Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

Beyond the Best: Real-Time Non-Invasive Collection of BGP Messages

L. Cittadini[1], V. Mezzapesa[1], M. Papagni[1], M. Pizzonia[1], L. Vergantini[1] and S. Vissicchio[1]

RT-DIA-165-2010 **February 2010**

(1) Dipartimento di Informatica e Automazione,
Università di Roma Tre,
Rome, Italy.
{ratm,mezzapes,papagni,pizzonia,verganti,vissicch}@dia.uniroma3.it

This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, FIRB project "Advanced tracking system in intermodal freight transportation" and under Project "ALGODEEP: Sfide algoritmiche per elaborazioni data-intensive su piattaforme di calcolo emergenti", MIUR Programmi di ricerca di Rilevante Interesse Nazionale.

ABSTRACT

Interdomain routing in the Internet has a large impact on network traffic and related economic issues. For this reason, BGP monitoring attracts both academic and industrial research interest. The most common solution for collecting BGP routing data is to establish BGP peerings between border routers and a *route collector*.

The downside of this approach is that it only allows us to trace changes of routes selected as *best* by border routers: this drawback hinders a wide range of economically relevant analyses that need access to *all* BGP messages received by border routers. Examples of these analyses include, e.g., verifying compliance to agreements involving BGP behavior, assessing the impact of unavailability of one upstream provider or peer, and evaluating the quality of received routing paths.

In this paper, we propose a novel technique enabling fast, non-invasive and scalable collection of *all* BGP messages received by a border router. By selectively cloning BGP traffic and sending it to a remote monitor, we are able to collect BGP messages without establishing additional BGP peerings. Our technique does not require any new feature to be implemented by routers and we experimentally show that our approach incurs a negligible processing overhead at the BGP routers. Our prototype implementation is able to process and archive BGP messages in near real-time on commodity hardware.

1 Introduction

The Internet provides connectivity among a great number of *Internet Service Providers* (ISPs) that exchange routing information via the Border Gateway Protocol (BGP) [23]. For inter-domain traffic, BGP has the final say on routing decisions and BGP messages received from neighboring ISPs can have a dramatic impact on the service provided by an ISP.

BGP monitoring enables ISPs to perform business-critical activities like troubleshooting and anomaly detection [20, 24]. Recently, it has been shown that BGP data can be exploited for business/market intelligence [13], traffic engineering [1], root cause analysis [4, 11], oscillation detection [9, 12], routing table analysis [16] and agreement compliance verification [10].

Despite such a rich set of potential applications, current BGP monitoring practices are quite limited: very often, they employ open source BGP daemon implementations to establish extra BGP peerings with production routers. The daemon acts as a *route collector*, in the sense that it collects information received via those extra peerings, dumps it in some format, and stores it for future analysis. For example, this is the approach adopted by RouteViews [22] to collect BGP data for the Internet community. Such a practice has two major drawbacks: *(i)* it is only able to collect those routes that have been selected as best by the routers that peer with the collector; and *(ii)* it is only able to collect BGP messages after ingress policy application, which can modify the messages themselves.

Unfortunately, these drawbacks prevent exploiting the monitoring system for interesting applications like, for example, fine tuning of ingress policies (e.g., for traffic engineering purposes), verification of Service Level Agreements (SLAs) that involve BGP updates received by the other party or analysis of what-if scenarios (e.g., what if an upstream provider goes down?). Recently, the BGP Monitoring Protocol [25] was proposed to overcome those limitations, but unfortunately it is still largely experimental, requires firmware support on the routers, and it does not mandate real-time monitoring of BGP messages.

In this paper we define a set of requirements that a BGP monitoring system should satisfy so that an ISP can take the biggest possible advantage from its deployment. We present a novel approach enabling real-time, non-invasive and scalable collection of all BGP messages received by production-level BGP routers. For this purpose, we exploit a usually overlooked feature that allows a router to selectively clone IP packets and send them to a remote collector. We make use of such a feature to copy every incoming TCP segment belonging to BGP sessions and send it to a collector. After possibly reordering out-of-order segments, our collector parses the BGP messages and stores the information in the standard MRT format [2].

By means of experimental evaluation on one of the cheapest commercial routers targeted to ISPs, we show that deploying our solution negligibly affects the performance of border routers with respect to traffic forwarding throughput and CPU usage. We show that our prototype implementation can monitor hundreds of BGP routers on commodity hardware. We also check the accuracy of the collected data. Finally, by comparing our approach to existing solutions for BGP data collection, we show that none of the previously proposed monitoring systems is able to fulfill all the requirements we considered.

The rest of this paper is organized as follows. In Section 2, we describe some realistic scenarios in which the two main drawbacks of existing solutions for building a monitoring system prevent an ISP from using BGP data in business critical analyses. In Section 3, we define the requirements we mandate for an ideal BGP monitoring system. Section 4 surveys existing BGP monitoring solutions. In Section 5, we describe our proposal for a BGP monitoring system, outlining its architecture and discussing the most relevant components. Then, based on the requirements defined in Section 3, we evaluate our solution (Section 6) and we compare it with existing solutions (Section 7). We conclude in Section 8.

2 Motivating Scenarios

In this section, we show some sample scenarios in which monitoring non-best routes and collecting BGP messages after ingress policy application enables ISPs to perform value-added activities which can not be performed otherwise.

2.1 What-If Analyses

Monitoring all the BGP messages received by border routers enables an ISP to perform business intelligence activities and better engineer its traffic.

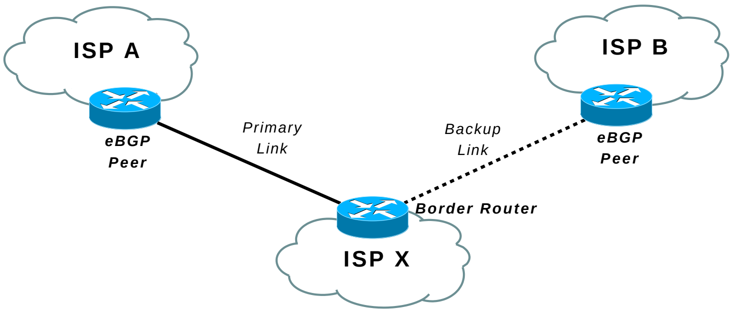


Figure 1: A simple network scenario wher collecting non-best routes can be very useful for business intelligence activities and detection of SLA violations.

Consider, for example, the scenario depicted in Figure 1. An ISP X configures one of its border routers to peer with two different upstream providers A and B. For economic reasons, X always prefers to send traffic to ISP A, and it is willing to use the peering with B only as backup. To respect this policy, the router is configured to assign a higher value of the `local-preference` attribute on the route announcements received from A with respect to those received from B.

In this setting, collecting non-best routes enables X to perform analyses of what-if scenarios, answering questions like the followings.

- What if the link with the upstream provider A goes down? Are all the Internet prefixes reachable through B?
- What happens to paying traffic if I set different values to some BGP attributes (e.g., local preference or MED) on routes announced by ISP A?
- What is the effective redundancy provided by the backup link? Does B provide different paths to destination prefixes with respect to those provided by A? Can a single failure on the Internet affect the capability of X to reach some destination using both links with A and B?

2.2 SLA and Quality Monitoring

Monitoring only best routes prevents the BGP monitoring system from detecting potential SLA violations and assessing the quality of the service offered by upstream providers.

Refer again to Figure 1. Assume that X has a SLA with both its providers stating that the full Internet routing table must be announced and available for 99.9% of the time. In this case, under normal circumstances, a BGP monitoring system which does not support the collection of non-best routes would only get information regarding provider A, and would be unable to verify B's SLA compliance.

Moreover, the collection and the analysis of non-best routes is necessary to monitor the quality (e.g., in terms of AS path length) of the routes announced by both A and B. Observe that quality monitoring can also trigger refined business intelligence activities, and help answering questions like "Is it convenient for X to keep the link with A as primary or it is better to send the traffic relative to some or all the prefixes to B?".

2.3 Historical Data

BGP messages should be collected as they are before border routers apply the configured BGP policies.

Consider the scenario in which an AS X decides to maintain historical BGP data for several months, in order to perform some analyses on the updates received from its providers over time. In this case, a single change in the policy applied by AS X on one of its border routers would invalidate the entire historical dataset, since data collected in the past would be no longer consistent with the new one. In particular, differences in the value of some attributes could be caused by the changed policy, by a change in the announcements sent by one of the providers of X or both.

Notice that, even if the routing policy locally applied by the collector ISP is known, the original BGP messages sent by neighboring ISPs can not be reconstructed in the most general case, since policies are not always reversible (e.g., if an attribute gets overwritten).

3 Requirements for a BGP Monitor

In this section, we describe a set of requirements that a BGP monitoring system should ideally fulfill.

Collection of non-best routes updates. Due to the way BGP is designed, BGP routers select a single *best* route among a set of feasible routes to a given destination, and forward traffic accordingly. Although updates related to routes not selected as best have no impact on where packets are forwarded, collecting them allows an ISP to perform business critical activities which it can not perform otherwise (see, for example, Section 2).

Accurate BGP update collection. Collected BGP data should reproduce the BGP updates received by other ISPs to the highest possible level of accuracy. This implies at least the ability to reconstruct all the BGP attributes that are involved in the best route selection process. We expect an ideal collection system to reconstruct the original BGP messages as sent by neighboring ISPs, that is, collected data should not be affected by locally configured routing policies in the collector ISP (as pointed out in Section 2). Peculiar applications, e.g. scientific research, might also need accurate reproduction of BGP update timings.

Real-time data collection. A BGP monitoring system should be able to collect data in real-time, or at least in near real-time. Namely, a BGP update should be available for applicative analysis within few seconds. This is a crucial requirement for troubleshooting and network diagnosing applications: network administrators want to know what is going on while it is going on, not hours later.

Low impact on router resources. A typical constraint on management systems is to have a small impact in terms of extra resource demand (e.g., CPU usage, throughput and bandwidth) on the network infrastructure. This is especially true for BGP monitoring, given that BGP border routers typically have to forward huge amounts of traffic.

Cost-efficient deployment. To be realistically deployable in large scale networks, the monitoring system should be able to handle hundreds of border routers employing a handful of machines equipped with commodity hardware.

Low management overhead. A monitoring system cannot require the existing network topology to be modified. Also, deployment and management overhead, e.g., the amount of extra configuration needed to set up and maintain the system, should be as low as possible.

4 Related Work

In this section we describe existing solutions for BGP monitoring. We refer to Section 7 for a comparative analysis of prior work which shows that no existing solution satisfies all the requirements defined in Section 3, thus motivating new contributions in this area.

Two naive approaches for BGP monitoring can be enabling debug option on router devices and installing optical taps and ad-hoc filtering boxes near each BR. However, debugging output stream “is assigned high priority in the CPU process; therefore, it can affect system performance and might render the system unusable” [7]. On the other hand, the usage of one optical tap and one filtering box for each BR does not scale very well and is not practical to be implemented.

The most adopted solutions can be broadly classified in two categories: those employing some kind of route collectors to which BGP messages are pushed by border routers, and those adopting separate protocols to pull BGP information from the routers.

The typical architecture of a BGP monitoring system belonging to the first category consists in a route collector that is deployed inside the network and keeps iBGP peerings with every border router. Quagga [17] and OpenBGPd [3] are probably the most famous and widespread tools to set up a route collector. Essentially, they simulate the behavior of real routers, but they also support dumping BGP messages in MRT [2] format. The Python Routing Toolkit (PyRT) [21], on the other hand, only implements a minimal set of features, and is more lightweight and scalable.

BGP monitoring systems based on separate management protocols are designed to pull information from the devices. The main advantage of such an approach is that it typically does not need any extra configuration at the border routers: support for the management protocol suffices. SNMP has a number

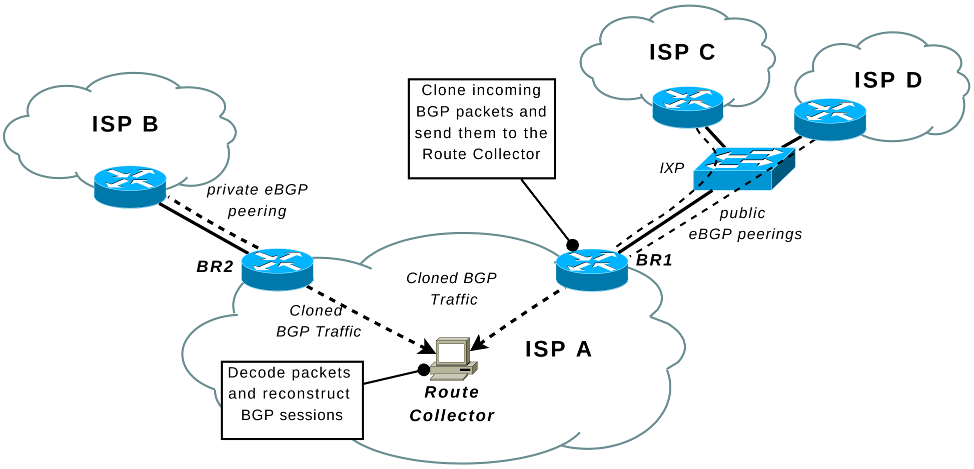


Figure 2: A deployment scenario of the BGP monitoring system proposed in the paper.

of MIB objects that are dedicated to BGP monitoring activities [15]. Often, operators pull information by *screen scraping*, i.e., using software that connects to the device, e.g., via Telnet or SSH, issues a specific command, e.g., `show ip bgp`, and collects the output.

Recently, a new ad-hoc protocol has been proposed in the IETF (the BGP Monitoring Protocol, or BMP) [25]. The basic idea is to send received BGP messages via a TCP connection with a monitoring station.

5 Proposed Architecture

In this section we propose an architecture for a BGP monitoring system that aims at satifying all the requirements listed in Section 3. The main idea is to mandate *border routers* to capture all the incoming TCP segments belonging to BGP sessions with eBGP peers and forward them to a remote *route collector*. The route collector is responsible for reassembling the TCP segments, decoding BGP messages and storing them in the standard MRT format [2]. In the following, we show that this technique can be implemented using a feature commonly available on routers together with ad-hoc software employed on the collector side.

Figure 2 depicts the architecture of our solution in a typical deployment scenario. In this example, ISP A configures its border routers $BR1$ and $BR2$ to clone BGP packets and send copies to a remote Route Collector. Since packet cloning is performed before applying local policies, the route collector will receive BGP messages exactly as they are sent by eBGP peers. This feature allows ISP A to monitor what routes are announced by its peers B, C, and D. Of course, this approach supports private peerings between ISPs as well as peerings at public Internet exchange points (IXPs).

It is easy to see in Figure 2 that our solution requires cooperation between two main architectural components: the border router (BR) and the route collector (RC). We now provide details regarding each component.

5.1 Border Routers: Cloning BGP Traffic

Most commercial routers provide the feature to clone IP packets and send copies to a remote machine. This is mostly used for copying traffic to Intrusion Detection Systems [8]. Leading vendors also provide filtering capabilities that allow operators to specify which packets must be cloned. Since such a feature is usually highly optimized, filters can usually be expressed based on IP source and destination addresses, and TCP source and destination ports only. However, for the purpose of cloning TCP segments belonging to BGP sessions, such simple filters suffice. To maintain a vendor-independent terminology, throughout the paper we will refer to this feature as *Selective Packet Cloning* (SPC).

A BR configured to perform SPC copies packets received from user-specified *source interfaces* and matching an optional *filter* that selects which packets should be cloned. Cloned traffic is forwarded via a *destination interface*.

RITE Configuration Steps

Step 1 - Define a *filter* to select BGP traffic

```
7201(config)#access-list 100 permit tcp
                              any any eq bgp
```

Step 2 - Define a *destination* interface

```
7201(config)#ip traffic-export
                              profile myProfile
7201(config-rite)#interface vlan1
7201(config-rite)#incoming access-list 100
                              mac-address <addr>
```

Step 3 - Select one or more *source* interfaces

```
7201(config)#interface ge0/0
7201(config-if)#ip traffic-export
                              apply myProfile
```

(a) Configuration of Cisco RITE

Port Mirroring Configuration Steps

Step 1 - Define a *filter* to select BGP traffic

```
M7i# edit firewall family inet filter myPr
M7i# set term mirror from port 179
M7i# set term mirror then port-mirror
M7i# set term mirror then accept
```

Step 2 - Define a *destination* interface

```
M7i# edit forwarding-options port-mirroring
M7i# set family inet input rate 1
M7i# set family inet output
             interface <vlan1> next-hop <addr>
```

Step 3 - Select one or more *source* interfaces

```
M7i# edit interfaces fe-1/3/1 unit 0
M7i# set family inet filter input myPr
```

(b) Configuration of Juniper Port Mirroring

Figure 3: Steps for configuring Selective Packet Cloning on Cisco and Juniper devices.

Depending on the capabilities of the device, a destination interface can be either a physical interface (e.g., an Ethernet interface), a VLAN interface (via 802.1q encapsulation), or even a tunnel interface (e.g., IP-in-IP encapsulation or Generic Routing Encapsulation). Observe that forwarding cloned packets to a physical destination interface forces the ISP to place the RC so that it is directly connected to the BR, and is therefore unpractical.

In the following, we briefly describe the SPC feature as implemented in Cisco and Juniper devices.

The cheapest Cisco devices targeted to ISPs (e.g., Cisco 7200 and 7300 routers) provide the *Router IP Traffic Export (RITE)* feature [8]. A RITE-enabled router can select packets received on certain interfaces applying IP- and TCP-based filters, and forward cloned packets over a VLAN interface. More expensive Cisco routers (i.e., 7600 series or greater) support the *Encapsulated Remote SPAN (ERSPAN)* feature [5], which provides a superset of the functionalities offered by RITE, e.g., the possibility to forward cloned traffic over a tunnel. Both RITE and ERSPAN can be used to implement the SPC feature on Cisco devices.

Juniper's SPC support is called *Port Mirroring* [6]. Traffic received via user-specified ingress interfaces is cloned and forwarded over a VLAN or a tunnel (IP-in-IP or GRE) interface. On M7i devices and greater, cloning on a VLAN interface is performed in hardware, while tunnel interfaces are handled in software unless ad-hoc hardware (i.e., Tunnel Services PIC [18]) is plugged into the router.

Depending on the SPC implementation, the TTL value might be decreased before cloning a packet. This is true both for Juniper's Port Mirroring and Cisco's RITE features. We cannot exclude that other SPC implementations behave the same. Unfortunately, this means that packets received with a TTL value equal to one cannot be cloned.

Since the default value of the TTL in eBGP is one, we must provide workarounds that make SPC usable within our context. For Cisco RITE, using a standard access control list, i.e., a filter that only matches fields in the IP header, solves the problem. Observe that this approach is reasonable because TCP traffic exchanged between border routers can be assumed to be mostly BGP traffic. Unfortunately, this workaround seems to be Cisco-specific. A more general workaround is to request peers to set up BGP session using a TTL greater than one. The recommended use of Generalized TTL Security Mechanism satisfies our needs since it forces the TTL to be set to the maximum value in order to protect BGP peerings from CPU-utilization based attacks [14].

From the management point of view, our approach requires a small amount of extra configuration on the SPC-enabled border routers. Figures 3a and 3b show the amount of extra configuration that is needed to enable the SPC feature on Cisco and Juniper routers, respectively. Steps 1 and 2 only need to be performed once, while Step 3 has to be repeated for each of the BR's interfaces that are used for BGP peerings.

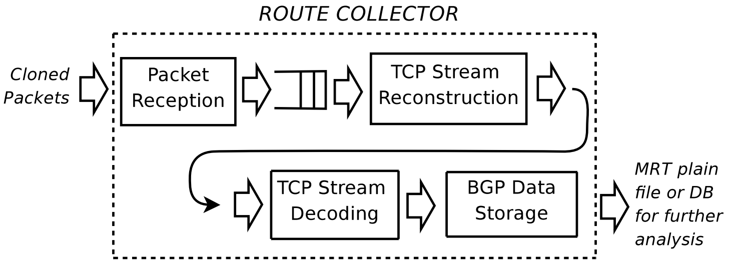


Figure 4: Main activities performed by the route collector software.

5.2 Route Collector: Receiving, Reconstructing, and Storing BGP messages

Cloned TCP segments are sent to the RC which is designed to provide BGP messages to applications for further analysis. The RC performs the following activities, as summarized in Fig. 4.

Packet reception. The RC receives cloned packets from one of its network interfaces, and buffers them for further elaboration.

TCP stream reconstruction. Since the RC does not establish a TCP session with the BR, cloned TCP segments might arrive out of sequence. Therefore, for each eBGP peering the RC needs to reorder packets in order to extract the TCP stream. Duplicated segments are discarded. To keep resource consumption at the BR as low as possible, the RC silently ignores lost cloned TCP segments, if any.

BGP message decoding. The reconstructed TCP stream is analyzed to decode BGP messages and infer BGP session state changes.

BGP message storing. BGP messages and inferred state changes are stored in the standard MRT format [2] or, optionally, inserted into a database ready to be analyzed by an application.

Our prototype RC implementation is based on the standard `tcpdump` utility for receiving cloned packets. We use `nice` to schedule the receiving process with high priority, and then send the received packets to a Perl script that is able to perform TCP stream reconstruction in pipeline. Finally, another Perl script takes the reconstructed stream in input and writes BGP messages in MRT format on a file.

6 Evaluation

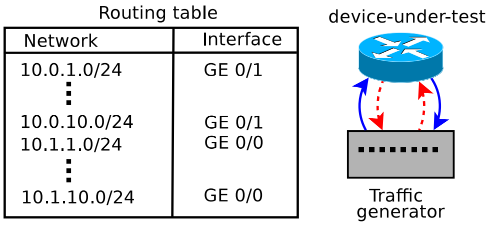
In this section, we evaluate the extent to which our proposal meets the requirements we defined in Section 3. In particular, we measured performance, accuracy and scalability of the proposed monitoring system.

In all the experiments we ran, we found that **no cloned packet was dropped** and **BGP messages were always correctly reconstructed and stored on disk**. Hence, we focus on the performance degradation at the BRs and on the scalability of the RC component.

6.1 Border Router Performance

We evaluate the router load in terms of frame loss (throughput), average CPU usage, and average packet latency. In our experiments, we used a Cisco 7201 router, referred to as *device-under-test* (*DUT*) in the following. The router is equipped with four Gigabit Ethernet ports, 1 Gigabyte of RAM, and a 1.67 GHz Motorola Freescale 7448 processor. The vendor's datasheet states that this router is able to route a maximum of 2 million packets per second. We chose the Cisco 7201 because it is considered one of the cheapest router targeted to ISPs.

Since the performance of the DUT is highly dependant on the total amount of traffic it has to route, we evaluate the impact of SPC features in different scenarios. In the following, we describe each scenario, discussing the results of our tests.


Routing table

Network	Interface
10.0.1.0/24	GE 0/1
⋮	
10.0.10.0/24	GE 0/1
10.1.1.0/24	GE 0/0
⋮	
10.1.10.0/24	GE 0/0

Figure 5: Baseline test topology.

6.1.1 Baseline Measurement

First of all, we measured the performance of the device without any special configuration. We use the results of this experiment as a baseline for evaluating the impact of enabling the SPC feature on the DUT.

Figure 5 illustrates the baseline test topology. Our traffic generator (a SmartBits 600B) only has two interfaces and we connected both of them to the router. Note that a unidirectional traffic flow on a full-duplex Gigabit Ethernet link can generate a maximum of $1,488,095$ packets per second [19], which would not be enough to measure the maximum throughput of the router. For this reason, we configured our traffic generator to send bidirectional traffic, that is, traffic was sent from interface 1 to interface 2 and viceversa at the same time.

To make the router work properly in this setting, we configured it with 20 static routes, 10 for each interface connected to the traffic generator. We programmed the traffic generator to generate 100 unidirectional IP flows (i.e., source-destination pairs) by randomly picking a source address in each of the 10 prefixes configured on interface ge0/0 and a destination address in each of the 10 prefixes configured on interface ge0/1. The same was done in the opposite direction (from ge0/1 to ge0/0), for a total of 200 simulated IP flows. Traffic was sent at a fixed packet rate, evenly distributed among all flows (i.e., each flow got 1/200 of the traffic). Each packet was 64 bytes long, the minimum size allowed on Ethernet.

We measured packet loss at different packet transmission rates. Results are summarized in Figure 7, where we also show the results presented in the next section for comparison. The x-axis represents packet rate, expressed as the percentage with respect to maximum packet rate for full duplex Gigabit Ethernet. The y-axis represents frame loss, expressed as the ratio between lost frames and sent frames.

In our setting, the Cisco 7201 router can handle circa $1,845,000$ packets per second (near 60% of the maximum packet rate) experiencing a negligible frame loss (less than 0.01%). The router was not able to handle the two million packets per second that the vendor's datasheet claims (vertical dashed line in Figure 7) without dropping frames. This is possibly a side effect of using only two interfaces or it might be due to our flows setting. Nevertheless, this fact does not affect the validity of this measure as a baseline for the following experiments.

6.1.2 Single Peering Scenario

After having performed the baseline measurement described in the previous section, we evaluated the router performance in a single BGP peering scenario. Namely, we set up a testbed using the topology depicted in Figure 6.

The DUT was connected to the traffic generator as in the baseline experiment. Also, the DUT was configured in the same way and the same 200 IP flows were sent by the traffic generator to the router. On a third interface of the router we set up a BGP peering with a medium sized ISP. From this BGP peering, the router received the full routing table, containing $310,000$ prefixes, and a continuous stream of real world BGP updates. We configured SPC such that incoming traffic belonging to the BGP peering was cloned on the fourth interface of the router over a VLAN. A packet sniffer was attached to the same VLAN and acted as a RC, capturing the cloned packets.

We performed the same experiment described in Section 6.1.1, the only difference being the size of the routing table, which, in this case, was increased by the full Internet routing table received over the BGP peering. We performed the test both with the SPC feature enabled (test "BGP-updates-mirror") and disabled (test "BGP-updates-no-mirror"). Results are presented in Figure 7. For convenience, we also report the baseline measurement results (test "baseline") on the same diagram. It is easy to see that activating the SPC feature essentially has no impact on the throughput achieved by the router. Moreover,

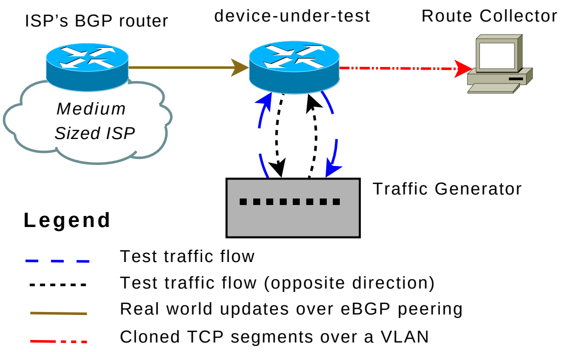

ISP's BGP router
device-under-test
Route Collector
Medium
Sized ISP
Traffic Generator
Legend
Test traffic flow
Test traffic flow (opposite direction)
Real world updates over eBGP peering
Cloned TCP segments over a VLAN

Figure 6: Topology in the single peering scenario.

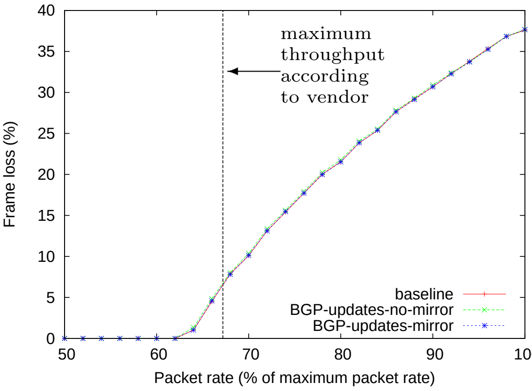


Figure 7: Frame loss versus packet rate in the single peering scenario.

we found that the presence of a single BGP peering does not cause more packets to be dropped. This can be explained by noting that, since the synthetic traffic is routed using static entries, the portion of the FIB that is accessed never changes, making BGP-induced FIB changes irrelevant to the test traffic.

6.1.3 Five Peerings Scenario

To verify the impact of enabling SPC in a more realistic situation, we performed more in-depth tests in the five peerings scenario.

Figure 8 shows the topology of this scenario. We interposed five BGP daemons (i.e., five Quagga [17] processes) between DUT and the ISP, in order to amplify the original stream five times. Each BGP daemon had a peering session with the ISP and one iBGP peering session with the DUT. This way, whenever a BGP update was sent by the ISP's BGP router, each BGP daemon sent an update to the DUT. The configuration of all the devices is analogous to that we used in the single peering scenario, the only difference being the exploitation of the BGP third-party next-hop mechanism instead of the static routes at the DUT.

In this scenario, we measured packet loss, average CPU usage and average latency when the router is solicited by the traffic generator with traffic at increasing packet rates. We ran tests both with SPC enabled and disabled and then compared the results. Since packet rates higher than 60% causes the router to drop non-negligible amounts of frames, we do not report results of the tests made for higher packet rates. We structured our tests in iterations of 5 minutes each, during which the SmartBits sends traffic to the DUT at the same packet rate. We chose this threshold estimating the interarrival time between BGP update bursts during daily hours of week days.

Figure 9 reports the results of our tests. The y-axis shows the difference (expressed as a percentage) between the performance of the router when SPC is enabled with respect to when it is disabled. The

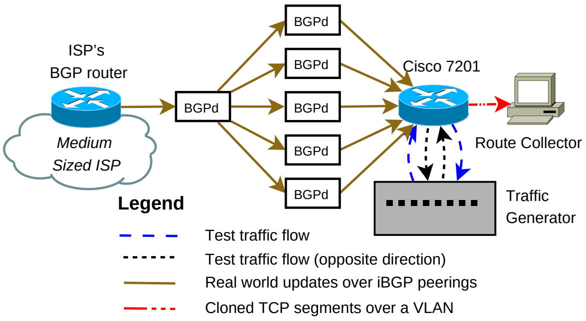


Figure 8: Topology in the five peerings scenario.

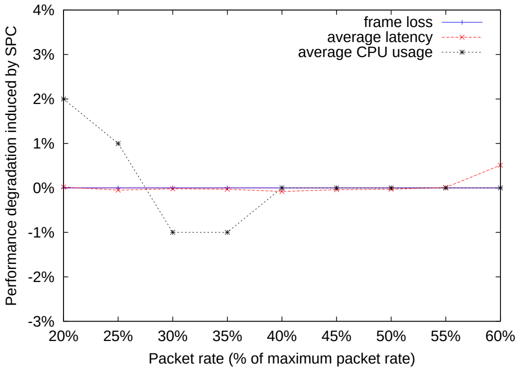


Figure 9: Performance degradation induced by SPC in the five peering scenario.

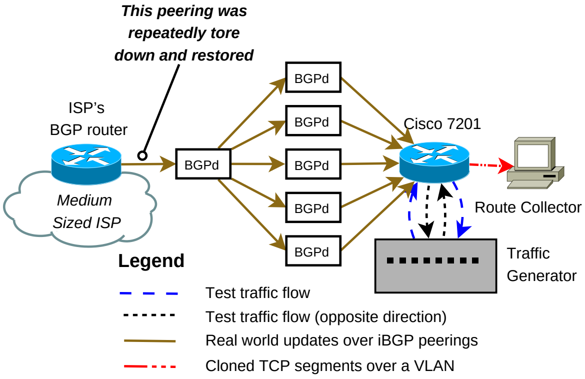

This peering was repeatedly tore down and restored
ISP's BGP router
Medium Sized ISP
BGPd
BGPd
BGPd
BGPd
BGPd
BGPd
Cisco 7201
Route Collector
Traffic Generator
Legend
Test traffic flow
Test traffic flow (opposite direction)
Real world updates over iBGP peerings
Cloned TCP segments over a VLAN

Figure 10: Test topology in the update bursts scenario.

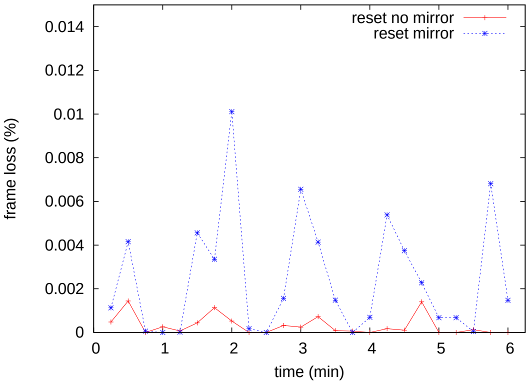


Figure 11: Packet loss in the update bursts scenario.

x-axis represents packet rate. It is easy to see that activating the SPC feature essentially has no impact on the frame loss and on the average latency. The worst latency we recorded was 375 μseconds with SPC enabled and 301 μseconds with SPC disabled.

Differences for CPU load are small and highly dependant on the presence of BGP bursts. Anyway, activating SPC never affected CPU load for more than 2%.

6.1.4 Update Bursts Scenario

We set up another experiment to evaluate how SPC affects the performance of the BRs under heavy BGP update bursts. The topology of the testbed is the same of the previous experiment (see Figure 10).

In this experiment, however, we tore down and restored the BGP session with the ISP's BGP router, at regular intervals of 1 minute each. This way, we were able to generate huge amounts of BGP updates. In fact, every time the BGP session was tore down (restored), the entire Internet full routing table was withdrawn (reannounced) by each of the five BGP daemons, and the DUT received almost 1.5 million route withdrawals (announcements). Moreover, we configured our traffic generator to send a considerable amount of traffic; namely the SmartBits generated traffic at the 45% of the maximum packet rate obtainable on a full-duplex Gigabit Ethernet.

We highlight that such a scenario is unrealistic. Indeed, notice that we stressed the DUT with traffic sent at the 75% of the maximum throughput that can be achieved by the device and routers of an ISP should not be (and typically are not) so overloaded by regular traffic. Moreover, real-world routers typically do not receive such huge amounts of BGP updates.

We run the experiment both with SPC disabled (test "reset no mirror") and enabled (test "reset mirror"). Figure 11 summarizes our results: the x-axis represents time, while the y-axis represents frame loss as measured by our traffic generator. We found that the DUT lost a very small fraction of traffic, about 0.001%, when working with SPC disabled, as shown by the red solid line in Figure 11. As predictable, packet loss spikes correspond to the reception of BGP update bursts. The spikes are higher when the SPC feature is activated on the router, but the performance of the router is affected to a small extent, as it is evident by observing that packet loss never exceeds 0.01%.

We also ran a 5 minutes experiment sending traffic at the 45% of the maximum packet rate, while repeatedly tearing down and bringing up the BGP peering with the ISP's router. We measured that, in these conditions, the router dropped less than 0.005% of packets even when SPC was enabled on it.

6.2 Performance of the Collector Software

From a theoretical point of view, scalability of the RC component should not be a problem. We now discuss the main factors that can affect the scalability of the collector software.

Receiving speed. To avoid dropping some TCP segments, the RC must be able to receive packets at the speed they are sent on the network. Note that cloned TCP segments are received by the RC at approximately the same time when the BR received the original segments, the only difference being the cloning delay introduced by the BR and the network latency from the BR to the RC.

The throughput of the TCP session between the BR and its BGP peer is limited by the TCP flow control mechanism, and it is roughly determined by the performance of the BGP software process running on the BR. The BGP software process, in turn, is bounded to the CPU speed of the BR. Given the current prices for commodity hardware, we can safely assume that the CPU speed of the RC exceeds, or is at least comparable with, the CPU speed of the BR. Moreover, the receiving process on RC just needs to buffer packets, a much less CPU-intensive task compared to what the BGP daemon on the BR needs to do. Hence, as long as the receiving process on the RC is scheduled with a sufficiently high priority, the receiving speed is not a problem.

Processing and storing speed. TCP stream reconstruction, BGP message decoding and data storage should be fast enough to sustain the average BGP traffic rate. Peak traffic rates are easy to accommodate by buffering received packets at the input of TCP stream reconstruction. All these activities take a constant amount of time for each BGP message, and the most critical with respect to processing time is the storage. A key feature of those three activities is that they are trivial to parallelize across multiple CPUs, allowing us to achieve good scalability by simply adding more processing resources to the RC. The possibility to improve write throughput of disks adopting RAID 0 is bounded only by the cost of additional disks.

We also perform some experimental tests to assess the amount of resources actually required on the RC side. During these tests, we separately measured the processing time needed for receiving the packets, reconstructing the TCP stream, decoding BGP messages and storing them in MRT format on commodity hardware (a laptop equipped with a dual-core 2.6 GHz CPU and 4G of RAM). We stress that summing the measures we obtained in this experiment provides an upper bound on the performance that can be achieved by a RC, since processing times can be greatly enhanced by enabling pipelining and parallel processing, as all the activities are trivial to parallelize across multiple processors.

We captured five BGP sessions during the initial full table transfer (nearly 1.5 million prefix updates, $37,157$ TCP segments, most of them of the maximum length). We re-played the capture file with `tcpreplay` using the topspeed option on a 100Mbit ethernet link connected to our prototypical RC. Actual throughput is about 80Mbit/sec, much higher than the throughput of regular BGP sessions. Replaying the capture file with `tcpreplay` took 3.38 seconds, while originally the BGP sessions lasted more than 2 minutes. A regular BGP session can reach such a high speed just sporadically. Even in this extreme experiment, we were able to capture all the packets with `tcpdump` and store them to an output file. TCP stream reconstruction from the output file took 2.6 seconds, while BGP session decoding and storage in MRT format took 1.7 seconds. Overall, a single prefix update was processed in less than 5.23 μseconds on average. Given that real world BGP sessions exhibit an average of less than 100 prefix updates per second, our prototype implementation can handle hundreds of BRs on commodity hardware.

Clearly, multiple route collectors can be deployed on the same network. However, given our experimental results, we expect that even tier-1 ISPs will only need a handful of collectors.

7 Comparison with Related Work

Table 1 summarizes the main differences between our approach and existing solutions. In the following, we discuss them in more detail.

Collection of Non-Best Routes Since Quagga, OpenBGPd, and PyRT rely on an iBGP peering, updates for routes that the BR does not select as best routes will never be collected at the RC. Non-best routes can be collected by screen scraping (e.g., via `show ip bgp` queries), and there exist SNMP managed objects for every route received. BMP and the solution we present in this paper are currently the only way to continuously monitor non-best routes.

Accuracy of BGP Session Reconstruction Quagga, OpenBGPd, and PyRT can only monitor routes selected as best, and they are forced to collect BGP messages after ingress policy application. Polling-based mechanisms are restricted to periodic snapshots of the received routes, hence, they are not suitable to accurately reconstruct the BGP sessions. BMP provides a more accurate view of the BGP session, however multiple BGP updates could be collapsed into a single one, and the timings of the messages could be altered. Moreover, BMP does not collect BGP control messages such as keepalives. Our solution, instead, clones each packet belonging to the BGP session as soon

	Quagga OpenBGPd	**PyRT**	**SNMP Screen scraping**	**BMP**	**Our Approach**
collection of non-best routes	no	no	yes	yes	yes
accuracy of BGP session reconstruction	bad	bad	bad	good	perfect
real-time data collection	no	no	no	almost	yes
impact on router resources	very low	very low	heavy	very low	very low
cost efficient deployment	no	yes	yes	yes	yes
management overhead	low	low	none	requires support from routers	low

Table 1: Comparison between our solution and related work with respect to the requirements defined in Section 3.

as it arrives to the BR, and also provides a very good approximation of the time when the BGP message was received.

Real-Time Data Collection Solutions that employ additional iBGP peerings, such as Quagga, OpenBGPd and PyRT, are, in principle, capable of collecting BGP messages in real time. However, if messages are dumped periodically, additional delay is introduced before data are available for an application to analyze. For example, Quagga can dump BGP data not faster than one file per minute. Real-time is of course unfeasible with SNMP and other polling-based mechanisms: their usage is restricted to periodic snapshots of BGP routes received by BRs. The current BMP specification asserts that BMP messages "are not real time replicated messages received from a peer" [25]. Section 6 shows that our approach can collect data in near real-time.

Low Impact on Router Resources Handling an iBGP peering is a lightweight task for a BR, hence solutions based on Quagga, OpenBGPd, or PyRT do not put stress on routers. On the other hand, polling-based solutions employing SNMP or screen scraping heavily affect the performance at the BR, since it must process the whole BGP table and send a snapshot to the monitor. Our experimental tests show that our approach affects the performance of the BR only minimally, see Section 6. We expect that also BMP has a low impact on router resources in most of the cases. See Section 7.1 for a more detailed comparison between BMP and our solution.

Cost Efficient Deployment Since Quagga and OpenBGPd emulate a real router, CPU cycles and memory are wasted at the route collector for activities that are useless to a BGP monitoring system, e.g., performing the best route selection process. This makes them unable to handle a large number of peers providing a full Internet routing table. PyRT is not affected by this problem since it only implements a minimal set of features, disregarding activities that are not relevant to the monitoring system. Since SNMP and screen scraping have no real-time constraint, a single monitor could be able to handle hundreds of BRs. The performance study in Section 6.2 ensures that our approach and, reasonably, also BMP can handle hundreds of BRs on a single RC.

Management Overhead Quagga, OpenBGPd and PyRT incur little management overhead, since all that is needed is to configure iBGP peerings between the BRs and the RC. SNMP and screen scraping virtually incur no management overhead, since they are commonly already used for other purposes in most ISP networks. The management overhead for our approach consists of extra router configuration as discussed in Section 5.1, plus the setup of a VLAN or tunnel from the BR to the RC. Deploying BMP, on the other hand, requires non-negligible firmware and/or hardware upgrade efforts: only JunOS versions later than 9.5 currently support BMP.

7.1 Comparison with BMP

Section 7 highlights that only our approach and BMP can reasonably be used in a monitoring system which aims at satisfying all the requirements listed in Section 3. However, BMP is not yet standardized and not yet widely supported, as already stated in this section.

The main technical difference between BMP and our approach is that BMP relies on TCP while our solution forwards packets from BRs to the RC over IP tunnels or VLAN. Our solution is based on enabling SPC at the BRs and does not need any additional daemon to be run. SPC involves only switching capabilities (either software or hardware) which are usually highly optimized. On the contrary, BMP is not implementable using switching mechanisms, must rely on conventional TCP implementation, and usually requires an additional daemon.

Adopting TCP, BMP guarantees reliable delivery of copied BGP messages to the collector. However, it is not clear what the router resource consumption would be under extreme circumstances, e.g., when the RC tries to slow down the BR by shrinking the TCP congestion window. Our proposal does not mandate the router to maintain any state. Observe that RCs can easily check whether some TCP segments are missing by analyzing sequence numbers of cloned traffic.

Finally, our solution is easy to extend to monitor other protocols than just BGP. It suffices to tune the SPC configuration at the BR and to enhance the software at the RC. On the contrary, extending BMP requires modifying the router firmware.

8 Conclusions

We envision that enhanced BGP monitoring techniques can allow ISPs to make better high-level economic decisions about peerings and commercial agreements. Within this context, ISPs could offer and buy new connectivity services with Service Level Agreements that involve BGP updates.

To support the above scenario, as well as better troubleshooting and other business intelligence analysis, we propose an innovative technique for real-time collection of all BGP messages sent by BGP peers.

Through experiments, we show that our approach accurately records the BGP updates received, it is easy to configure on current routers, it is scalable, and it has a negligible impact on the performance of the monitored border routers.

We believe that our approach based on selective packet cloning could turn out to be useful also for monitoring other signalling protocols like OSPF or LDP.

References

[1] Simon Balon and Guy Leduc. Combined intra- and inter-domain traffic engineering using hot-potato aware link weights optimization. In *Proc. SIGMETRICS*, 2008.

[2] L. Blunk, M. Karir, and C. Labovitz. MRT routing information export format. Internet-Draft, draft-ietf-grow-mrt-10.txt, 2009.

[3] H. Brauer and C. Jeker. OpenBGPd. `www.openbgpd.org`.

[4] Alessio Campisano, Luca Cittadini, Giuseppe Di Battista, Tiziana Refice, and Claudio Sasso. Tracking back the root cause of a path change in interdomain routing. In *Proc. NOMS*, 2008.

[5] Configuring local span, remote span (rspan), and encapsulated rspan (erspan). Cisco Systems, Inc. Official Cisco ERSPAN documentation.

[6] Configuring port mirroring. Juniper Networks, Inc. Official Juniper Port Mirroring Documentation.

[7] Using debug commands on cisco ios xr software. Cisco Systems, Inc. Official Cisco IOS XR documentation.

[8] Router ip traffic export packet capture enhancements. Cisco Systems, Inc., 2006. Official Cisco RITE documentation.

[9] Luca Cittadini, Massimo Rimondini, Matteo Corea, and Giuseppe Di Battista. On the feasibility of static analysis for BGP convergence. In *Proc. IM*, 2009.

[10] Nick Feamster, Zhuoqing Morley Mao, and Jennifer Rexford. BorderGuard: detecting cold potatoes from peers. In *Proc. IMC*, 2004.

[11] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet Routing Instabilities. In *Proc. SIGCOMM*, 2004.

[12] Ashley Flavel, Matthew Roughan, Nigel Bean, and Aman Shaikh. Where's Waldo? Practical Searches for Stability in iBGP. In *Proc. ICNP*, 2008.

[13] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6), 2001.

[14] V. Gill, J. Heasley, D. Meyer, P. Savola, and C. Pignataro. The generalized ttl security mechanism (GTSM). RFC 5082, 2007.

[15] J. Haas and S. Hares. Definitions of managed objects for BGP-4. RFC 4273, 2006.

[16] Geoff Huston. Analyzing the internet's BGP routing table. *The Internet Protocol Journal*, 4(1), 2001.

[17] K. Ishiguro, et al. Quagga routing suite. `www.quagga.net`.

[18] Inc. Juniper Networks. Tunnel service pic datasheet. Datasheet.

[19] Scott Karlin and Larry Peterson. Maximum packet rates for full-duplex ethernet. Technical Report TR64502, Department of Computer Science Princeton University, 2002.

[20] Jianning Mai, Lihua Yuan, and Chen-Nee Chuah. Detecting BGP anomalies with wavelet. In *Proc. NOMS*, 2008.

[21] Richard Mortier. PyRT. `research.sprintlabs.com/pyrt/`.

[22] Oregon RouteViews Project. `www.routeviews.org`.

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

[24] Matthew Roughan, Tim Griffin, Z. Morley Mao, Albert Greenberg, and Brian Freeman. IP forwarding anomalies and improving their detection using multiple data sources. In *Proc. SIGCOMM workshop on Network troubleshooting*, 2004.

[25] J. Scudder, R. Fernando, and S. Stuart. BGP monitoring protocol. Internet-Draft, draft-ietf-grow-bmp-02.txt, 2009.