Stealthy Attacks to the BGP Routing System

Project report

Jan Henke

27th February 2014

Department Computer Science
Faculty Engineering & Computer Science
Hamburg University of Applied Sciences

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

The Border Gateway Protocol (BGP) [1] provides the foundation for configuring and exchanging routing policies between providers and thus largely changed the Internet economics. Since BGP was first standardized about 20 years ago, the Internet has changed significantly. New business models and technological possibilities led to significant changes in the structure of the Internet. The increasing importance of the Internet for the economic development of the world lead to an increased awareness of the security weaknesses which were not prevented by the protocols design. Fixing the known weaknesses without altering the protocols definition too much has been challenging and no ultimate solution to be problem has been found yet.

BGP has no provisions for automated validation of incoming route updates. Whether an update is accepted or not is entirely delegated to the policy of the router. This policy is defined by hand and modified if needed.

Two common forms of attacks on BGP are **backholing**, which aims at denying network traffic for a certain autonomous system (AS), and **redirection**, which typically tries to gain access to the network traffic but afterwards delivers it to the original target, thereby allowing for intercepting specific information and executing man-in-the-middle attacks. Blackholing is the attack easier to accomplish. It does not require an unaffected route back to the target and the attacker does not need to have the routing capacity to handle all the affected traffic. It may simply discards packets. But blackholing is also much easier to detect since the attack causes a sudden drop in the traffic levels of the victim, thereby alarming the victim. These attacks typically aim at maximizing their impact.

To defend an AS from an ongoing attack and to estimate the threat potential for a given AS, it is vital to understand how attacks on the BGP propagate throughout the routing infrastructure of the Internet. But the rules describing the Internet's topology formation are changing over time. As a consequence our understanding of the propagation throughout the Internet has to be checked constantly and be adapted to structural changes.

1.1. Background on BGP

BGP is a path vector routing protocol announcing reachability information about IP prefixes between autonomous systems (AS). The concept of ASes is used to keep path information as simple as possible. "An AS is a connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy." [2]. Every AS has a globally unique number called autonomous system number (ASN). Every BGP update message contains the path of ASes this update has already traversed. Before forwarding a BGP update, the router must append its own ASN to the AS path of the update message. Routers are free to increase the length of the route by appending their own ASN several times. The first ASN in the path attribute is by definition the AS which owns the announced prefix, it is called the origin AS.

The behaviour of BGP is defined by the routing policy of the AS operator. Without knowledge of the policies the response to an attack of any AS can not reliably predicted.

Unfortunately routing policies are normally not released to public as they also contain information about the business model of the operator.

BGP in the default free zone has no single global view, but every AS has its own unique view on the global routing. Additionally, BGP speakers are free to aggregate routes for several prefixes to create less specific prefix before forwarding that route to its neighbours. For load balancing purposes, the opposite process can also occur.

BGP mandates use of four different sets for the processing of routes. Three routing information bases (RIBs) and the forward information base (FIB). Routes learned from other BGP speakers are initially inserted into the Adj-RIB-In, then filtered and prioritized according to the routing policy. If they pass the filter the routes are taken into the Loc-RIB. From the Loc-RIB one route for every prefix is chosen. If there are multiple routes with the same priority for the same prefix, a sequence of tie breaking rules is employed to reduce the amount of candidates to one. The first rule in this sequence considers the length of the AS path, this is often enough to have only one route for that prefix remaining. The route selected by the process is then put into the FIB, as well as taken over to the Adj-RIB-Out according to the routing policy. All routes in the Adj-RIB-Out are exported to other BGP speakers.

To forward an actual package, the router chooses the route from the FIB, whose prefix has the most bits matching with the target address of the packet (longest prefix match). That means the most specifc prefixes are always preferred over less specific prefixs.[1]

1.2. Outline

The next section describes in detail the proposed attack pattern. It starts with a definition of the prerequisites needed for a successful attack, followed by more detailed explanation of the attacks concept and an outline of possible motivations for doing such attacks. The section thereafter documents an implementation of this attack pattern in a lab scale network, using two different network topologies.

2. Stealthy BGP-level attacks

Over time a lot of different attacks in the default free zone (DFZ) of the Internet have been observed and documented. While simple but widely spreading attacks receive attention by the victims due to the changes in traffic levels they are causing, more sophisticated attacks are less easy to observe. Especially for the latter class of attacks it is vital to understand the concept behind the attack. Only with the knowledge of this concept, signs of an ongoing attack can be understood correctly and steps to mitigate the attack can be initiated.

This paper describes an attack pattern created by new combinations of known and exploitation strategies of the BGP. It is called "stealthy BGP-level attacks", because the most important goal of this pattern is the avoidance of its detection.

[1] Most ASes in the Internet do not accept prefixes smaller than a /24. Examples in this paper stick to the ip addresses reserved for documentation [3], therefore examples may contain prefixes more specific than a /24.

2.1. Prerequisites

The routing behaviour of any AS is defined by its policy. This policy is a consequence of the ASes business model [4, 5]. Thus inter-AS connections can be categorized by the type of business relation. Some of these are symmetrical, like peering connections, while others are asymmetrical, like customer to provider. Commonly the following categories are used. Each of it describes the outgoing view.

provider-to-customer connection The AS gets paid for the bandwidth use on this connection. Also called a downstream connection.

customer-to-provider connection The AS has to pay for the bandwidth usage on this connection. Also called an upstream connection.

peering connection These are normally settlement-free relations between two ASes. The agreement usually includes a limitation to route only traffic originating from the AS itself or one of the ASes customers.

sibling connection A special type involving two ASes that belong to the same operator. This type is not investigated further in this paper.

paid peering connection A relatively new business model. It is similar to a customer-provider-connection, just the "customer" gets paid for it instead of paying. Usually the part of the paying "provider" is taken by a content delivery network, like Google or Akamai.

transitive peering connection Similar to the normal peering connections, but with the addition that also routes learned from *peerings* of the peer are exported over this connection.

For this paper, the peering behaviour of ASes in the Internet is assumed to follow the following rules.

1. All ASes follow a similar economic model.
   a) Route updates coming from provider-to-customer connections are preferred over all other sources, since they ensure reachability for the customer to the rest of the Internet.
   b) Route updates coming from a peering connection are preferred over customer-to-provider connections, since traffic is free.
   c) Route updates coming from a customer-to-provider connection are chosen only if there is no known route via another type of connection.
   d) In case there is more than one route for any known prefix, the route with the shortest AS path is chosen.
2. Routes are exported from the FIB as follows.

a) Customers receive the full table content, as this ensures reachability.

b) Local routes and those learned from either customers or peering connections are exported to transitive peering connections.

c) Local routes and those learned from customers connections are exported to (non-transitive) peering connections.

d) Local routes and those learned from customers connections are exported to provider connections.

2.2. Concept of the attack pattern

This paper focuses on describing attacks on the routing system which aim for two goals. First the attacker wants to manipulate his victims routing table in such way that the victim routes all traffic destined for a set of prefixes though the AS controlled by the attacker. Secondly the attacker wants to keep the fact hidden that he is actually executing an attack.

An important factor in keeping an attack hidden is the limitation of the impact of the attack. A generic attack aimed at maximizing the amount of victims will most certainly be visible to many ISPs and on the different monitoring tools employed in the DFZ. An attacker therefore has to focus the attack on the individual victim to appear as legitimate as possible. Also it must be ensured that the malicious BGP updates are accepted into the FIB by the victim and they do not propagate to anybody else in the DFZ.

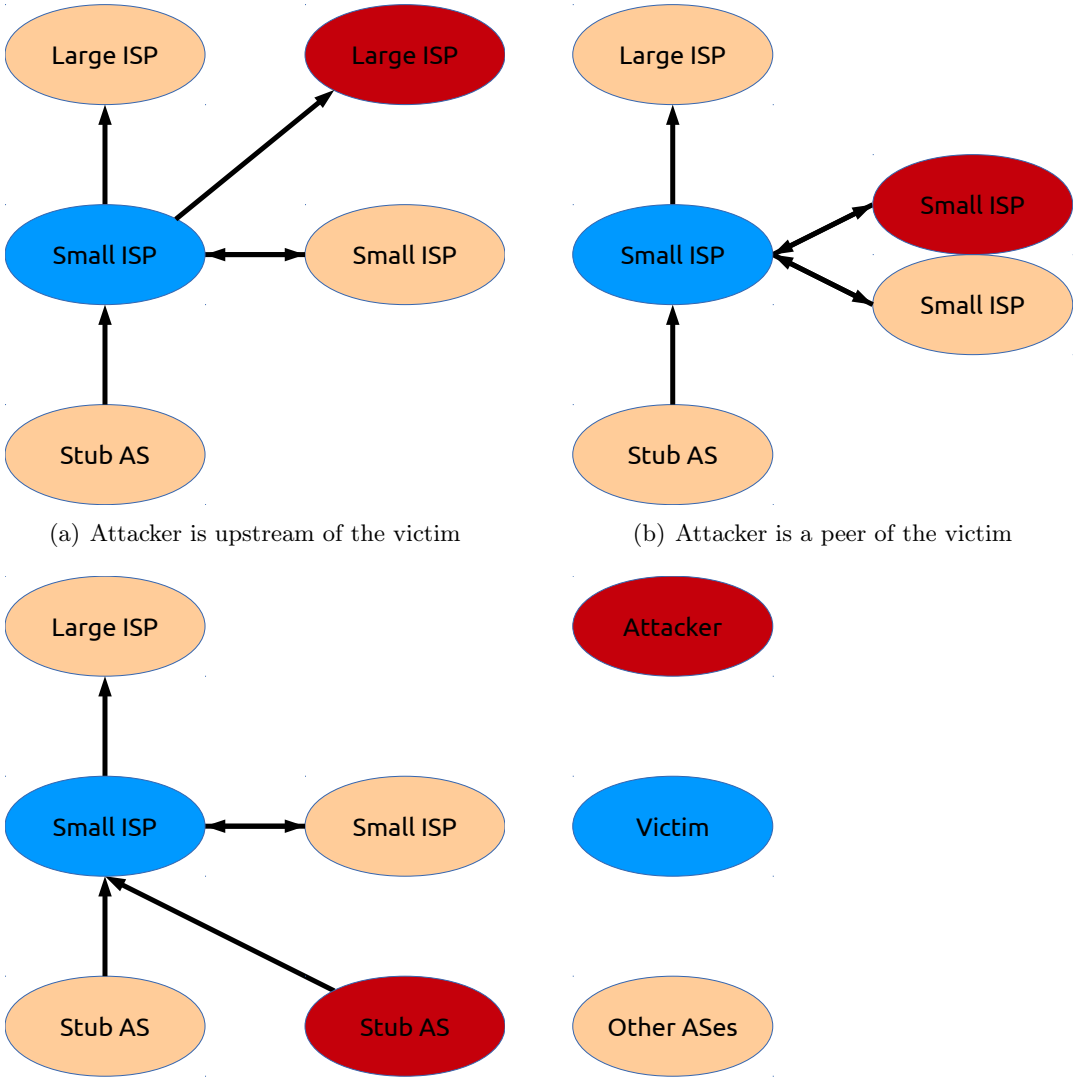
It becomes easier for the attacker if he is located as closely as possible towards the victim on the topological level. As soon as a spoofed update has to traverse a third party AS on the way to the victim it becomes visible to transit ASes. Also every AS applies an own policy and can therefore have potentially unwanted effects on the spoofed update the attacker distributes. This can be avoided by creating a direct BGP connection between the attacker and the victim. Today's widespread use of Internet-exchange-points (IXPs) makes it easy to create new inter AS connections to a large number of different ASes.

For any given prefix it is reasonable to assume that the victim learns several other regular routes in addition to the spoofed route the attacker announces. To archive the goal of making the victim accept the spoofed route into his FIB, the attacker has to take the victims routing policy into account. The exact policy is often not known, but the assumptions in section 2.1 provide an approximation to estimate over which type of topological link an attack is likely to succeed.

Three different scenarios have to be considered. In the first case, the attacker acts as an upstream provider of the victim (figure 1(a)). In the second case, the attacker is peering with the victim (figure 1(b)). In the last case, the attacker is a customer of the victim (figure 1(c)).

- If the attacker is upstream of the victim, the spoofed route is preferred against other upstream routes that have a longer AS path.

- If the attacker is on peer-to-peer-level with the victim, the spoofed route is preferred against upstream routes and other peering routes with a longer AS path.



(a) Attacker is upstream of the victim

(b) Attacker is a peer of the victim

(c) Attacker is downstream of the victim

(d) Legend of the colours used

Figure 1: Possible topologically positions of the attacker. [Customer $\rightarrow$ Provider, Peer $\leftrightarrow$ Peer]

- If the attacker is downstream of the victim, the spoofed route is preferred against upstream routes, peering routes and downstream routes with a longer AS path.

Once the attacker has gained an appropriate position in relation to the victim, the next step is the collection of data to craft a trustworthy spoofed routing update. For this purpose it can be useful to consult not only public sources like route reflectors in the Internet, but also becoming (non-AS) costumer of the victim and probing the routes currently used towards the prefix origin. Another point that has to be taken into account is the spreading of a route updates in the global routing. Assuming no better route is known, figures 2(a) to 2(d) illustrate the spread of a routing update in the routing topology. The set of ASes seeing such an update form roughly the shape of a cone. To limit the visibility of the attack, the AS path length of the spoofed update must be carefully chosen to prevent a spread of the malicious update past the intended target.

While making these preparations, the attacker should participate normally in the global routing and thereby build trust with peering partners. It also helps the attacker to seek more peering connections than the one used for the attack later on. By doing so, the attacker simulates the behaviour of a normal AS and makes himself known in the global routing community. Operators are more likely to allow a new route if the AS, from which they receive this route, is known to them.

Once all these preparations are completed the attacker can go forward and start with the attack. This is done by simply exporting the spoofed BGP route update towards the victim while maintaining normal route updates towards the rest of the BGP sessions. To reduce the visibility of the attack it could be advisable to export the spoofed update only for a limited amount of time, while still avoiding the route flap damping mechanisms [6].

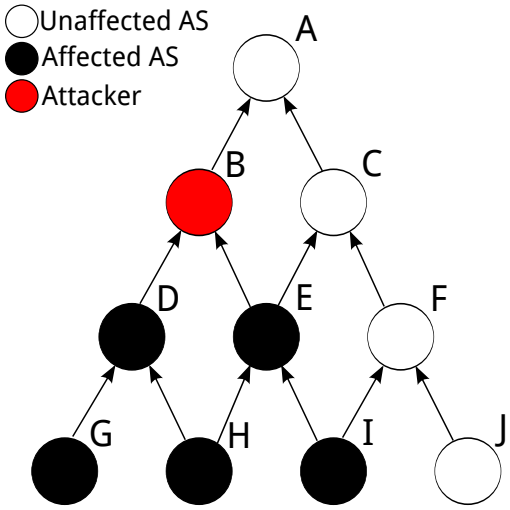
Another point of importance for any potential attacker is to ensure the route to the regular prefix owner is not affected by the attack [7]. Only then the attacker can act as a man-in-the-middle between the victim and the original prefix owner.

2.3. Use cases

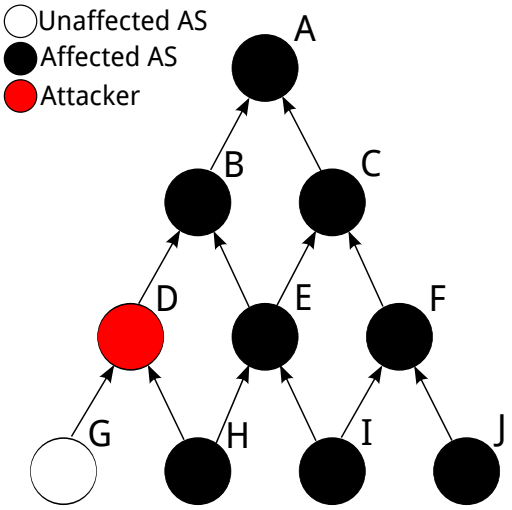
The motivation for any real world attacker to execute such an attack is essentially financial. Two possible use cases of such an attack are given here as examples to illustrate the possible impact.

2.3.1. Use case 1: Phishing account data

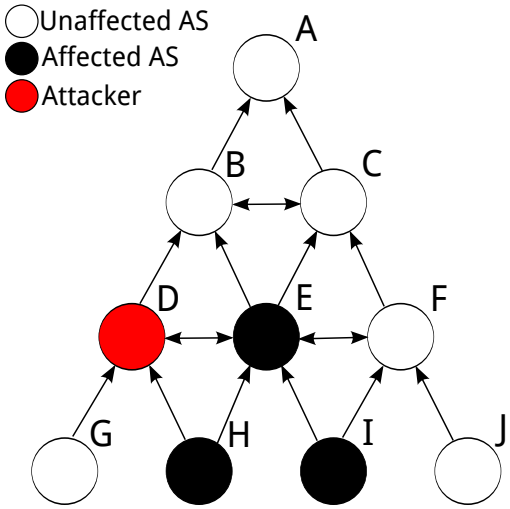
An attackers wishes to phish account details to execute malicious transactions. To prepare for this attack, he has to choose a website (e.g. an online banking website) he wants to impersonate and a target AS, for which typically an eyeball provider is preferred. The attacker could extend his attack to several websites at any time, in which case he has to execute all the steps for every website. The attacker then hosts a recreation of website on his own servers in his own AS. The copy is made to record any data entered by the victim before forwarding the modified data to the real website as if the data came directly from the victim. For this to work the attacker also needs a SSL/TLS certificate that will be accepted by the victim. How to gain access to this is out of scope of this



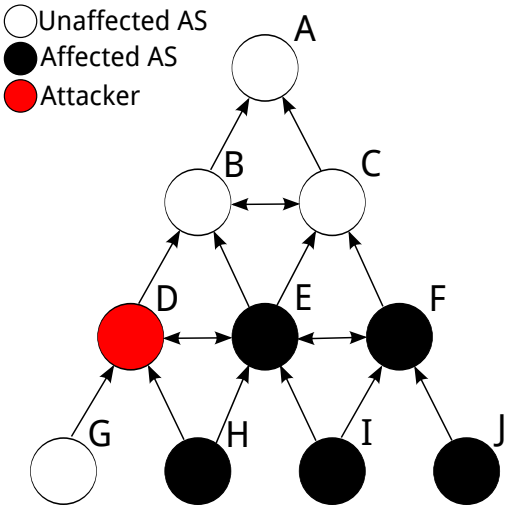
(a) Attacker annouces the malicous update downstream (without any peering connections involved).



(b) Attacker annouces the malicious update upstream (without any peering connections involved).



(c) Attacker annouces the malicious update to his peering partner (simple peering).



(d) Attecker annouces the malicious update to his peering partner (transitive perring).

Figure 2: Spreading of a malicious update in the topology, forming a cone of influence around the attacker. Assuming none of the ASes knows a better route.

paper, but recent history has shown this to be possible (e.g. DigiNotar hack [8]. To prevent a potential invalidation in the future the attacker should also provide a copy of the certification revocation list service. Once this infrastructure is ready the attacker hides it in his own AS making it accessible on demand under the IP addresses used by the real infrastructure.

As the next step the attacker has to get a peering connection to the victim AS. The attacker must be able to provide transit for prefixes without creating suspicion [7]. To facilitate this target the attacker could provide transit service relatively cheap. Once the victim is made to accept routes from the attacker, the attacker has to fake a trustworthy route for the prefixes of the online banking and certification revocation list infrastructure he is targeting. The fake route then leads to the shadow infrastructure hidden in the attackers AS. In order to make the faked route trustworthy the attacker could do some probing in the victims AS to learn the currently used routes from the victim to the real prefix owner.

Finally the attacker exports this faked routes towards the victim and all requests for the online banking infrastructure from the victims AS are from now on answered by the attacker.

2.3.2. Use case 2: Pretending copyright infringement

At the time of this writing this use case was speculative. Recent press releases indicate its application to the real world [9].

A company A cooperates with copyright holders to identify illegal sharing activity of their protected works in peer-to-peer networks. The company is in charge to collect evidence of the copyright infringement to be later used in demands for compensation. Unknown to the rest of the company the management of this company cooperates with a second company B, whose job is to generate the evidence of copyright infringements. These faked infringements can the be recorded and used as evidence by company A.

To hide the faked nature and make the evidence look genuine, company B employs the described attack pattern. This case requires less preparation by company B compared to scenario 1. Here the AS of company A is in the role of the "victim" and company B is the "attacker". The attacker has to chose one or more eyeball ASes to harvest potential customer addresses. The attacker then prepares an infrastructure inside his AS, which listens to addresses from the chosen prefixes. On this infrastructure the attacker prepares a large number of peer-to-peer clients. These clients will start to download and share files from a defined list, which company A created beforehand. The infrastructure is set up to mimic the behaviour of real end users, to avoid any suspicion. This includes random start and end times of activity by the individual clients and graceful termination of the peer-to-peer sessions.

The attacker then starts a peering agreement with the victim (company A) and exports routes for all the chosen prefixes which lead through the attackers AS. To make this believable the same techniques as in scenario 1 can be used.

At this point the attacker can start his infrastructure at any time making the victim believe clients in the eyeball networks would access the content.

3. Implementation

We have build a network to test our attack model. It consists of Bird Internet Routing Daemon (BIRD) instances and one hardware router (Brocade NetIron CER 2024C), each simulating a single AS and originating one IP-Prefix.[2] To record the behaviour and impact of the attack, I observe the routing table entries of the affected BIRD instances with the standard `birdc` command line tool.

Implementing the attack on a single BIRD node proved to be difficult, as the standard BIRD configuration only allows adding ASNs to the AS path and other modifications to it. To craft route update messages with arbitrary AS paths, we had to use an additional BIRD node as a pure generator for BGP update messages with arbitrary AS paths. This node is not a regular member of the network and only connected to the attackers primary BIRD node.

Besides the need for a BGP route generator I make some further changes to the BIRD configuration of the attacker. We create a new routing table, called `malicious`. A BIRD specific mechanism synchronizes the routes between this routing table and the main routing table. Only for the prefix that is going to be attacked a filter prevents the flow of routing updates from the malicious table back to the main table. A new BGP connection to the route generator is also created and connected to the malicious route table. Listing 1 shows the necessary additions to the BIRD configuration file. As the last step of the preparations, the attacker changes the routing table used for the connection to the future victim to malicious one. By enabling and disabling the connection with the generator the attacker can easily start and stop the attack at any time.

Listing 1: Configuration excerpt for the attacker node using BIRD.

```
1  table malicious;
2  protocol pipe {
3      peer table malicious;
4      export all;
5      import filter {
6              if net = <prefix to be attacked> then # Do not feed malicious route
                   back to main routing table
7                    reject;
8
9              accept;
10     };
11 }
12 protocol bgp RouteGenerator {
13     description "Connection which receives the crafted rotue updates from the
           generator.";
14     table malicious;
15     local as <attacker ASN>;
16     next hop self;
17     neighbor <generator IP> as <ASN of attacked prefix>;
18     import all;
```

[2]The prefixes announced in the network as well as the used ASNs are designated for private use [10, 2].

Table 1: Routers used in the experiment

Node	Router ID	ASN	IP-Prefix	BGP Implementation
1	141.22.28.121	65121	172.16.0.0/15	BIRD 1.3.10
2	141.22.28.122	65122	172.18.0.0/15	BIRD 1.3.10
3	141.22.28.199	65133	172.29.0.0/16	BIRD 1.3.11
4	141.22.28.25	65025	172.27.0.0/16	BIRD 1.3.10
5	141.22.28.124	65124	172.22.0.0/15	BIRD 1.3.10
6	176.28.11.244	65131	172.31.0.0/16	BIRD 1.3.11
7	141.22.27.145	65130	172.30.0.0/16	NetIron CER 2024C
8	141.22.28.123	65123	172.20.0.0/15	BIRD 1.3.10
9	62.75.143.240	65132	172.28.0.0/16	BIRD 1.3.11
10	141.22.28.125	65125	172.24.0.0/15	BIRD 1.3.10
11	141.22.28.126	65126	172.26.0.0/16	BIRD 1.3.10

```
19      export none;
20  }
```

The above pattern is used to simulate such attacks in two different network topologies. Demonstrating the correctness of the previously made assumptions. The first of the two topologies implements the model described by L. Gao and J. Rexford[4]. The second topology is based on the work of C. Labovitz et. al.[5]. It focuses on the concepts of Internet-Exchange-Points (IXPs) and Hypergiants (e.g. CDNs) and the associated changes to routing topology.

3.1. Tier based Topology

In this section all 11 nodes are assigned to act like one of the four tiers commonly found in the Internet. Figure 3 depicts the connections between the nodes. In particular there are no peering connections between nodes of different tiers. Communication is mainly done via upstream providers. The behaviour of the nodes is enforced by the import and export filters. Appendix A contains the used configuration files without any attack occurring.

Table 1 lists the details of the used routers. This is the starting point for several different attacks scenarios executed.

3.1.1. Small ISP attacking a stub network

This first scenario is chosen deliberately simple to verify that all configuration is indeed working correctly. Node 4 acts as the attacker, attacking node 2 by hijacking the prefix 172.30.0.0/16 normally originating from node 7.

Listing 2 shows node 2's view before the attack. Both upstream providers export one route each. The route via node 5 is chosen as it is three hops shorter.

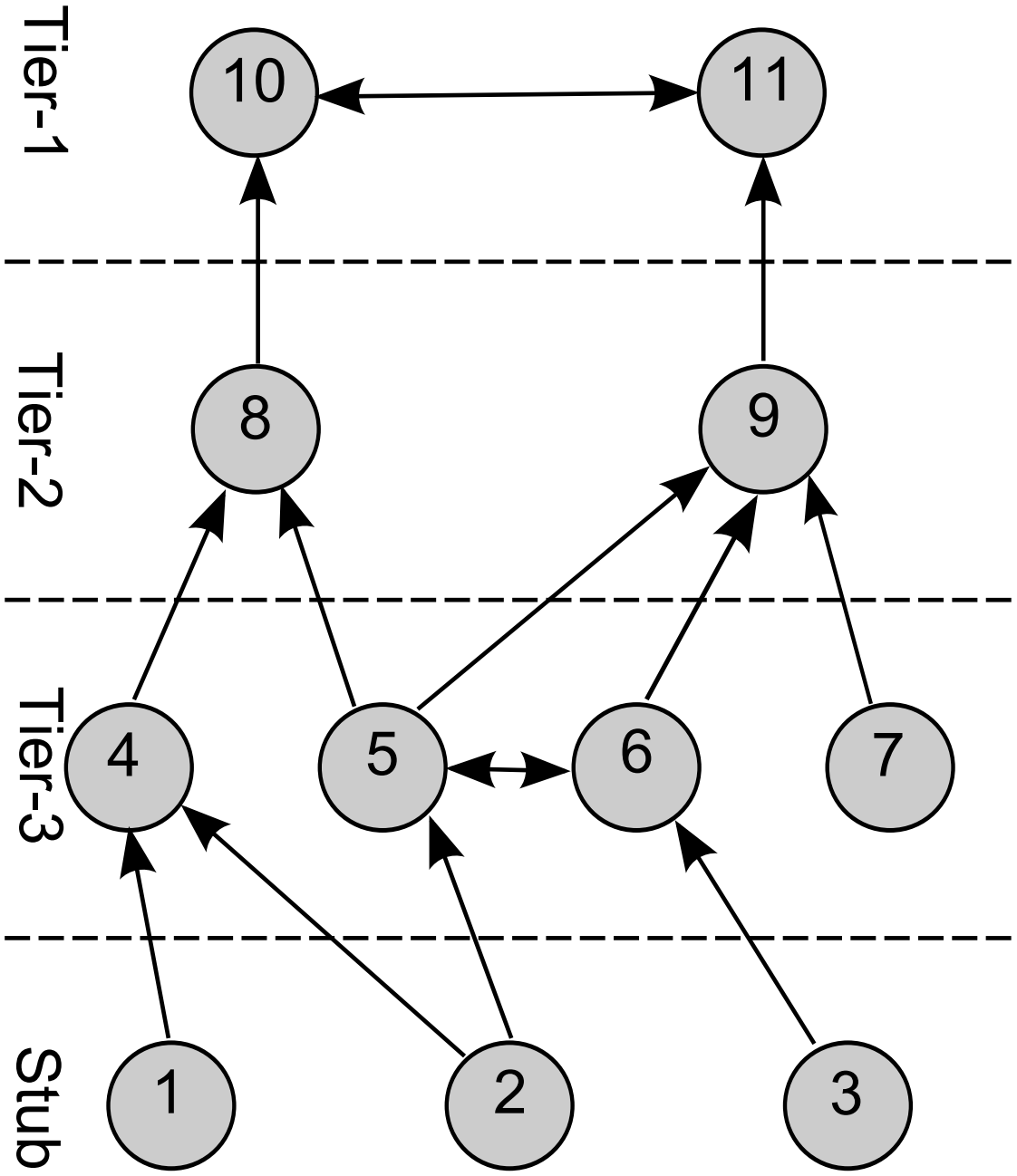


Figure 3: Topology of the test network during the first part of the experiments. [Customer → Provider, Peer ↔ Peer]

Listing 3 documents the consequence of the attack. Node 4 pretending to have a direct connection with the origin AS. Announcing the prefix with a two hop AS path. Consequently this path is chosen by node 2.

Listing 2: node 2's view for prefix 172.30.0.0/16 before the attack

```
172.30.0.0/16    via 141.22.28.124 on eth0 [node5 10:59] * (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65124 65132 65130
    BGP.next_hop: 141.22.28.124
    BGP.med: 0
    BGP.local_pref: 100
                 via 141.22.28.25 on eth0 [node4 Jan17] (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65025 65123 65125 65126 65132 65130
    BGP.next_hop: 141.22.28.25
    BGP.med: 0
    BGP.local_pref: 100
```

Listing 3: node 2's view for prefix 172.30.0.0/16 during the attack

```
172.30.0.0/16    via 141.22.28.25 on eth0 [node4 16:51] * (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65025 65130
    BGP.next_hop: 141.22.28.25
    BGP.local_pref: 100
                 via 141.22.28.124 on eth0 [node5 10:59] (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65124 65132 65130
    BGP.next_hop: 141.22.28.124
    BGP.med: 0
    BGP.local_pref: 100
```

To verify that I also succeeded in hiding this attack from the other nodes in the network, I verify node 1's view for the attacked prefix. As node 4 is node 1's only upstream, this differentiates this attack from the classical attack pattern. Node 1's view for the attacked prefix (listing 4) confirms everything is working as expected. Node 1 sees the same AS path like it was visible to node 2 before the attack started.

Listing 4: node 1's view for prefix 172.30.0.0/16 during the attack

```
172.30.0.0/16    via 141.22.28.25 on eth0 [node4 Jan17] * (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65025 65123 65125 65126 65132 65130
    BGP.next_hop: 141.22.28.25
    BGP.med: 0
```

```
7     BGP.local_pref: 100
```

With the confirmation that the attack works in principle we analyse further scenarios.

3.1.2. Large ISP attacking a small ISP

In the next scenario a large ISP (node 8) wants to attack a small ISP (node 5), which is one of its customers. In this scenario the limitations of my test bed influence the expected results. For the small ISPs in my network most ASes are reachable with just two hops. Generating longer AS paths, while maintaining a realistic topology between the individual nodes, requires a much larger network. But larger networks are also more difficult to create and to handle during experiments. Consequently this attack should be easier to execute in a real world environment.

Despite the aforementioned limitation of the test network, the attacker captures the prefix 172.30.0.0/16 again. Before the start of the attack node 5's view of the network is without any suspicious entries and reflects the actual network topology (listing 5). As seen in listing 6, the attacker is successful in attracting all traffic from node 5 to the prefix 172.30.0.0/16.

Listing 5: node 5's view for the prefix 172.30.0.0/16 before the attack

```
1  172.30.0.0/16    via 62.75.143.240 on eth0 [node9 12:47] * (75) [AS65130i]
2      Type: BGP unicast univ
3      BGP.origin: IGP
4      BGP.as_path: 65132 65130
5      BGP.next_hop: 62.75.143.240
6      BGP.med: 0
7      BGP.local_pref: 100
8                    via 141.22.28.123 on eth0 [node8 12:47] (75) [AS65130i]
9      Type: BGP unicast univ
10     BGP.origin: IGP
11     BGP.as_path: 65123 65125 65126 65132 65130
12     BGP.next_hop: 141.22.28.123
13     BGP.med: 0
14     BGP.local_pref: 100
```

Listing 6: node 5's view for the prefix 172.30.0.0/16 during the attack

```
1  172.30.0.0/16    via 141.22.28.123 on eth0 [node8 13:21] * (75) [AS65130i]
2      Type: BGP unicast univ
3      BGP.origin: IGP
4      BGP.as_path: 65123 65130
5      BGP.next_hop: 141.22.28.123
6      BGP.local_pref: 100
7                    via 62.75.143.240 on eth0 [node9 12:47] (75) [AS65130i]
8      Type: BGP unicast univ
9      BGP.origin: IGP
10     BGP.as_path: 65132 65130
11     BGP.next_hop: 62.75.143.240
```

```
12      BGP.med: 0
13      BGP.local_pref: 100
```

Comparing the view for the attacked prefix with the one of node 4 shows that node 4 does not see any sign of the ongoing attack (listing 7). Node 2 does see the change in the AS path caused by the attack (compare listings 8 and 9, the AS path changes from 65124 **65132** 65130 to 65124 **65123** 65130), but it does not affect the chosen route for the prefix. Node 2 was using Node 5 as preferred route anyway as the alternative route via node 4 us much longer. As node 8 does not export the malicious route to node 4, this route does not change. In the route via node 5 one element of the AS path do change, but since it is in the middle of the AS path it is difficult for node 2 to verify the legitimacy of the change. Comparing the two routes shows an irregularity as AS 65123 is apparently exporting two different routes for the same prefix, but since this also happens neither with the direct peering partner nor with the origin AS, this is not a clear indication of an attack.

Consequently one can argue that node 2 affected by the attack, but not likely to consider the route change caused by a malicious attacker.

Listing 7: node 4's view for the prefix 172.30.0.0/16 during the attack

```
1  172.30.0.0/16    via 141.22.28.123 on eth1 [node8 Jan24] * (75) [AS65130i]
2      Type: BGP unicast univ
3      BGP.origin: IGP
4      BGP.as_path: 65123 65125 65126 65132 65130
5      BGP.next_hop: 141.22.28.123
6      BGP.med: 0
7      BGP.local_pref: 100
```

Listing 8: node 2's view for the prefix 172.30.0.0/16 before the attack

```
1  172.30.0.0/16    via 141.22.28.124 on eth0 [node5 12:48] * (75) [AS65130i]
2      Type: BGP unicast univ
3      BGP.origin: IGP
4      BGP.as_path: 65124 65132 65130
5      BGP.next_hop: 141.22.28.124
6      BGP.med: 0
7      BGP.local_pref: 100
8                   via 141.22.28.25 on eth0 [node4 Jan24] (75) [AS65130i]
9      Type: BGP unicast univ
10     BGP.origin: IGP
11     BGP.as_path: 65025 65123 65125 65126 65132 65130
12     BGP.next_hop: 141.22.28.25
13     BGP.med: 0
14     BGP.local_pref: 100
```

Listing 9: node 2's view for the prefix 172.30.0.0/16 during the attack

```
1  172.30.0.0/16    via 141.22.28.124 on eth0 [node5 Jan24] * (75) [AS65130i]
2      Type: BGP unicast univ
```

```
3     BGP.origin: IGP
4     BGP.as_path: 65124 65123 65130
5     BGP.next_hop: 141.22.28.124
6     BGP.local_pref: 100
7                   via 141.22.28.25 on eth0 [node4 Jan24] (75) [AS65130i]
8     Type: BGP unicast univ
9     BGP.origin: IGP
10    BGP.as_path: 65025 65123 65125 65126 65132 65130
11    BGP.next_hop: 141.22.28.25
12    BGP.med: 0
13    BGP.local_pref: 100
```

3.1.3. Tier-1 attacking a large ISP

Going up another level from the last attack, simulating the consequences of a Tier-1 provider attacking one of it's customer ISPs, is problematic. It is to be expected that Tier-1 provider have an enormous influence on the whole routing topology. That makes any attack by one of them both widespread and difficult to hide. Unfortunately an accurate simulation requires a much larger test network. In my network both Tier-1 provider are either already the preferred route for many prefixes or cannot provide a malicious route that would be shorter than the legitimate route. In any case it is trivial to see that any malicious route update by one of the Tier-1 providers would widely spread in the topology. Therefore such an attack would not be different from the classical attack pattern and be highly visible to most of the Internet.

3.1.4. Attacking upstream provider

After the series of examined attacks on the downstream customers done before, one interesting question left to discuss about this topology are those cases, where the attacker chooses to direct the attack at his own upstream provider. It quickly becomes apparent that such an attack violates the basic idea of keeping the attack confined and therefore invisible.

The underlying model of routing policy leads to the fact that the victim will not only choose the malicious route, but also propagate it (see figure 2d). Thus the attacker does not have any control to prevent unwanted spreading of the malicious update through the routing topology. One possible exception are upstream provider which are known to filter route updates, only propagating prefixes known to be owned by a customer of these providers. But if an upstream provider filters and how this filter varies greatly and is difficult to describe in general. An investigation of this constellation is out of scope of this paper due to the limited size of the network used and a lack of a consistent model for an filtering upstream provider. It is therefore left to future work to investigate this aspect further.

When we do not take filter into consideration, it is therefore to never attack an upstream AS with the attack pattern described in this paper. Doing so would loose the benefits of this specific attack pattern.

3.2. Modified IXP/Tier based Topology

While the network topology used in the previous experiments served well to strengthen the understand of the implications of the attack pattern, the strictly hierarchical tier based topology does not reflect today's real world topology of the Internet. C. Labovitz et. al. conducted a more recent survey [5] of the Internet topology. Important changes are the concentration of the traffic in the Internet, flowing either in or out of just a small number of ASes, and the importance of IXPs for the global traffic flows, with a high amount of traffic passing at least one IXP.

The concentration of the traffic to a few very high traffic nodes creates so called "hyper giants". The business model of these ASes is often based on distributing as much content as possible. For that purpose they enter into a large number of direct peering connections, especially with stub networks. The result is ASes with very good interconnection, similar to Tier-1 provider, which do not offer transit services. Those networks are interesting in terms of their ability to launch attacks.

The other key point is the importance of Internet-Exchange-Points. While IXPs have existed for a long time already, their importance in terms of share from the global traffic depends on the prevailing business models among the ISPs. Their importance increased in connection with increasing diversity of ISPs participating in the global routing. Consequently recent studies put more focus on the influence they have on the routing topology [11]. One important effect of IXPs is the fact that direct peering happens more frequently and also across tier levels.

For the second part of the experiment the test network has therefore been changed to investigate those effects. The nodes are still the same (see table 1 on page 10), but they are connected differently this time. Figure 4 shows the complete topology. Notable changes include node 10, which acts here as a hyper giant, and the change of the connection between nodes 5 and 8 to a peering connection, which changes the routing behaviour of the left side of the topology.

3.2.1. Large ISP attacking a small ISP

This scenario is identical to the one covered in section 3.1.2 now applied to the new network. Now nodes 5 and 8 are connected via a peering connection. Before the attack node 5 only receives the prefix 172.30.0.0/16 via it's upstream connection to node 9 (listing 10) as expected.

Listing 10: node 5's view for the prefix 172.30.0.0/16 before the attack

```
172.30.0.0/16    via 62.75.143.240 on eth0 [node9 12:47] * (75) [AS65130i]
    Type: BGP unicast univ
    BGP.origin: IGP
    BGP.as_path: 65132 65130
    BGP.next_hop: 62.75.143.240
    BGP.med: 0
    BGP.local_pref: 100
```

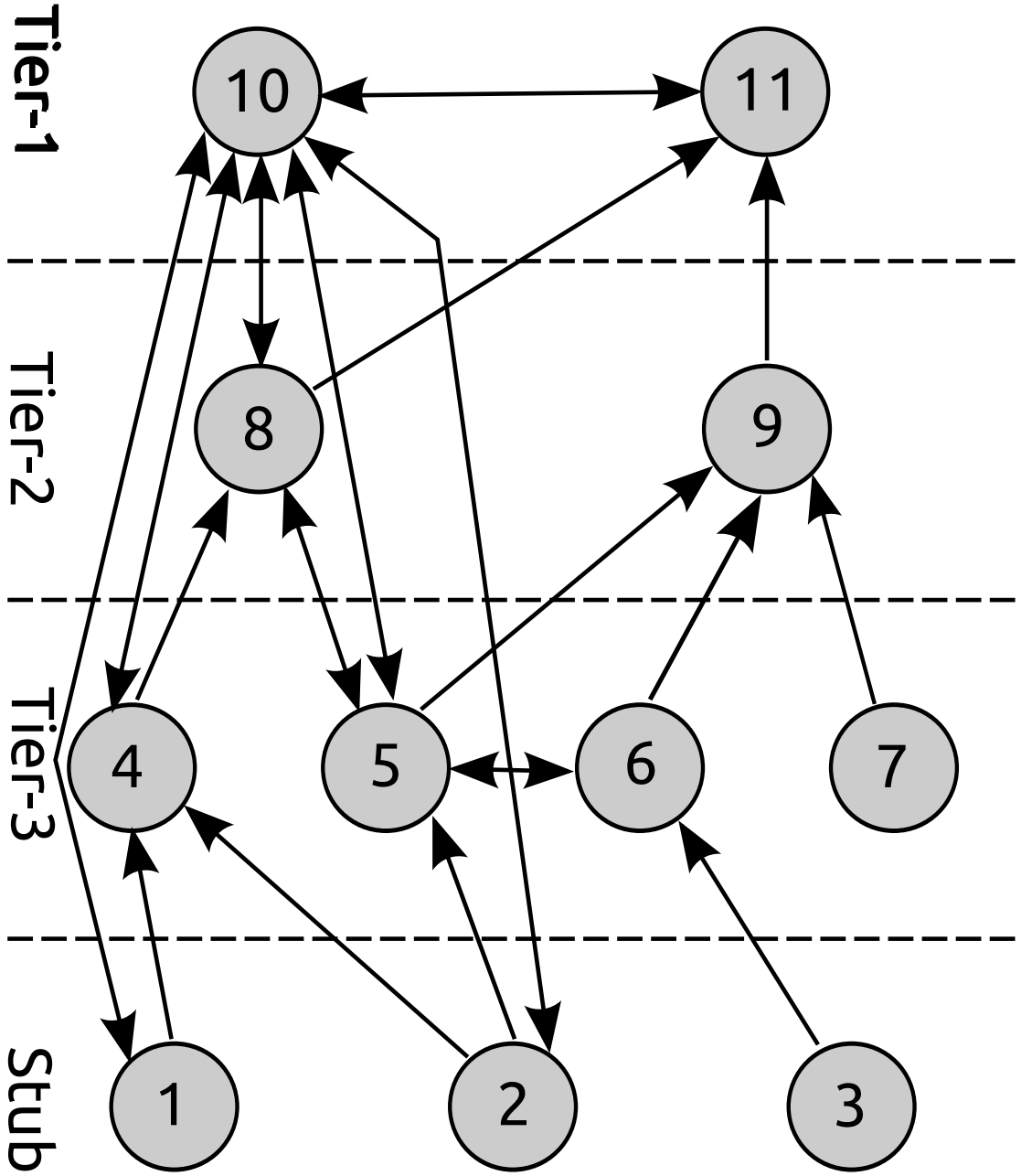


Figure 4: Topology of the test network during the second part of the experiments. [Customer → Provider, Peer ↔ Peer]

Node 8 starts the attack by exporting the prefix via the peering connection to node 5. As peering connections are preferred over provider ones, node 5 immediately uses the new route via node 8 for the prefix (listing 11). Consequently the attack works even better, as peering connections are in general preferred over provider connections.

Listing 11: node 5's view for the prefix 172.30.0.0/16 during the attack

```
172.30.0.0/16    via 141.22.28.123 on eth0 [node8 13:21] * (100) [AS65130i]
   Type: BGP unicast univ
   BGP.origin: IGP
   BGP.as_path: 65123 65130
   BGP.next_hop: 141.22.28.123
   BGP.local_pref: 100
                 via 62.75.143.240 on eth0 [node9 12:47] (75) [AS65130i]
   Type: BGP unicast univ
   BGP.origin: IGP
   BGP.as_path: 65132 65130
   BGP.next_hop: 62.75.143.240
   BGP.med: 0
   BGP.local_pref: 100
```

Other nodes in the network behave identical to the attack in the Gao-Rexford-model.

Hyper giants attacking a stub network

Compared to the tier-1 providers hyper giants have the big advantage to have a direct peering connection with many other networks, especially stub networks. As stub networks do not have any customer ASes themselves the peering connection with the hyper giant will always take precedence. It is trivial to show that the attack pattern works perfectly in this constellation.

4. Conclusion

In this paper we have described an attack pattern that aims at being harder to detect than previously known attacks. Based on a standard set of routing policies we developed the circumstances which allow this attack to be executed. We went through the different stages and provided two use cases. We also used a small test network to verify that the theory can in principle be but in practice.

Not part of this work is implementing the attack in a larger scale network, in a setup with the Internet. That is left to our next work.

References

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

[2] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)," IETF, RFC 1930, March 1996.

[3] J. Arkko, M. Cotton, and L. Vegoda, "IPv4 Address Blocks Reserved for Documentation," IETF, RFC 5737, January 2010.

[4] L. Gao, "On Inferring Autonomous System Relationships in the Internet," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 733–745, 2001.

[5] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-domain Traffic," in *Proc. of the ACM SIGCOMM '10*. New York, NY, USA: ACM, 2010, pp. 75–86.

[6] G. Huston, M. Rossi, and G. Armitage, "A Technique for Reducing BGP Update Announcements through Path Exploration Damping," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 8, pp. 1271–1286, October 2010.

[7] H. Ballani, P. Francis, and X. Zhang, "A Study of Prefix Hijacking and Interception in the Internet," in *Proc. of SIGCOMM '07*. New York, NY, USA: ACM, 2007, pp. 265–276.

[8] J. Prins and B. U. Cybercrime, "DigiNotar Certificate Authority breach'Operation Black Tulip'," *Fox-IT, November*, 2011.

[9] K. Biermann, "Die hinterhältigen Tricks der Porno-Abmahner."

[10] Y. Rekhter, R. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," IETF, RFC 1918, February 1996.

[11] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger, "Anatomy of a Large European IXP," in *Proc. of the ACM SIGCOMM*. New York, NY, USA: ACM, 2012, pp. 163–174.

A. BIRD configuration files for Rexford-Gao model

Listing 12: Configuration of Node 1

```
/*
 * BIRD configuration file of node 1.
 */

log "/var/log/bird.log" all;

router id 141.22.28.121;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference = customer_pref || preference = local_pref then # route came from provider, peer, customer or us
        accept;

    reject;
}
```

```
40
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.16.0.0/15 via "eth0";
54 }
55
56 template bgp bgp_template {
57     local as 65121;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65025 ] then {
62             preference = provider_pref;
63             accept;
64             }
65             reject;
66     };
67
68     export none;
69 }
70
71 protocol bgp node4 from bgp_template {
72     description "BGP session with haw3";
73     neighbor 141.22.28.25 as 65025;
74     export filter export_provider;
75 }
```

Listing 13: Configuration of Node 2

```
1  /*
2   * BIRD configuration file of node 2.
3   */
4
5  log "/var/log/bird.log" all;
6
7  router id 141.22.28.122;
8
9  define provider_pref = 75;
10 define peer_pref = 100;
11 define customer_pref = 125;
```

```
12 define local_pref = 200;
13
14 filter export_provider {
15     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
16         reject;
17
18     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
19         accept;
20
21     reject;
22 }
23
24 filter export_peer {
25     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
26         reject;
27
28     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
29         accept;
30
31     reject;
32 }
33
34 filter export_customer {
35     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
36         accept;
37
38     reject;
39 }
40
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.18.0.0/15 via "eth0";
54 }
```

```
55
56 template bgp bgp_template {
57     local as 65122;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65025, 65124 ] then {
62             preference = provider_pref;
63             accept;
64             }
65             reject;
66     };
67
68     export none;
69 }
70
71 protocol bgp node4 from bgp_template {
72     description "BGP session with haw3";
73     neighbor 141.22.28.25 as 65025;
74     export filter export_provider;
75 }
76
77 protocol bgp node5 from bgp_template {
78     description "BGP session with vm4";
79     neighbor 141.22.28.124 as 65124;
80     export filter export_provider;
81 }
```

Listing 14: Configuration of Node 3

```
1  table db;
2
3  filter zweihundertdran {
4  bgp_path.prepend(201);
5  bgp_path.prepend(1337);
6  bgp_path.prepend(201);
7  accept;
8  }
9
10
11 #protocol bgp Brocade {
12 #
13 #local as 200;
14 #neighbor 141.22.27.145 as 300;
15 #table db;
16 ##export all;
17 #import all;
18 #path metric 100;
19 #export filter zweihundertdran;
20 #default bgp_local_pref 300;
```

```
#next hop self;
#}

#protocol bgp Notebook{
#local as 200;
#neighbor 141.22.26.77 as 100;
#table db;
#export all;
#import all;
#path metric 200;
#export filter zweihundertdran;
#}

protocol bgp hosteurope{
local as 65133;
neighbor 176.28.11.224 as 65131;
table db;
export all;
import all;
multihop 255;
path metric 300;
#export filter zweihundertdran;
}


/*
 *  This is an example configuration file.
 */

# Yes, even shell-like comments work...

# Configure logging
#log syslog { debug, trace, info, remote, warning, error, auth, fatal, bug };
#log stderr all;
#log "tmp" all;

# Override router ID
router id 141.22.28.199;

# You can define your own symbols...
#define xyzzy = (120+10);
#define '1a-a1' = (30+40);

# Define a route filter...
#filter test_filter {
#   if net ~ 10.0.0.0/16 then accept;
#   else reject;
#}

```

```
#filter sink { reject; }
#filter okay { accept; }

#include "filters.conf";

# Define another routing table
#table testable;

# Turn on global debugging of all protocols
#debug protocols all;

# The direct protocol automatically generates device routes to
# all network interfaces. Can exist in as many instances as you wish
# if you want to populate multiple routing tables with device routes.
#protocol direct {
#   interface "-eth*", "*"; # Restrict network interfaces it works with
#}

# This pseudo-protocol performs synchronization between BIRD's routing
# tables and the kernel. If your kernel supports multiple routing tables
# (as Linux 2.2.x does), you can run multiple instances of the kernel
# protocol and synchronize different kernel tables with different BIRD tables.
protocol kernel {
#   learn;         # Learn all alien routes from the kernel
    persist;       # Don't remove routes on bird shutdown
    scan time 20;      # Scan kernel routing table every 20 seconds
#   import none;       # Default is import all
    export all;    # Default is export none
#   kernel table 5;   # Kernel table to synchronize with (default: main)
}

# This pseudo-protocol watches all interface up/down events.
protocol device {
    scan time 10;      # Scan interfaces every 10 seconds
}

# Static routes (again, there can be multiple instances, so that you
# can disable/enable various groups of static routes on the fly).
protocol static {
table db;
#   disabled;      # Disable by default
#   table testable;   # Connect to a non-default table
#   preference 1000;  # Default preference of routes
#   debug { states, routes, filters, interfaces, events, packets };
#   debug all;
#   route 0.0.0.0/0 via 198.51.100.13;
#   route 198.51.100.0/25 reject;
#   route 10.0.0.0/8 reject;
#   route 10.1.1.0:255.255.255.0 via 198.51.100.3;
```

```
#   route 10.1.2.0:255.255.255.0 via 198.51.100.3;
#   route 10.1.3.0:255.255.255.0 via 198.51.100.4;
#   route 10.2.0.0/24 via "arc0";
#   route 10.2.0.0/16 via "eth2";
#   route 192.168.2.0/24 via "eth2";
route 172.29.0.0/16 via "eth2";
}

# Pipe protocol connects two routing tables... Beware of loops.
#protocol pipe {
#   peer table testable;
# Define what routes do we export to this protocol / import from it.
#   import all;    # default is all
#   export all;    # default is none
#   import none;      # If you wish to disable imports
#   import filter test_filter;   # Use named filter
#   import where source = RTS_DEVICE; # Use explicit filter
#}

# RIP aka Rest In Pieces...
#protocol rip MyRIP { # You can also use an explicit name
#   preference xyzzy;
#   debug all;
#   port 1520;
#   period 7;
#   infinity 16;
#   garbage time 60;
#   interface "*" { mode broadcast; };
#   honor neighbor;   # To whom do we agree to send the routing table
#   honor always;
#   honor never;
#   passwords {
#       password "nazdar";
#   };
#   authentication none;
#   import filter { print "importing"; accept; };
#   export filter { print "exporting"; accept; };
#}

#protocol ospf MyOSPF {
#       tick 2;
#   rfc1583compat yes;
#   area 0.0.0.0 {
#       stub no;
#       interface "eth*" {
#           hello 9;
#           retransmit 6;
#           cost 10;
#           transmit delay 5;
```

```
#          dead count 5;
#          wait 50;
#          type broadcast;
#          authentication simple;
#          password "pass";
#      };
#      interface "arc0" {
#          rx buffer large;
#          type nonbroadcast;
#          poll 14;
#          dead 75;
#          neighbors {
#              10.1.1.2 eligible;
#              10.1.1.4;
#          };
#          strict nonbroadcast yes;
#      };
#      interface "xxx0" {
#                      passwords {
#              password "abc" {
#                  id 1;
#                  generate to "22-04-2003 11:00:06";
#                  accept to "17-01-2004 12:01:05";
#              };
#              password "def" {
#                  id 2;
#                  generate from "22-04-2003 11:00:07";
#                  accept from "17-01-2003 12:01:05";
#              };
#          };
#                      authentication cryptographic;
#      };
#  };
#  area 20 {
#      stub 1;
#      interface "ppp1" {
#          hello 8;
#          authentication none;
#      };
#              interface "fr*";
#              virtual link 192.168.0.1 {
#                      password "sdsdffsdfg";
#                      authentication cryptographic;
#              };
#  };
#}


#protocol bgp {
```

```
#   disabled;
#   description "My BGP uplink";
#   local as 65000;
#   neighbor 198.51.100.130 as 64496;
#   multihop;
#   hold time 240;
#   startup hold time 240;
#   connect retry time 120;
#   keepalive time 80; # defaults to hold time / 3
#   start delay time 5; # How long do we wait before initial connect
#   error wait time 60, 300;# Minimum and maximum time we wait after an error (when consecutive
#               # errors occur, we increase the delay exponentially ...
#   error forget time 300; # ... until this timeout expires)
#   disable after error;  # Disable the protocol automatically when an error occurs
#   next hop self;    # Disable next hop processing and always advertise our local address as nexthop
#   path metric 1;    # Prefer routes with shorter paths (like Cisco does)
#   default bgp_med 0; # MED value we use for comparison when none is defined
#   default bgp_local_pref 0; # The same for local preference
#   source address 198.51.100.14; # What local address we use for the TCP connection
#   password "secret"; # Password used for MD5 authentication
#   rr client;     # I am a route reflector and the neighor is my client
#   rr cluster id 1.0.0.1; # Use this value for cluster id instead of my router id
#   export where source=RTS_STATIC;
#   export filter {
#       if source = RTS_STATIC then {
#           bgp_community = -empty-; bgp_community = add(bgp_community,(65000,5678));
#           bgp_origin = 0;
#           bgp_community = -empty-; bgp_community.add((65000,5678));
#           if (65000,64501) ~ bgp_community then
#               bgp_community.add((0, 1));
#           if bgp_path ~ [= 65000 =] then
#               bgp_path.prepend(65000);
#           accept;
#       }
#       reject;
#   };
#}
#
# Template usage example
#template bgp rr_client {
#   disabled;
#   local as 65000;
#   multihop;
```

```
#   rr client;
#   rr cluster id 1.0.0.1;
#}
#
#protocol bgp rr_abcd from rr_client {
#   neighbor 10.1.4.7 as 65000;
#}
```

Listing 15: Configuration of Node 4

```
/*
 *  BIRD configuration file for node 4.
 */

# Configure logging
log "/var/log/bird.log" all;

# Override router ID
router id 141.22.28.25;

table malicious;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}
```

```
37
38 filter export_customer {
39     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
40         accept;
41
42     reject;
43 }
44
45 protocol kernel {
46     scan time 20;      # Scan kernel routing table every 20 seconds
47     import none;       # Default is import all
48     export all;        # Default is export none
49 }
50
51 protocol device {
52     scan time 10;
53 }
54
55 protocol static localAS {
56     description "Static route to simulated own AS.";
57     route 172.27.0.0/16 via "eth1";
58 }
59
60 template bgp bgp_template{
61     local as 65025;
62     next hop self;
63
64     import filter {
65         if bgp_path.first ~ [ 65123 ] then {
66             preference = provider_pref;
67             accept;
68         }
69
70         if bgp_path.first ~ [ 0 ] then {
71             preference = peer_pref;
72             accept;
73         }
74
75         if bgp_path.first ~ [ 65121, 65122 ] then {
76             preference = customer_pref;
77             accept;
78         }
79         reject;
80     };
81
82     export none;
83 }
```

```
84
85  protocol bgp node1 from bgp_template {
86      description "BGP session with vm1";
87      neighbor 141.22.28.121 as 65121;
88      export filter export_customer;
89  }
90
91  protocol bgp node2 from bgp_template {
92      description "BGP session with vm2";
93      neighbor 141.22.28.122 as 65122;
94      export filter export_customer;
95  }
96
97  protocol bgp node8 from bgp_template {
98      description "BGP session with vm3";
99      neighbor 141.22.28.123 as 65123;
100     export filter export_provider;
101 }
```

Listing 16: Configuration of Node 5

```
1   /*
2    * BIRD configuration file of node 5.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.124;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
22  }
23
24  filter export_peer {
25      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
26          reject;
```

```
27
28      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
29          accept;
30
31      reject;
32  }
33
34  filter export_customer {
35      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
36          accept;
37
38      reject;
39  }
40
41  protocol kernel {
42      scan time 20;
43      import none;
44      export all;
45  }
46
47  protocol device {
48      scan time 10;
49  }
50
51  protocol static localAS {
52      description "Static route to own AS.";
53      route 172.22.0.0/15 via "eth0";
54  }
55
56  template bgp bgp_template {
57      local as 65124;
58      next hop self;
59
60      import filter {
61          if bgp_path.first ~ [ 65123, 65132 ] then {
62              preference = provider_pref;
63              accept;
64              }
65
66              if bgp_path.first ~ [ 65131 ] then {
67              preference = peer_pref;
68              accept;
69              }
70
71              if bgp_path.first ~ [ 65122 ] then {
72              preference = customer_pref;
```

```
            accept;
            }
            reject;
    };

    export none;
}

protocol bgp node2 from bgp_template {
    description "BGP session with node2";
    neighbor 141.22.28.122 as 65122;
    export filter export_customer;
}

protocol bgp node6 from bgp_template {
    description "BGP session with node6";
    neighbor 176.28.11.224 as 65131;
    multihop 255;
    export filter export_peer;
}

protocol bgp node8 from bgp_template {
    description "BGP session with node8";
    neighbor 141.22.28.123 as 65123;
    export filter export_provider;
}

protocol bgp node9 from bgp_template {
    description "BGP session with node9";
    neighbor 62.75.143.240 as 65132;
    multihop 255;
    export filter export_provider;
}
```

Listing 17: Configuration of Node 6

```
/*
 *  BIRD configuration file for node 6.
 */

# Configure logging
#log "/var/log/bird.log" all;


# Override router ID
router id 176.28.11.224;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
```

```
15 define local_pref = 200;
16
17 filter export_provider {
18     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
19         reject;
20
21     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
22         accept;
23
24     reject;
25 }
26
27 filter export_peer {
28     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
29         reject;
30
31     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
32         accept;
33
34     reject;
35 }
36
37 filter export_customer {
38     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
39         accept;
40
41     reject;
42 }
43
44 protocol kernel {
45     scan time 20;     # Scan kernel routing table every 20 seconds
46     import none;      # Default is import all
47     export all;       # Default is export none
48 }
49
50 protocol device {
51     scan time 10;
52 }
53
54 protocol static localAS {
55     description "Static route to simulated own AS.";
56     route 172.31.0.0/16 via "venet0";
57 }
```

```

template bgp bgp_template{
    local as 65131;
    next hop self;
    multihop 255; # added


    import filter {
        if bgp_path.first ~ [ 65132 ] then {
            preference = provider_pref;
            accept;
        }

        if bgp_path.first ~ [ 65124 ] then {
            preference = peer_pref;
            accept;
        }

        if bgp_path.first ~ [ 65133 ] then {
            preference = customer_pref;
            accept;
        }
        reject;
    };

    export none;
}

protocol bgp S4Y from bgp_template {
    description "BGP session with S4Y";
    neighbor 62.75.143.240 as 65132;
    export filter export_provider;
}

protocol bgp FHRechner from bgp_template {
    description "BGP session with FH-Rechner";
    neighbor 141.22.28.199 as 65133;
    export filter export_customer;
}

protocol bgp vm4 from bgp_template {
    description "BGP session with vm4";
    neighbor 141.22.28.124 as 65124;
    export filter export_peer;
}
```

Listing 18: Configuration of Node 8

```
/*
 * BIRD configuration file of node 8.
```

```
 */

log "/var/log/bird.log" all;

router id 141.22.28.123;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference = customer_pref || preference = local_pref then # route came from provider, peer, customer or us
        accept;

    reject;
}

protocol kernel {
    scan time 20;
    import none;
    export all;
}
```

```
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.20.0.0/15 via "eth0";
54 }
55
56 template bgp bgp_template {
57     local as 65123;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65125 ] then {
62             preference = provider_pref;
63             accept;
64             }
65
66             if bgp_path.first ~ [ 65025, 65124 ] then {
67             preference = customer_pref;
68             accept;
69             }
70             reject;
71     };
72
73     export none;
74 }
75
76 protocol bgp node4 from bgp_template {
77     description "BGP session with node4";
78     neighbor 141.22.28.25 as 65025;
79     export filter export_customer;
80 }
81
82 protocol bgp node5 from bgp_template {
83     description "BGP session with node5";
84     neighbor 141.22.28.124 as 65124;
85     export filter export_customer;
86 }
87
88 protocol bgp node10 from bgp_template {
89     description "BGP session with node10";
90     neighbor 141.22.28.125 as 65125;
91     export filter export_provider;
92 }
```

Listing 19: Configuration of Node 9

```
/*
 *  BIRD configuration file of node 9.
 */

# Configure logging
#log "/var/log/bird.log" all;

# Override router ID
router id 62.75.143.240;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference = customer_pref || preference = local_pref then # route came from provider, peer, customer or us
        accept;

    reject;
}

```

```
protocol kernel {
    scan time 20;     # Scan kernel routing table every 20 seconds
    import none;      # Default is import all
    export all;       # Default is export none
}

protocol device {
    scan time 10;
}

protocol static localAS {
    description "Static route to simulated own AS.";
    route 172.28.0.0/16 via "venet0";
}

template bgp bgp_template{
    local as 65132;
    next hop self;
    multihop 255;  # added

    import filter {
        if bgp_path.first ~ [ 65126 ] then {
            preference = provider_pref;
            accept;
        }

        if bgp_path.first ~ [ 0 ] then {
            preference = peer_pref;
            accept;
        }

        if bgp_path.first ~ [ 65131, 65130 ] then {
            preference = customer_pref;
            accept;
        }
        reject;
    };

    export none;
}

protocol bgp vm6 from bgp_template {
    description "BGP session with vm6";
    neighbor 141.22.28.126 as 65126;
    export filter export_provider;
}

protocol bgp brocade from bgp_template {
    description "BGP session with brocade";
```

```
92      neighbor 141.22.27.145 as 65130;
93      export filter export_customer;
94  }
95
96  protocol bgp hosteurope from bgp_template {
97      description "BGP session with hosteurope";
98      neighbor 176.28.11.224 as 65131;
99      export filter export_customer;
100 }
```

Listing 20: Configuration of Node 10

```
1   /*
2    * BIRD configuration file of node 10.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.125;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
22  }
23
24  filter export_peer {
25      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
26          reject;
27
28      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
29          accept;
30
31      reject;
32  }
33
34  filter export_customer {
```

```
35      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
36          accept;
37
38      reject;
39  }
40
41  protocol kernel {
42      scan time 20;
43      import none;
44      export all;
45  }
46
47  protocol device {
48      scan time 10;
49  }
50
51  protocol static localAS {
52      description "Static route to own AS.";
53      route 172.24.0.0/15 via "eth0";
54  }
55
56  template bgp bgp_template {
57      local as 65125;
58      next hop self;
59
60      import filter {
61              if bgp_path.first ~ [ 65126 ] then {
62              preference = peer_pref;
63              accept;
64              }
65
66              if bgp_path.first ~ [ 65123 ] then {
67              preference = customer_pref;
68              accept;
69              }
70              reject;
71      };
72
73      export none;
74  }
75
76  protocol bgp node8 from bgp_template {
77      description "BGP session with node8";
78      neighbor 141.22.28.123 as 65123;
79      export filter export_customer;
80  }
81
```

```
protocol bgp node11 from bgp_template {
    description "BGP session with node11";
    neighbor 141.22.28.126 as 65126;
    export filter export_peer;
}
```

Listing 21: Configuration of Node 11

```
/*
 * BIRD configuration file of node 11.
 */

log "/var/log/bird.log" all;

router id 141.22.28.126;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came
        from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came
        from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came
        from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came
        from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference =
        customer_pref || preference = local_pref then # route came from provider
        , peer, customer or us
        accept;
```

```
37
38      reject;
39  }
40
41  protocol kernel {
42      scan time 20;
43      import none;
44      export all;
45  }
46
47  protocol device {
48      scan time 10;
49  }
50
51  protocol static localAS {
52      description "Static route to own AS.";
53      route 172.26.0.0/16 via "eth0";
54  }
55
56  template bgp bgp_template {
57      local as 65126;
58      next hop self;
59
60      import filter {
61          if bgp_path.first ~ [ 65125 ] then {
62              preference = peer_pref;
63              accept;
64              }
65
66              if bgp_path.first ~ [ 65124, 65132 ] then {
67              preference = customer_pref;
68              accept;
69              }
70              reject;
71      };
72
73      export none;
74  }
75
76  protocol bgp node10 from bgp_template {
77      description "BGP session with node10";
78      neighbor 141.22.28.125 as 65125;
79      export filter export_peer;
80  }
81
82  protocol bgp node9 from bgp_template {
83      description "BGP session with node9";
84      neighbor 62.75.143.240 as 65132;
85      export filter export_customer;
```

```
86      multihop 255;
87  }
```

B. BIRD configuration files for IXP model

Listing 22: Configuration of Node 1

```
1   /*
2    * BIRD configuration file of node 1.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.121;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
22  }
23
24  filter export_peer {
25      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
26          reject;
27
28      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
29          accept;
30
31      reject;
32  }
33
34  filter export_customer {
35      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
```

```
36          accept;
37
38      reject;
39  }
40
41  protocol kernel {
42      scan time 20;
43      import none;
44      export all;
45  }
46
47  protocol device {
48      scan time 10;
49  }
50
51  protocol static localAS {
52      description "Static route to own AS.";
53      route 172.16.0.0/15 via "eth0";
54  }
55
56  template bgp bgp_template {
57      local as 65121;
58      next hop self;
59
60      import filter {
61          if bgp_path.first ~ [ 65025 ] then {
62              preference = provider_pref;
63              accept;
64              }
65
66          if bgp_path.first ~ [ 65125 ] then {
67              preference = peer_pref;
68              accept;
69              }
70              reject;
71      };
72
73      export none;
74  }
75
76  protocol bgp node4 from bgp_template {
77      description "BGP session with node 4";
78      neighbor 141.22.28.25 as 65025;
79      export filter export_provider;
80  }
81
82  protocol bgp node10 from bgp_template {
83      description "BGP session with node 10";
84      neighbor 141.22.28.125 as 65125;
```

```
85      export filter export_peer;
86  }
```

Listing 23: Configuration of Node 2

```
1   /*
2    * BIRD configuration file of node 2.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.122;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
22  }
23
24  filter export_peer {
25      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
26          reject;
27
28      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
29          accept;
30
31      reject;
32  }
33
34  filter export_customer {
35      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
36          accept;
37
38      reject;
39  }
```

```
40
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.18.0.0/15 via "eth0";
54 }
55
56 template bgp bgp_template {
57     local as 65122;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65025, 65124 ] then {
62             preference = provider_pref;
63             accept;
64             }
65
66         if bgp_path.first ~ [ 65125 ] then {
67             preference = peer_pref;
68             accept;
69             }
70             reject;
71     };
72
73     export none;
74 }
75
76 protocol bgp node4 from bgp_template {
77     description "BGP session with haw3";
78     neighbor 141.22.28.25 as 65025;
79     export filter export_provider;
80 }
81
82 protocol bgp node5 from bgp_template {
83     description "BGP session with vm4";
84     neighbor 141.22.28.124 as 65124;
85     export filter export_provider;
86 }
87
88 protocol bgp node10 from bgp_template {
```

```
    description "BGP session with node 10";
    neighbor 141.22.28.125 as 65125;
    export filter export_peer;
}
```

Listing 24: Configuration of Node 3

```
table db;

filter zweihundertdran {
bgp_path.prepend(201);
bgp_path.prepend(1337);
bgp_path.prepend(201);
accept;
}


#protocol bgp Brocade {
#
#local as 200;
#neighbor 141.22.27.145 as 300;
#table db;
##export all;
#import all;
#path metric 100;
#export filter zweihundertdran;
#default bgp_local_pref 300;
#next hop self;
#}

#protocol bgp Notebook{
#local as 200;
#neighbor 141.22.26.77 as 100;
#table db;
#export all;
#import all;
#path metric 200;
#export filter zweihundertdran;
#}

protocol bgp hosteurope{
local as 65133;
neighbor 176.28.11.224 as 65131;
table db;
export all;
import all;
multihop 255;
path metric 300;
#export filter zweihundertdran;
}
```

```
44
45
46  /*
47   *  This is an example configuration file.
48   */
49
50  # Yes, even shell-like comments work...
51
52  # Configure logging
53  #log syslog { debug, trace, info, remote, warning, error, auth, fatal, bug };
54  #log stderr all;
55  #log "tmp" all;
56
57  # Override router ID
58  router id 141.22.28.199;
59
60  # You can define your own symbols...
61  #define xyzzy = (120+10);
62  #define '1a-a1' = (30+40);
63
64  # Define a route filter...
65  #filter test_filter {
66  #   if net ~ 10.0.0.0/16 then accept;
67  #   else reject;
68  #}
69
70  #filter sink { reject; }
71  #filter okay { accept; }
72
73  #include "filters.conf";
74
75  # Define another routing table
76  #table testable;
77
78  # Turn on global debugging of all protocols
79  #debug protocols all;
80
81  # The direct protocol automatically generates device routes to
82  # all network interfaces. Can exist in as many instances as you wish
83  # if you want to populate multiple routing tables with device routes.
84  #protocol direct {
85  #   interface "-eth*", "*"; # Restrict network interfaces it works with
86  #}
87
88  # This pseudo-protocol performs synchronization between BIRD's routing
89  # tables and the kernel. If your kernel supports multiple routing tables
90  # (as Linux 2.2.x does), you can run multiple instances of the kernel
91  # protocol and synchronize different kernel tables with different BIRD tables.
92  protocol kernel {
```

```
93  #   learn;          # Learn all alien routes from the kernel
94      persist;        # Don't remove routes on bird shutdown
95      scan time 20;       # Scan kernel routing table every 20 seconds
96  #   import none;        # Default is import all
97      export all;     # Default is export none
98  #   kernel table 5;     # Kernel table to synchronize with (default: main)
99  }
100
101 # This pseudo-protocol watches all interface up/down events.
102 protocol device {
103     scan time 10;       # Scan interfaces every 10 seconds
104 }
105
106 # Static routes (again, there can be multiple instances, so that you
107 # can disable/enable various groups of static routes on the fly).
108 protocol static {
109 table db;
110 #   disabled;       # Disable by default
111 #   table testable;     # Connect to a non-default table
112 #   preference 1000;    # Default preference of routes
113 #   debug { states, routes, filters, interfaces, events, packets };
114 #   debug all;
115 #   route 0.0.0.0/0 via 198.51.100.13;
116 #   route 198.51.100.0/25 reject;
117 #   route 10.0.0.0/8 reject;
118 #   route 10.1.1.0:255.255.255.0 via 198.51.100.3;
119 #   route 10.1.2.0:255.255.255.0 via 198.51.100.3;
120 #   route 10.1.3.0:255.255.255.0 via 198.51.100.4;
121 #   route 10.2.0.0/24 via "arc0";
122 #   route 10.2.0.0/16 via "eth2";
123 #   route 192.168.2.0/24 via "eth2";
124 route 172.29.0.0/16 via "eth2";
125 }
126
127 # Pipe protocol connects two routing tables... Beware of loops.
128 #protocol pipe {
129 #   peer table testable;
130 # Define what routes do we export to this protocol / import from it.
131 #   import all;     # default is all
132 #   export all;     # default is none
133 #   import none;        # If you wish to disable imports
134 #   import filter test_filter;   # Use named filter
135 #   import where source = RTS_DEVICE; # Use explicit filter
136 #}
137
138 # RIP aka Rest In Pieces...
139 #protocol rip MyRIP { # You can also use an explicit name
140 #   preference xyzzy;
141 #   debug all;
```

```
142 #   port 1520;
143 #   period 7;
144 #   infinity 16;
145 #   garbage time 60;
146 #   interface "*" { mode broadcast; };
147 #   honor neighbor;   # To whom do we agree to send the routing table
148 #   honor always;
149 #   honor never;
150 #   passwords {
151 #       password "nazdar";
152 #   };
153 #   authentication none;
154 #   import filter { print "importing"; accept; };
155 #   export filter { print "exporting"; accept; };
156 #}
157 
158 #protocol ospf MyOSPF {
159 #       tick 2;
160 #   rfc1583compat yes;
161 #   area 0.0.0.0 {
162 #       stub no;
163 #       interface "eth*" {
164 #           hello 9;
165 #           retransmit 6;
166 #           cost 10;
167 #           transmit delay 5;
168 #           dead count 5;
169 #           wait 50;
170 #           type broadcast;
171 #           authentication simple;
172 #           password "pass";
173 #       };
174 #       interface "arc0" {
175 #           rx buffer large;
176 #           type nonbroadcast;
177 #           poll 14;
178 #           dead 75;
179 #           neighbors {
180 #               10.1.1.2 eligible;
181 #               10.1.1.4;
182 #           };
183 #           strict nonbroadcast yes;
184 #       };
185 #       interface "xxx0" {
186 #                       passwords {
187 #               password "abc" {
188 #                   id 1;
189 #                   generate to "22-04-2003 11:00:06";
190 #                   accept to "17-01-2004 12:01:05";
```

```
191 #             };
192 #             password "def" {
193 #                 id 2;
194 #                 generate from "22-04-2003 11:00:07";
195 #                 accept from "17-01-2003 12:01:05";
196 #             };
197 #         };
198 #                     authentication cryptographic;
199 #     };
200 #  };
201 #  area 20 {
202 #      stub 1;
203 #      interface "ppp1" {
204 #          hello 8;
205 #          authentication none;
206 #      };
207 #             interface "fr*";
208 #             virtual link 192.168.0.1 {
209 #                     password "sdsdffsdfg";
210 #                     authentication cryptographic;
211 #             };
212 #  };
213 #}
214
215
216 #protocol bgp {
217 #  disabled;
218 #  description "My BGP uplink";
219 #  local as 65000;
220 #  neighbor 198.51.100.130 as 64496;
221 #  multihop;
222 #  hold time 240;
223 #  startup hold time 240;
224 #  connect retry time 120;
225 #  keepalive time 80; # defaults to hold time / 3
226 #  start delay time 5; # How long do we wait before initial connect
227 #  error wait time 60, 300;# Minimum and maximum time we wait after an error (
        when consecutive
228 #             # errors occur, we increase the delay exponentially ...
229 #  error forget time 300; # ... until this timeout expires)
230 #  disable after error;  # Disable the protocol automatically when an error
        occurs
231 #  next hop self;    # Disable next hop processing and always advertise our
        local address as nexthop
232 #  path metric 1;    # Prefer routes with shorter paths (like Cisco does)
233 #  default bgp_med 0; # MED value we use for comparison when none is defined
234 #  default bgp_local_pref 0; # The same for local preference
235 #  source address 198.51.100.14; # What local address we use for the TCP
        connection
```

```
#   password "secret"; # Password used for MD5 authentication
#   rr client;     # I am a route reflector and the neighor is my client
#   rr cluster id 1.0.0.1; # Use this value for cluster id instead of my router id
#   export where source=RTS_STATIC;
#   export filter {
#       if source = RTS_STATIC then {
#           bgp_community = -empty-; bgp_community = add(bgp_community,(65000,5678));
#           bgp_origin = 0;
#           bgp_community = -empty-; bgp_community.add((65000,5678));
#           if (65000,64501) ~ bgp_community then
#               bgp_community.add((0, 1));
#           if bgp_path ~ [= 65000 =] then
#               bgp_path.prepend(65000);
#           accept;
#       }
#       reject;
#   };
#}
#
# Template usage example
#template bgp rr_client {
#   disabled;
#   local as 65000;
#   multihop;
#   rr client;
#   rr cluster id 1.0.0.1;
#}
#
#protocol bgp rr_abcd from rr_client {
#   neighbor 10.1.4.7 as 65000;
#}
```

Listing 25: Configuration of Node 4

```
/*
 *  BIRD configuration file for node 4.
 */

# Configure logging
log "/var/log/bird.log" all;

# Override router ID
router id 141.22.28.25;

table malicious;

define provider_pref = 75;
define peer_pref = 100;
```

```
15 define customer_pref = 125;
16 define local_pref = 200;
17
18 filter export_provider {
19     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
20         reject;
21
22     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
23         accept;
24
25     reject;
26 }
27
28 filter export_peer {
29     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
30         reject;
31
32     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
33         accept;
34
35     reject;
36 }
37
38 filter export_customer {
39     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
40         accept;
41
42     reject;
43 }
44
45 protocol kernel {
46     scan time 20;     # Scan kernel routing table every 20 seconds
47     import none;      # Default is import all
48     export none;          # Default is export none
49 }
50
51 protocol device {
52     scan time 10;
53 }
54
55 protocol static localAS {
56     description "Static route to simulated own AS.";
57     route 172.27.0.0/16 via "eth1";
```

```
}

template bgp bgp_template{
    local as 65025;
    next hop self;

    import filter {
        if bgp_path.first ~ [ 65123 ] then {
            preference = provider_pref;
            accept;
        }

        if bgp_path.first ~ [ 65125 ] then {
            preference = peer_pref;
            accept;
        }

        if bgp_path.first ~ [ 65121, 65122 ] then {
            preference = customer_pref;
            accept;
        }
        reject;
    };

    export none;
}

protocol bgp node1 from bgp_template {
    description "BGP session with vm1";
    neighbor 141.22.28.121 as 65121;
    export filter export_customer;
}

protocol bgp node2 from bgp_template {
    description "BGP session with vm2";
    neighbor 141.22.28.122 as 65122;
    export filter export_customer;
}

protocol bgp node8 from bgp_template {
    description "BGP session with vm3";
    neighbor 141.22.28.123 as 65123;
    export filter export_provider;
}

protocol bgp node10 from bgp_template {
    description "BGP session with node 10";
    neighbor 141.22.28.125 as 65125;
    export filter export_peer;
```

```
}
```

Listing 26: Configuration of Node 5

```
/*
 * BIRD configuration file of node 5.
 */

log "/var/log/bird.log" all;

router id 141.22.28.124;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came
        from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came
        from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came
        from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came
        from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference =
        customer_pref || preference = local_pref then # route came from provider
        , peer, customer or us
        accept;

    reject;
}

```

```
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.22.0.0/15 via "eth0";
54 }
55
56 template bgp bgp_template {
57     local as 65124;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65132 ] then {
62             preference = provider_pref;
63             accept;
64             }
65
66             if bgp_path.first ~ [ 65123, 65125, 65131 ] then {
67             preference = peer_pref;
68             accept;
69             }
70
71             if bgp_path.first ~ [ 65122 ] then {
72             preference = customer_pref;
73             accept;
74             }
75             reject;
76     };
77
78     export none;
79 }
80
81 protocol bgp node2 from bgp_template {
82     description "BGP session with node2";
83     neighbor 141.22.28.122 as 65122;
84     export filter export_customer;
85 }
86
87 protocol bgp node6 from bgp_template {
88     description "BGP session with node6";
89     neighbor 176.28.11.224 as 65131;
```

```
90      multihop 255;
91      export filter export_peer;
92  }
93
94  protocol bgp node8 from bgp_template {
95      description "BGP session with node8";
96      neighbor 141.22.28.123 as 65123;
97      export filter export_peer;
98  }
99
100 protocol bgp node9 from bgp_template {
101     description "BGP session with node9";
102     neighbor 62.75.143.240 as 65132;
103     multihop 255;
104     export filter export_provider;
105 }
106
107 protocol bgp node10 from bgp_template {
108     description "BGP session with node 10";
109     neighbor 141.22.28.125 as 65125;
110     export filter export_peer;
111 }
```

Listing 27: Configuration of Node 6

```
1   /*
2    *  BIRD configuration file for node 6.
3    */
4
5   # Configure logging
6   #log "/var/log/bird.log" all;
7
8
9   # Override router ID
10  router id 176.28.11.224;
11
12  define provider_pref = 75;
13  define peer_pref = 100;
14  define customer_pref = 125;
15  define local_pref = 200;
16
17  filter export_provider {
18      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
19          reject;
20
21      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
22          accept;
23
```

```
24      reject;
25  }
26
27  filter export_peer {
28      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
29          reject;
30
31      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
32          accept;
33
34      reject;
35  }
36
37  filter export_customer {
38      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
39          accept;
40
41      reject;
42  }
43
44  protocol kernel {
45      scan time 20;     # Scan kernel routing table every 20 seconds
46      import none;      # Default is import all
47      export all;       # Default is export none
48  }
49
50  protocol device {
51      scan time 10;
52  }
53
54  protocol static localAS {
55      description "Static route to simulated own AS.";
56      route 172.31.0.0/16 via "venet0";
57  }
58
59  template bgp bgp_template{
60      local as 65131;
61      next hop self;
62      multihop 255; # added
63
64
65      import filter {
66          if bgp_path.first ~ [ 65132 ] then {
67              preference = provider_pref;
68              accept;
```

```
        }

        if bgp_path.first ~ [ 65124 ] then {
            preference = peer_pref;
            accept;
        }

        if bgp_path.first ~ [ 65133 ] then {
            preference = customer_pref;
            accept;
        }
        reject;
    };

    export none;
}

protocol bgp S4Y from bgp_template {
    description "BGP session with S4Y";
    neighbor 62.75.143.240 as 65132;
    export filter export_provider;
}

protocol bgp FHRechner from bgp_template {
    description "BGP session with FH-Rechner";
    neighbor 141.22.28.199 as 65133;
    export filter export_customer;
}

protocol bgp vm4 from bgp_template {
    description "BGP session with vm4";
    neighbor 141.22.28.124 as 65124;
    export filter export_peer;
}
```

Listing 28: Configuration of Node 8

```
/*
 * BIRD configuration file of node 8.
 */

log "/var/log/bird.log" all;

router id 141.22.28.123;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

```

```
14 filter export_provider {
15     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
16         reject;
17
18     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
19         accept;
20
21     reject;
22 }
23
24 filter export_peer {
25     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
26         reject;
27
28     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
29         accept;
30
31     reject;
32 }
33
34 filter export_customer {
35     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
36         accept;
37
38     reject;
39 }
40
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.20.0.0/15 via "eth0";
54 }
55
56 template bgp bgp_template {
```

```
57      local as 65123;
58      next hop self;
59
60      import filter {
61          if bgp_path.first ~ [ 65126 ] then {
62              preference = provider_pref;
63              accept;
64              }
65
66          if bgp_path.first ~ [ 65124, 65125 ] then {
67              preference = peer_pref;
68              accept;
69              }
70
71              if bgp_path.first ~ [ 65025 ] then {
72              preference = customer_pref;
73              accept;
74              }
75              reject;
76      };
77
78      export none;
79  }
80
81  protocol bgp node4 from bgp_template {
82      description "BGP session with node4";
83      neighbor 141.22.28.25 as 65025;
84      export filter export_customer;
85  }
86
87  protocol bgp node5 from bgp_template {
88      description "BGP session with node5";
89      neighbor 141.22.28.124 as 65124;
90      export filter export_peer;
91  }
92
93  protocol bgp node10 from bgp_template {
94      description "BGP session with node10";
95      neighbor 141.22.28.125 as 65125;
96      export filter export_peer;
97  }
98
99  protocol bgp node11 from bgp_template {
100     description "BGP session with node 11";
101     neighbor 141.22.28.126 as 65126;
102     export filter export_provider;
103 }
```

Listing 29: Configuration of Node 9

```
/*
 *  BIRD configuration file of node 9.
 */

# Configure logging
#log "/var/log/bird.log" all;

# Override router ID
router id 62.75.143.240;

define provider_pref = 75;
define peer_pref = 100;
define customer_pref = 125;
define local_pref = 200;

filter export_provider {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_peer {
    if preference = provider_pref || preference = peer_pref then # route came from provider or peer
        reject;

    if preference = customer_pref || preference = local_pref then # route came from customer or us
        accept;

    reject;
}

filter export_customer {
    if preference = provider_pref || preference = peer_pref || preference = customer_pref || preference = local_pref then # route came from provider, peer, customer or us
        accept;

    reject;
}

```

```
protocol kernel {
    scan time 20;     # Scan kernel routing table every 20 seconds
    import none;      # Default is import all
    export all;       # Default is export none
}

protocol device {
    scan time 10;
}

protocol static localAS {
    description "Static route to simulated own AS.";
    route 172.28.0.0/16 via "venet0";
}

template bgp bgp_template{
    local as 65132;
    next hop self;
    multihop 255;  # added

    import filter {
        if bgp_path.first ~ [ 65126 ] then {
            preference = provider_pref;
            accept;
        }

        if bgp_path.first ~ [ 0 ] then {
            preference = peer_pref;
            accept;
        }

        if bgp_path.first ~ [ 65131, 65130 ] then {
            preference = customer_pref;
            accept;
        }
        reject;
    };

    export none;
}

protocol bgp vm6 from bgp_template {
    description "BGP session with vm6";
    neighbor 141.22.28.126 as 65126;
    export filter export_provider;
}

protocol bgp brocade from bgp_template {
    description "BGP session with brocade";
```

```
92      neighbor 141.22.27.145 as 65130;
93      export filter export_customer;
94  }
95
96  protocol bgp hosteurope from bgp_template {
97      description "BGP session with hosteurope";
98      neighbor 176.28.11.224 as 65131;
99      export filter export_customer;
100 }
```

Listing 30: Configuration of Node 10

```
1   /*
2    * BIRD configuration file of node 10.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.125;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
22  }
23
24  filter export_peer {
25      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
26          reject;
27
28      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
29          accept;
30
31      reject;
32  }
33
34  filter export_customer {
```

```
35      if preference = provider_pref || preference = peer_pref || preference =
            customer_pref || preference = local_pref then # route came from provider
            , peer, customer or us
36          accept;
37
38      reject;
39  }
40
41  protocol kernel {
42      scan time 20;
43      import none;
44      export all;
45  }
46
47  protocol device {
48      scan time 10;
49  }
50
51  protocol static localAS {
52      description "Static route to own AS.";
53      route 172.24.0.0/15 via "eth0";
54  }
55
56  template bgp bgp_template {
57      local as 65125;
58      next hop self;
59
60      import filter {
61              if bgp_path.first ~ [ 65025, 65121, 65122, 65123, 64124, 65126 ]
                    then {
62              preference = peer_pref;
63              accept;
64              }
65      };
66
67      export none;
68  }
69
70  protocol bgp node1 from bgp_template {
71      description "BGP session with node 1";
72      neighbor 141.22.28.121 as 65121;
73      export filter export_peer;
74  }
75
76  protocol bgp node2 from bgp_template {
77      description "BGP session with node 2";
78      neighbor 141.22.28.122 as 65122;
79      export filter export_peer;
80  }
```

```
81
82  protocol bgp node4 from bgp_template {
83      description "BGP session with node 4";
84      neighbor 141.22.28.25 as 65025;
85      export filter export_peer;
86  }
87
88  protocol bgp node5 from bgp_template {
89      description "BGP session with node 5";
90      neighbor 141.22.28.124 as 65124;
91      export filter export_peer;
92  }
93
94  protocol bgp node8 from bgp_template {
95      description "BGP session with node8";
96      neighbor 141.22.28.123 as 65123;
97      export filter export_peer;
98  }
99
100 protocol bgp node11 from bgp_template {
101     description "BGP session with node11";
102     neighbor 141.22.28.126 as 65126;
103     export filter export_peer;
104 }
```

Listing 31: Configuration of Node 11

```
1   /*
2    * BIRD configuration file of node 11.
3    */
4
5   log "/var/log/bird.log" all;
6
7   router id 141.22.28.126;
8
9   define provider_pref = 75;
10  define peer_pref = 100;
11  define customer_pref = 125;
12  define local_pref = 200;
13
14  filter export_provider {
15      if preference = provider_pref || preference = peer_pref then # route came
            from provider or peer
16          reject;
17
18      if preference = customer_pref || preference = local_pref then # route came
            from customer or us
19          accept;
20
21      reject;
```

```
22 }
23
24 filter export_peer {
25     if preference = provider_pref || preference = peer_pref then # route came
           from provider or peer
26         reject;
27
28     if preference = customer_pref || preference = local_pref then # route came
           from customer or us
29         accept;
30
31     reject;
32 }
33
34 filter export_customer {
35     if preference = provider_pref || preference = peer_pref || preference =
           customer_pref || preference = local_pref then # route came from provider
           , peer, customer or us
36         accept;
37
38     reject;
39 }
40
41 protocol kernel {
42     scan time 20;
43     import none;
44     export all;
45 }
46
47 protocol device {
48     scan time 10;
49 }
50
51 protocol static localAS {
52     description "Static route to own AS.";
53     route 172.26.0.0/16 via "eth0";
54 }
55
56 template bgp bgp_template {
57     local as 65126;
58     next hop self;
59
60     import filter {
61         if bgp_path.first ~ [ 65125 ] then {
62             preference = peer_pref;
63             accept;
64             }
65
66             if bgp_path.first ~ [ 65123, 65132 ] then {
```

```
67              preference = customer_pref;
68              accept;
69              }
70              reject;
71      };
72
73      export none;
74  }
75
76  protocol bgp node10 from bgp_template {
77      description "BGP session with node10";
78      neighbor 141.22.28.125 as 65125;
79      export filter export_peer;
80  }
81
82  protocol bgp node9 from bgp_template {
83      description "BGP session with node9";
84      neighbor 62.75.143.240 as 65132;
85      export filter export_customer;
86      multihop 255;
87  }
88
89  protocol bgp node8 from bgp_template {
90      description "BGP session with node 8";
91      neighbor 141.22.28.123 as 65123;
92      export filter export_customer;
93  }
```