Tools for Experimenting Routing in the Data Centers

IETF 107

Mariano Scazzariello
Roma Tre University
mariano.scazzariello@uniroma3.it

Tommaso Caiazzi
Roma Tre University
tom.caiazzi@stud.uniroma3.it

Kathará

Kathará

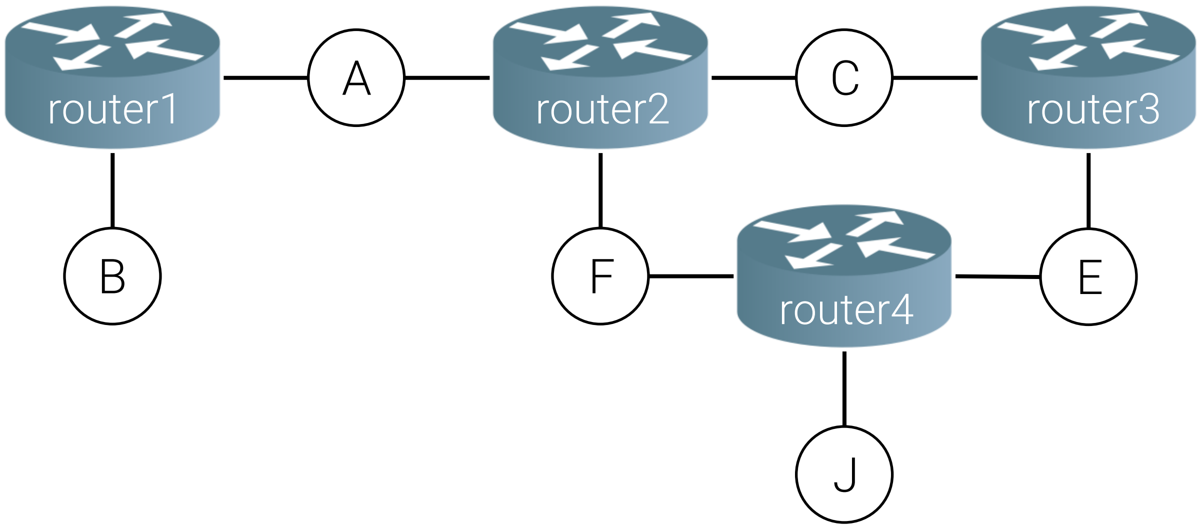
- Website: https://www.kathara.org/
- Kathará is a network emulation system
- Puts at user’s disposal a virtual environment that can be exploited for tests, experiments, and measures on networks
- Stable software (with more than 10.000 downloads)

Kathará – Base Concepts

- Device: a virtual entity which acts like a real network device. It is mainly composed of one or more virtual network interfaces, a virtual CPU with RAM, and a virtual disk.
- Collision Domain: a virtual Layer 2 LAN that interconnects devices. It acts like a physical link between device's interfaces, forwarding all packets received from an interface to all the other interfaces on that collision domain, without modifying them.
- Network Scenario: a set of devices connected by means of collision domains. It provides a simple representation of complex networks consisting of several devices and collision domains.

Kathará – Network scenario

- Simple configuration language to describe a network scenario
- A network scenario is represented as a directory
  - Containing a file with the network topology (**lab.conf**)
  - For each device, files and folders containing the real configuration of that device



Network Scenario Directory Structure

```
|-- router1
|   \-- etc
|       \-- quagga
|           |-- bgpd.conf
|           |-- daemons
|-- router1.startup
|-- router2
|   \-- etc
|       \-- quagga
|           |-- bgpd.conf
|           |-- daemons
|-- router2.startup
|-- router3
|   \-- etc
|       \-- quagga
|           |-- bgpd.conf
|           |-- daemons
|-- router3.startup
|-- router4
|   \-- etc
|       \-- quagga
|           |-- bgpd.conf
|           |-- daemons
|-- router4.startup
|-- lab.conf
```

lab.conf content

```
router1[0]=A
router1[1]=B

router2[0]=A
router2[1]=C
router2[2]=F

router3[0]=C
router3[1]=E

router4[0]=E
router4[1]=F
router4[2]=J
```

Kathará – Command Line Interface

- Kathará exposes a set of commands accessible through a shell which allows to interact with it
- Three categories:
  - *v-prefixed* commands: commands used to configure, start up, and shut down a single network device
  - *l*-prefixed commands: commands used to configure, start up and shut down a whole network scenario
  - General commands: general purpose commands for system settings, for getting running devices information, and for performing global actions on devices and scenarios
- A REST API Server will be implemented in the future…

Kathará – Virtualization Technologies

- Can support different virtualization technologies
- Currently it is mainly based on Docker
- A set of images for devices is available on Docker Hub
  - Based on Debian
  - Built-in support for Quagga, FRRouting, OpenVSwitch, and P4…
  - … but you can use any Docker image (rift-python image ☺)
- Two different type of deployments:
  - Single node, called Kathará using Docker
  - Distributed, called Megalos using Kubernetes (to appear NOMS 2020) (supports ~5M devices in a network scenario!)

Fat Tree Generator

Kathará/Megalos Tool

Fat Tree Generator

- Tool to generate Kathará/Megalos (in what follows just Kathará) network scenarios for Fat Tree Topologies
- Uses fundamental fat tree topology parameters (K_{leaf}, K_{top}, redundancy factor)
- Automatic routing-protocol configuration on nodes
- Currently supported routing protocols:
  - BGP (FRRouting)
  - Openfabric (FRRouting)
  - RIFT (rift-python)

Fat Tree Generator – Formulas to build Fat Trees

- Input parameters:
  - K_{leaf}
  - K_{top}
  - R (redundancy factor)
  - Number of servers per rack
  - Desired protocol
- Given K_{leaf}, K_{top}, R:
  - N (Number of ToF planes) = K_{leaf}/R
  - P (Number of PoDs) = $(K_{leaf} + K_{top})/R$

Fat Tree Generator – RIFT

- Ring between ToF planes are automatically added
- Currently, only IPv4 subnets are automatically assigned to nodes
  - Add IPv6 subnet assignment:
    - for servers
    - for the whole fabric
- Generated RIFT configuration is minimal
  - ZTP
  - However, it is possible to manually edit RIFT configuration files as needed!

Kathará Test Framework

Kathará Test Framework

- Platform (written in Python) to automatically test Kathará network scenarios
- Basic idea
  - Provide a default template that is easily extensible and adaptable to desired scenarios
- Basic functioning:
  - Deploy a Kathará network scenario
  - Wait that all the nodes are running
  - Undeploy the scenario

Kathará Test Framework – Code Annotations

- Extending the basic scenario is possible using **code annotations**
  - Allow the definition of new methods to execute, with a priority, in different phases of the test
- To create a new scenario:
  - extend the basic scenario
  - define specific methods needed by the scenario
  - tag methods with annotations to define when they will be executed
    - (e.g. before/after protocol convergence)

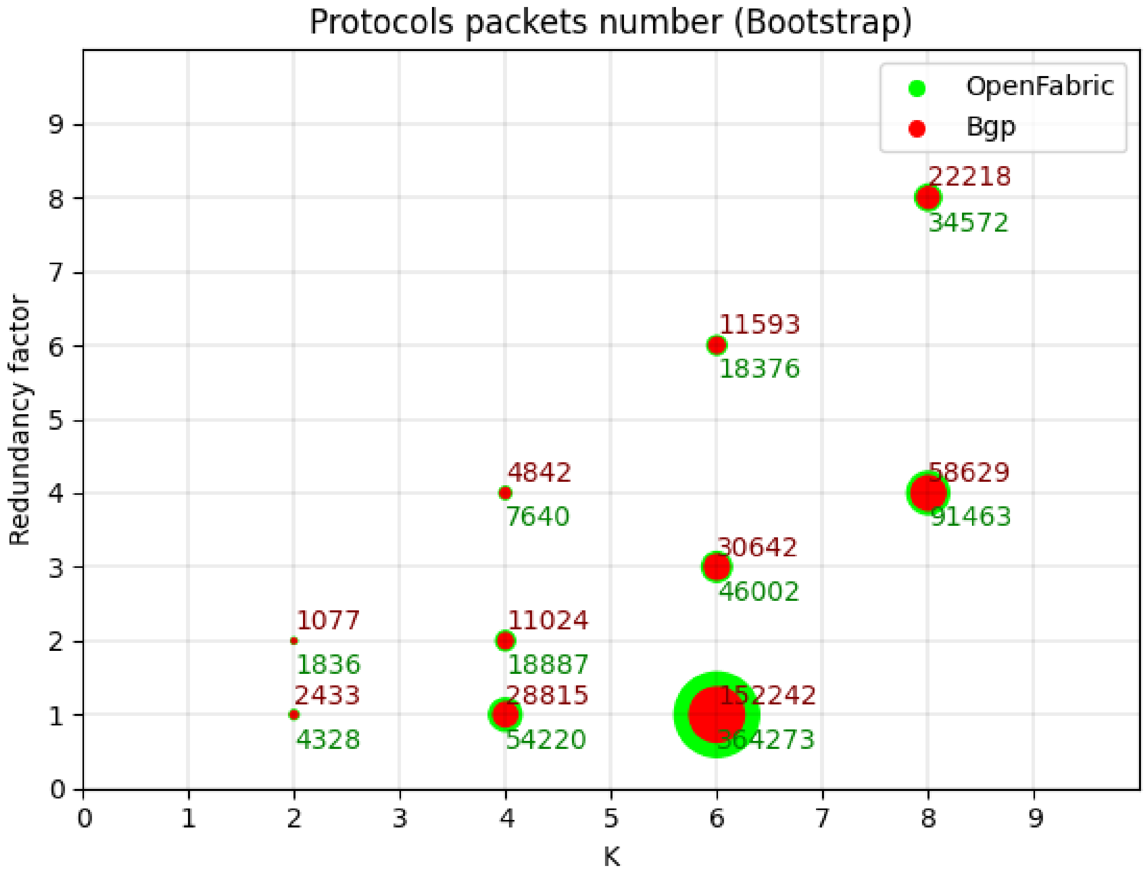
Test Framework – Integration with Fat Tree Generator

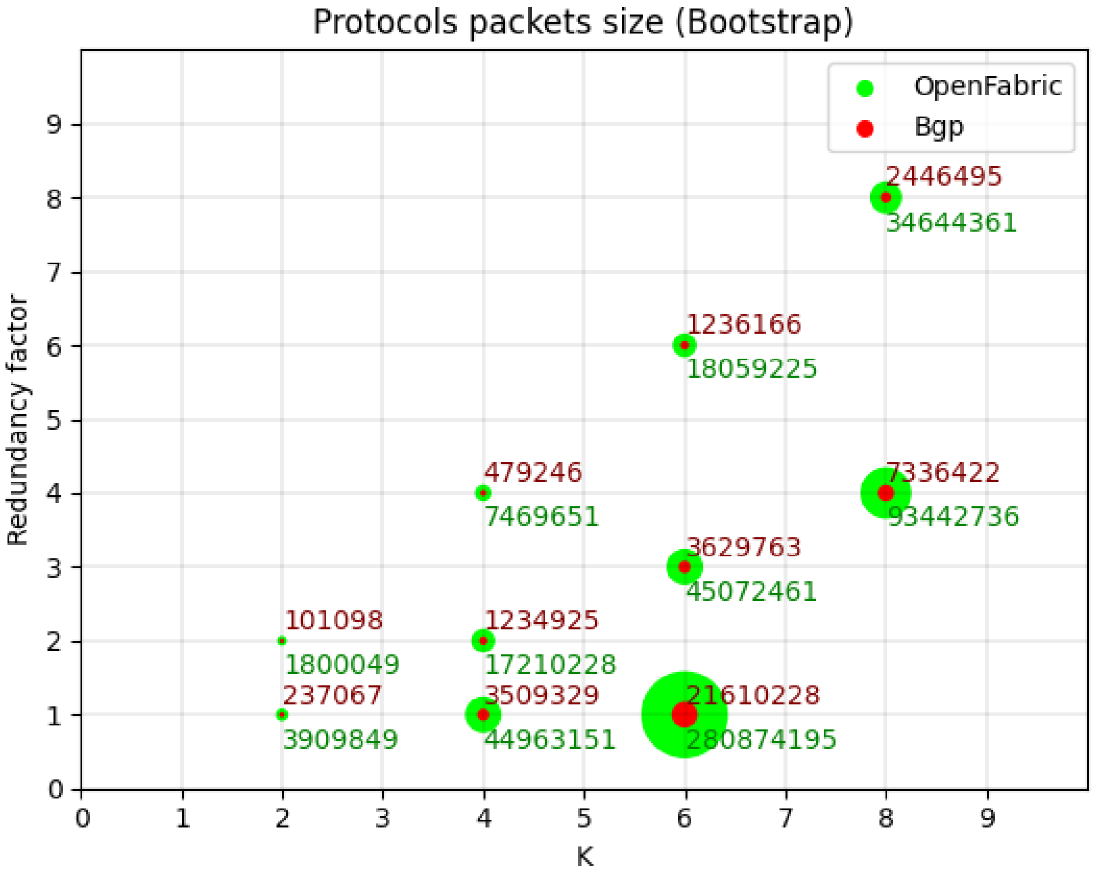
- Using Fat Tree Generator it is possible to automatically deploy and test Fat Tree scenarios
- Basic functioning:
  - Generate a Fat Tree topology
  - Deploy the resulting Kathará network scenario
  - Wait that all the nodes are running
  - Start the routing protocol on each node that needs it
  - Wait for protocol convergence
  - Undeploy the scenario
- It is also possible to dump *.pcap* files for each node interface to analyze traffic and gather statistics

Test Framework – Use Cases

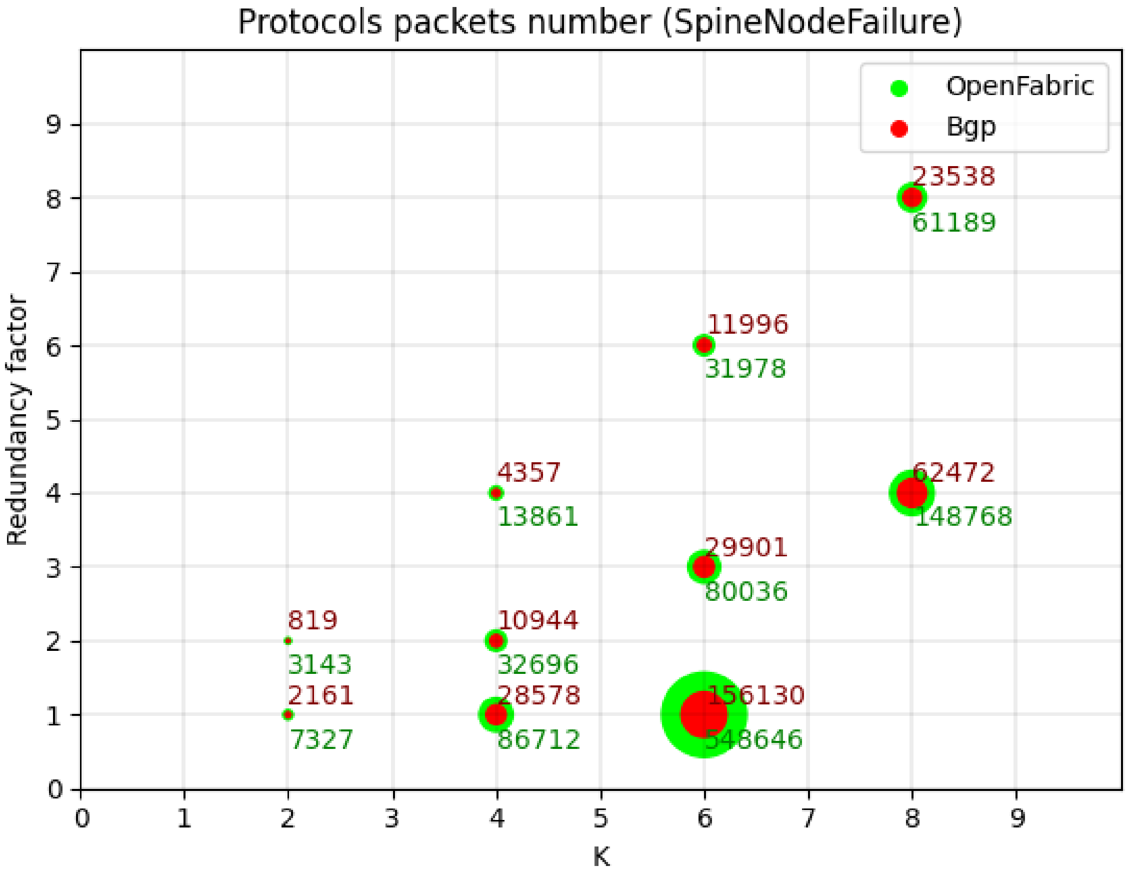
- Protocol Comparison
  - We are actively working together with the Uruguay University to compare the most popular routing protocols for Fat Trees
- RIFT Development
  - We are exploiting the Framework to develop and test negative disaggregation on RIFT

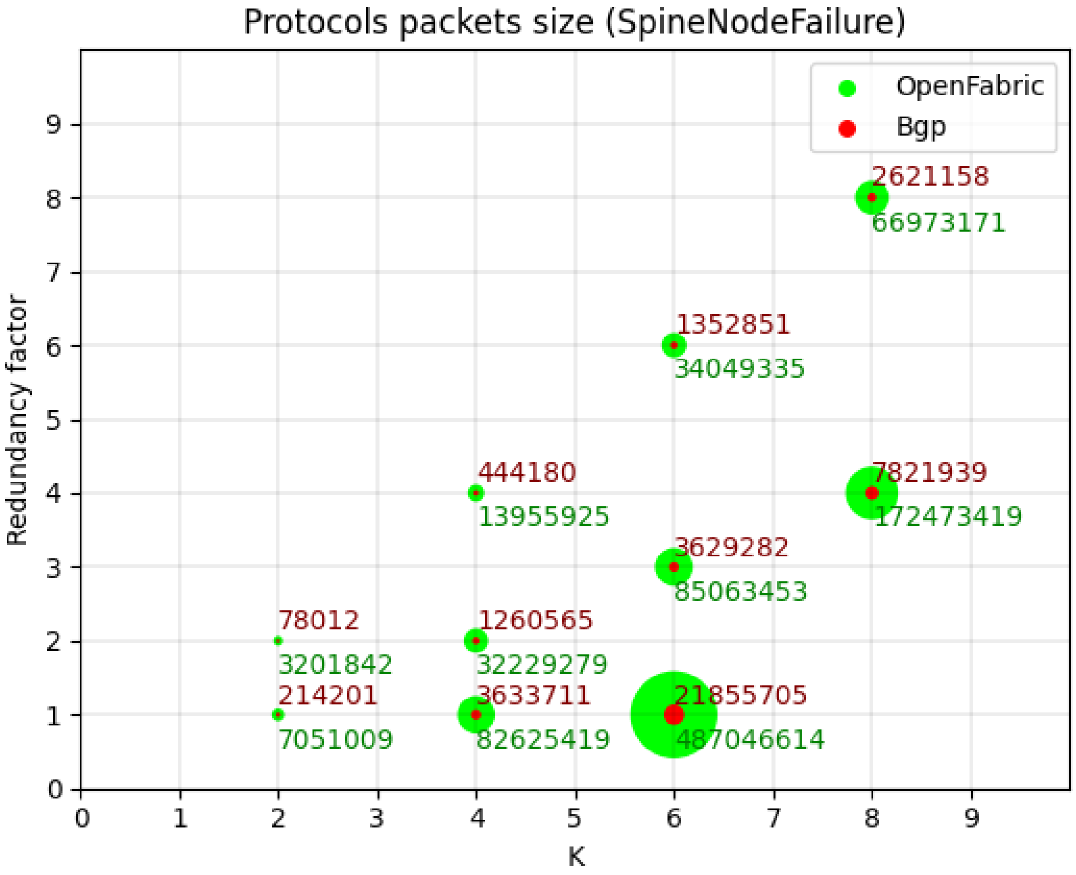
Test Framework – Example of Protocol Comparison





Test Framework – Example of Protocol Comparison



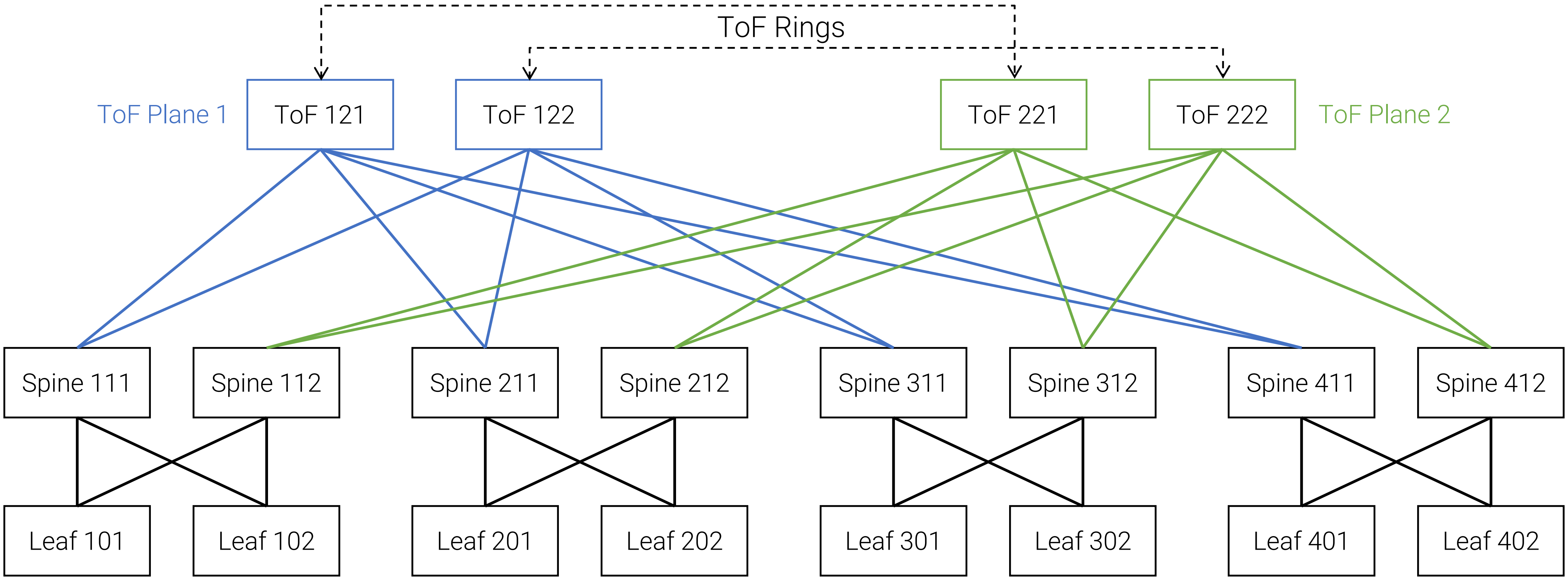


Demo

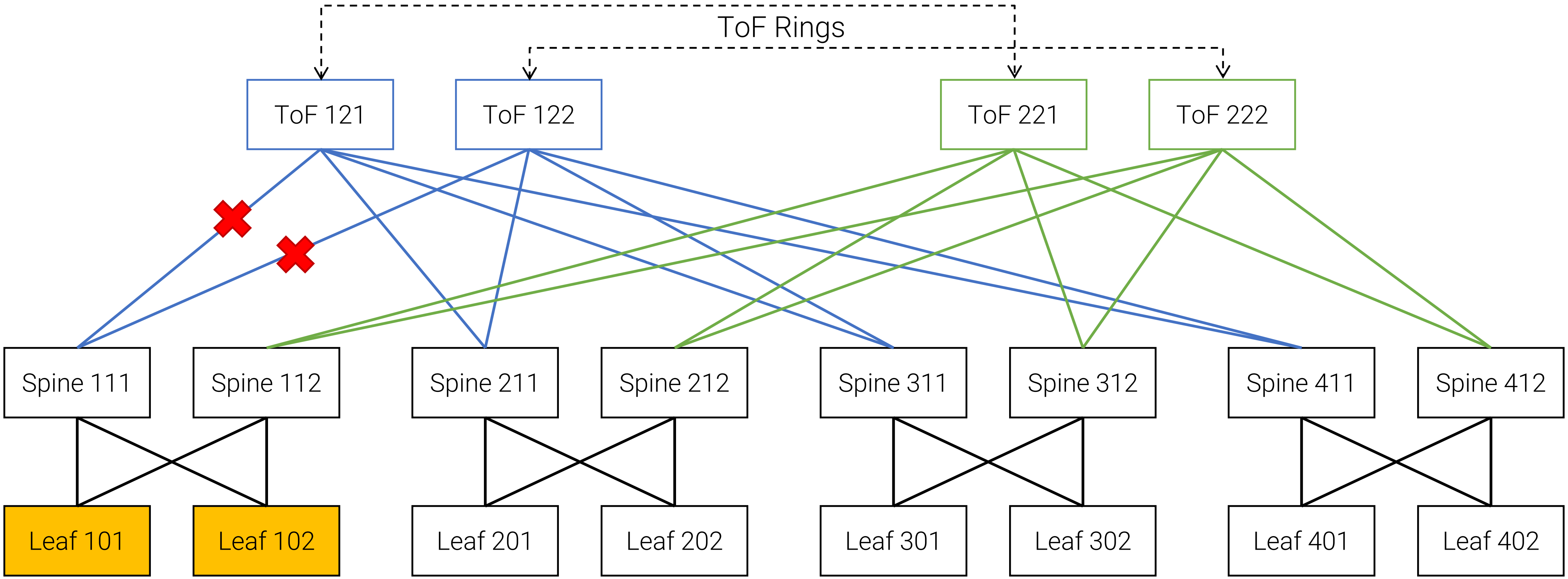
Demo – Summary

- The demo will show the usage of Kathará and Fat Tree Generator to build and deploy a Multi-Plane Fat Tree topology
- Fat Tree Parameters:
  - $K_{leaf} = 2$
  - $K_{top} = 2$
  - $R = 1$
- The topology will be exploited to create a fallen leaf scenario and show our progresses in the negative disaggregation implementation

Demo – Multi-Plane Topology



Demo – Multi-Plane Topology (Fallen Leaf Scenario)



Fallen Leaves for ToF Plane 1

Tools for Experimenting Routing in the Data Centers

IETF 107

Mariano Scazzariello
Roma Tre University
mariano.scazzariello@uniroma3.it

Tommaso Caiazzi
Roma Tre University
tom.caiazzi@stud.uniroma3.it