# What do parrots and BGP routers have in common?

David Hauweele*, Bruno Quoitin
University of Mons (UMONS)
{first.last}@umons.ac.be

Cristel Pelsser†
University of Strasbourg
pelsser@unistra.fr

Randy Bush
Internet Initiative Japan (IIJ)
randy@psg.com

## ABSTRACT

The Border Gateway Protocol propagates routing information accross the Internet in an incremental manner. It only advertises to its peers changes in routing. However, as early as 1998, observations have been made of BGP announcing the same route multiple times, causing router CPU load, memory usage and convergence time higher than expected.

In this paper, by performing controlled experiments, we pinpoint multiple causes of duplicates, ranging from the lack of full RIB-Outs to the discrete processing of update messages. To mitigate these duplicates, we insert a cache at the output of the routers. We test it on public BGP traces and discuss the relation of the cache performance with the existence of bursts of updates in the trace.

## 1. INTRODUCTION

The Border Gateway Protocol [1] (BGP) is the de facto standard used to exchange inter-AS routing information on the Internet. Its correct and scalable behavior is critical to the operation of the Internet. One of the keys to BGP scalability is the use of *incremental routing updates*: only changes in destination prefix reachability are advertised. These changes include the reachability of a new prefix, the unreachability of an existing destination (withdrawal), or a modification of the path attributes associated with a destination. Path attributes are involved in routing decisions and also ensure proper protocol behavior such as avoiding routing loops. According to the protocol specification, a BGP speaker should not issue an update containing the same BGP information as was most recently advertised for the prefix.

Anomalous BGP behavior has been observed as early as 1998 [2]. Based on a 9 months trace of the BGP traffic exchanged between backbone networks, Labovitz et al. showed lack of aggregation and high routing instability with up to 99% of exchanged routing information not being related to topological changes. In particular, they observed the occurrence of redundant BGP update messages that they called *duplicate updates*. At that time, most of the duplicates where due to bogus stateless BGP implementations. The authors noted that the observed high level of instability was detrimental to the operations of the Internet, causing high router CPU load, making routers unresponsive and in the worst cases leading to packet or routing information losses. In addition, they may sometimes trigger unreachability when interacting with route flap damping [3].

*David started this work during his internship at IIJ.

†The credits go to IIJ for supporting Cristel's work.

Several studies later revisited BGP dynamics [4–8] and its impact on router CPU load [9], some focused on BGP duplicates. Although the number of pathological updates declined over time, duplicates still constitute a significant part of the BGP traffic with up to 15% of the updates observed at RIPE monitors in 2006 [5]. It was later shown that the duplicate problem is even worse for routers in the core of the Internet with the portion of duplicates varying from 7% to 60% in 2008 [7]. More recently, in 2009, Park et al. [6] studied over 90 RouteViews/RIPE monitors and showed that the duplicates make up 13.5% of the aggregated BGP traffic. Routers can receive up to 86.4% of duplicates during their busiest time. These previous works show that duplicates are a continuing problem. We confirm this observation by looking at all sessions from EQUINIX, ISC, LINX and WIDE RouteViews collectors from 2009 to 2014. 48.5% of the traces we observed had more than 10% of duplicates. The traces also display a high variability with an average of $(18.84 \pm 22.31)\%$ duplicates. Finally, [6] hinted that a change in attributes attached to iBGP routes may trigger eBGP duplicates. To the best of our knowledge, so far, no thorough study has explained their origin or tried to mitigate the problem.

In this paper, we make the following contributions:

- We discuss in Section 2 the causes of today's duplicates. Although the majority of duplicates in 1998 were bogus route withdrawals, this is not the case today (less than 0.5% on almost all traces). To understand what causes duplicates, we inject carefully crafted BGP updates into a router and we correlate the input and output BGP traffic. Based on this, we identify different causes for duplicates. Most duplicates today are due to implementions trading off between memory footprint and statefulness.

- In Section 3, we devise a caching mechanism that mitigates duplicates. The benefit of using a cache is that the amount of memory used can be controlled. We evaluate the efficiency of our caching mechanism on several real world BGP traces, using several replacement strategies. We show that our cache significantly reduces duplicates for prefixes in the default free zone even with a small cache size.

## 2. THE ORIGIN OF DUPLICATES

To investigate the origin of BGP duplicates, we follow two different approaches. First we look at a router that receives live BGP feeds. We capture all the BGP traffic and we man-

ually correlate duplicates observed in the outbound traffic with messages in the inbound traffic. This is an approach similar to that used by Park et al. in [6] that gives us some initial insight on potential causes for duplicates.

Second, we perform a fully controlled experiment where we inject crafted sequences of messages into a test router. We then look for duplicates in the output messages. Our experiment allows to confirm the hypotheses of Park et al. on the origin of duplicates. We also go much further as we establish three additional causes for duplicates.

This section explains our methodology and subsequent observations.

## 2.1 Definitions

We define a duplicate as a *redundant* prefix advertisement with the *same attributes* as the most recent update for this prefix on the same session and not interleaved with a withdrawal or a session reset. This definition is stricter than the one in [2] where an update is considered a duplicate (AADup) if its AS-Path and Next-Hop do not change. When we count duplicates, we include the initial duplicated route advertisement.

We also define the *ratio of duplicates* as the number of duplicates (including the original messages) over the total number of messages. With this definition, a trace where every advertisement is duplicated will have a ratio of 100%.

## 2.2 Real BGP feed experiment

The objective of this experiment is to manually investigate some occurrences of duplicates by correlating the duplicates observed at the output of a router with the messages it receives. Our setup is shown in Fig. 1. Devices $r0$, $r1$ (Cisco) and $r2$ (Juniper) are real routers while *mon*0 is a dedicated host running a software BGP router (Quagga).

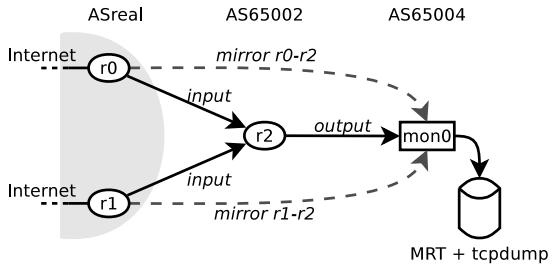


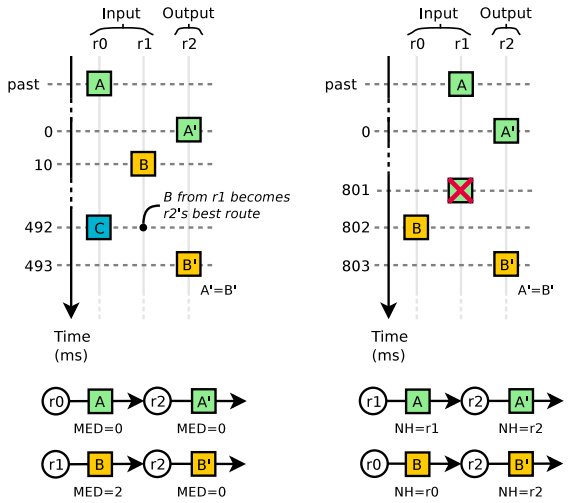

Figure 1: Setup for the I/O correlation.

The router under test is $r2$. It receives BGP messages from $r0$ and $r1$ through *input* eBGP sessions. After selecting its best routes, $r2$ sends BGP messages over a single *output* eBGP session to *mon*0. The routes learned by $r0$ and $r1$ are from real BGP feeds received in September 2013 for a duration of 23 days.

The *mon*0 host captures all the BGP messages received on the *mirror* and *output* sessions. The *mirror* sessions (dashed lines on Fig. 1) allow to capture the *input* routes advertised by the upstream routers $r0$ and $r1$. To reduce timing differences between the *input* and *mirror* sessions, both sessions are placed in the same update group on $r0$ and $r1$. The *Minimum Route Advertisement Interval* (MRAI) is also set to zero on these routers.

The messages are stored in MRT format. MRT records route advertisements, route changes and route withdrawals. Each record contains a timestamp and the path attributes. TCP-level traces of all the BGP messages received are also captured. This allows us to validate the MRT capture and dwelve deeper in the BGP message packet details e.g. to check the ordering of attributes.

We describe in the following paragraphs two common cases we observed. The first case involves the Multi-Exit-Discriminator (MED) attribute while the second case involves a rewritten Next-Hop. We do not know the exact frequency of these cases, as we have to manually extract the data.




(a) MED case (b) Next-Hop case

Figure 2: Common causes of duplicates. Timeline of the updates seen at the output of each router.

In the MED case, illustrated in Fig. 2a, we believe the duplicate is caused by a MED attribute stripped at the output of $r2$. Three different *input* routes are involved, all for the same IPv4 prefix. The first route, $A$, has an AS-Path of length 5 and a MED value of 0. The second route, $B$, has the same AS Path as $A$ but a MED value of 2. The third route, $C$, has an AS Path of length 6 and a MED value of 0. At time 0ms, $r2$ announces route $A$ learned from $r0$. Before announcing $A$, $r2$ updates the AS-Path and strips the MED, which produces route $A'$. At time 10ms, $r1$ announces route $B$ to $r2$. The decision process of $r2$ ranks route $A$ better than route $B$, causing no change in $r2$'s best route. At time 492ms, $r0$ announces to $r2$ route $C$ which has a longer AS-Path. Route $C$ implicitly withdraws route $A$. As a consequence, $r2$ now selects route $B$ as best. Before announcing $B$, $r2$ strips the MED value, producing $B'$. *Output* routes $A'$ and $B'$ are equal, hence $B'$ is a duplicate of $A'$.

In the case illustrated in Fig. 2b, we believe the duplicate is caused by the next-hop attribute. This case involves two routes. Route $A$ announced first by router $r1$, is selected as best by $r2$ and announced on the *output* session at time 0ms. Before announcing route $A$, $r2$ rewrites the next-hop and emits route $A'$. At time 801ms, router $r1$ explicitly withdraws route $A$. At time 802ms, router $r0$ announces route $B$ although it does not trigger any change in $r2$ yet. Finally, at time 803ms, router $r2$ selects route $B$ as best. Before announcing route $B$, $r2$ rewrites the next-hop value with its own IP address, leading to route $B'$. Routes $A$ and $B$ only differ by their next-hop (resp. $r1$ and $r0$), hence routes $A'$ and $B'$ are identical.

## 2.3 Controlled experiment

To confirm the hypotheses of the previous section, we perform the same input/output matching in a fully controlled experiment. We systematically test a large set of situations that may not have appeared in the setting with a real, live BGP feed. We are able to find additional causes of duplicates and pinpoint more precisely the reasons behind these duplicates.

The setup depicted in Fig. 3 is similar to the previous experiment except we use a machine *inj*0, running Linux, to inject crafted updates to the router under test, $r0$, and another to capture its *output*. Router $r0$ is a Cisco 7200 running IOS v15.3. On *inj*0, we use ExaBGP [10] to inject synthetic updates. The monitoring host *mon*0 collects the routes observed on the *output* and *mirror* sessions with a Quagga BGP daemon and with tcpdump. The *mirror* session is used to validate *inj*0's program. We check the ability of this program to send BGP messages accurately. We measure that the minimum interval between two consecutive updates sent by ExaBGP is 1ms.

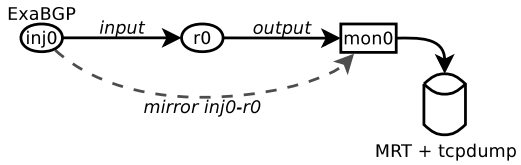



Figure 3: Setup for the injection.

Table 1 summarizes the results of the injection experiment. Due to space limitations, only results for a small number of test cases are presented. For each experiment, the first column shows the average delay between messages observed on the *input* and its standard deviation. Second column shows the same information for the *output*. The last column shows the ratio of duplicates. That is, the number of duplicates including the initial update over the number of updates (see Section 2.1).

| Test case | Input (ms) | Output (ms) | Dup. |
|---|---|---|---|
| `NotVisible` | – | – | 100% |
| `RFlap` (1 ms) | $1.23 \pm 0.50$ | $3.47 \pm 3.46$ | 69.0% |
| `RFlap` (2 ms) | $2.07 \pm 0.39$ | $2.84 \pm 0.99$ | 25.9% |
| `RFlap` (3 ms) | $3.07 \pm 0.44$ | $3.06 \pm 0.48$ | 0.1% |
| `AFlap` (1 ms) | $1.22 \pm 0.69$ | $3.74 \pm 17.25$ | 95.1% |
| `AFlap` (2 ms) | $2.07 \pm 0.36$ | $2.07 \pm 0.10$ | 4.7% |
| `AFlap` (3 ms) | $3.07 \pm 0.44$ | $3.06 \pm 0.09$ | 0.1% |

Table 1: Results of selected injection test cases.

### 2.3.1 Internal / non-transitive / filtered attributes

This first set of experiments (`NotVisible`) considers the case of attributes whose changes should not be visible from the outside of an AS as they are either internal, non-transitive or filtered/rewritten by output policies. The objective of these experiments is to test whether or not such attributes could cause duplicate routes to be sent by the router.

For this purpose, we repeatedly send a sequence of 2 route updates $(A, B)$ for the same destination prefix. Route $B$ differs from route $A$ for only a specific internal / non-transitive / filtered attribute. The expected behavior is as follows. When route $A$ is received, it is selected as best as there is no other choice. It is then propagated on the output session. When route $B$ is received, it replaces route $A$ (implicit withdraw). Route $B$ should not be propagated to the *output* session as it differs from route $A$ only by an attribute that is either internal, non-transitive, or removed by a filter. Hence, on the *output* session, routes $A$ and $B$ are identical.

We observe a duplicate ratio of 100% for experiments in this class, as shown in Table 1 for the `NotVisible` test case. The router was not able to detect that the second route was a duplicate of the previous. We explain this behavior on the statelessness of the BGP implementation.

These results held for the following attributes: MED, Local Pref, Cluster List, and Originator ID. We also observed a 100% duplicates ratio for non-transitive Community values, for Community values stripped by outgoing policies and for rewritten Next-Hop (as already observed in Section 2.2).

### 2.3.2 Fast flapping route

In a second set of experiments (`RFlap`) we investigate the impact of a flapping route on the generation of duplicates. The experiment relies on the repetition of a simple sequence of 2 BGP updates $(A, W)$ for the same prefix. $A$ announces a route while $W$ withdraws it.

The objective of this experiment is to trigger duplicates by forcing a route to change multiple times before the router has the opportunity to propagate it. To understand this behavior, we need to refine our model of how a router generates updates. When a route towards a prefix changes, the main BGP process does not send an update immediately. Instead, this task is delegated to a separate thread that periodically reads the RIB and advertises the routes marked as changed.

The following scenario illustrates how the transmission of a duplicate update can be caused. When the first Announce is received, the route is marked as changed in the RIB. The RIB is then scanned and an update is sent. Then, the Withdraw is received and the route is again marked as changed. However, before the RIB is scanned, the third message (second Announce) is received and the route is again marked as changed. When the RIB is scanned, the second Announce, identical to the first one is sent. It is a duplicate as the router did not have time to send a Withdraw between the two Announces.

We repeat this experiment with increasing delay between updates: 1ms, 2ms and 3ms. The results are in Table 1 for test case `RFlap`. We observe that with a 1ms interval, almost 70% of output updates are duplicates. When the interval between input updates increases, the ratio of duplicates decreases. With a 2ms interval, the ratio is almost 26% and at 3ms, there are almost no duplicates.

We also tested the impact of the MRAI on the generation of duplicates. We conducted the same experiment with a larger interval of 2 seconds and a MRAI set to 6 seconds. With this experiment we still generated more than 30% of duplicates.

### 2.3.3 Flapping attribute

This third set of experiments (`AFlap`) looks at flapping attributes. The principle is identical to the *RFlap* experiment except that the second message is not a withdraw but an update with a transitive attribute that flaps from one value to another and back. As an example, we present the results for routes where the origin AS in the AS-Path has value $x$ in the first and third updates and has value $y \neq x$ in the sec-

ond update. We see in Table 1 for the `AFlap` test cases that the ratio of duplicates decreases with an increasing interval between the *input* BGP messages.

The explanation for these results is analogous to the *RFlap* experiment. When the interval between messages is small, the router marks the route as changed after the second message, but the third message, reversing the second update, is received before the second message is propagated downstream.

# 3. MITIGATING DUPLICATES

In Section 2, we found several causes explaining the generation of duplicates. According to the BGP specification, such duplicates should not appear. When a router advertises a route for a given prefix, it should store this route in the RIB-Out associated with the peer. When it later advertises a route for the same prefix, it looks at the current entry in the RIB-Out. If the current entry is the same as the new advertisement, the router does not send it because it would be a duplicate update.

We found out that although most router implementations support a RIB-Out, the implementation might be partial or operators might disable it to spare memory, especially on older hardware. Some vendors [11] explicitly recommend to disable the RIB-Out when the router has a large number of peers.

For this reason, we need to devise a solution that is not a full RIB-out but that still significantly reduces the number of BGP duplicates. This new mechanism must come at a lower cost than a RIB-Out in terms of memory consumption.

To obtain a baseline on the possible load reduction, we count the legitimate updates after filtering all duplicates. We compare this count to the number of updates in the original trace. We use a BGP trace obtained from the Equinix RouteViews collector and focus on the session with peer AS5769 (EQUIX-1). Fig. 4 shows two 12 hours excerpts of this session starting on 2013-9-17 at 0:00 (left) and 2013-9-18 at 4:00 (right). The Figure shows the total amount of updates received during the last hour (dark gray) and the same information after all duplicates have been filtered (light gray). On the left the trace has a relatively low rate of duplicates. We observe an average of 5,188 duplicates per hour. By filtering all duplicates, the number of updates on this period is reduced by an average factor of 1.62. On the right the trace features two large spikes of updates. On the largest spike, we count $5.46 * 10^5$ duplicates. By filtering all duplicates, the number of updates in this spike is reduced by a factor of 5.08.

We observe that a significant reduction in BGP traffic can be achieved by filtering duplicate updates. If CPU usage is proportional to the number of updates, sizable improvement in performance can be expected by getting rid of duplicates especially on small routers with limited CPU.

## 3.1 Caching router

Instead of a RIB-Out, we propose a small cache at the output of the router which can significantly reduce the number of duplicates at a far less memory cost. The advantage of this solution is that it can easily be added to the output of a router with little modifications of the BGP implementation.

A cache at the output of the router works similarly to a RIB-Out but using less memory. When a cache reaches its maximum capacity, it must remove one of its entries to add

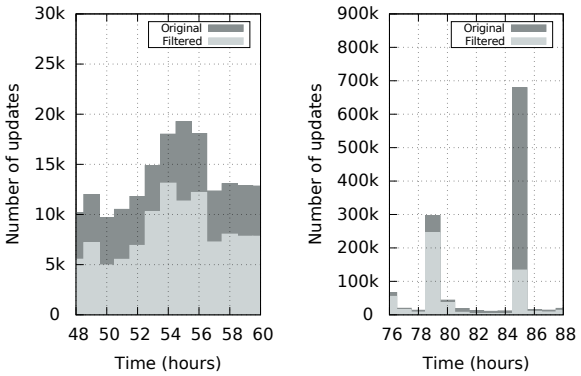



Figure 4: Two excerpts of the EQUIX-1 trace. Low rate of duplicates on the left. Spikes of duplicates on the right. We compare the original trace to the same trace with all duplicates filtered.

| Name | Eviction strategy |
|---|---|
| `lru` / `mru` | Least/most recently queried entry. |
| `lrh` / `mrh` | Least/most recently hit entry. |
| `lfu` / `mfu` | Least/most frequently queried entry. |
| `lfh` / `mfh` | Least/most frequently hit entry. |
| `random` | Random entry. |

Table 2: Eviction strategies

a new prefix. There are multiple ways to choose which prefix to remove when the cache is full. These selection methods are called *eviction strategies*. A cache is defined by its size and its eviction strategy.

In our case, the cache can be viewed as an Abstract Data Type (ADT) with the following operations: `query`, `remove` and `clear`. The `query` operation tells if an entry for a given key and value exists. If the given value is different from the entry in the cache, the entry is updated. If the cache does not contain an entry for this key, it adds this new entry to the cache. When the size reaches the cache limit, the cache eviction strategy comes into play. An entry is removed before the addition of the new entry to the cache. These two cases are considered *miss* queries. Instead, if the cache contains an entry for this key with the same value, the query is considered a *hit*.

The `remove` operation takes a key and if it exists, removes the associated entry from the cache. The `clear` operation removes all entries from the cache.

When the router advertises a given prefix and set of attributes, it queries the cache with the prefix as the key and the set of attributes as the value. In the case of a hit, the advertisement is a duplicate caught by the cache, and the router inhibits the advertisement. In the case of a miss, an advertisement is sent to the peer. When the router withdraws a given prefix, it removes the cache entry with the prefix as key and sends the withdraw to the peer. Finally when the router opens or reopens a session, the cache content is cleared and the router sends an open message to the peer.

## 3.2 Evaluation methodology

We assess the performance of the cache with the different eviction strategies listed in Table 2. The `random` cache

uses a pseudo random number generator to select an entry to remove. We use this strategy as a baseline to determine if other strategies are able to exploit characteristics of the input trace or if there is no specific pattern to exploit. Any such strategy should perform better in average than the random strategy.

In order to test the performance of the cache, we replay through the cache a previously captured trace. The cache then filters the duplicates. Since time does not matter for the eviction strategy, the cache can replay the trace without taking into account the elapsed time between each message. As a result it is possible to simulate the behavior of the cache on a captured trace much more rapidly than playing it directly on a router.

We use the Minimum Collection Time [12] (MCT) algorithm to accurately identify the start and duration of the routing table transfers in the BGP trace. We add an implicit OPEN message at the beginning of each detected table transfer so that updates within the table transfer do not count as duplicates.

## 3.3 Dataset

We measured the updates rate and duplicates ratio of several sessions at the RouteViews collectors from 2009 to 2014. We observed that the duplicate ratio was higher than 10% on 48.5% of the traces. The quantity of updates and duplicates also varies greatly from one session to another. The average rate of updates and duplicates per week across all traces observed in 2014 is of $(3.6 \pm 10.8)$ millions updates and $(1.0 \pm 3.7)$ millions duplicates respectively.

In order to take this variability into account, we apply the cache on three different sessions obtained from RouteViews collectors during one week period. We choose these three sessions as they contain a significant number of updates ($> 1$ million/week) but exhibit 3 extreme behaviours for what concerns the duplicates. Fig. 5 shows the hourly number of duplicates over time for these three traces.

| | EQUIX-1 | EQUIX-2 | WIDE |
|---|---|---|---|
| **Peer ASN** | 5769 | 2914 | 7500 |
| **Start** | 2013-09-15 | 2014-10-15 | 2013-09-15 |
| **End** | 2013-09-22 | 2014-10-22 | 2013-09-22 |
| **Updates** | $4.5 * 10^6$ | $1.55 * 10^7$ | $1.2 * 10^6$ |
| **Duplicates** | 59.38% | 98.36% | 2.17% |
| **Spikes** | Large | No | Small |

Table 3: Characteristics of three different traces.

Table 3 summarizes the characteristics of the traces. The number of updates and the ratio of duplicates observed vary greatly from one trace to another. The first trace, EQUIX-1, exhibits a large number of updates ($4.5{*}10^6$) and a high ratio of duplicates (59.38%), a large fraction of which (41%) visible as two large spikes of duplicates. In comparison EQUIX-2 has a higher number of updates ($1.55 * 10^7$) and a higher ratio of duplicates (98.36%) but displays no major spike. Finally the WIDE trace has a very low ratio of duplicates ($1.2 * 10^6$) and does not contain any large spike.

## 3.4 Results

We apply the cache on the WIDE and EQUIX-1 traces presented in Section 3.3. We also apply the cache on the third trace, EQUIX-2 with a fixed size of 65k entries and

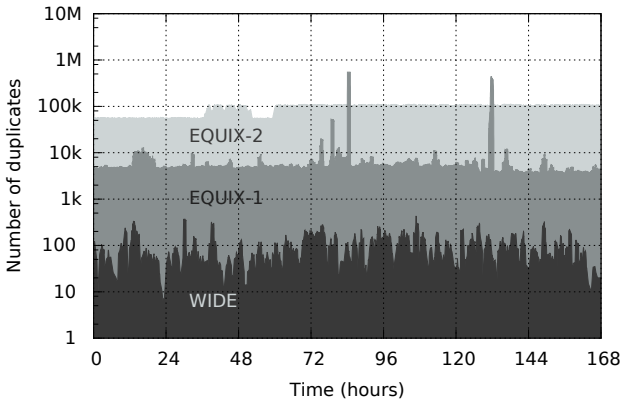



Figure 5: Three traces with different duplicates ratio. Each point shows the number of duplicates seen during the last hour.

| | WIDE | | EQUIX-1 | |
|---|---|---|---|---|
| **Cache** | **32k** | **65k** | **32k** | **65k** |
| No cache | 2.172% | | 59.38% | |
| `lfh` | 1.351% | 0.885% | 49.14% | 45.50% |
| `lfu` | 1.324% | 0.818% | 49.09% | 45.45% |
| `lrh` | 0.040% | **0.009%** | 42.91% | 42.27% |
| `lru` | **0.039%** | 0.016% | **42.90%** | 42.25% |
| `mfh` | 1.556% | 1.121% | 53.85% | 50.30% |
| `mfu` | 0.830% | 0.173% | 52.97% | 48.17% |
| `mrh` | 1.555% | 1.078% | 53.34% | 49.68% |
| `mfu` | 1.518% | 1.014% | 52.93% | 49.04% |
| `random` | 0.042% | 0.020% | 42.98% | **41.87%** |

Table 4: Percentage of duplicates at the output of the EQUIX-1 and WIDE traces for different cache eviction strategies and sizes expressed in number of different routes.

the `lru` strategy. These traces were captured at different locations and time. They show different behaviours against which we test our solution.

Table 4 summarizes the percentage of duplicates found at the output of the WIDE and EQUIX-1 traces for two cache sizes, 32768 (32k) and 65536 (65k) different routes, and multiple strategies. The first line gives the duplicate ratio of the original trace (no cache applied). For the WIDE trace, the `lru` and `lrh` eviction strategies provide the best results. The best cache, `lrh`, reduces the original duplicate ratio by a factor 241. Further, the larger cache provides better results. In the case of the WIDE trace, the `lru` cache is 2.44 times as effective in filtering the duplicates with a cache that is twice as large.

On the EQUIX-1 trace, the cache performs poorly. With a 32k cache, the best results are achieved with the `lru` strategy. However, the output duplicate ratio remains high, at 42.9%. Doubling the cache size does not provide as much benefit as with the WIDE trace. Moreover, a striking result is that in the case of the large cache, the `random` eviction performs better than the other techniques. This indicates that the eviction strategies are not able to properly exploit the characteristics of the trace.

These results suggest that a higher duplicate ratio inhibits the performance of the cache. However, when we apply the `lru` cache of 65k on the EQUIX-2 trace, which exhibits a

higher duplicates ratio than EQUIX-1, the duplicate ratio drops from 98.36% to 5.83%. This reduces the number of updates for the trace by a factor of 50.

This shows that a cache is able to filter a session with a very high number of duplicates. I.e., the performance does not depend on the number of duplicates but rather on other characteristics of the trace. Actually, it depends on the number of distinct prefixes at the origin of those duplicates. During the EQUIX-2 trace this number stays at an average of 1000 prefixes per hour. During the EQUIX-1 trace this number stays at the same value most of the time. However when the largest spike of duplicates occurs more than $2*10^5$ distinct prefixes are involved during less than one hour. As a result the cache did not retain most of the route changes occurring during this period. Hence subsequent duplicates caused by these routes were not filtered by the cache.

## 3.5 Discussion

Although a cache is effective in filtering feeds with a high ratio of duplicates (e.g. EQUIX-2), we observed that spikes of updates involving a large number of distinct prefixes are detrimental to the performance of the cache. These spikes can have multiple origins. First, spikes of updates can be caused by large routing events beyond the router. Second, spikes can be caused by routing table transfers following a session reset or a change in outbound policies. It is indeed common for network operators to prompt a table transfer with a ROUTE REFRESH message in order to apply changes in their inbound policies. However spikes in this second category must have been filtered by the MCT algorithm applied beforehand.

While we can explain the origin of spikes, we do not know if these spikes represent a frequent feature of the BGP sessions. We now measure the maximum spike size in term of distinct prefixes for all RouteViews sessions we observed during the year 2014. We also apply a `lru` cache of 65k entries on all these traces to map the performance of the cache to the size of the spikes observed in the sessions. The sample size for all measured sessions is of 1339 traces.

We define attenuation as the ratio of the number of duplicates seen in the original trace over the number of duplicates seen after the cache. The average attenuation of duplicates for all observed traces is 300.47. If we distinguish the traces by the size of their maximum spikes, the average attenuation for traces with spikes larger and smaller than the size of the cache are 1.26 and 370.06 respectively.

The existence of updates spikes can negatively impact the possibility to mitigate the duplicates. We measured the presence of spikes among all observed sessions in 2014. For this purpose, we consider there is a spike in a trace when more than 65k distinct prefixes at the origin of future duplicates are transferred in less than one hour. According to this definition, 11.73% of the traces displayed large spikes of duplicates.

# 4. CONCLUSION

Redundant consecutive BGP announcements consume unnecessary bandwidth and CPU in routers. In addition, these messages delay the propagation of useful routing information. We observed that BGP sessions exhibit different behaviors. For some session the number of duplicates is low. But other sessions can exhibit a very high ratio of duplicates. We identified large spikes of duplicates in 11.73% of the sessions we observed in 2014. This may be a problem on chatty sessions.

We then identified three causes of duplicates: changes in attributes that are not propagated further, flapping of routes or attributes and, finally, incorrect implementations for sets in AS-Paths. We verified these causes by performing thorough controlled experiments.

To mitigate the problem we propose use of a cache to find the right trade-off between additional memory consumption and the reduction of duplicates. We show that the performance of a cache highly depends on the characteristics of the BGP trace, in addition to the eviction strategy. While a cache is suitable on some traces, it is not always the case. The current trend of pushing control functions outside the router, to devices that are not as limited memory-wise, opens the door to full Adj-RIB-Outs and thus enable to avoid using pretty hacks to get rid of BGP duplicates completely in the future.

# 5. REFERENCES


[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006.

[2] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–528, 1998.

[3] C. Pelsser, O. Maennel, P. Mohapatra, R. Bush, and K. Patel, "Route flap damping made usable," in *Passive and Active Measurement*, 2011, pp. 143–152.

[4] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *ACM SIGCOMM CCR*, vol. 30, no. 4, pp. 175–187, 2000.

[5] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," *ACM SIGCOMM CCR*, vol. 37, no. 2, pp. 5–16, 2007.

[6] J. H. Park, D. Jen, M. Lad, S. Amante, D. McPherson, and L. Zhang, "Investigating occurrence of duplicate updates in BGP announcements," in *Passive and Active Measurement*, 2010, pp. 11–20.

[7] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "BGP churn evolution: a perspective from the core," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 571–584, 2012.

[8] A. Elmokashfi and A. Dhamdhere, "Revisiting bgp churn growth," *ACM SIGCOMM CCR*, vol. 44, no. 1, pp. 5–12, Dec. 2013.

[9] S. Agarwal, C. Chuah, S. Bhattacharyya, and C. Diot, "Impact of BGP dynamics on router CPU utilization," in *Passive and Active Network Measurement*, 2004, pp. 278–288.

[10] "ExaBGP," http://github.com/Exa-Networks/exabgp, 2014.

[11] "EXOS," http://documentation.extremenetworks.com/exos_commands/EXOS_All/EXOS_Commands_All/r_disable-bgp-adjribout.shtml, 2015.

[12] P.-C. Cheng, B. Zhang, D. Massey, and L. Zhang, "Identifying BGP routing table transfers," *Computer Networks*, vol. 55, no. 3, pp. 636–649, 2011.