

# BGP pySim documentation

Lorenzo Ghio

## Abstract

A python simulator has been developed to replicate the exponential path exploration problem described in [1]. The simulator workflow and kind of events, together with the BGP node logic implemented by the pySim, are described in this document.

## 1. Simulator high-level architecture

The simulator requires:

1. The network topology, described by a graphml file (–graphml)
2. The output folder

### Initialization

The graphml is parsed to:

1. initialize node objects with their TYPE and prefixes to be exported
2. setup neighbourhood relationships. This includes peering or customer/provider role assignment and per-neigh MRAI assignment

At node initialization, prefixes exported by nodes are put in those nodes' receiving buffer. Then, for all nodes, a 'CHECK-RX' event is triggered so that nodes install those prefixes in their RT and can start advertising them.

### pySim main loop and events

Events are described by a tuple of the form: (*actor*, *action*, *params*). The actor is a nodeID indicating which node should perform an action, with all info necessary to perform the action contained in *params*. Actions can be of 2 kinds:

1. 'CHECK-RX': the actor controls whether new updates are in its rx-buffer, and process them.
2. 'MRAI-DEADLINE': the actor MRAI deadline for advertising a given prefix expired, so the node send an update immediately

The logic of processing and sending updates is described in Section 2 about Node logic implementation.

## 2. Node implementation

### Node attributes

A node has/is described by, and keeps updated the following:

1. nodeID and nodeType
2. **rxQueue**: the updates receiveing buffer

3. **neighs**: a dictionry with neighID as keys and 'realtion' and MRAI as neighbour attributes
4. **exportPrefixes**: a list of prefixes exported by this node
5. **RoutingTable**: an object with convenient methods to install routes and to remeber received updates, so to be ready to install backup routes

### Routing table

A routing table is a dictionary indexed by known prefixes. For each prefix these info are kept updated:

1. NH and AS-PATH
2. PREFERENCE, computed according to the **policy function**<sup>1</sup>
3. MRAIs: a dictionary indexed by neighbours' ids. For each neigh the time after which is possible to send an update is maintained.
4. SHARED-FLAG: again a per-neigh indexed dictionary. A flag per neighbour is maintained to remember if an update has been or not already sent to this neigh for this prefix. Thanks to these flags and assuming no losses in sending updates over TCP connections, we will see the network "silent" at convergence.

### SENDING updates

After receiving an update, a node may decide to send and update for these reasons:

1. the route is new
2. some route's attributes changed

If the MRAI for this prefix with a given neighbour is expired, the update can be really sent, appending the sender-id to the route's AS-PATH and pushing the update in the neigh's rxQueue.

After sending an update for a given prefix to a given neigh, the SHARED-FLAG in the RT[prefix][SHARED-FLAGS][neigh] must be set to TRUE and **the MRAI must be updated!**

### PROCESSING received updates

Periodically (each second +- jitter), every node flushes its receiving buffer processing all found updates. The update processing workflow is:

<sup>1</sup>The policy function comes as a separate py file, to ease extension and multiple versions implementation in the future

1. Compute PREFERENCE applying the policy function to the route's attributes
2. Proceed with a standard Bellman-Ford, installing the route if it is new or with a strictly higher preference.
3. If preference is decreasing or a withdraw is received, a node must consider to install a backup route!

Come triggerare l'installazione delle backup routes?  
Mattia questo e' il punto su cui dovremo ragionare  
tanto domani :)

## References

- [1] A. Fabrikant, U. Syed, and J. Rexford, "There's something about mrai: Timing diversity can exponentially worsen bgp convergence," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 2975–2983.