Using BGP Flow-Spec for distributed micro-segmentation

February, 2020

Davide Pucci
M.Sc. Security and Network Engineering
University of Amsterdam
Amsterdam, The Netherlands
davide.pucci@os3.nl

***Abstract*—Data Centers need proper security constraints implementation, proper ways of hardening internal services, isolating multiple tenants and filtering hosted applications: this leads to the segmentation sphere, with particular attention to the micro-segmentation methodology. The advent large-scale applications led to a huge shift of traffic from outside Data Centers, the so-called *north-south* traffic, to within them, hence called *east-west*. Such change imposed a rethinking of the traditional way to build Data Centers. Among such changes, one of the most successful has been the adoption of Border Gateway Protocol (BGP) as internal routing protocol. Furthermore, at the end of the last decade, a new BGP Subsequent Address Family Identifier (SAFI) has been defined, the Flow Specification extension, with the purpose of propagating traffic filtering information, mainly as a countermeasure to Distributed Denial-of-Service (DDoS) attacks. This research shows how distribution of policies with regards to micro-segmentation in modern Data Centers can be achieved with the adoption of BGP Flow Specification protocol.**

***Index Terms*—bgp, routing, data center, micro-segmentation**

I. Introduction

Border Gateway Protocol (BGP) is an *inter*-Autonomous System (AS) routing protocol, originally intended to work as an Exterior Gateway Protocol (EGP), first defined almost thirty years ago [1]. It makes use of Network Level Reachability Information (NLRI) to exchange data between BGP routers, the so-called Speakers. The NLRIs are used to build a virtual graph that coherently apply to the physical architecture, that links Speakers with each others. The EGP relies on this mechanism to both drop loops and enforce possible policies. BGP runs over Transmission Control Protocol (TCP), which basically removes the need to implement reliability mechanisms, as already guaranteed by the transport level, such as fragmentation, re-transmission, acknowledgement and sequencing.

If BGP was originally — and still is — meant to regulate the highest hierarchy layer of routing in the Internet, in the last half a decade the situation has evolved in a specific direction. Given the relatively simple architecture of the routing protocol and its hierarchical distribution in two sub-layers, Internal BGP (iBGP) and External BGP (eBGP), it has been proven to be a good protocol not only for its original purpose of routing the Internet. More specifically, the use of reserved AS numbers [2], along with the adoption of iBGP capabilities like AS Confederations [3] and Route Reflection [4], all brought the routing protocol, to be able to scale better and more elastically. Also, last decade has been very crucial in the sense that it reported a huge growth of third-wave applications, which heavily rely on distributed and large infrastructures. Differently from the first two waves, which were both monolithic single-machine applications, third-wave ones brought a massive increase of traffic within a single Data Center. This highly affected the conception and implementation of modern Data Centers, which needed to handle higher amounts of internal traffic, to better scale to support up to hundred thousands servers in single physical location, to constraint the "blast radius" of a failure to limit its impact on the whole infrastructure and to support multi-tenancy with flawless and automated deployments and tear-downs of virtual networks. To meet these requirements, BGP revealed to be a good option to handle routing inside a single and private Data Center, as described in Request for Comments (RFC) 7938 [5]. With virtualization, the way to build elastic and extensible Data Centers has been massively revised, due to the need to hold large-scale systems. Such environments present specific network requirements, with emphasis on security, operational simplicity and network stability, and BGP has been reported to be usable as the only routing protocol, doing its job stably, according to RFC 7938.

Furthermore, BGP has been designed to allow extensibility, thanks to the use of combination of specific Address Family Identifier (AFI), Subsequent Address Family Identifier (SAFI) and specific NLRI encoding formats. Relying on this approach, BGP Flow Specification [6] protocol has been defined with the intent to isolate Distributed Denial-of-Service (DDoS) attacks at a BGP level. With its specific NLRI format, it aims to associate specific actions, e.g., reduce traffic rate (down to discard it, possibly), mark it, redirect it or a combination of each one, to specific traffic flows, identified by a collection of filters, starting from the simple destination and/or source prefix, ending with more complex settings like fragmentation status or TCP flags inspection.

II. Problem

BGP is worth the use in Data Center for internal routing, but it is not necessarily applicable, nor capable, to cover all the roles needed in such environments: choices like virtualization solutions, hardware capabilities, customer requirements, ..., are not obviously inherently depending on the routing protocol. But it is not necessarily true for security requirements and, more specifically, to the micro-segmentation sphere. In last decade, network traffic has been indirectly and gradually redistributing: originally, the greatest portion was *north-south*, i.e., traffic between servers and clients, between a Data Center and someone else outside it. With the rise of big and large-scale systems, the largest amount of traffic has been redistributing in a *east-west* variant, i.e., traffic inside a single Data Center, making the communication *intra*-Data Center much higher than the one between the client and the large-scale system. This gives the idea of the size of a single Data Center and how it is crucial to guarantee proper network isolation inside these environments: if an attacker is able to compromise and force an entry point to a node inside the Data Center and the hardening has not been well thought, the intruder would be free to move inside the Data Center, just thanks to a single hole in the infrastructure. Technologies like Virtual Local Area Network (VLAN), Virtual Routing and Forwarding (VRF) and roles like firewalls are of great help to accomplish such a purpose, but, as one of the most important advantages of the adoption of BGP as Data Centers local routing protocol is flexibility and simplicity in the management, delegate the control plane to the same protocol would be a profit to network operators, making the environment even simpler. Also, BGP Flow Specification seems to be the most suitable open source candidate to cover this kind of role. In order to accomplish this aim, though, it is critical to understand which are the requirements of Data Centers, in terms of networks and components isolation, and which are the bounds of BGP Flow Specification official specification, to check if the first ones fit into the latter. Being BGP Flow Specification a protocol invented to counteract DDoS attacks, influencing specific traffic flows, it could theoretically fit into the generic firewall scope. This research aims to deepen this topic, trying to give a definitive answer to the feasibility of the purpose.

III. Research Question

This section presents the main research question and sub-questions that follow from it.

> Is the BGP Flow Specification applicable for Data Center micro-segmentation?

This question can be divided into these sub-questions:

1) What are the requirements for Data Center micro-segmentation, with regards to the security aspect of component isolation?
2) Is the current specification of BGP Flow Specification capable to cover these requirements?
3) Is it feasible to build such a scenario with the actual open source implementations of the protocol?

IV. Literature

T. Arnold et al., in *Beating BGP is Harder than we Thought* [7], demonstrated how the relatively old age of BGP does not impact on its performance. Even if it is a protocol which has not been thought to be performance-oriented, but policy, instead, it is evidently hard to replace it to achieve better speed. This enables it to be considered thanks to its flexibility, without representing a trade-off with performance.

V. Giotsas et al., in *Inferring BGP Blackholing Activity in the Internet* [8], showed how BGP is, still more and more, adopted to directly apply traffic filtering using blackholing. One of the most important limitations of this approach, differently from the use of BGP Flow Specification, is that the prefix to be filtered out is limited to a certain maximum amount (typically, `/24`): this makes the filtering too wide, discarding legitimate traffic, e.g., when in the need to filter out specific `/32` prefixes. This research, though, shows the evidence of the need of a solution to apply filtering at a BGP level which has to be more flexible.

D. Bakker, in *Impact-based optimisation of BGP Flowspec rules for DDoS attack mitigation* [9], showed how BGP Flow Specification is indeed allowing that flexibility, whilst needing on the other hand a proper handling of rules specification, possibly reducing the amount of filters making coherent prefixes aggregation.

D. Thaler, in RFC 7288 [10], deepened the advantages of using host-based firewalls, instead of conventional network-based ones. Host-based firewalls represent the base idea behind a distributed firewall, as shown by S. Ioannidis et al. in *Implementing a Distributed Firewall* [11]: a central firewall is still used to generate rules and they get spread into the network accordingly and coherently. The advantage is that each host is aware of the rules it needs to apply for what is concerned to itself, without the need to make traffic always flow through the central firewall to be possibly filtered and/or manipulated in any way.

To conclude, there is no specific research on whether is possible and how to join the idea behind distributed firewalls and the approach used by BGP Flow Specification, with regards to Data Centers micro-segmentation.

V. Research Considerations

With regards to Data Centers segmentation, two main mechanisms naturally come up: level 2 and level 3 segmentation. Level 2 segmentation is virtual network isolation made relying on VLAN [12], while level 3 one on VRF. But when it comes to micro-segmentation, a further separation has to be done. This kind of isolation is within both the same VRF and VLAN: network architects want to assure that, if needed, even nodes within the same segment cannot speak to each other. One of the leaders in this sphere is VMware NSX [13], which allows to achieve Zero-Trust security by defining and enforcing policies consistently on a single management pane. With Zero-Trust security, every single "visitor", regardless of whether it is part of the internal network, is treated as foreign entity. Therefore, it has to prove to be capable and authorised for each specific request is doing towards a service. The added value of the micro-segmentation is that it comes along with the attempt of moving policies application as down and close to the servers as possible.

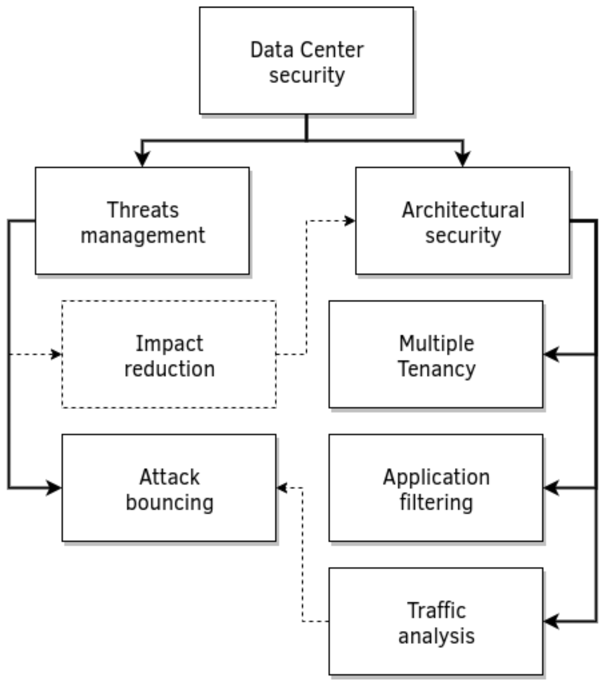


Fig. 1. Data Center security can be divided into two main groups, *threats management* and *architectural security*. The first one is mainly wrapping all the security requirements oriented to neutralise an inbound attack. The latter contains all the requirements needed to provide multiple-tenancy, application filtering and traffic analysis.

It is important as well to understand what kind of policies get usually applied in such contexts. On that regard, fig. 1 shows an overall distinction of rules that are generally applied. The two main spheres into which policies can be split are the architectural security and the threats management.

For what is concerning the latter, it wraps all the requirements which are defined *ad-hoc* to mitigate an in-going attack: the network operators have to guarantee a doable and quick way to make the threat be bounced and isolated as fast as possible. Isolating an attack means blocking any option that the attacker has to make any lateral movement in the network. Hence, this point highly relies on the architectural setup of the network. On the other hand, bouncing an attack means forcing it to stop, injecting rules that make the attacker be kicked out from the attack surface itself.

Speaking of architectural security, coherent segmentation is needed to make the network components separation. With micro-segmentation, the network architects can achieve a fine-grained level of security constraints. These constraints can be further split into three groups:

1) *Multiple-tenancy*. Network is shared between multiple parties (e.g., internal departments, or external customers) where the services should not at all or only communicate limited. This kind of isolation is typically applied using different VRFs.
2) *Application filtering*. Even within a single tenant, multiple application could be hosted and exposed. On such level, it often happens that filtering is made on a application-role basis, e.g., the front-end can reach the back-end, but not the storage, while the back-end can reach the storage.
3) *Traffic analysis*. Part of this group are all the counter-measures or requirements, not strictly related to security, which are needed to manipulate or forward traffic for analysis purposes, e.g., to be confidently aware of whether an attack is in action.

With regards to the effective policies implementation, what all these requirements have in common is need of being able to filter traffic in or out. A common distinction that it is usually done at this regard, is to roll a default policy for which all the filters represent an exception. There can be either one of the following cases:

1) Drop by default.
   In this case, if no exception is given, traffic between two nodes is completely forbidden. This represents the option used in most cases.
2) Accept by default.
   In this case, instead, all the traffic which is not explicitly forbidden, is allowed.

What is usually related to the traffic analysis capability, is to make traffic be routed to some other nodes for further investigation.

Furthermore, what is very likely present in a Data Center is the role of load-balancers: in this case, traffic redirection is done to achieve high availability or better performance of a specific application, making the load be distributed over several nodes. Anyway, in order to let load-balancers do their job good, they usually need Deep Packet Inspection (DPI) capabilities. Though this is a very common feature, as of the size of this topic, it needs a proper and standalone research, to look deeply at what is needed to be extended to enable such functionalities, and it has been thus excluded by this research.

A. Using BGP Flow Specification

Based on BGP Flow Specification RFC 5575 [6], the protocol relies on well-known extended community values to act in a certain way against the traffic flow. The first value defined, the `traffic-rate` extended community (type `0x8006`), is composed by a 2-bytes AS number and a 4-bytes float. The latter is used apply an upper-bound to the traffic rate. Setting this value to a certain amount would shape the traffic. On the other side, setting it to zero would result to a complete traffic drop, which would comply to the disabling traffic requirement.

The BGP Flow Specification also defines the `redirect` and the `traffic-marking` extended communities (respectively, type `0x8008` and `0x8009`). Using these fields coherently, would enable the traffic to be correctly forwarded for possible traffic analysis purposes.

Theoretically, then, it is feasible to translate normal and usual firewall rules in a BGP Flow Specification fashion. On the other hand, though, rules ordering is a very critical topic. Flow Specification RFC dedicates a whole section to this point. In fact, in section 5.1, *Order of Traffic Filtering Rules*, it can be read the following:

> For IP prefix values (IP destination and source prefix) precedence is given to the lowest IP value of the common prefix length; if the common prefix is equal, then the most specific prefix has precedence.

For what is concerning firewalls, though, the ordering is usually manually imposed using sequence numbers: if a specific rule has a sequence number which is lower than another, then it will be processed before the latter. To impose such a scheme to BGP Flow Specification, then, an additional information has to be carried along with the route. All the information in the protocol are handled using well-known extended communities: the only ones allocated specifically for the protocol are the already introduced `traffic-rate`, `traffic-marking` and `redirect`, and also the unmentioned `traffic-action`, which is used either to regulate the evaluation of multiple and subsequent rules or the sampling and logging of the traffic. There is no other option than to allocate a new extended community for that purpose. Extended communities are of the size of 64 bits, with the first 16 bits dedicated to the type label. This means that the remaining 48 bits could be used to carry the sequence number associated to a specific rule. Hence, this scheme, relying on BGP, would enable a maximum amount of $2^{48} = 2.8147498 \times 10^{14}$ rules.

Even with such a high threshold, it is undesirable to make all the rules spread on the whole network nodes, especially if a given rule does not apply at all to a specific node. For this purpose, an additional extended communities comes to help. In fact, the RFC 4360 [14] not only specifies the extended variant of communities for the first time, but also the first well-known ones: the Route Target (RT) and Route Origin (RO). The first one perfectly applies to the introduced Data Center use-case, as it states:

> The Route Target Community identifies one or more routers that may receive a set of routes (that carry this Community) carried by BGP. This is transitive across the Autonomous System boundary.

Looking further on the subsequent RFC 4364 [15], the role of this extended community is more deepened:

> Every VRF is associated with one or more Route Target (RT) attributes.
> [...]
> A Route Target attribute can be thought of as identifying a set of sites. (Though it would be more precise to think of it as identifying a set of VRFs.) Associating a particular Route Target attribute with a route allows that route to be placed in the VRFs that are used for routing traffic that is received from the corresponding sites.

Hence, if a given rule only applies to specific segments, the RT extended community is the most suitable and well-known option to reduce the rule recipients range to those VRFs. This is highly recommended to make the Data Center even higher scalable. On the other hand, though, this could be not sufficient: moving the firewall to a host-based variant would be, instead. But this specific focus is beyond the scope of this research.

VI. Experiments

In order to realise the proof of concept relying on an open source implementation of the BGP protocol, which shipped with the Flow Specification protocol, the range of possibilities is not wide.

One of the oldest open source alternatives is Quagga [16]. It is a routing software suite, which provides the implementation of not only BGP, but also of Open Shortest Path First (OSPF) and Routing Information Protocol (RIP). It has started as a fork of GNU Zebra [17], which was developed by Kunihiro Ishiguro. The drawback with Quagga is that it is no more actively mantained, nor shipping with the Flow Specification implementation and has been therefore discarded. Also, OpenBGPD [18], which is a BGP-only router, has been discarded as well, as only implementing a limited set of RFC related to the BGP protocol, among which there is no the Flow Specification one. On the other hand, ExaBGP [19] correctly implements BGP as well as the Flow Specification protocol, but it does not manipulate the local Forwarding Information Base (FIB).

These constraints led the choice to a combination of Bird [20] and Free Range Routing (FRR) [21]. The first one was originally developed as a school project at Faculty of Math and Physics, Charles University Prague, and during last years added support for several routing protocols, as well as BGP and Flow Specification. The latter is born as a fork of Quagga and aims to seamlessly integrate with the native Linux/Unix IP networking stacks. Anyway, both share a specular drawback: they lack of full Flow Specification protocol implementation.

In fact, the Flow Specification protocol works relying on two distinct roles: the controller and the client. The controller is assigned of spawning of the Flow Specification routes: it configures them and transmits them to its peers. The latter receives the Flow Specification routes from the controller and possibly forwards them to its peers. The reason behind choosing both Bird and FRR, then, is that the first one only implements the Flow Specification for the controller use-case, while the second for the client one.

As starting from version 2, Bird added support for Flow Specification protocol, version 2.0.7 has been picked. FRR 7.0.1 is running on the other switches.

Furthermore, none of the two software suites is able to inject Flow Specification routes entries into the underlying system. This lack led the research to cover also the realisation of the code needed to make this sphere working, which relies on rules application over iptables, the utility program used to configure the Linux kernel firewall.

A. Protocol additions

As discussed in the previous section, the only capability BGP Flow Specification is missing for the purpose aiming to

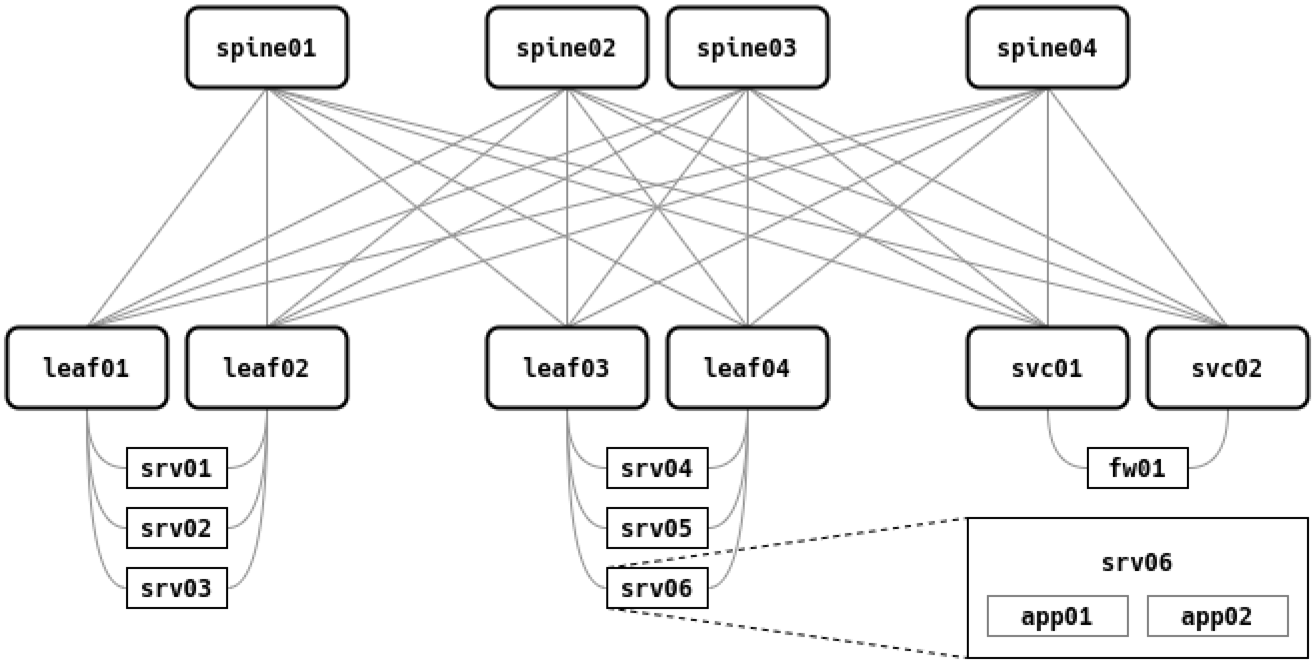


Fig. 2. The *Clos* network topology adopted to bring on the proof of concept, composed of four main spine switches, three pairs of leaf switches and nine servers. The BGP Flow Specification controller is configured in the server `fw01`, while all the other switches are configured as clients.

be achieved is a way to carry, along with the rule, the sequence number associated to it. Given the nature of a proof of concept, the complexity of extending the protocol specification to add such information and the inactivity, regarding the specific use-case, of already specified extended communities, the `traffic-marker` one has been chosen to carry that information.

Furthermore, it is needed to represent in some way a rule which defined the default policy: to do that, the route, regardless of the sequence number associated to it, for which filters point to source and destination prefixes of 0.0.0.0/0 is considered the default.

B. Network topology

Fig. 2 represents the virtual network that has been built to realise the proof of concept. Its topology shape is commonly known as a *Clos* network [22] and it is in a simple two-tier variant: switches at the top are called *spine* nodes, while the lower ones are *leaf* nodes. The spines connect the leaves with one another, whereas the leaves define how servers get connected to the network. In this kind of network, in which spines are working as connectors, the functionality and the capabilities are pushed out to the edges rather than pulled into the spines. This is why this scaling model is called *scale-out* [23].

The specific network used to perform the experiment is composed by four spine and six leaf switches. Last pair of leaves, `svc01` and `svc02`, has been connected to the server `fw01`, which works as a Flow Specification controller. All the other switches are supposed to work as Flow Specification clients.

It is worth to mention that every switch node in the network is not a layer-2 switch, but a Cumulus Linux [24] 4.0.0 machine, while all the servers are Ubuntu [25] 18.04 machines. Exception made for `fw01` machine, all the other servers are supposed to work as hypervisors which host several virtual machines, each with at least one dedicated IP.

C. Flow Specification configuration

First operation has been configuring Bird on `fw01`, to make it act as a BGP speaker and establish a session with its neighbours, the leaves `svc01` and `svc02`.

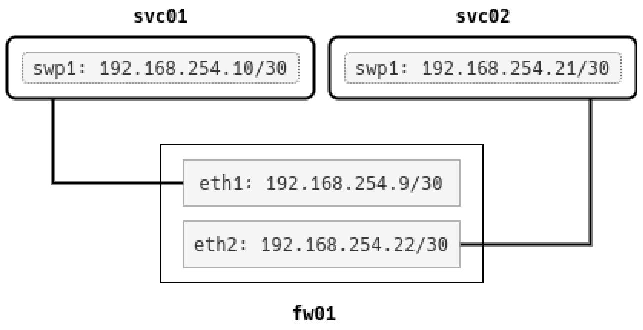


Fig. 3. Leaves `svc01` and `svc02` are connected to server `fw01` relying on their `swp1` interfaces. Each interface is given a `/30` IP.

Fig. 3 shows how node `fw01` is connected to leaves `svc01` and `svc02`, which have, respectively, AS numbers 65105 and 65106. The Bird server has been configured with the following parameters to correctly establish a BGP session with the switches:

```
template bgp bgp_fs {
   local 192.168.254.9 as 65309;
   ipv4 {
      import all;
      export all;
   };
}
protocol bgp bgp_svc01 from bgp_fs {
   neighbor 192.168.254.10 as 65105;
}
protocol bgp bgp_svc02 from bgp_fs {
   neighbor 192.168.254.21 as 65106;
}
```

On the other end, both the other FRR nodes needed to allow the new neighbour: this has been done using FRR shell, vtysh, with the instructions below (which apply to `svc01`).

```
svc01# configure terminal
svc01(conf)# router bgp 65105
svc01(conf-rtr)# neighbor swp1 interface
     peer-group underlay
```

Using this configuration, Bird was capable of exchanging routes with the switches and constructing a routing table based on the network BGP graph.

Next step has been enabling the session to exchange Flow Specification routes. The configuration, controller-side, needed to incorporate in `bgp_fs` template configuration the `flow4` protocol is the following:

```
flow4 table flowtab4;
protocol static {
    flow4 {
      import all;
      export all;
    };
    # sample rule
    route flow4 {
        src 254.254.254.1/32;
        dst 254.254.254.254/32;
    } {
        bgp_ext_community.add( # traffic-rate: 0.0
            (generic, 0x80060000, 0x0)
        );
    };
}
template bgp bgp_fs {
    [...]
    flow4 {
        table flowtab4;
        import all;
        export all;
    };
}
```

FRR nodes, on the other hand, needed activate Flow Specification SAFI:

```
svc01# configure terminal
svc01(conf)# router bgp 65105
svc01(conf-rtr)# address-family ipv4 flowspec
svc01(conf-rtr-af)# do show ip bgp neighbor swp1
[...]
For address family: IPv4 Flowspec
 underlay peer-group member
 Update group 18, subgroup 20
 Packet Queue length 0
 Community attribute sent to this neighbor(all)
 0 accepted prefixes
```

As clearly visible, Flow Specification SAFI was enabled but prefixes were neither received or applied. Looking further at the reason behind that discrepancy, FRR log showed the following:

```
FS Rx Update IPv4 to 254.254.254.254/32 from
     254.254.254.1/32 EC{FS:rate 0.000000}
%NOTIFICATION: rcvd End-of-RIB for IPv4 Flowspec
     from swp1 in vrf default
%NOTIFICATION: received from neighbor swp1 3/1
     (UPDATE Message Error/Malformed Attribute
     List) 0 bytes
```

Apparently, as a response to the service switches try to relay Flow Specification routes, the controller was giving back a *Malformed Attribute List* error. So, the UPDATE message sent by the controller was valid, as correctly parsed by the FRR receivers and the problem came up when receivers needed to redistribute the same routes, where they got corrupted. The reason behind this bug was a manual alteration of Flow Specification NLRI length, which caused a mismatch between the NLRI length calculation and its effective one. Working together with Cumulus Networks engineers, a new contribution has been done at that regard, to properly fix FRR Flow Specification routes redistribution [26].

In fact, patching the FRR instance installed on the switches, fixed the issue and now peers were able to both establish a Flow Specification enabled BGP session and relay routes:

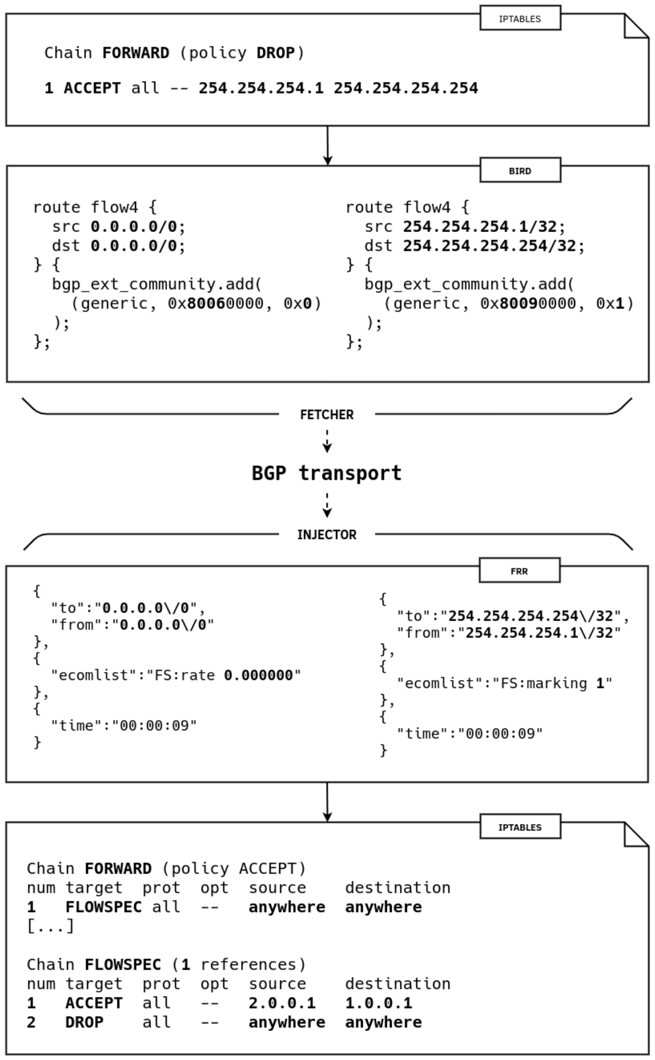
```
svc01# show bgp ipv4 flowspec
[...]
Network Next Hop    Metric  LocPrf  Weight  Path
*>  to 254.254.254.254/32 from 254.254.254.1/32 0
     65309 i

Displayed  1 routes and 1 total paths
svc01# show bgp ipv4 flowspec detail json
[
  {
    "to":"254.254.254.254\/32",
    "from":"254.254.254.1\/32"
  },
  {
    "ecomlist":"FS:rate 0.000000"
  },
  {
    "time":"00:20:28"
  }
]
```

Spawning the same instructions to enable Flow Specification SAFI for all peers sessions, all the switches were able to receive the routes:

```
leaf01# show bgp ipv4 flowspec detail json
[
  {
    "to":"254.254.254.254\/32",
    "from":"254.254.254.1\/32"
  },
  {
    "ecomlist":"FS:rate 0.000000"
  },
  {
    "time":"00:22:33"
  }
]
```

D. Flow Specification injection

As already noted, the injection of Flow Specification rules over the underlying system has been done using iptables. It is worth the mention, then, that this choice not only has been possible because of the switches not being effectively layer-2 switches, but Linux machines. It also brought an important additional simplification. In fact, when applying a rule on iptables, the utility itself is covering also all the possible cases of traffic that is inherently related to the original rule, e.g., adding a rule to accept traffic from a specific IP prefix to another one for TCP protocol connections, against a specific port, will automatically enable also the traffic coming from that port to the active socket that has been generated to open up the aforementioned connection. The fig. 4 shows a perspective of the whole proof of concept flow.



```
Chain FORWARD (policy DROP)

1 ACCEPT all -- 254.254.254.1 254.254.254.254
```

```
route flow4 {                          route flow4 {
  src 0.0.0.0/0;                         src 254.254.254.1/32;
  dst 0.0.0.0/0;                         dst 254.254.254.254/32;
} {                                    } {
  bgp_ext_community.add(                 bgp_ext_community.add(
    (generic, 0x80060000, 0x0)             (generic, 0x80090000, 0x1)
  );                                     );
};                                     };
```

```
{                                      {
  "to":"0.0.0.0\/0",                     "to":"254.254.254.254\/32",
  "from":"0.0.0.0\/0"                    "from":"254.254.254.1\/32"
},                                     },
{                                      {
  "ecomlist":"FS:rate 0.000000"          "ecomlist":"FS:marking 1"
},                                     },
{                                      {
  "time":"00:00:09"                      "time":"00:00:09"
}                                      }
```

```
Chain FORWARD (policy ACCEPT)
num target   prot  opt  source    destination
1   FLOWSPEC all   --   anywhere  anywhere
[...]

Chain FLOWSPEC (1 references)
num target   prot  opt  source    destination
1   ACCEPT   all   --   2.0.0.1   1.0.0.1
2   DROP     all   --   anywhere  anywhere
```

Fig. 4. From the top, the fetcher iterates over the iptables FORWARD chain, parsing each entry as a Flow Specification route. It produces a Bird-supported configuration. The routes get distributed from Bird — the Flow Specification controller — to FRR — the client — relying on the BGP protocol. The injector collects the routes from the FRR vtysh shell and inject them into a iptables dedicated FLOWSPEC chain.

The injector utility has been thought to work in a dual direction: the first one is effectively used to inject Flow Specification routes as iptables rules to the underlying system. The latter, to generate Bird-compatible `flow4` table configurations starting from an iptables instance. From now on, the first role will be referred as injector, while the latter as fetcher.

The fetcher iterates over the iptables FORWARD chain and parses each entry as a `flowspec.Entry` object:

```
type Entry struct {
     // source prefix
    Src *net.IPNet
     // source port
    SrcPort *[]int64
     // destination prefix
    Dst *net.IPNet
     // destination port
    DstPort *[]int64
     // IP protocol
    Proto *[]int64
     // ICMP type
    ICMPType *[]int64
     // ICMP code
    ICMPCode *[]int64
    Params *struct {
         // line number
       SeqID *int64
         // traffic-rate
       Rate *float64
       // unsupported actions
       Unknown map[string]string
    }
    Time time.Duration
}
```

The parsing procedure works by mapping iptables rules options to the `flowspec.Entry` object fields, which is aiming to represent a Flow Specification route. While it is obvious that most of fields map easily to each others, few exceptions are made:

1) the default chain policy is parsed as an entry without a sequence number, the `SeqID` field, as it will be always positioned as last rule by the injector
2) the default chain policy is translated as an entry with `Src` and `Dst` fields set to 0.0.0.0/0
3) if the policy is set to DENY, the entry will have the `Params.Rate` field set to 0.0

Once proper `flowspec.Entry` object instances are generated for each rule of the FORWARD chain, the fetcher generates a Bird-supported configuration file shipping with each of the parsed routes.

The injector carries the BGP Flow Specification routes, that have landed into the FRR nodes as BGP UPDATE packets, to the underlying system, using iptables, still.

As already shown, the FRR vtysh shell offers a way to display the BGP Flow Specification routes in the JavaScript Object Notation (JSON) format. The injector iterates over these JSON entries to construct the corresponding `flowspec.Entry` objects. Relying on these instances, then, it applies the rules in a dedicated iptables chain, called FLOWSPEC, which is referenced as first "jump" instruction on the FORWARD chain. This approach has been adopted to not let the injector interfere with other system protocols that could have already filled the FORWARD chain for their own purposes. The FLOWSPEC chain contains all the rules corresponding to the BGP Flow Specification received routes, ordered by their sequence number.

VII. Conclusions

The proof of concept demonstrated how the theoretical idea can be actually implemented relying on the bare BGP Flow Specification capabilities. Other than that, few clarification are to be done.

First, the proof of concept relied on iptables to inject the rules to the underlying system. As already pointed out, this choice brings a massive simplification, but also the need of carrying rules numbering. In fact, classical switches operating systems, on the other hand, do not use iptables to represent Access Control List (ACL)s, because all switches are level-2 or level-3, nowadays. Even though a proper application of the idea in that case should be covered by a dedicated research use-case, rules ordering remains an issue. As BGP Flow Specification defines a specific way of ordering rules, either the underlying system should be following the same ordering convention, or a proper and formal extended community sub-type should be defined for ordering purposes, such as the sequence numbers used in the proof of concept.

Also, the research showed how the open source landscape is not ready at all to offer proper implementation of the protocols involved in the topic, mainly due to BGP Flow Specification shortcomings. Other than that, the implementation of rules injection is completely missing.

A. Further research

First, this research could not deepen into what kind of additions would be actually needed to extend the role of BGP Flow Specification to cover also load-balancing capabilities, adopting the `redirect` and the `traffic-marking` extended community sub-types to let it be able to route traffic coherently.

Furthermore, as already mentioned, rules ordering remains a critical issue, that has been solved in a very specific way which was applicable to iptables. On the other hand, though, a formal way to handle rules ordering should be defined that applies to all the sub-systems to which the routes are to be injected.

To conclude, this research only shows how the distribution of rules can be limited a (set of) VRF. It would be definitely useful to deepen the possibilities available to reach an even further limit, reducing the distribution on a per-host basis, instead of a per-VRF one.

VIII. Acknowledgement

The role of Attilla de Groot, the research supervisor, has been of great help. In his person, a very good practical point of view has always been found, always accompanied frankness, both for encouraging or straighten the aim.

Finally, a very sincere thank you goes to Paul Dunn, without which the whole research would not have been possible: being colleagues does not necessarily mean being friends, but being both can only help, both in a professional and human perspective.

References

[1] Lougheed, K. and Rekhter, Y. *A Border Gateway Protocol (BGP)*. RFC 1105. RFC Editor, June 1989, pp. 1–17. URL: https://www.rfc-editor.org/rfc/rfc1105.txt.

[2] Hawkinson, J. and Bates, T. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. RFC 1930. RFC Editor, Mar. 1996, pp. 1–10. URL: https://www.rfc-editor.org/rfc/rfc1930.txt.

[3] Traina, P. *Autonomous System Confederations for BGP*. RFC 1965. RFC Editor, June 1996, pp. 1–7. URL: https://www.rfc-editor.org/rfc/rfc1965.txt.

[4] Bates, T. and Chandra, R. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*. RFC 1966. RFC Editor, June 1996, pp. 1–12. URL: https://www.rfc-editor.org/rfc/rfc1966.txt.

[5] Lapukhov, P., Premji, A., and Mitchell, J. *Use of BGP for Routing in Large-Scale Data Centers*. RFC 7938. RFC Editor, Aug. 2016, pp. 1–35. URL: https://www.rfc-editor.org/rfc/rfc7938.txt.

[6] Marques, P. et al. *Dissemination of Flow Specification Rules*. RFC 5575. RFC Editor, Aug. 2009, pp. 1–22. URL: https://www.rfc-editor.org/rfc/rfc5575.txt.

[7] Arnold, T. et al. “Beating BGP is Harder than we Thought”. In: Nov. 2019, pp. 9–16. ISBN: 978-1-4503-7020-2. DOI: 10.1145/3365609.3365865.

[8] Giotsas, V. et al. “Inferring BGP blackholing activity in the internet”. In: Nov. 2017, pp. 1–14. DOI: 10.1145/3131365.3131379.

[9] Bakker, D. *Impact-based optimisation of BGP Flowspec rules for DDoS attack mitigation*. Mar. 2019. URL: http://essay.utwente.nl/77598/.

[10] Thaler, D. *Reflections on Host Firewalls*. RFC 7288. RFC Editor, June 2014, pp. 1–13. URL: https://www.rfc-editor.org/rfc/rfc7288.txt.

[11] Ioannidis, S. et al. “Implementing a Distributed Firewall”. In: *7th ACM Conference on Computer and Communications Security (CCS)* (Apr. 2001).

[12] Keen, H. “IEEE 802.1Q: Virtual Bridged Local Area Networks”. In: *IEEE Network* 14 (July 2000), pp. 3–3.

[13] *Micro-Segmentation with VMware NSX*. https://www.vmware.com/solutions/micro-segmentation.html. Accessed: 2020-01-30.

[14] Sangli, S., Tappan, D., and Rekhter, Y. *BGP Extended Communities Attribute*. RFC 4360. RFC Editor, Feb. 2006, pp. 1–12. URL: https://www.rfc-editor.org/rfc/rfc4360.txt.

[15] Rosen, E. and Rosen, E. *BGP/MPLS IP Virtual Private Networks (VPNs)*. RFC 4364. RFC Editor, Feb. 2006, pp. 1–47. URL: https://www.rfc-editor.org/rfc/rfc4364.txt.

[16] *Quagga*. https://www.quagga.net. Accessed: 2020-01-29.

[17] *GNU Zebra*. http://www.zebra.org. Accessed: 2020-01-29.

[18] *OpenBGPD*. http://www.openbgpd.org. Accessed: 2020-01-29.

[19] *ExaBGP*. https://github.com/Exa-Networks/exabgp. Accessed: 2020-01-29.

[20] *Bird*. https://bird.network.cz. Accessed: 2020-01-29.

[21] *FRRouting*. https://frrouting.org. Accessed: 2020-01-29.

[22] Clos, C. “A study of non-blocking switching networks”. In: *The Bell System Technical Journal* 32.2 (Mar. 1953), pp. 406–424. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[23] Vahdat, A. et al. “Scale-Out Networking in the Data Center”. In: *IEEE Micro* 30 (July 2010), pp. 29–41. DOI: 10.1109/MM.2010.72.

[24] *Cumulus Linux*. https://cumulusnetworks.com/products/cumulus-linux. Accessed: 2020-01-29.

[25] *Ubuntu*. https://ubuntu.com. Accessed: 2020-01-29.

[26] *Flowspec issue redistribute*. https://github.com/FRRouting/frr/pull/5717. Accessed: 2020-01-31.