Implementation of a BGP Route Flap Damping Algorithm for the Bird Routing Project

Alexandre Chappuis, Bastian Marquis and David Klopfenstein @ EPFL.ch

***Abstract*—Today's Internet stability strongly relies on the good behaviour of dynamic routing protocols such as BGP (Border Gateway Protocol), which enables routing between Autonomous Systems. Route flapping is a well-known and undesirable phenomenon occuring in both commercial and private networks. In this report, we explain our implementation of the RFC 2439, BGP Route Flap Damping, for one famous Open Source routing software suite, the Bird Routing Project.**

We also present results of an experiment where we compare the number of BGP updates sent with and without route damping enabled, in a simplified setup where only one of the bgp router received advertisments for flapping routes.

I. Introduction

The inter-domain routing protocol BGP is still surviving to the gigantic growth of the Internet that started during the last decade. However, some widely used applications, such as Skype, still suffer from weaknesses in that protocol. The main problems are twofolds: firstly, BGP has a slow convergence rate, meaning that a change at one location can take long before it is propagated to the other ends of the network. Secondly, if a node becomes unstable, for example if its connectivity constantly comes up and down, it will have bad consequences on the network, both in terms of useless processing at BGP routers and unnecessary routing traffic. Routes advertised and withdrawn at regular interval of time are said to be *flapping*.

Many approaches to counter route flapping were developped in the late 90's. RFC 2439[1] standardized one such approach : it basically blocks updates from flapping routes, and does so until the routes are deemed stable again. This RFC has been used extensively for many years, in both commercial and open source routers.

Although this standard is not recommended anymore[2] for today's routers, we wanted to implement it for the Bird Routing Project[3], hoping that it would serve as a good basis for future possible improvements and extensions to this RFC. There exist many variants of the Route Flap Damping alorithm and the community has not lost its interest in finding robust mechanisms that could allow BGP to be more resilient.

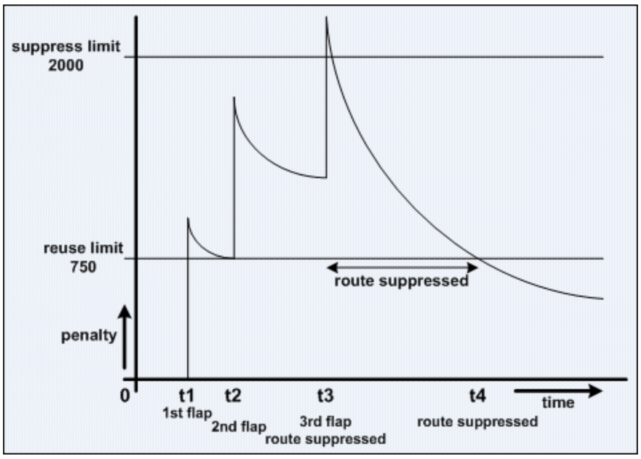
II. Overview

RFC 2349 seeks to limit the impact of route flapping by "damping" (*i.e.* ignoring packets of) missbehaving routes. The solution must be able to distinguish flapping routes from good routes and consume few resources, both in terms of memory usage and process time. The RFC solves these problems by assigning each route a penalty term. Whenever this penalty term for a given route reaches a certain threshold, further advertisements for that route are ignored. This penalty term varies over time : it is increased when the route becomes unreachable and decays as long as the route stays stable. As soon as the figure of merit goes below a *reuse threshold*, the route can be used again.

The figure of merits decays exponentially over time. Exponential decay has several advantages : it can be implemented very efficiently using precomputed *decay arrays*. Also, with exponential decay, the figure of merit keeps trace of previous instabilities for a fairly long time : old instabilities become less and less important over time, while newer ones have more weight.

Network administrators have lots of freedom in choosing the behavior of the penalty term : they can control the half-life of the penalty term, its maximum value (thus controlling the maximum time a route can be suppressed) and both the reuse and cut thresholds.

Here is an example showing how the figure of merit evolves over time (the illustration comes from [5]). The route flaps three times, exceeding the cut threshold only after the third flap. The route is reused as soon as its penalty term goes below the reuse limit.



Penalty terms are re-computed at regular time intervals using timers. Also, every time an update for a route is received, the penalty term for that route is re-computed.

The RFC proposes several optimizations to decrease processing time, at the cost of a slightly bigger memory footprint. *E.g. reuse lists* are used to not have to recompute the figure of merit of all routes : damped routes with similar penalty terms are grouped together in a same reuse list. Penalty terms of the routes in a given reuse list are then re-computed only when necessary.

III. Implementation

Our implementation contains most of the data structure definitions and functions needed for the route damping algorithm (RFC 2439). As the RFC leaves little freedom regarding the implementation and the design of the data structures, our implementation closely matches the RFC. The code, along with the list of commits, is publicly available on `https://github.com/alexchap/Albatros-Project`.

Several parts of BIRD had to be modified for the implementation of route damping. The configuration file parser had to be modified in order to correctly handle the parameters required by the RFC. Also, we had to make similar modifications to the command line parser so that suppressed routes could be listed using the `show dampened paths` command. The `show protocols all` was also extended with a "damped" field, indicating the total number of currently damped routes. Last but not least, we had to add hooks to BIRD's BGP implementation : whenever a route is added or withdrawn, our code is executed. Route damping is enabled at configure time by adding the `--enable-route-damping` switch.

A. Data structures

The data structures used by the implementation are very close to those described in the RFC (§4.4, *Run Time Data Structures*). The most important of which are certainly `damping_config` and `damping_info`. Their definitions can be found in *damping.h*.

`damping_config` groups all configurations parameters for a BGP peer. This allows network administrators to have different sets of parameters for differents routes. *I.e* `damping_config`s have a one-to-one relation to BIRD's `bgp_proto`. This structure contains all the parameters controlling the behaviour of the damping algorithm : `reuse_threshold`, `cut_threshold`, `ceiling`, `tmax_hold` and `half_time`. Additionally, it contains some informations used to speedup the algorithm (at the cost of a more important memory consumption) : both the decay arrays and the reuse lists are contained here. The former speeds up the computation of new figure of merits by pre-computing a predefined set of (costy) exponentials and using them when figures of merit needs to be updated. The latter limitates the time of computation at every timer tick by grouping damped routes according to their penalty term. Instead of being updated at every tick, a group of routes is updated once every n ticks, where n is the number of reuse lists. Reuse lists are implemented using `slists`, a doubly linked list implementation provided by BIRD.

BIRD's timer are used to regularly update the penalty terms of routes : every `bgp_proto` structure contains a `reuse_list_timer`. These timers are set up to call `damping_reuse_timer_handler` at every tick. This function is responsible for updating the penalty terms of damped routes and re-inserting damped routes into the routing table if their penalty term is low enough.

The core of the damping algorithm lies with the `damping_info` structure. The penalty term is stored in that structure, along with a timestamp indicating when the last update of the penalty term was. Other informations, such as the IP prefix of the route, reuse list pointers and route attributes (BIRD specific) are also included in `damping_info`. One such structure is allocated once a route starts to flap, and is deallocated whenever the penalty term becomes "small enough". There is a one-to-one mapping between routes (`rte` in BIRD) and `damping_info`. This mapping was made possible by FIBs, an hash table implementation provided by BIRD's API. FIBs basically map IP prefixes to any other data structure, `damping_info`s in our case.

Last but not least, the fact that BIRD uses only one thread made our job much simpler as we didn't have to worry about concurrency.

B. Configuration parameters

Our extended version of BIRD gives network administrators a set of four parameters to control the behaviour of route damping. In particular, `cut_threshold`, `reuse_threshold`, `tmax_hold` and `half_time` are all modifiable from the configuration file. Both `tmax_hold` (the maximum time a route can be kept damped, it defines the ceiling value) and `half_time` (the time taken by a penalty term to diminish by half its value) are expressed in seconds. Here is an example of configuration :

```
protocol bgp bgp_config {
  descriptionl "BGP daemon";
  debug all;

  # Specific config for BGP
  local as 65011;
  source address 10.10.10.11;
  multihop;
  next hop self;
  path metric 1;

  # Route damping configuration
  route damping;
  cut_threshold 2500;
  reuse_threshold 1000;
  tmax_hold 3000;
  half_time 900;
}
```

The default parameters (hard coded in *damping.h*) are :

`cut_threshold`	1500
`cut_threshold`	750
`half_time`	900
`tmax_hold`	3000

The `route damping` parameters can be used to enable/disable route damping for a particular neighbour. All the other parameters used by the RFC (*e.g.* decay arrays, scaling factor, ...) are derived from those four values.

BIRD uses Bison to parse the configuration files and we had to extend it to support the new set of keywords. Those parameters are passed to the `damping_config_new` function, which allocates and initializes a new `damping_config` and computes the other parameters. Finally, checks are made to ensure that the parameters are consistent with each other. *E.g.* the `cut_threshold` must be bigger than `reuse_threshold`, and damping should be enabled only for E-BGP.

C. Detecting BGP Flapping

Our implementation sticks to the RFC regarding route flapping detection. Hooks were added in the BGP module of BIRD for that purpose. Those hooks ensure that the `damping_add_route` or `damping_remove_route` functions are called when routes are advertised or withdrawn, respectively.

As those functions are similar to their equivalent in the RFC, only the relevant implementation details are covered here. Please refer to the RFC for additional details.

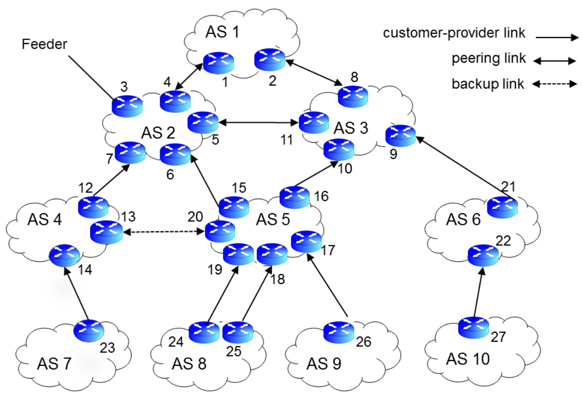
When a route is removed for the first time, a new `damping_info` is allocated in a `damping_info_fib`, and its penalty term is set to a default value. The penalty term of this route is updated as the route is re-advertised and removed again. In particular, when the route is removed, its penalty term is updated and then incremented by a default value : 1000, whereas when it is added, its penalty term is just updated. The penalty term is represented by a scaled integer, so that only the right level of precision is kept during execution. The main idea is that floating point operations are used when the figure of merit needs to be updated, but the result is stored as an integer (thus saving memory space). This usually results in an accidental loss of precision, except that in our case, the decrease in precision is controlled so as not to crash the application. Penalty terms are all implicitely multiplied by 1000, implying that only three digits after the dot are kept.

`damping_info` are de-allocated when the penalty gets smaller than `reuse_threshold` divided by two. The rationale being that when the penalty term gets much smaller than the `reuse_threshold` index, keeping the `damping_info` structure with a small figure of merit is equivalent to de-allocating it (in the worst —very unlikely — case, it will only add the cost of re-allocating a new `damping_info` structure).

IV. Evaluation

A. Overview

We used a topology of 27 routers in the NSL cluster to evaluate our implementation. This topology is composed of 10 Autonomous Systems :



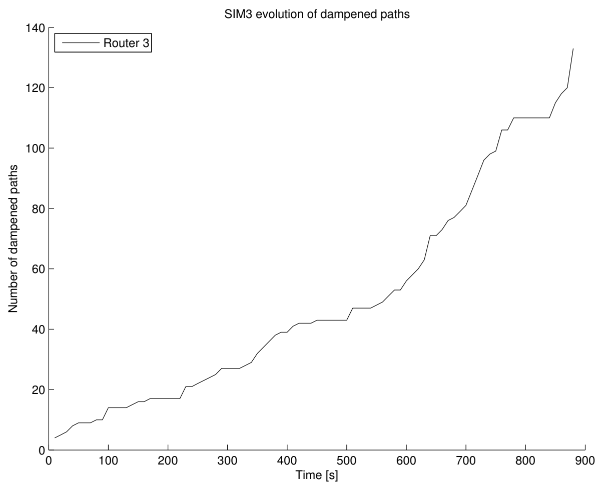
The experiment replays updates contained into an archive, *updates.20100401.1729* (the archive comes from [6]). This archive is composed of advertisements sent by 11 BGP routers and collected over a period of time of 15 minutes. All updates are sent to router 3 (`10.0.0.3`), which then relays them in the network. We ran 3 simulations, with different settings :

1) The first simulation was run with the original BIRD code, without route damping.
2) The second one had route damping enabled and used the default parameters.
3) The third one used a more aggressive settings : `cut_threshold` was lowered to 1000 (vs 1500 for the default setting).

During the experiment, the number of updates and withdrawals were collected every 10 seconds (*i.e.* we collected the total number of update/withdrawals received at every router). The number of dampened routes was also collected, for the settings where route damping was enabled. The number of updates/withdrawals cumulates over time, whereas the number of dampened routes is immediate.

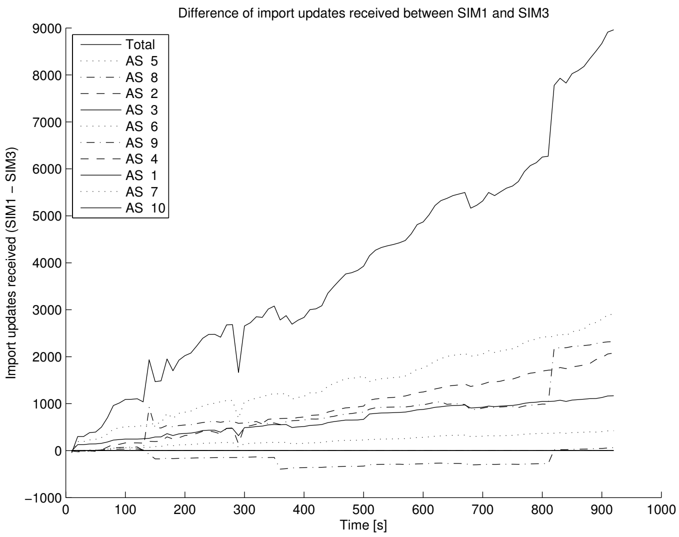
B. Evolution of dampened paths

The following figure shows the accumulated number of dampened routes for router 3. Snapshots were taken every 10 seconds. The results are shown for the third setting. Unsurprisingly, the results for the second setting (with more passive damping) were not as good : most of the route flappings went undetected (this figure is not shown here).



C. Number of route updates

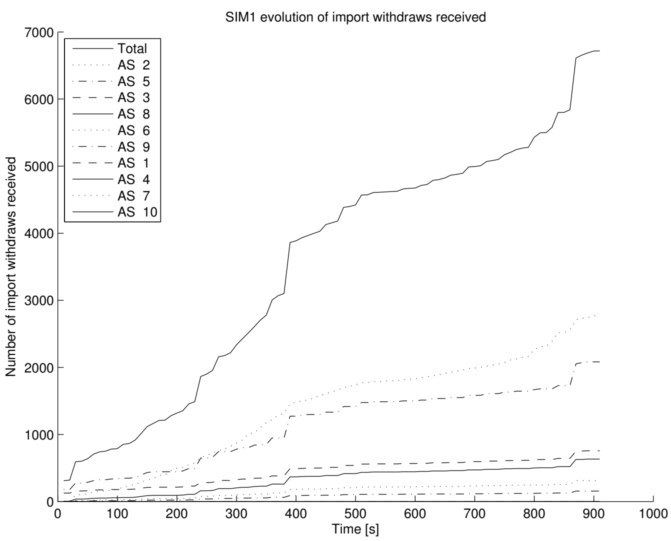
While it is interesting to know the number of dampened paths, it is much more relevant to know how many updates were blocked, as limiting irrelevant traffic is precisely the aim of the RFC. The following figure shows the difference of the number of updates received in the first and third settings, per AS.



As shown in the figure, the number of blocked updates is around 10^4 for a period of about 15 minutes. We didn't expect so many messages to be blocked. Also, we can notice that the number of updates varies a lot between different ASes. This can certainly be exploited to enable route damping on only the most vulnerable nodes (typically ASes 5, 8 and 2 in this case).

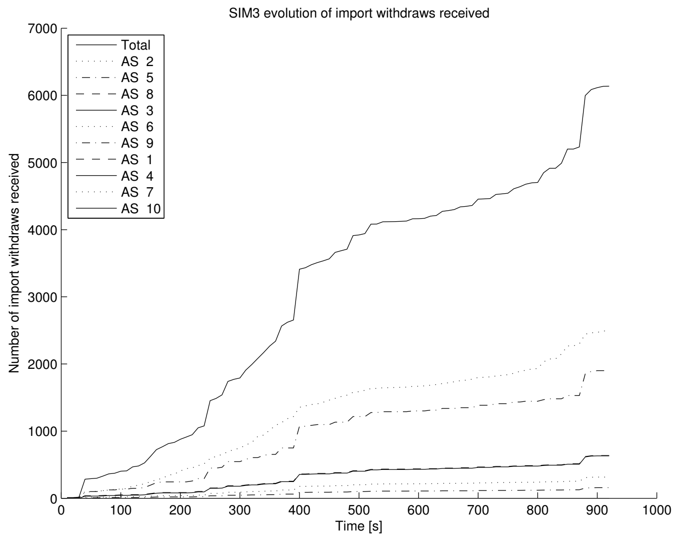
D. *Number of withdrawals*

Similarly, we measured the number of withdrawals per AS.



The previous figure shows the number of withdrawals for the first setting. The next one shows the number of withdrawals for the third setting. Once again, the second setting was much too laxist and yielded results similar to that of the first setting.

The differences are really small even between the first and third settings though. The biggest differences are in the order of the hundred, which contrasts a lot from the previous results. This can be explained by the ratio between updates and withdrawals. There are about 10^5 updates received, much more than the 10^3 withdrawals received. It is thus logic that the difference between the first and third settings have the same ratio.



V. Conclusion

We have shown that a "simple" method (around 1000 lines of C code) like that presented in RFC 2439 can significantly improve the overall health of a whole network. It not only reduces the traffic, but can also help BGP converge by preventing unwanted traffic from spreading to the whole network (we didn't study this part).

A. *Future work*

Several extensions proposed in the RFC still need to be implemented. For instance, the RFC suggests that a network administrator may want penalty terms to behave differently depending on whether their route is dampened or not. Initially, we also wanted to improve the algorithm of the RFC, but understanding BIRD's code and debugging our own took us more time than expected, and we eventually ran out of time and could barely finish the testing phase.

Finally, it would have been really interesting to test route damping on a real scale system, and for a longer time span than 15 minutes.

References

[1] The RFC 2439, BGP Route Flap Damping, http://www.ietf.org/rfc/rfc2439.txt

[2] RIPE Recommendations On Route-flap Damping, http://www.ripe.net/ripe/docs/ripe-378

[3] Bird Routing Project, http://bird.network.cz

[4] Our publicly available repository, https://github.com/alexchap/Albatros-Project

[5] itcertnotes, http://www.itcertnotes.com

[6] http://routeviews.org