ns-3 RPL module:
IPv6 Routing Protocol for Low power and Lossy Networks

Lorenzo Bartolozzi
Tommaso Pecorella
Romano Fantacci

Università degli Studi di Firenze

Wns3 2012, March 23,
Desenzano, Italy.

This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.

Low power and Lossy Networks

- What is a LLN ?
- LLNs: open issues

RPL standard

- RPL: Routing Protocol for LLNs

ns-3 implementation

- Motivations

Test results

- Basic scenario
- Large Scale scenario

Conclusions

LLN: terminology

LLN: Low power and Lossy networks (LLNs) are typically composed of many embedded devices with limited power, memory, and processing resources interconnected by a variety of links, such as IEEE 802.15.4 or Low Power WiFi. There is a wide scope of application areas for LLNs, including industrial monitoring, building automation (HVAC, lighting, access control, fire), connected home, healthcare, environmental monitoring, urban sensor networks, energy management, assets tracking and refrigeration.

Reference IETF Working Groups:

- Routing Over Low power and Lossy networks WG (ROLL)
- IPv6 over Low power WPAN (6lowpan)

LLNs: facts

LLNs have not to be confused with Wireless Sensor Networks.
A WSN is an LLN, but not the opposite !

LLNs are not necessarily wireless.
Industrial automation networks are wired.

LLNs will use IPv6. No discussions about this.
IPv4 is a dead horse.

LLN nodes can not be easily configured.
Typically they're cheap, small and with very limited memory/CPU.

LLNs: open issues

Transmission efficiency (6lowpan)

Nodes run on battery. Saving energy is a premium. IPv6 Headers must be compressed.

Node bootstrap (6lowpan-nd)

Configuration is a nightmare, and 'normal' IPv6 auto configuration is too chatty, NS/NA as well, etc,.

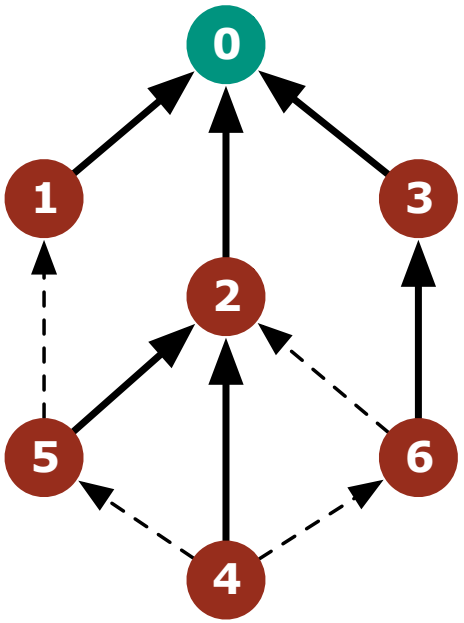
Routing (ROLL)

Routing requirements are scenario-based. Mainly:

- one-to-one (configuration, data reading)
- many-to-one (data gathering)
- one-to-many (data dissemination)

RPL: basics

RPL is not a standard... yet. There is a draft (19th release right now).



Routing based on graph construction

- DAG: Direct Acyclic Graph
- DODAG: Destination Oriented DAG

New ICMPv6 Messages;

- DIS: DODAG Information Solicitation
- DIO: DODAG Information Object
- DAO: Destination Advertisement Object

Many options for each kind of message, carried in sub-headers.

RPL: basics

An *extremely* simplified scenario:

Node not joined to any DODAG:

- Send a multicast DIS to FF02::1A,
- Receives one (or more) DIS,
- Join the DODAG and announce itself with a DAO (*).

A node joined to a DODAG performs maintenance duties, like:

- Keeps sub-DODAG consistent,
- Performs Neighbor Unreachability Detection,
- Finds if there is any route *optimization* for itself.

RPL: more complex than it seems

The source of complexity in RPL is its flexibility.

- **Mobility**: multiple 'up' routes are maintained. There can be even multiple *roots*.
- **Route optimization**: Optimal upward routes are calculated according to *Metrics* and *Objective Functions*.

Metrics

A metric defines how a node (or link) will affect the path.

- Additive and min-max metrics.
- Node energy, hop count, node reliability, etc.

Objective Functions

OFs will decide if the route optimization have to be performed. E.g., a route might be not "better enough" to perform an optimization.

RPL: Objective Functions

An Objective Function is able to compare two objects' metrics and decide:

- The Preferred Parent between two candidates A and B
- The node *rank* w.r.t. the Preferred Parent.

Objective Function 0 (Of0)

Extremely simple OF, the node will choose always the node with the minimum rank.

Minimum Rank Hysteresis Objective Function (Mrhof)

The Preferred Parent is changed only if the rank difference is greater than a given value. Less aggressive optimization and more stable DAGs.

RPL: Metrics (and Constraints)

A metric is used to compute the node's rank *and* to decide about preferred Parents.

Node Metric/Constraint

Node-related informations, such as:
Node State and Attributes, **Node Energy**, **Hop-Count**

Link Metric/Constraint

Link-related informations, such as:
Throughput, Latency, Link Reliability, Link Color

Each metric is carried in a *DAG Metric Container*, embedded in the relevant RPL control messages.

Why another RPL implementation ?

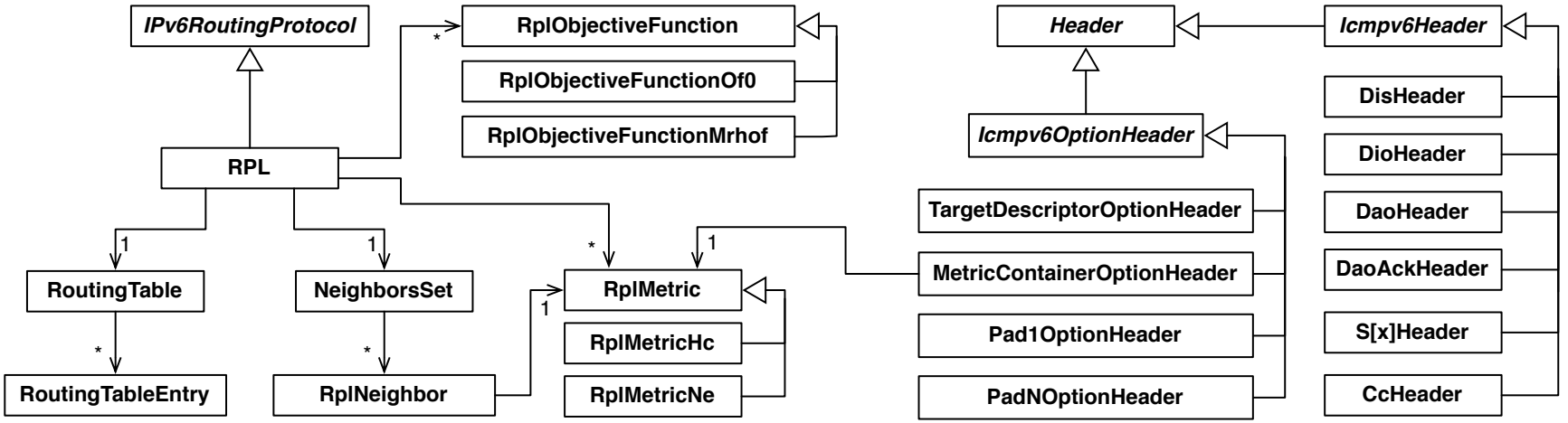
There are a number of implementation, but we did not like any.

1. Omnet++ (Castalia): incomplete.
2. Cooja (Contiki): it's not a simulator, it's a real implementation.
3. TOSSIM (TinyOS): same as for Cooja.

We wanted a tool to:

- Learn for real what's RPL and all it's implementation's headaches.
- Have the maximum parameter flexibility.
- Experiment new OFs, Metrics, Constraints, etc.
- Be able to run experiments *beyond* what's actually implemented.

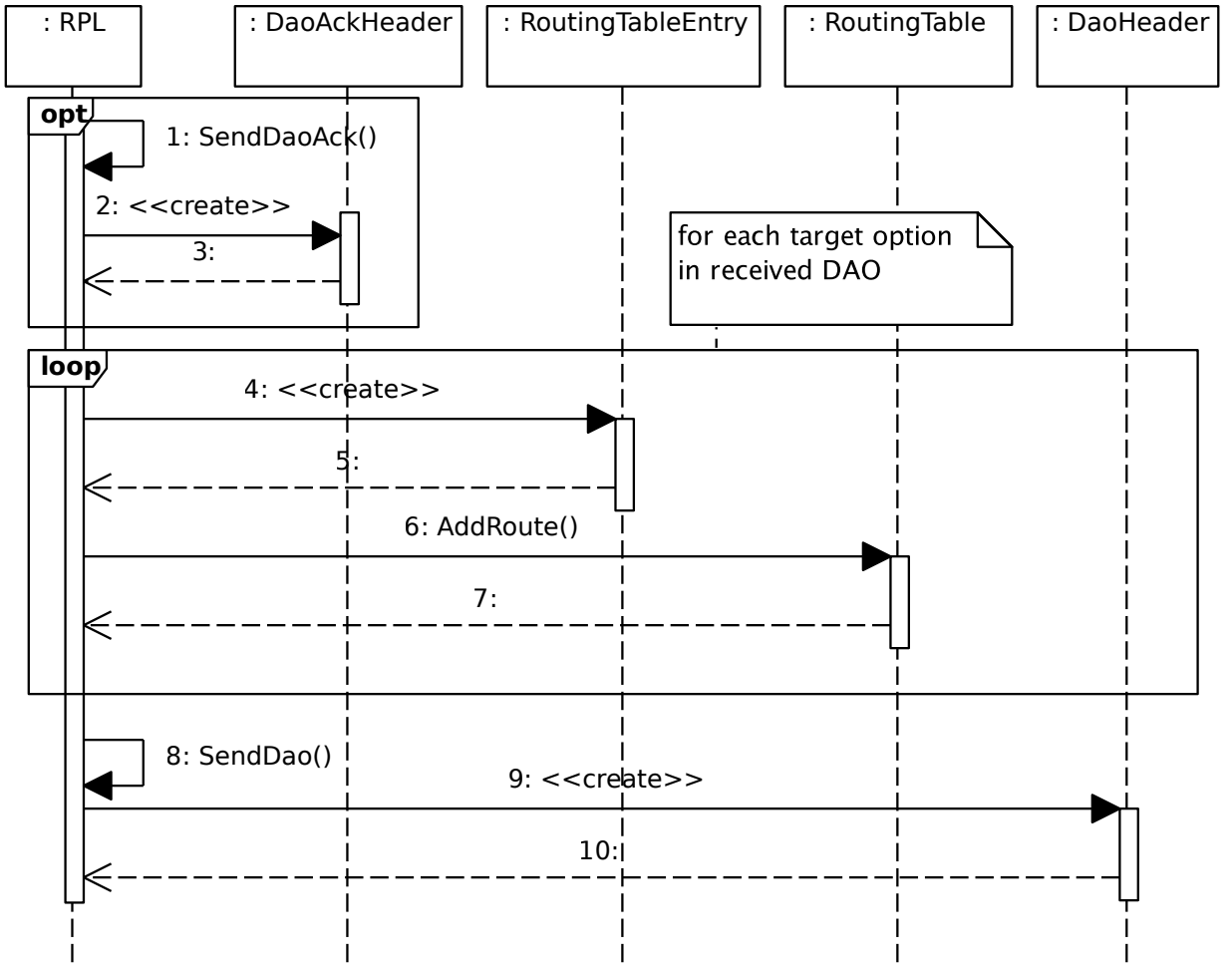
Classes



Noteworthy classes:

- NeighborSet: storing neighbors relationships.
- RplObjectiveFunction and RplMetric: extensible set.
- RoutingTable: still trying to downgrade it to a “normal” one.

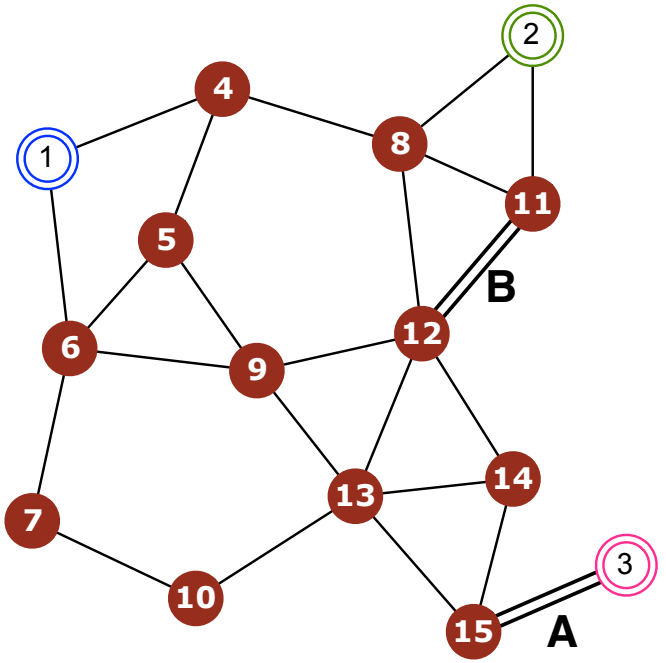
DAO Operations



Other sequence diagrams are in the paper.

Basic scenario

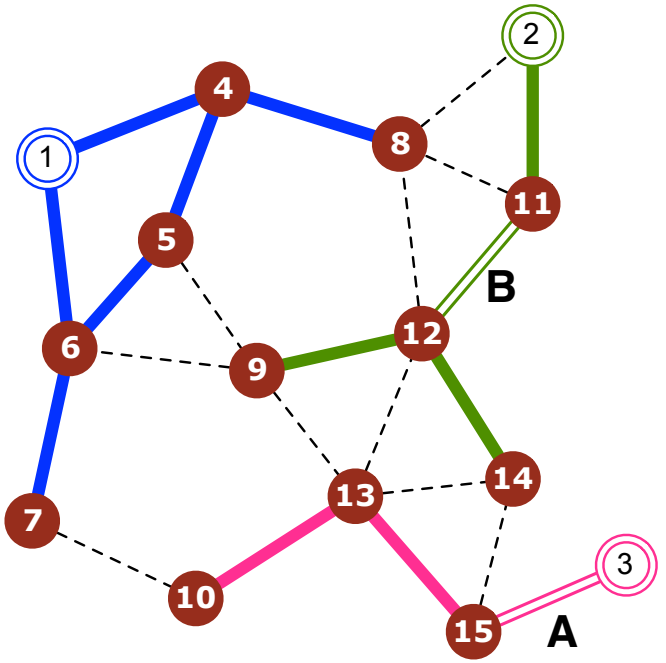
The basic scenario is 15 nodes and point-to-point links.



Links A and B will break during the simulation.

Basic scenario

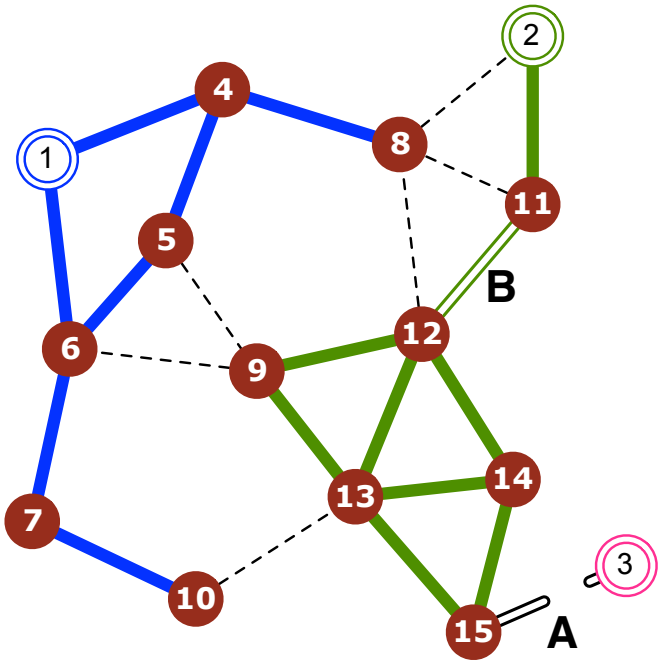
The basic scenario is 15 nodes and point-to-point links.



Race conditions might change the topology.

Basic scenario

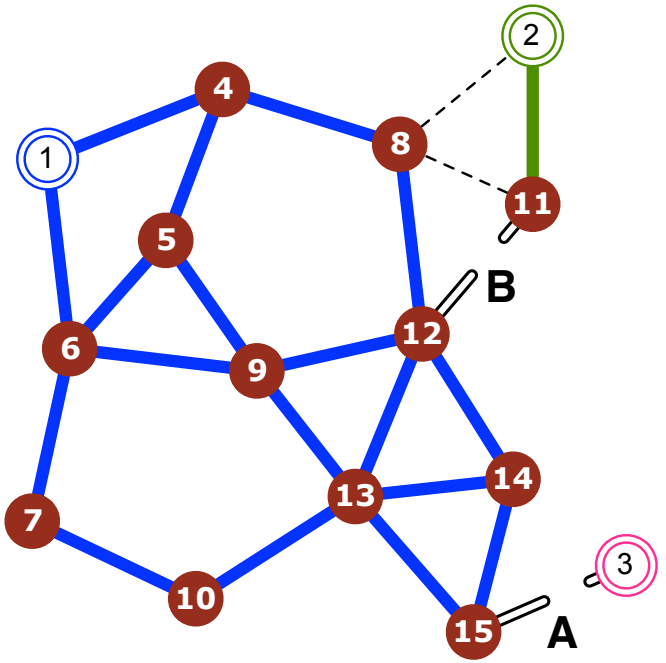
The basic scenario is 15 nodes and point-to-point links.



Node 10 behavior is interesting.

Basic scenario

The basic scenario is 15 nodes and point-to-point links.



All nodes are connected to a root, always.

Large scale scenario

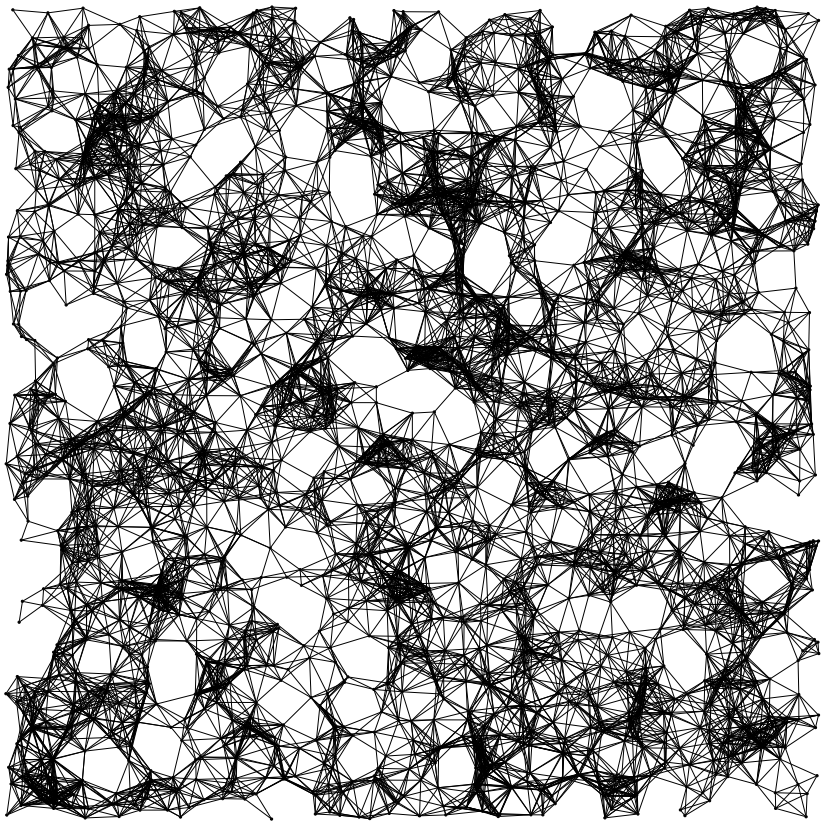
The large scale scenario is made of:

- 2000 nodes over a 1Km x 1Km area
- Random positions, Poisson distribution.
- Radio coverage of each node is 50 m.
- 14.967 *equivalent* point-to-point links.

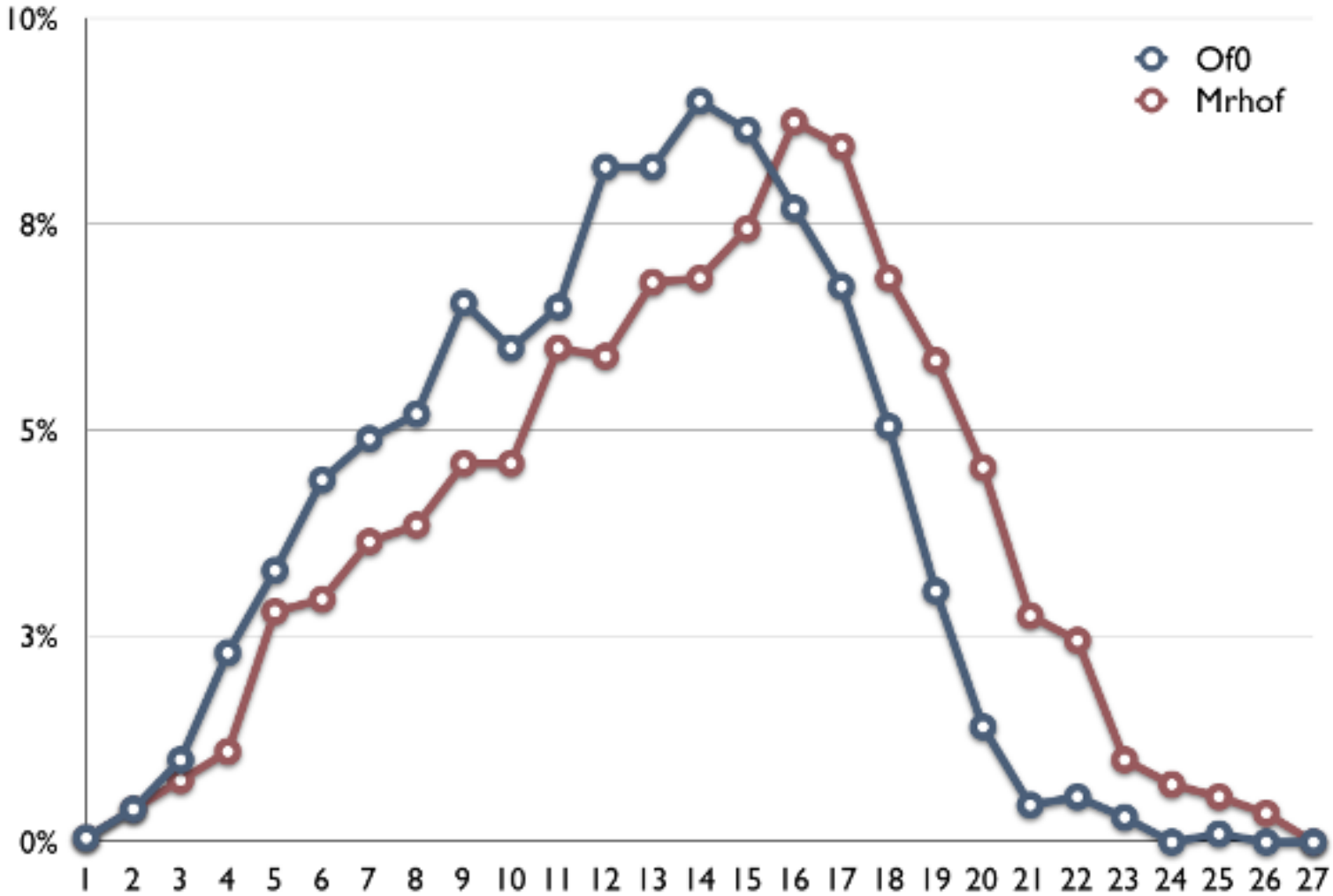
Obtained in this way:

- Custom program (few lines) to randomize node position, calculate the links and cut out disconnected nodes (if any).
- TopologyRead to read the resulting topology.
- Normal ns-3 simulation.

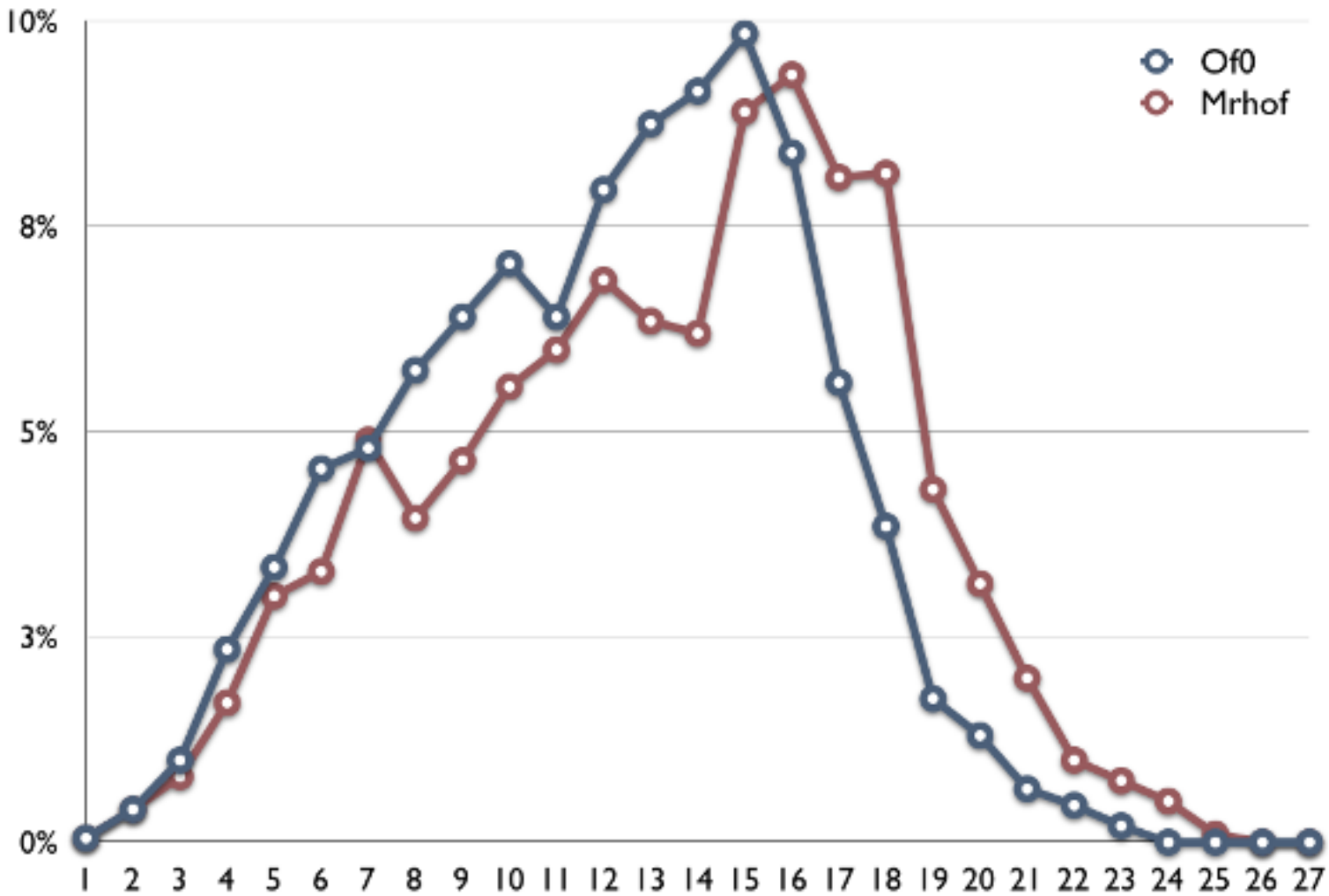
Large scale scenario



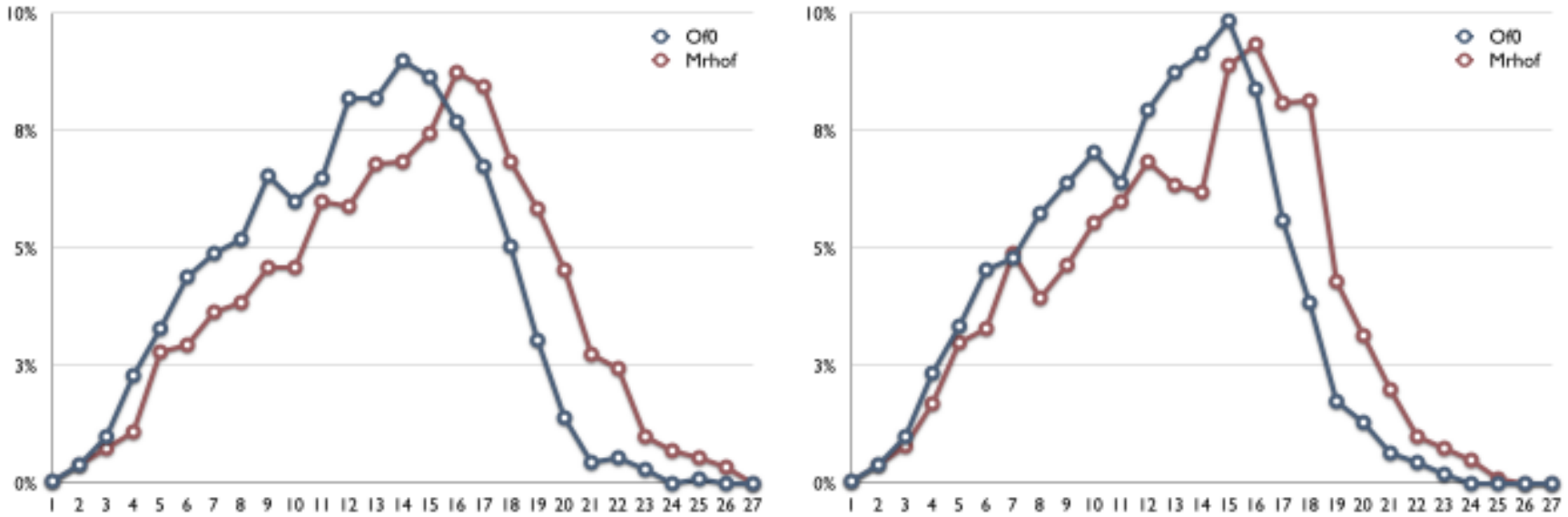
Large scale scenario results



Large scale scenario results



Large scale scenario results



- Some bugs exterminated after paper submission.
- Of0 is more 'aggressive' in route optimization
- Mrhof generates less overhead and less topology reconfiguration (during the transitory setup).

Lessons learned

- Started one year and half ago. And it's not yet completed.

SLOC	model	helper	examples
rpl	8793	158	113
olsr	3864	78	242
dsdv	2046	51	256

Some things in life can never be fully appreciated understood unless experienced firsthand. Some things in networking can never be fully understood by someone who neither builds commercial networking equipment nor runs an operational network. [RFC 1925]

Corollary: Some things in protocols can never be fully understood by someone who did not write the code.

Next steps

The plan is to:

1. Complete the implementation and validate it. 1 year
2. Write a nice paper with the simulation results. 1 months
3. Submit rpl for public revision. 3 months
4. Integrate it with the main ns-3 codebase.

That's bound to fail miserably.

Next steps

The plan is to:

1. Complete the implementation and validate it. 1 year
2. Write a nice paper with the simulation results. 1 months
3. Submit rpl for public revision. 3 months
4. Integrate it with the main ns-3 codebase.

That's bound to fail miserably.

Next steps

The plan is to:

1. Complete the implementation and validate it. 1 year
2. Write a nice paper with the simulation results. 1 months
3. Submit rpl for public revision. 3 months
4. Integrate it with the main ns-3 codebase.

That's bound to fail miserably.

Next steps

The *real* plan is to:

1. Iron out some bugs and features - 1-2 months.
2. Write a nice paper with the simulation results - 1 month.
3. Check the implementation against Contiki/TinyOS traces.
4. Submit rpl for public revision.
5. Integrate it with the main ns-3 codebase.

The following points will be done through 'normal' long-term maintenance:

- Implementation validation using mixed-mode emulation.
- Add more features toward a full standard support.

Important features lacking

The main features remaining to be added are:

1. Full Lollipop counter support.
2. Architectural support for multi-DAG (not useful for simulations, but used in real systems).
3. Smarter check against long RPL messages (they can easily exceed the MTU).

The 'nice to have' features to be included are:

1. Upward route preference support (colored links).
2. Multi-root support (virtual roots).
3. External networks gateways (beside the root one).

Thanks for the attention.

Questions ?