Improving BGP Convergence with Fed4FIRE+ Experiments

Mattia Milani*, Marco Nesler*, Michele Segata*, Luca Baldesi*, Leonardo Maccari†, Renato Lo Cigno‡
*Dept. of Information Engineering and Computer Science, University of Trento, Italy
†Ca' Foscari University of Venice, Italy
‡Dept. of Information Engineering, University of Brescia, Italy
mattia.milani@studenti.unitn.it, {marco.nesler,michele.segata,luca.baldesi}@unitn.it,
leonardo.maccari@unive.it, renato.locigno@unibs.it

***Abstract*—The Border Gateway Protocol (BGP) is the single routing protocol that glues the Internet together. Its performance, especially the convergence speed after path changes, is key to global efficiency, also in light of the fact that the number of Autonomous Systems (ASes) and Subnets has reached a level that makes path changes a frequent event. This work presents a testbed-based experimental analysis of BGP convergence time under different hypothesis of Minimum Route Advertisement Interval (MRAI) setting and a proposal to improve it by setting MRAI based on the topological position of the ASes. MRAI is a timer that regulates the frequency of successive `UPDATE` messages sent by a BGPs router for a given route and destination. The work is based on the modifications of the BIRD BGP daemon and shows that it is possible to execute experiments on testbeds with topologies that have Internet-like characteristics scaling up to thousands of ASes.**

I. Introduction

The protocol architecture of the Internet follows an "hourglass" design with IP (v4 or v6) as unifying layer, the slim waist of the hourglass, managing global addresses and routing. Maybe less known is that the IP layer has an even "slimmer" waist when it comes to global routing with Border Gateway Protocol (BGP) as the single Exterior Routing Protocol (ERP) that interconnects Autonomous Systems (ASes), both for IPv4 and IPv6.

BGP is a very complex protocol defined in tens of different RFCs, and its description and details are beyond the scope of this paper; however it is not difficult to imagine that, as the single protocol that keeps the global internet working, experimenting with it and proposing innovations and improvements is very difficult, if feasible at all. On-line experiments are out of question as a failure would mean interrupting Internet connectivity for millions of users, and simulations are too approximate for global operators to trust them to modify their routing policies. Exemplary in this context is the seminal study of Fabrikant, Rexford et al. [1] that shows that modifying one parameter of BGP, namely the Minimum Route Advertisement Interval (MRAI), to try to improve performance without appropriate coordination can lead in specific cases to the exponential growth of the number of `UPDATE` messages required to achieve a new stability point after a route change. The result in [1] is purely theoretical (no experiments or simulations are presented to verify the expected results), but correctly a negative theoretical results is enough to stop pursuing an innovative direction. Unfortunately BGP is way too complex to attempt an overall theoretical modeling that not only shows that in some specific cases the network may become unstable (exponential growth of signaling messages), but that there might be other sound strategies that lead to improve performance without hampering stability.

This research is supported by the European Commission, H2020 Programme, Grant Number 732638 'Fed4FIRE+' through the Open Call 5 Experiment IoF (Internet on FIRE).

Our work has the ambition to show that it is possible to experiment with a real implementation of BGP, namely BGP Internet Routing Daemon (BIRD),[1] reproducing with a real implementation and measures the results in [1], and then building topologies that correctly mimics Internet characteristics with thousands of nodes. We also show that there are strategies that allow to improve the convergence of BGP without hampering its stability.

II. Background and Motivation

Global routing and BGP are, strangely enough, somewhat niche topics in research, often deemed more technical issues to be solved by providers, than scientific arguments needing sound analysis and solutions. Clearly there exist several papers dealing with BGP and its characteristics and features (see [2] and citations therein or [3] to name a few), but the core itself of BGP, i.e., the Path Vector descriptors exchanged by Exterior BGP (eBGP) routers, and the policy-based routing algorithms, do not lend themselves to rigorous theoretical analysis and modeling, thus preventing elegant and sound theoretical results similar to those available for link state and distance vector routing protocols.

BGP research has traditionally been based on simulations [4]–[6] and/or small-scale testbeds [7], [8], with a few works like the already cited paper by Fabrikant et al. [1], addressing specific issues with a theoretical analysis based on heuristics considerations on BGP dynamics. Other significant works on the subject are based on measures, like the seminal analysis in [9], that clearly identifies the key role of BGP in properties and

[1]BIRD is one of the most used and well maintained BGP open source implementations; see https://bird.network.cz/

performance of the global Internet, or [10], [11] analyzing prefix scaling and router ownership boundaries, but they normally document the behavior or the properties and consequences of BGP rather than exploring possible improvements; the PEERING testbed [12] offers an infrastructure to experiment with BGP, but does not seem prone to explore performance or major protocol modifications.

One of the prominent themes in this body of literature is the trade-off between the generated overhead and the convergence speed of BGP after a reconfiguration event. BGP is known to be subject to *path exploration*, a transitory phenomenon that happens when a router adopts and publishes a sequence of non-optimal paths for a destination before reaching a stable state. Path exploration can generate thousands of update messages in networks made of as few as tens of nodes [13]. In order to reduce the message overhead, BGP uses MRAI, the minimum time between two consecutive `UPDATE` for the same destination sent to the same neighbor, which is set by default to 30 s [14]. MRAI reduces the overhead but strongly impacts BGP convergence [15] and its default value was often discussed [16]. Unfortunately, changing BGP is a tricky task: any viable solution, as highlighted in [2], should encompass at least the following characteristics:

- Incremental deployment: it cannot require to “reboot" the Internet;
- Computational overhead: it must be usable on current hardware;
- Changing BGP behavior: it should not change the macro behavior of BGP;
- Constraints to BGP expressiveness: it cannot limit the current flexibility of BGP; and,
- Coordination between ASes: it cannot require explicit cooperation among ASes.

On top of this, Fabrikant et al. introduced a key observation: an uncoordinated change in MRAI could produce in certain realistic topologies an exponential growth in the generated overhead. This theoretical result produced a research deadlock, since a coordinate change is not possible and an uncoordinated change can lead to instability, MRAI is still set to 30 s.[2]

The goal of this paper is to open the way to automate MRAI configuration dynamically in BGP with a solution respecting all the five points above, which requires three key steps.

a) Provide a scalable and repeatable BGP experimentation framework: This paper provides an experimental approach that enables the emulation of networks made of thousands of routers, in order to compare different approaches on real code. For this purpose we used the open source BIRD project as already mentioned, one of the most popular BGP implementations worldwide and we reproduce two kinds of network topologies, the ones analyzed by Fabrikant, and Internet-Like topologies. We modified BIRD to implement MRAI as described by RFC 4271 [14], as MRAI is not currently supported. Our experience confirms that we can emulate networks made of thousands of nodes, and we are confident we can reach tens of thousands following sound experimental research principles. All the decribed tools (topology generators, deployment scripts, logging parsers, ...) were developed during the Internet on FIRE (IoF) project in the context of the Fed4FIRE+ experimentation facilities, and are documented and published on-line.[3] All experiments have been replicated several times during the project to ensure that results are consistent and due to the phenomena described and explained, and not to random variations of the environment;

[2]As a relevant example, Cisco suggests 30 s MRAI for any regular eBGP peer [17].

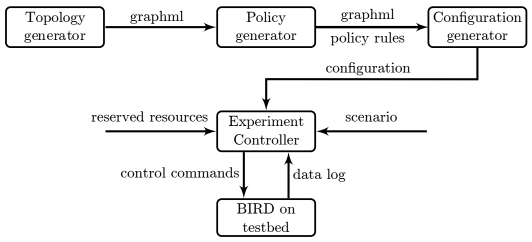


Figure 1: Sketch of the testing system.

b) Confirm Fabrikant Results: We use our emulation framework to confirm Fabrikant’s results, which is is key to understand the risks associated to dynamically setting MRAI. We reproduce and confirm Fabrikant observation in Section V;

c) Towards a dynamic MRAI strategy: We provide the initial design of a strategy to dynamically set MRAI. We introduce a proposal to set MRAI timers based on Destination Partial Centrality (DPC) a destination-based centrality metric which can be computed by every router without coordination with other ASes. Our initial results are based on the assumption of knowledge of the BGP topology, but confirm experimentally that we can improve the performance of the standard BGP configuration while avoiding the pitfall described in [1].

III. BGP Experimental Framework on Fed4FIRE+

Emulating BGP involves a number of complex steps that are needed to recreate realistic conditions. First, we need to generate a topology of BGP routers, which strongly influences the final outcome. Then we need to set BGP policies on the routers and the MRAI timers for each node or edge (which can depend on the topology), finally we need to translate this model information into configuration files for BIRD. Only then we can run a number of repetitions of the experiment and collect results. Figure 1 shows a sketch of the steps involved.

A. BGP topology

Let us review the model we use to represent a BGP network providing the minimum level of detail needed to understand our testing framework and results.

We model a BGP network as an undirected graph $G(V, E)$ (no self-loops, no multi-edges) in which nodes in V are eBGP speaker routers representing an AS, and en edge in E represents a BGP advertisement (ADV) communication link (hence undirected). We extended the model provided by

[3]See https://iof.disi.unitn.it, and the related github repository https://github.com/iof

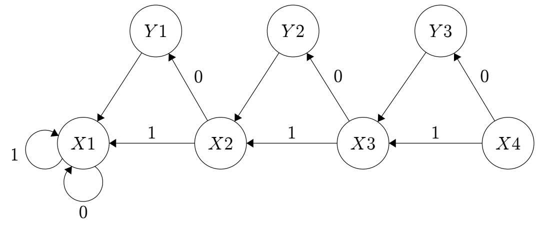
Elmokashfi et al. [18] with nodes and edges attributes necessary for our analysis. For brevity, we do not re-define all the terms which can be found in the original paper. With respect to nodes the attributes are:

- type {T, M, C, CP}: defines the type of AS, i.e., tier-1, mid-level, customer, or content provider, respectively;
- destinations: a string of comma separated IPv4 network identifiers with the respective netmask, indicating the list of networks the AS exposes.

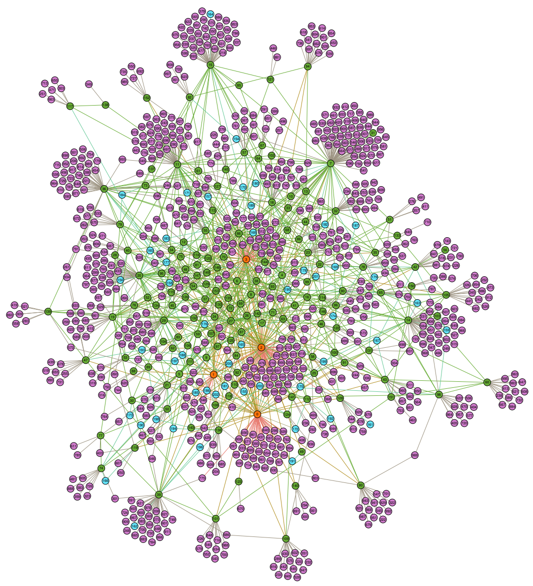
Given $i, j \in V$, the attributes for an edge $(i, j) \in E$ are:

- type {transit, peer}: indicates whether there is a customer-provider (i.e., i pays j for transit or vice versa) or a peering relationship (i.e., i and j agreed to exchange traffic under certain conditions for free);
- customer $z \in V$: if the edge (i, j) is of type transit, z identifies which node between i and j is the customer, else, this attribute is set to "none";
- mrai1 (seconds): indicates the per-link (i, j) MRAI timer set on node i (initially left blank);
- mrai2 (seconds): indicates the per-link (i, j) MRAI timer set on node j (initially left blank).

The topology information is stored in a GraphML file, a standard format to represent graphs. For the time being we realized two graph generators, the first reproduces the so-called "chain gadget" described in the work of Fabrikant et al. [1] that we use to verify their findings. Gadgets are chains of rings of arbitrary length, depicted in Fig. 2a in which each link is of type transit, to generate a pathological case of exponential `UPDATE` messages proliferation. The second generator produces Internet-like graphs of arbitrary dimension using the algorithm proposed by Elmokashfi et al. [18] that preserves the structure of the Internet from a stochastic perspective (i.e., the distribution of tier-1, mid-level, customer, and content provider ASes as well as their relationships). Figure 2b reports an example network.



(a) Gadget topology derived from [1] with 3 rings



(b) Elmokashfi (Internet-like) topology with 1000 nodes

Figure 2: The two types of BGP topologies the tool currently supports. (a) Small hand-build (gadget) topologies to show particular phenomena; (b) Internet-like topologies following [18] model where tier-1, mid-level, customer, and content-provider nodes are represented in orange, cyan, purple, and green, respectively.

B. Policy and MRAI Generator

BGP decides the next hop for a certain destination based on the knowledge of the path to the destination and on the policies that are configured in the router. A policy is a set of preferences on which neighbor to prefer and on which path to prefer when multiple ones are available. We implement the widely-used "valley-free" policy. Routers announce to all neighbors routes learned from customers, while they announce only to customers routes learned from peers or providers. An AS chooses the path following the default breaking ties rules of BGP [14] (preference policies, shortest AS path, lowest origin AS number, etc.). In addition to standard policies we can generate specific BGP policies such as the rules that are required in order to reproduce the exponential convergence theorized by Fabrikant in chain gadgets. Policies are produced as configuration files for BIRD. In this phase we also define the MRAI timers depending on the strategy to be evaluated (for example, a constant value for all nodes, a value drawn from a uniform distribution, etc.).

The output of this phase is again a GraphML file with the same format as the one used to describe the AS topology, only with additional attributes, plus a file expressing policies.

C. Configuration Generator

The next step is the generation of the configurations for the different ASes meaning, in fact, generating the configuration files used by BIRD. Each AS is managed by a BIRD instance, so we generate the configuration files required by each daemon starting from the GraphML and the policies. The tool also generates the scripts to configure the interfaces and the IP addresses to enable the communication between the different ASes. The configuration setup used during the experiment is shown in Fig. 3. In IoF we test potentially very large topologies (we successfully ran experiments up to 8000 nodes, results not presented here), so having one BIRD instance per physical testbed node might not be feasible. We thus run several BIRD instances for CPU inside a Linux Namespace, so that each instance is isolated from the others. To enable the communication between different instances (even on different testbed nodes), we have one virtual interface per namespace which is bridged with all the others and with the physical interface of the node. As a result, each virtual interface is

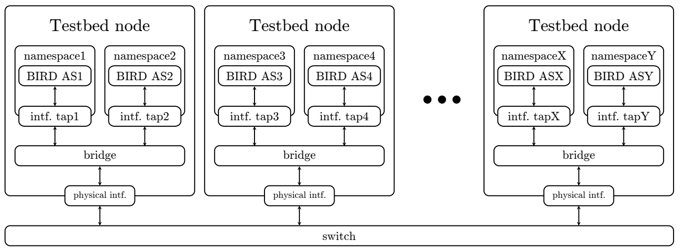


Figure 3: Configuration of the BGP network on the testbed.

in fact connected to the same physical network through the switch, so by simply configuring a /30 IP subnet per each AS pair, the ASes can establish BGP peering sessions through TCP. The example in Fig. 3 shows two BGP instances per physical node, but in fact this number is dynamic and depends on the hardware characteristics of the node.

D. Running the Experiments

What we described so far is a generic procedure that can be used to prepare experiments in any testbed. The Fed4FIRE+ project provides several testbeds available to researchers for networking experiments. With specific reference to the Virtuall Wall 1 and 2,[4] which we used in IoF the final step includes reserving the resources, launching the experiment, and collecting the results. The tools developed in IoF can query for available resources on multiple testbeds (e.g., Virtual Wall 1 and 2) and automatically generate an rspec file (a configuration file for experiments) that can be opened in jFed to reserve the nodes. Once this step is complete, the tool obtains the number of CPUs for each node and decides how many BGP instances will be deployed on that node. The tool then installs the required software and deploys the configuration files generated in the previous step. Once this is completed, the experiment is ready to be started. The user can connect to one node of the testbed which will act as an experiment controller, and from there launch the experiment. The experiment controller takes in input the configuration files for BIRD, the reserved resources (i.e., the hardware description of each testbed node), and the scenario. The scenario indicates which AS (or ASes) exposes an IP destination and a triggering router T_R which will increase the weight of one of its links to start a network reconfiguration and measure the time required to reach convergence. The link to use is also a user parameter. Once the experiment is completed, it is possible to fetch the log files and process the results using the provided scripts.

IV. Comparing MRAI Strategies

Fabrikant et al. showed that a combination of a particular topology and MRAI setting can make the number of `UPDATE` messages explode. The key for this to happen is that MRAI decreases as the distance from T_R increases.[5] In the case MRAI is halved at every hop the number of `UPDATE` messages increases exponentially with the network size, but this effect happens with lower probability also with any decreasing sequence.

A simple solution to this problem would be to always use an increasing MRAI compared to the previous hop, which would have two drawbacks, the first is that routers would need to implement some form of dynamic coordination per each propagation path, the second is that increasing MRAI indefinitely will slow down convergence. The intuition at the base of centrality-based MRAI tuning is that we would like MRAI to increase in the initial phase, close to T_R, and then, when the core of routers around T_R stabilized, it should start decreasing in order to quickly propagate the new stable situation to the rest of the Internet. To verify the validity of this intuition we set-up a strategy that exploits the previous knowledge of the network graph together with the concept of Destination Partial Centrality (DPC).

DPC is a variant of so-called load centrality which is defined in its general form as follows [19]: Consider a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and an algorithm to identify the (potentially multiple) minimum weight path(s) between any pair of vertices s, d. Let $\theta_{s,d}$ be a quantity of a generic commodity that is sent from vertex s to vertex d. We assume the commodity is always passed to the next hop following the minimum weight paths. In case of multiple next hops, the commodity is divided equally among them. We call $\theta_{s,d}(v)$ the amount of commodity forwarded by vertex v. The *load centrality* of v is then given by:

$$LC(v) = \sum_{s,d \in \mathcal{V}} \theta_{s,d}(v) \tag{1}$$

DPC adapts load centrality to represent the propagation of routes in an IP network. In DPC *load* represents the number of networks that a BGP node exports. Not all nodes that run BGP generate *load*, but all nodes that forward traffic have a non-zero DPC centrality. We call $\mathcal{C} \subseteq \mathcal{V}$ the set of nodes that can be source and/or destination of traffic (they export at least one network) and N_s, N_d the number of networks that are exported by node s and d, respectively, then $\theta_{s,d} = \frac{N_s + N_d}{2}$. DPC $\Delta(v)$ of any vertex $v \in \mathcal{V}$ is defined as

$$\Delta(v) = \sum_{s,d \in \mathcal{C}} \theta_{s,d}(v) \tag{2}$$

With DPC we model the fact that some Internet routers export network addresses, and for this reason they generate changes in the network state, while other routers only forward traffic, but still their centrality can be larger than zero. In all our experiments we assign one destination per node, so that $\theta_{s,d}$ is always unitary but this is an arbitrary choice that can be replaced with any other suitable one. In a previous work we have shown that load centrality can be computed in a distributed way with minimal modifications to a Distance-Vector routing protocol. We also experimentally verified that computing DPC is possible with a custom BGP extension, and thus, it can be incrementally deployed on the Internet without requiring any global coordination.[6] Further theoretical details are outside the

[4]See https://doc.ilabt.imec.be/ilabt/virtualwall/hardware.html
[5]Note that this is not strictly true for all the topologies, but due to space constraints we use this simplification.

[6]A brief explanation on how to calculate in a distributed way the centrality can be found at: https://iof.disi.unitn.it/docs/DPCOnTopOfBGP.pdf

scope of this paper, but principles of centrality-based routing can be found in [20], [21].

Our proposal configures MRAI as a function of DPC with the following model: We assume the information contained in the `UPDATE` message propagates in the network in three phases, which identify three propagation graphs:

- ***Ascending phase graph*** $\mathcal{G}_{\mathcal{A}}(\mathcal{V}^{\mathcal{G}_{\mathcal{A}}}, \mathcal{E}^{\mathcal{G}_{\mathcal{A}}})$*:* made by the nodes updated without reaching tier one nodes;
- ***Tier one graph*** $\mathcal{G}_{\mathcal{T}}(\mathcal{V}^{\mathcal{G}_{\mathcal{T}}}, \mathcal{E}^{\mathcal{G}_{\mathcal{T}}})$*:* made by tier-1 nodes;
- ***Descending graph phase*** $\mathcal{G}_{\mathcal{D}}(\mathcal{V}^{\mathcal{G}_{\mathcal{D}}}, \mathcal{E}^{\mathcal{G}_{\mathcal{D}}}) = \mathcal{G}(\mathcal{V}, \mathcal{E}) - \mathcal{G}_{\mathcal{A}}(\mathcal{V}^{\mathcal{G}_{\mathcal{A}}}, \mathcal{E}^{\mathcal{G}_{\mathcal{A}}}) - \mathcal{G}_{\mathcal{T}}(\mathcal{V}^{\mathcal{G}_{\mathcal{T}}}, \mathcal{E}^{\mathcal{G}_{\mathcal{T}}})$*:* the rest of the graph.

Considering a graph-wide maximum timer $T = 30\,\mathrm{s}$ and DPC $c_i \in [0, 1]$ for node i, DPC-based MRAI T_{ij} used by node i with neighbor j is set as follows:

$$T_{ij} = \begin{cases} \frac{T}{2} c_i & \forall i \in \mathcal{V}^{\mathcal{G}_{\mathcal{A}}} \\ \frac{T}{2} & \forall i \in \mathcal{V}^{\mathcal{G}_{\mathcal{T}}} \\ \frac{T(1-c_i)}{2} + \frac{T}{2} & \forall i \in \mathcal{V}^{\mathcal{G}_{\mathcal{D}}} \end{cases} \tag{3}$$

In this work we pre-compute the propagation graphs and the DPC in advance, in order to verify that our intuition is correct. In future works we will relax these requirements and move towards a fully distributed and incrementally deployable solution, based on the theory provided by our previous works [21].

A. Emulation Scenarios

In our emulations we use the two aforementioned topologies of Fig. 2. In the gadget topology T_R is the leftmost node, which changes its preference from the 1-labeled edge to the 0-labeled one. We use chain gadgets of increasing size. We set the MRAI of X_i equal to the MRAI of Y_i and the MRAI of X_{i+1} half of the MRAI of X_i as described in [1]. In the Internet-like topology we choose T_R from the set of the nodes that will induce the worst case situation i.e., changing the edge weight will cause a reconfiguration of the highest possible number of nodes.

In both topologies, in order to trigger the change on the network we use a well known technique called *AS_PATH Prepending*.[7] For the chain gadget we repeat each emulation ten times, while for the Internet-like topologies we do 5 repetition per choice of T_R with 10 different choices of T_R. In all emulations MRAI is subject to a jitter (5% in chain gadgets and 5% in Internet-like topologies) to avoid perfect synchronization.

In all cases DPC is pre-computed assuming one exposed destination network per AS (except for tier-1), even if in the emulation we use only one destination in total, in order to speed-up the initial convergence of the network. We run the network, we let the routing tables of all nodes stabilize, then we trigger the change in T_R. We then wait for all nodes to have a stable routing table. We measure the convergence time (the time between the configuration change and the convergence of the slowest node) and the total number of updates generated. We evaluate these metrics on the following MRAI strategies:

- **30 s *fixed***: the default value according to BGP RFC [14];
- ***No MRAI***: `UPDATE` messages will be generated without delay ($T_{ij} = 0$);
- ***Fabrikant style***: MRAI will be set on each node according to the policies described in Fabrikant work, to reproduce the worst case scenario; and,
- ***DPC-based***.

V. Results on Chain Gadgets

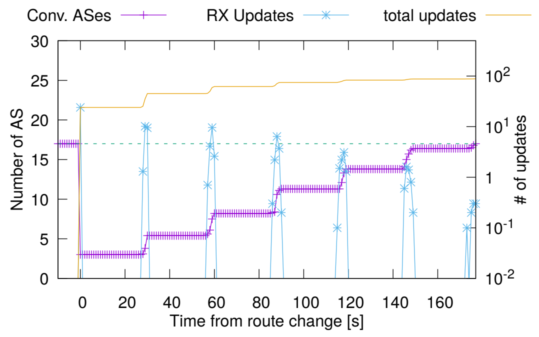
Figure 4 reports the convergence time and number of `UPDATE` messages for chain gadgets made of 17 nodes. Each line reports the average of 10 curves (one per run) measured in a 1 s interval (1 ms for No MRAI) and Table I reports aggregated statistics. At time zero T_R triggers a reconfiguration, and we can clearly see the difference in the behavior of the four strategies. The 30 s MRAI strategy creates bursts of `UPDATE` which make nodes converge gradually, but takes in average more than 150 s with an average of 94 `UPDATE` messages generated. The Fabrikant configuration has a long phase in which all the nodes (excluding one-hop neighbors) don't have a valid path, in average it takes roughly 23 s to converge and requires more than 200 `UPDATE` messages. The DPC-based strategy in average converges in less than 17 s, with 66 `UPDATE` messages, which is a clear improvement over the previous strategies. Finally, the No MRAI strategy behaves as expected, convergence is almost immediate but path exploration generates 130 `UPDATE` messages concentrated in less than 250 ms.

Figure 5a reports the average number of `UPDATE` messages on chains of growing length. We can confirm the abnormal growth of the number of `UPDATE` messages in the Fabrikant configuration compared to any other strategy. The No MRAI strategy has the second highest number of generated messages followed by the 30 s and the DPC-based strategy. Figure 5b reports the convergence time: DPC substantially improves Fabrikant configuration and outperforms the 30 s strategy. No MRAI always converges in less than a second, and thus is not reported.

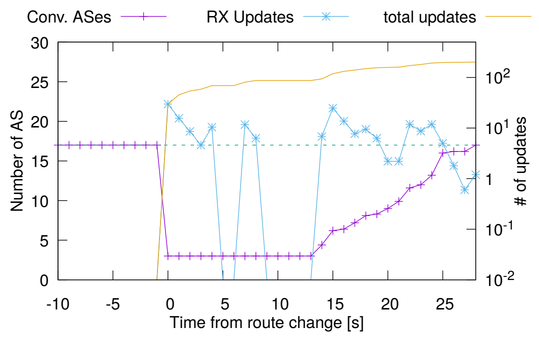
Note that since Fabrikant configuration halves the MRAI at every hop we could not test chains longer than 17 nodes as the MRAI value would be negligible, so it is hard to numerically confirm the exponential growth. Nevertheless we can confirm that:

- Fabrikant configuration is systematically outperformed by any other strategy, thus we verified its abnormal trend;
- The No MRAI strategy produces the fastest convergence but the time-density of `UPDATE` messages is not sustainable since it produces tens of reconfiguration per node in a few hundreds of milliseconds. While this can be handled in a small gadget, the computational overhead needed to update routing tables made of tens of thousands of destinations would not be acceptable;
- The 30 s strategy prevents path exploration but strongly impacts convergence time; and,

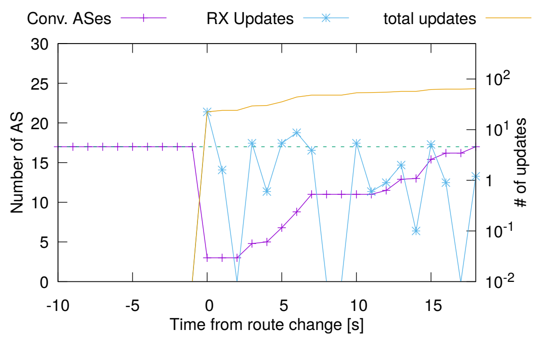
[7] AS_PATH Prepending is a simple way to do traffic engineering on BGP, it consists in the addition of an arbitrary number of entries in the AS_PATH list. Changing the AS_PATH length will force the generation of a new BGP UPDATE on the network.



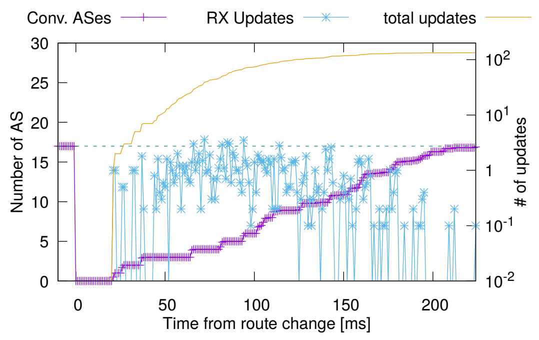
(a) 30 s MRAI strategy



(b) Fabrikant MRAI strategy



(c) DPC-based MRAI strategy



(d) No MRAI strategy

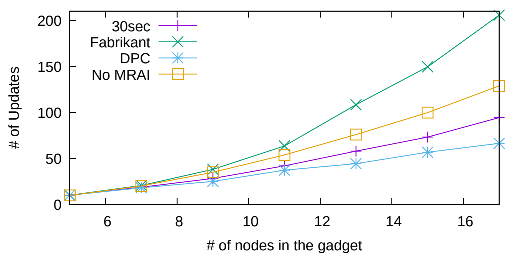
Figure 4: Time evolution of MRAI strategies on 17 nodes chain.

- DPC-based configuration seems to provide the best trade-off between convergence speed and number of UPDATE messages.

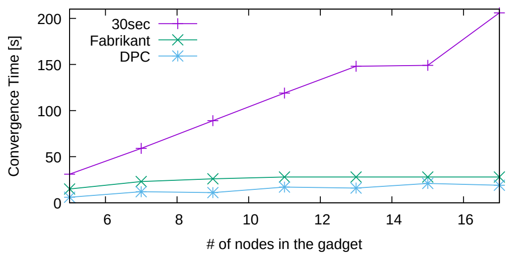
VI. Internet-like Topologies and Scaling

Once verified that in the critical topologies designed by Fabrikant DPC-based MRAI proves to be a viable solution, we report preliminary results obtained on a 4000 node Internet-like topology. Here we compare only the DPC-based and 30 s MRAI strategies since Fabrikant configuration can not be ported to a generic topology, and No MRAI is not a viable solution with thousands of nodes. Table I reports the main statistics for both strategies, and show that DPC produces in average more UPDATE messages but converges in just 38% of time needed by the 30 s strategy.

As further comparison we report the time evolution of all the emulations in Fig. 6 which outlines some key differences. The first is that the number of UPDATE bursts (the blue curve spikes in the figure) is reduced with DPC-based MRAI. This means that the increase in the total number of messages is compensated by a smaller number or "rounds" necessary to make BGP converge. DPC-based centrality does not only reduces the interval between bursts, it eventually makes each round or UPDATE exchanges more effective. The second is that with DPC-based MRAI some rounds of UPDATE exchanges have a greater effect than others on the number of nodes that reach convergence; it's an interesting phenomenon that needs further study, as it suggests that there are some nodes that are more



(a) Number of messages required to reach convergence

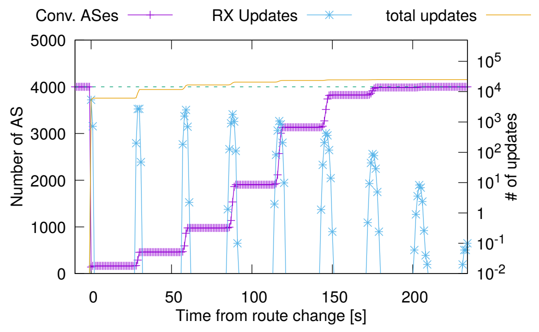


(b) Time in second required to reach convergence

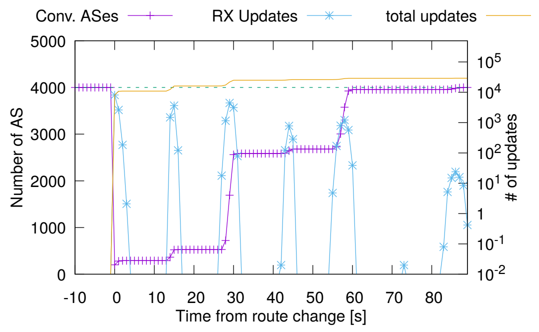
Figure 5: Convergence time and UPDATE messages with chains of growing length.

		Fabrikant			Internet-like		
	Strategy	10th	mean	90th	10th	mean	90th
updates	Fabrikant	157.1	201.3	220.8			
	No MRAI	131.3	133.0	138.1			
	30 s	81.6	88.5	95.2	18790	24806	34298
	DPC	63.0	64.2	66.0	21861	29044	38793
conv. (s)	Fabrikant	18.14	22.98	25.35			
	No MRAI	0.19	0.20	0.21			
	30 s	146.58	156.40	177.22	174.35	184.6	206.37
	DPC	15.60	16.83	18.60	59.09	70.86	88.39

Table I: Statistics on the number of updates and the convergence time for all the experiments.



(a) Evolution in time of the 30 s fixed MRAI strategy



(b) Evolution in time of the DPC MRAI strategy

Figure 6: Evolution in time of the two main MRAI strategies on the same Elmokashfi topology of 4000 nodes, 50 repetitions for each strategy.

important than others, and should converge as early as possible. With centrality-based MRAI tuning we would like to help the convergence of those nodes, and then quickly propagate "good" information to the rest of the network.

VII. Conclusions and Future Work

Conducting Internet-scale research is extremely challenging for many reasons. One of them is the performance evaluation of the proposals, which cannot easily be conducted with (realistic) simulations or simple lab experiments. The use of large-scale testbeds is thus a nearly mandatory tool to make this kind of research credible, but very often testbed results are not reproducible (e.g., they are run on a private testbed), thus hampering the scientific value of the work.

We have explored in this work the use of Fed4FIRE+ to evaluate changes in the management of the MRAI timer of BGP using the BIRD open source implementation. The experimental work has been carefully crafted to make results easily reproducible (see the appendix), offering the community not only a detailed description of all the experimental machinery we have developed, but also the code developed, the scripts to run the experiments and the post-processing tools to obtain results and graphs.

Besides the intrinsic value of an experimental setup that can be used to carry out additional research on BGP, our work also shows that it is indeed possible to improve Internet convergence after a route change by properly managing MRAI, and this without the risk of signaling overhead explosion.

Future work includes pushing even further (to more than 10 000 nodes) the scale of experiments, as well as fully developing the theoretical part behind this experimental work (impossible to do here due to space constraints) to improve the scientific value of our proposal, and make it more appealing for adoption on the Internet.

References

[1] A. Fabrikant, U. Syed, and J. Rexford, "There's something about MRAI: Timing diversity can exponentially worsen BGP convergence," in *30th IEEE Int. Conf. on Computer Communications (INFOCOM 2011)*, Shanghai, China, Apr. 2011.
[2] R. Bennesby da Silva and E. Souza Mota, "A survey on approaches to reduce BGP interdomain routing convergence delay on the Internet," *IEEE Comm. Surveys & Tutorials*, vol. 19, no. 4, pp. 2949–2984, Jun. 2017.
[3] D. Perouli, T. G. Griffin, O. Maennel, S. Fahmy, I. Phillips, and P. Cristel, "Detecting the unintended in BGP policies," in *20th IEEE Int. Conf. on Network Protocols (ICNP)*, Oct. 2012, pp. 1–2.
[4] D. Pei, , X. Zhao, D. Massey, and L. Zhang, "A study of BGP path vector route looping behavior," in *24th International Conference on Distributed Computing Systems*, Mar. 2004, pp. 720–729.
[5] X. Dimitropoulos and G. Riley, "Large-scale simulation models of BGP," in *12th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Tel. Systems (MASCOTS 2004)*, Oct. 2004, pp. 287–294.
[6] "A performance evaluation of BGP-based traffic engineering," *Wiley Int. J. Network Mgmt*, vol. 15, p. 177–191, Feb. 2005.
[7] K. Zhang, S. Teoh, S. Tseng, R. Limprasittipom, K. Ma, S. Wu, and C. Chuah, "Performing BGP experiments on a semi-realistic Internet testbed environment," in *25th IEEE Int. Conf. on Distributed Computing Systems Workshops*, Columbus, OH, USA, Jun. 2005, pp. 130–136.
[8] Y. Song, A. Venkataramani, and L. Gao, "Identifying and Addressing Reachability and Policy Attacks in "Secure" BGP," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2969–2982, Oct. 2016.
[9] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush, "10 Lessons from 10 Years of Measuring and Modeling the Internet's Autonomous Systems," *IEEE Jou. on Selected Areas in Comm. (JSAC)*, vol. 29, no. 9, pp. 1810–1821, Oct. 2011.
[10] T. Krenc and A. Feldmann, "BGP Prefix Delegations: A Deep Dive," in *Internet Measurement Conference (IMC '16)*, Santa Monica, CA, USA, 2016, p. 469–475.
[11] A. Marder, M. Luckie, A. Dhamdhere, B. Huffaker, k. Claffy, and J. M. Smith, "Pushing the Boundaries with BdrmapIT: Mapping Router Ownership at Internet Scale," in *Internet Measurement Conference (IMC '18)*, Boston, MA, USA, 2018, p. 56–69.
[12] B. Schlinker, T. Arnold, I. Cunha, and E. Katz-Bassett, "PEERING: Virtualizing BGP at the Edge for Research," in *ACM CoNEXT*, Orlando, FL, Dec. 2019.
[13] S. Deshpande and B. Sikdar, "On the impact of route processing and MRAI timers on BGP convergence times," in *IEEE Global Telecomm. Conf. (GLOBECOM)*, Dallas, USA, Nov. 2004.
[14] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Internet Engineering Task Force, Tech. Rep. 4271, Jan. 2006, updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705.
[15] X. Wang, O. Bonaventure, and P. Zhu, "Stabilizing BGP routing without harming convergence," in *30th IEEE Int. Conf. on Computer Comm. Workshops (INFOCOM WKSHPS)*, Apr. 2011, pp. 840–845.
[16] P. Jakma. Internet-Draft: Revisions to the BGP Minimum Route Advertisement Interval. [Online]. Available: https://tools.ietf.org/id/draft-ietf-idr-mrai-dep-04.html
[17] V. Jain. (2018) Troubleshooting BGP. [Online]. Available: https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2017/pdf/BRKRST-3320.pdf
[18] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "On the Scalability of BGP: The Role of Topology Growth," *IEEE Jou. on Selected Areas in Comm. (JSAC)*, vol. 28, no. 8, pp. 1250–1261, 10 2010.
[19] U. Brandes, "On Variants of Shortest-Path Betweenness Centrality and their Generic Computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, May 2008.
[20] L. Maccari and R. Lo Cigno, "Improving Routing Convergence With Centrality: Theory and Implementation of Pop-Routing," *IEEE/ACM Trans. on Networking*, vol. 26, no. 5, pp. 2216–2229, Oct. 2018.
[21] L. Maccari, L. Ghiro, A. Guerrieri, A. Montresor, and R. Lo Cigno, "On the Distributed Computation of Load Centrality and Its Application to DV Routing," in *37th IEEE Int. Conf. on Computer Comm. (INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 2582–2590.

Appendix – Reproducing IoF

This tutorial enables the reader to run a single repetition of one experiment using a pre-built Fabrikant topology with a Fabrikant-based MRAI strategy, which produces a convergence graph similar to the one in Fig. 4. Reproducing all the results would require several experiment hours and several testbed nodes, so we limit this tutorial to a single experiment. The on-line resources include the configuration files required to reproduce all the results in this work. Please follow the instructions on our website to run different experiments and to learn how to conduce further research using our tools.[8] The main assumptions for this document are the following:

- The reader has a general knowledge of Linux systems;
- The reader has knowledge of jFed[9] and how to use it;
- The reader has an account on a Fed4Fire+/GENI authority and is able to reserve resources on imec Virtual Walls[10];
- The reader has his own ssh public and private key associated with the iMinds Authority account already configured, respectively in ~/.ssh/iminds.pub and ~/.ssh/iminds.key.

What follows has been tested on GNU/Linux Ubuntu 18.04, to help those running a different operating system we provide a pre-configured Virtual Machine on our website.

First of all, we have to set up the system for the experiment (not necessary on the Virtual Machine). The first thing to do is to clone the repository with the whole project, with the following commands:

```
mkdir ~/src && cd ~/src/
git clone https://github.com/internetonfire/\
iof-tools.git
```

Now you will find inside the `src` folder a new folder called `iof-tools`. Inside this folder you'll find all the resources needed to execute the experiments. As a first step it is necessary to set up the environment, installing all the required software and libraries. This can be done running the following command (again, not necessary in the VM):

```
./configure_env.sh
```

As previously mentioned, we assume that the user has an iMinds account, with the public and the **unencrypted** private keys stored in `~/.ssh/iminds.pub` and `~/.ssh/iminds.key`, respectively. If not, please follow the instructions provided inside `~/src/iof-tools/README.md`.

Inside the iof-tools folder, you will find a folder named *experimentFiles*, where you find the configuration files created by the configuration generator that we used in the paper. Here we use such pre-configured files, but the IoF website and the source repository includes detailed instructions for generating such topologies.

Before launching the experiment, we have to reserve the resources on the Testbed. The tools include a script (`gen-rspec.py`) that can find available resources and generate an .rspec file for jFed automatically. For our Fabrikant experiment, only two machines are needed and the fastest way to reserve them is to use a ready-to-use rspec file inside the repository `utils/2nodes.rspec`. Run jFed launching `jFed-Experimenter` from command line, then open the file inside jFed and click on "Run". Give a unique name to the experiment and once the nodes are available, save the rspec file by clicking on the "Save Manifest" and store it in the iof-tools folder with the name "demo.mrspec".

Now you can set up the experiment environment with the following command (substitute `IMINDSUSER` with your iMinds authority account username):

```
python3 gen-config.py -r demo.mrspec -u \
    IMINDSUSER -k ~/.ssh/iminds.key
```

Now you can install all the software on the nodes (this can take up to 15 to 20 minutes to complete) by running

```
./setup-nodes-environment.sh
```

After the installation and the configuration of the nodes you can deploy your experiment running:

```
cp -r experimentFiles/fabrikant/\
bird-config-files/0_fabrikant_f17n-dest/ .
./deploy-experiment.sh -d \
    0_fabrikant_f17n-dest
```

After the deploy you can start the experiment running:

```
ssh -F ssh-config node0
./run-experiment.sh -a 19 -n 17 -r 1
```

The first argument is the AS that triggers the change, the second one is the link for which the weight is changed (the AS_Id of the neighboor identifies the link), and the third is how many repetitions to perform.

After the experiment is complete, exit the ssh session and fetch the results with

```
./fetch-results.sh
```

This script will take care of copying the logs from the Testbed to your local machine. When prompted by the script whether the results are for a Fabrikant topology, type "y". Now the results can be found inside the folder `iof-tools/RESULTS/`.

Once you have all the logs you can parse them with the script inside the folder logHandlers.

```
cd logHandlers/parser/
python3 log_parser.py \
-f ../../RESULTS/run1/logs/* -c -t > logs
```

With this tool it is possible to average multiple repetitions of the same experiment (see `README.md`). Here we only have a single repetition to plot, and we can do so by running

```
gnuplot -e "outfile='fab.pdf'; \
inputfile='logs'" \
../../plotsGenerator/Gnuplot/plot_logs.gnuplot
```

The output is a file showing the number of ASes that has reached convergence over time, together with the number of `UPDATE` messages, similarly to what is shown in Fig. 4.

[8]https://iof.disi.unitn.it

[9]https://jfed.iminds.be - If you dont't have jFed installed you can download it from http://jfed.ilabt.imec.be/downloads/stable/jar/jfed_gui.tar.gz

[10]https://authority.ilabt.iminds.be