MPLS-Kit: An MPLS Data Plane Toolkit – Extended details

Juan Vanerio
Faculty of Computer Science
University of Vienna
Austria

Stefan Schmid
TU Berlin, Germany &
University of Vienna, Austria

Morten Konggaard Schou
Dept. of Computer Science
Aalborg University
Denmark

Jiří Srba
Dept. of Computer Science
Aalborg University
Denmark

I. Library Usage

To allow researchers to integrate MPLS-Kit into their projects and automate the generation of different MPLS data planes with different topologies and protocol configurations, as well as to automatically perform simulations, MPLS-Kit provides interface functions, which we describe next. The admitted input arguments are described in Table I.

- `generate_fwd_rules`: Generates the forwarding rules in a network. It acts as a building recipe orchestrating all network components, resulting in an in-memory network object. This function is the only interface provided for computing the data plane.
  It returns a network object that can be further modified, used for simulation purposes or its embedded data plane printed in an external JSON file.
- `generate_topology`: Given n nodes and an edge generation mode, returns a randomly created weighted topology as a networkX [?] graph.
  By default, the topology edges are chosen at random. Alternatively, connectivity can be ensured by generating links by iterating through all nodes and linking them at random with previously visited nodes. Finally, MPLS-Kit also has functions to load existing topologies in GraphML or custom JSON format.
- `Network.build_flow_table()`: Method provided by the Network class to build a table of MPLS flows.
  In a nutshell, it returns a list of flows that can be expected to be seen on real networks.
  To accomplish this goal, the function examines each router's LIB to identify only FECs representing networking resources that on a real network indicates that the router is a valid packet source for said FEC. Flows are identified by their FEC type:
  - LDP FECs: All routers can initiate packets to any link and any router (except itself).
  - RSVP Tunnels: only initialize packets on the initial router of each tunnel. Never initialize a packet to use backup routes as only existing tunnels may use these.
  - VPN Service FEC: For each VPN, packets may only reach other routers that instantiate the same VPN.

  Following these rules on every matching router and FEC, the initial header of the packet is extracted from the LIB and the valid destinations computed from the specific resource represented by the FEC.
  The list of flows can be written to a file.

For simulation purposes the Simulator class offers the following main function:

- `run`: Takes a network object, a list of failure scenarios and list of all flows to simulate as input arguments. If the latter is not provided, it uses the output from `Network.build_flow_table()`. For each flow, it instantiates and forwards a packet, and evaluates if the forwarding was succesful by matching against the set of valid targets. The result, exit code and traceroute of each simulated packet is stored in a table and can be printed to an external file.

A. Command Line Interface Tools

The Python functions described previously are the application programming interfaces designed to be used by external tools to interact with the network models efficiently. To enable easy usage, we also provide the following Command Line Interface (CLI) utility tools:

- `tool_generate.py`: Script to generate the MPLS data plane. Accepts all of `generate_fwd_rules()` inputs as inline arguments or defined on an external YAML configuration file, and prints the resulting data plane to standard output or saves it to a file in AalWiNes' JSON format.
- `tool_simulate.py`: Script to make simulations on the MPLS data plane. Extends the functionality of generator.py by first creating the data plane (optionally also writing it to a file) and then instantiating a Simulator object. This script accepts all the same arguments as generator.py and a YAML file containing a list of failure scenarios to simulate in sequential order. It also accepts a folder to store the simulation result files; otherwise it prints the results to standard output.
- `create_confs.py`: Script to generate configuration files for the generator and failure files for the simulator. Examples are shown in Figure 1 and Figure 2 respectively.

Parameter	Type (default)	Description
		`generate_fwd_rules`
`enable_PHP`	Boolean (True)	Activate Penultimate Hop Popping functionality.
`enable_LDP`	Boolean (False)	Enables Label Distribution Protocol. A label will be allocated for each link and each node (loopback address) in the network in a straightforward fashion (as in a single IGP area/level).
`enable_RSVP`	Boolean (True)	Enables Resource Reservation Protocol (RSVP-TE). RSVP defines tunnels (LSPs) between a headend and a tailend allowing for traffic engineering.
`num_lsps`	Integer (10) or List	Number of TE tunnels to compute between different (headend,tailend) pairs, or explicit list of such pairs.
`tunnels_per_pair`	Integer (3)	Number of tunnels between each (headend,tailend) pair.
`enable_services`	Boolean (True)	Enables MPLS VPN services. These services abstract away the specific type of service (VPWS, VPLS or VPRN), using a single class to represent them all.
`num_services`	Integer (2)	Number of VPN services to instantiate.
`PE_s_per_service`	Integer (3)	Number of provider edge (PE) routers per VPN service.
`CEs_per_PE`	Integer (1)	Number of customer edge (CE) routers attached to each service PE.
		`generate_topology`
n	Integer	Number of nodes.
`mode`	{`log_degree`, `large_degree`}	Method for generating the number of edges: `log_degree`: a random value between boundaries roughly proportional to $n \log_2 n$, and `large_degree` where boundaries are proportional to n^2
`gen_method`		Method for generating the random topology.
`weight_mode`	{`equal`, `random`, `distance`}	Control link weights generation, for shortest-path computations. Option `equal` sets all links with a weight of 1, while `random` sets a random value between 1 and 10. Option `distance` computes the link weights from the distance between link nodes if their geographical coordinates are provided.
		`Simulator.run`
`network`	Network	Pointer to MPLS-Kit's network object containing the data plane required for the simulation.
`restricted_topology`		A subgraph of the topology containing only links that are not failed. Such a subgraph can easily be generated from a list of failed links using networkx's `subgraph_view` function.
`flows`	List	List of flows to simulate. Defaults to get the list from `Network.build_flows_table()`.

TABLE I: The input interface of MPLS-Kit

As an example of CLI execution using the example configuration files, the following command will create a data plane according to instructions of Figure 1 and store it in a JSON file in `data_plane.out`. Immediately after, it will generate simulations for packets for each on each failure scenario included in Figure 2. The results of said simulations are stored in individual files in the folder called results.

```
python tool_simulate.py --conf config.yaml \
--failure_chunk_file fail.yaml \
--output_file data_plane.out \
--result_folder results
```

B. *Jupyter Notebook Usage*

A researcher can also use the library directly in her own Jupyter Notebook, or any other tool allowing for interactive and visual Python code execution. The main benefit from this approach comes from the enhanced control and flexibility provided by the possibility of keeping the network and Simulator objects in memory, available for further exploration, interaction and modification.

In such usage the researcher imports MPLS-Kit (e.g., `from MPLS_Kit import *`), and then calls `generate_fwd_rules` with the desired arguments. The returned network object can be explored in detail by accessing its attributes, or modified making new calls to its methods. If a simulation is desired, the researcher just has to instantiate an appropiate MPLS packet object, or a simulator and provide a list of flows and a failure scenario. Notice that no external files are required when using this approach.

```
random_topology: True
random_mode: "random.log_degree"
num_routers: 20
random_weight_mode: "random"
random_gen_method: 1
php: True
ldp: False
rsvp: True
rsvp_num_lsps: 25
rsvp_tunnels_per_pair: 3
vpn: True
vpn_num_services: 7
vpn_pes_per_services: 4
vpn_ces_per_pe: 3
random_seed_gen: 321354
```

Fig. 1: Example YAML config file “config.yaml”.

```
-["R1","R4"]
-["R1","R2"],["R1","R4"],["R7","R8"]
-["R1","R2"],["R5","R6"]
```

Fig. 2: Example YAML failure scenarios file “fail.yaml”.