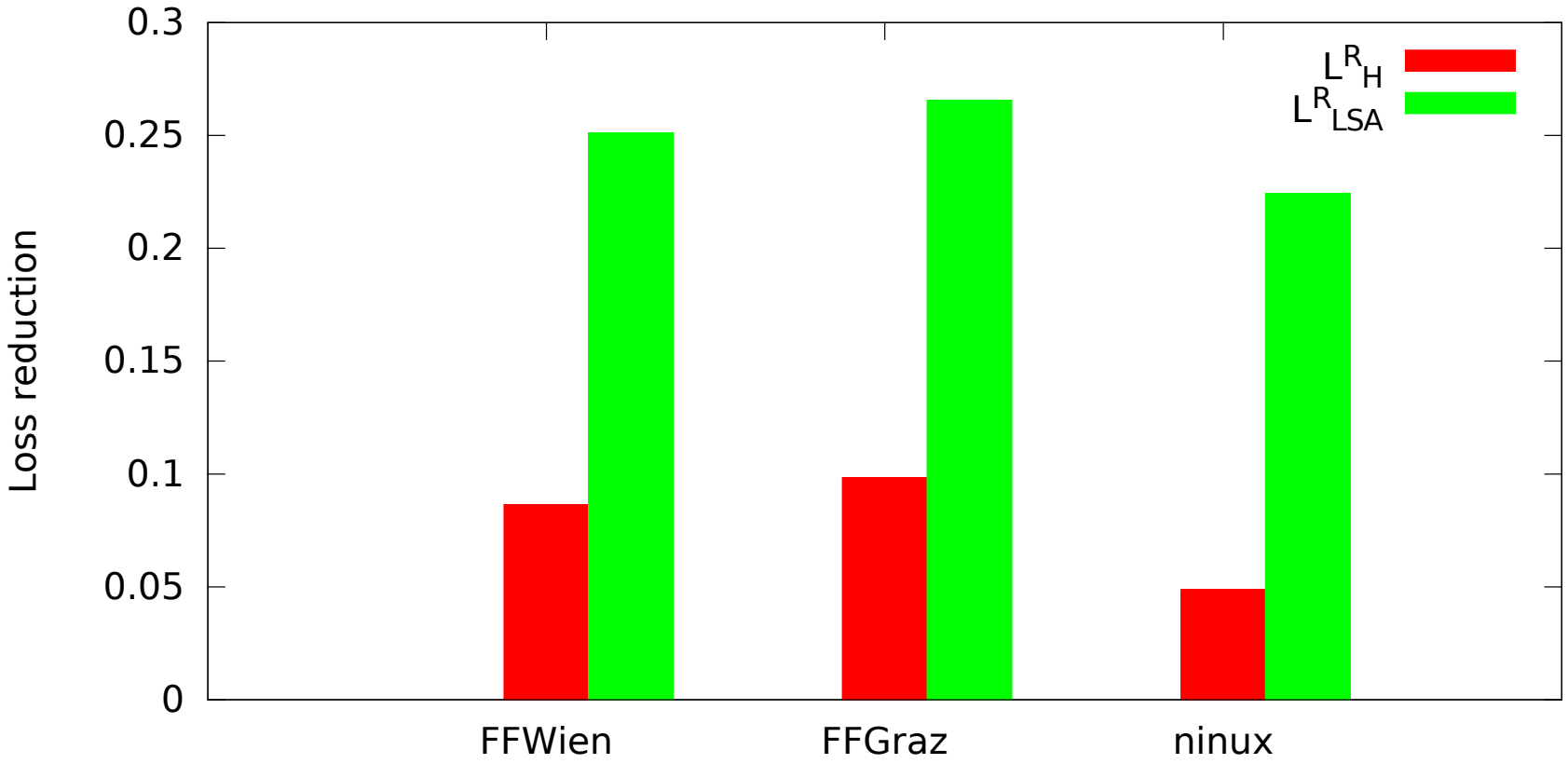
- I will spare you the math, but the optimal trade-off between route convergence speed and control message overhead is given when [3]:

$$H_t(i) = \frac{\sqrt{d_i}}{\sqrt{B_i}} \frac{1}{O_H} \sum_{j=1}^{N} \sqrt{B_j d_j} \tag{1}$$

$$T_t(i) = \frac{\sqrt{L}}{\sqrt{B_i}} \frac{\sum_{j=1}^{N} \sqrt{B_j L}}{O_H} \tag{2}$$

[3] Leonardo Maccari, Renato Lo Cigno, *Pop-Routing: Centrality-based Tuning of Control Messages for Faster Route Convergence*, Infocom 2016

Pop-Routing: what is this for?



Emulated networks, real topologies: route convergence upon node failure is faster, with the same number of control messages.

Pop-Routing: requisites

What a node needs to know to compute its optimal values:

- The betweenness of every node
- The degree of every node

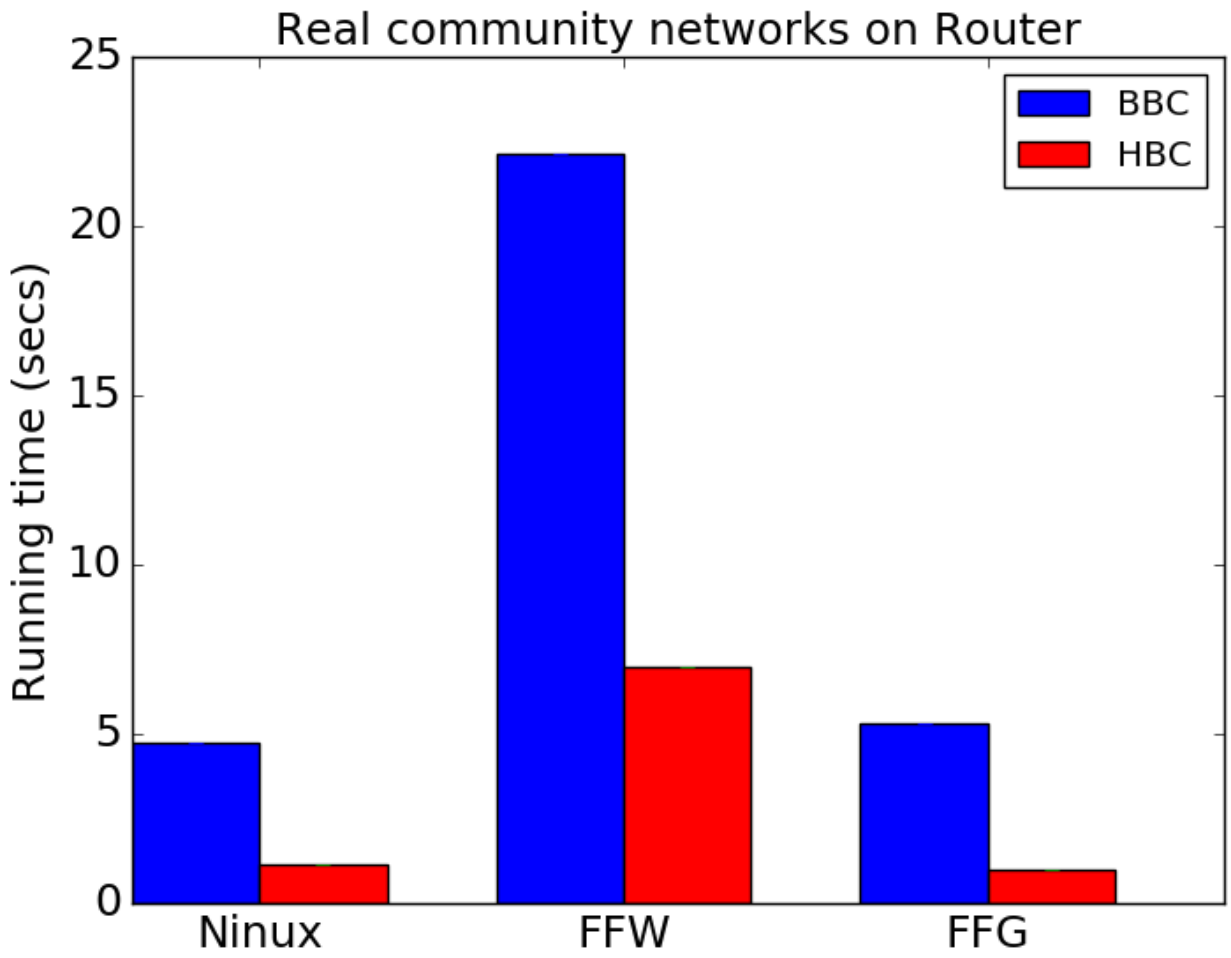
Pop-Routing: from emulation to real networks.

Issue one: how complex is to compute betweenness?

1. Dummy approach: first you compute all the shortest paths. then you count how many times each node falls in one shortest path $\Rightarrow O(N^3)$
2. Better approach: you accumulate centrality as you compute all the shortest paths $\Rightarrow O(N^2 log(N))$
3. Better heuristic approach: first you preprocess the network graph, then you compute centrality.

Pop-Routing: from emulation to real networks.

Computation time: three networks (120-240 nodes, Ubiquiti M2)



Pop-Routing: from emulation to real networks.

- centrality does not need to be computed every second, network topology changes slowly.
- 1 to 7 seconds with 240 nodes is not bad, can be done, once per X minutes
- but it is not something you can do in the main of the routing protocol daemon

Nework topologies

To fine-tune the algorithm I need to access snapshots of network topologies, do you publish topologies in a place I can reach?

Pop-Routing: from emulation to real networks.

Issue two: what about distance vector protocols?

- They should be loop-free but they are slower to converge
- They do not know the whole topology, so they can not use the optimized
- There can be solutions:
  - Gossiping algorithms can be used to compute one own's centrality (TBD)
  - Distribute the value to all nodes