
SDN Internet Router (sir) Documentation

Release 1

David Barroso

December 10, 2016

1	Features	3
1.1	Analytics	3
1.2	Variables	8
2	Architecture	11
2.1	Local Architecture	11
2.2	Global Architecture	13
3	API v1.0 Documentation	15
3.1	API	15
3.2	Endpoints	16
4	Use Cases	27
4.1	A commodity switch as a Peering Router	27
4.2	A Software Defined Content Delivery Network	29
5	How To: Generic Router	31
5.1	Scenario	31
5.2	Enabling pmacct	32
5.3	Enabling SIR agent	37
6	How To: EOS	41
6.1	Scenario	41
6.2	Deploying SIR Manually	43
6.3	Deploying SIR Automatically	45
6.4	Configuring EOS	46
6.5	Accessing SIR	46
6.6	Logging	47
6.7	fib_optimizer	48

The SDN Internet Router, abbreviated SIR, is an agent that you can add to your router. The agent exposes information that your router can't expose by itself like the BGP table, traffic per BGP prefix or traffic per ASN. This data is provided both via a WebUI and an API to access this data.

The agent is vendor agnostic as it gathers data using both BGP and netflow/sflow/ipfix. This means it can be attached to any router or switch that supports those protocols.

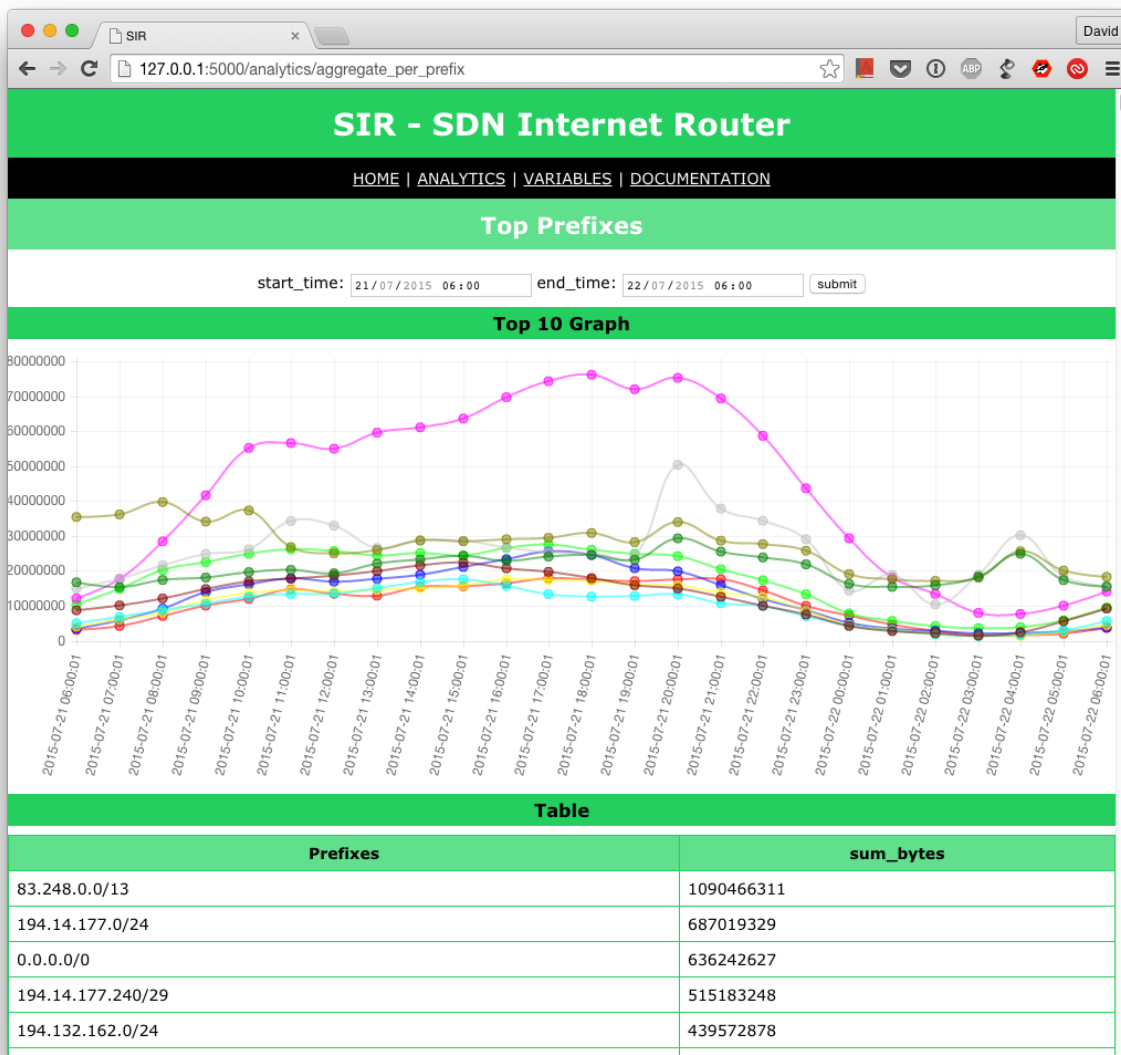
Features

The agent enhances your router adding some nice features that are exposed both via the WebUI and the API. You can see some examples of the features in the following pages. Although the examples are shown using the WebUI all the features are also available via the API. Actually, features are built first for the API so there is the possibility that you can do something with the API but not with the WebUI.

1.1 Analytics

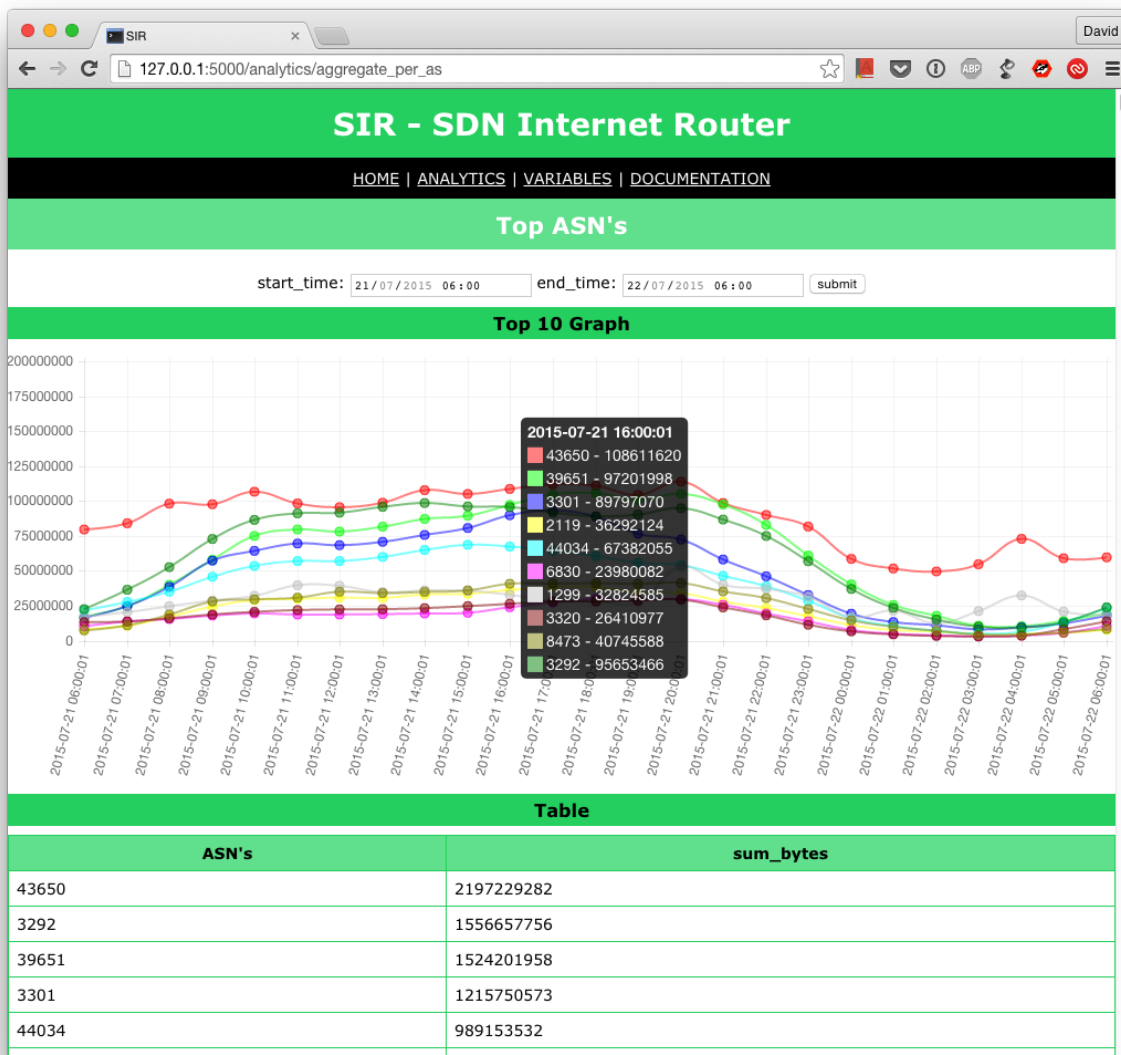
1.1.1 Top Prefixes

You can get the top N prefixes for a given period of time.



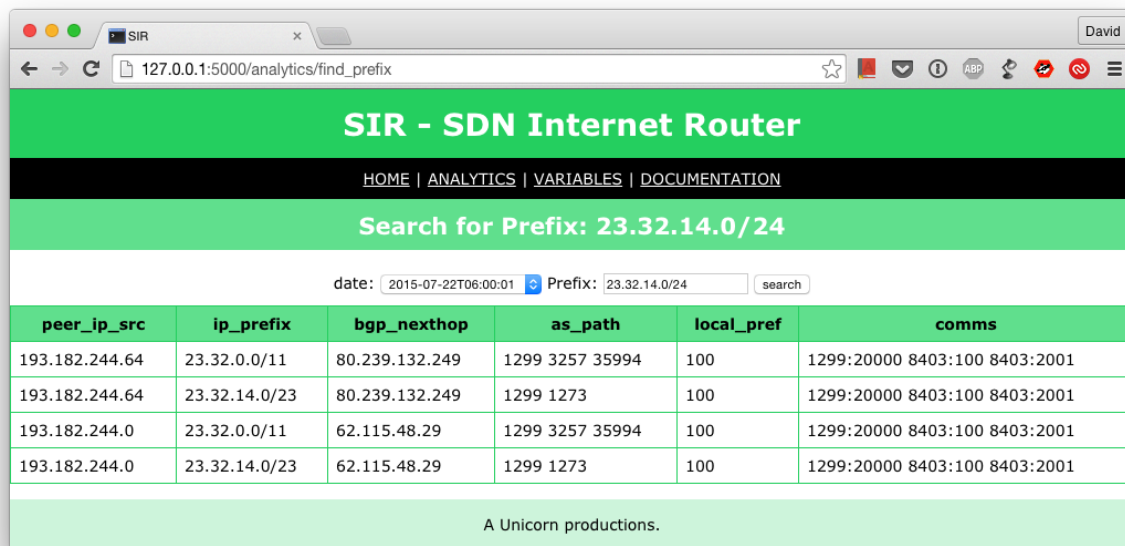
1.1.2 Top ASN's

You can get the top ASN's for a given period of time.



1.1.3 Search Prefixes

You can search for any prefix of any prefix-length and get back all the possible routes that you could choose to reach it. You will not get only the longest prefix match but all the possible matches.



The screenshot shows a web browser window with the URL `127.0.0.1:5000/analytics/find_prefix`. The page title is "SIR - SDN Internet Router". Below the title is a navigation bar with links: [HOME](#) | [ANALYTICS](#) | [VARIABLES](#) | [DOCUMENTATION](#). The main heading is "Search for Prefix: 23.32.14.0/24". Below this is a search bar with "date: 2015-07-22T06:00:01" and "Prefix: 23.32.14.0/24" with a "search" button. The results are displayed in a table with 6 columns: `peer_ip_src`, `ip_prefix`, `bgp_nexthop`, `as_path`, `local_pref`, and `comms`. The table contains 4 rows of data. At the bottom of the page, it says "A Unicorn productions."

peer_ip_src	ip_prefix	bgp_nexthop	as_path	local_pref	comms
193.182.244.64	23.32.0.0/11	80.239.132.249	1299 3257 35994	100	1299:20000 8403:100 8403:2001
193.182.244.64	23.32.14.0/23	80.239.132.249	1299 1273	100	1299:20000 8403:100 8403:2001
193.182.244.0	23.32.0.0/11	62.115.48.29	1299 3257 35994	100	1299:20000 8403:100 8403:2001
193.182.244.0	23.32.14.0/23	62.115.48.29	1299 1273	100	1299:20000 8403:100 8403:2001

Note: `peer_ip_src` is useful if you are sending data to the agent from different routers as it shows the IP of the router that provided those prefixes. In this particular example we have two different routers sending data to the same agent.

1.1.4 Search ASN's

You can search for any ASN and see all the prefixes that either traverse and/or originated in that ASN. Set the option `origin_as_only` to indicate if you want to see only prefixes originated in that ASN or if you want prefixes transiting that ASN as well.

SIR - SDN Internet Router

HOME | ANALYTICS | VARIABLES | DOCUMENTATION

Search for ASN: 345

date: 2015-07-22T06:00:01 origin_as_only: True ASN: 345 search

peer_ip_src	ip_prefix	bgp_nexthop	as_path	local_pref	comms
193.182.244.64	55.3.0.0/16	80.239.132.249	1299 209 721 27064 575 306 345	100	1299:25000 8403:100 8403:2001
193.182.244.64	156.112.250.0/24	80.239.132.249	1299 209 721 27065 6025 345	100	1299:20000 8403:100 8403:2001
193.182.244.0	55.3.0.0/16	62.115.48.29	1299 209 721 27064 575 306 345	100	1299:25000 8403:100 8403:2001
193.182.244.0	156.112.250.0/24	62.115.48.29	1299 209 721 27065 6025 345	100	1299:20000 8403:100 8403:2001

A Unicorn productions.

Note: `peer_ip_src` is useful if you are sending data to the agent from different routers as it shows the IP of the router that provided those prefixes. In this particular example we have two different routers sending data to the same agent.

1.1.5 Offloaded Traffic

For a given period of time and a number of prefixes N , returns the amount of traffic matched if you had only the top N prefixes installed in the FIB vs the Total Traffic.

SIR - SDN Internet Router

HOME | ANALYTICS | VARIABLES | DOCUMENTATION

Offloaded Traffic

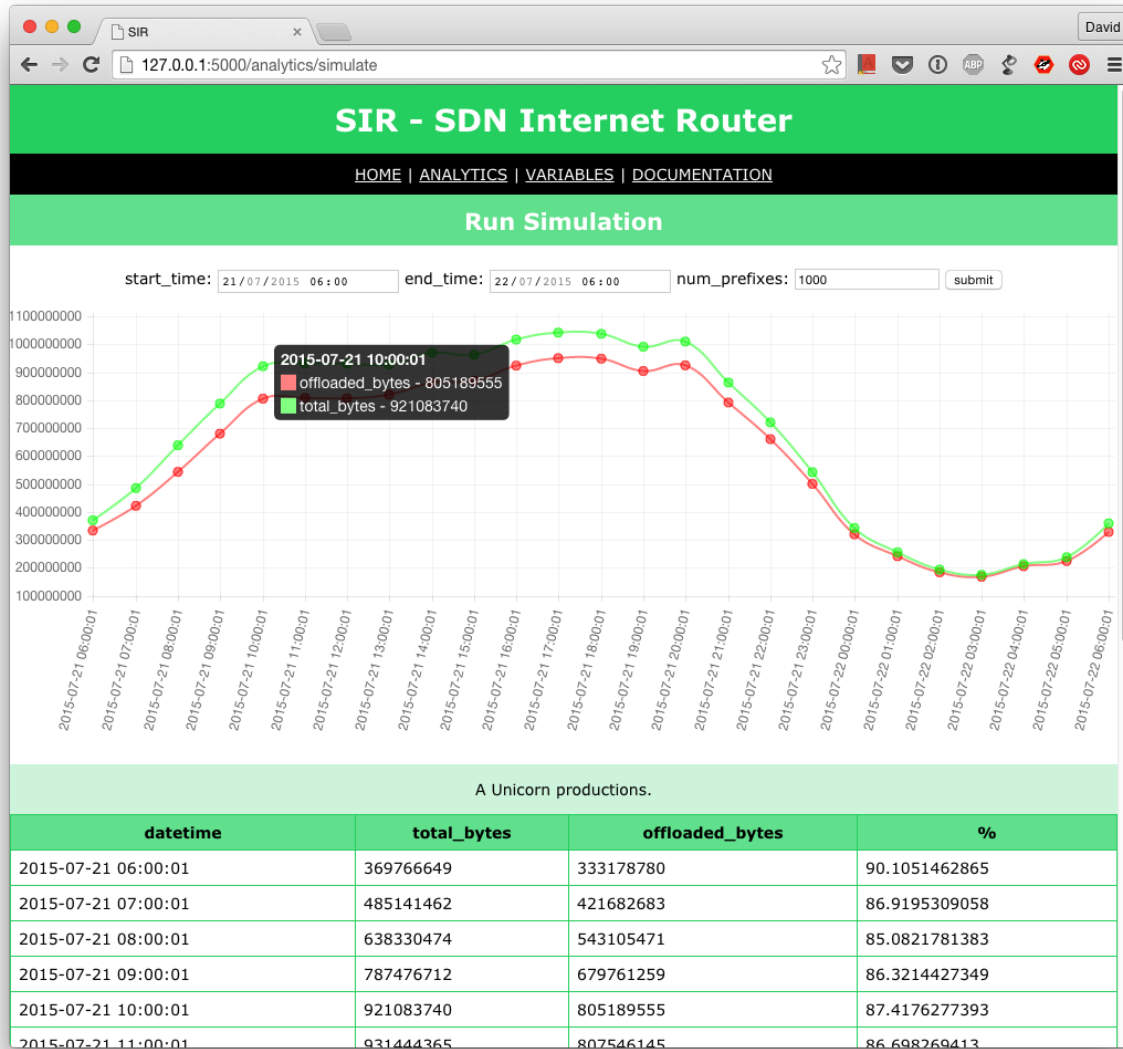
start_time: 21/07/2015 06:00 end_time: 22/07/2015 06:00 num_prefixes: 1000 submit

Total	Offloaded	%
16,918,693,346	15,064,278,917	89.0392573996

A Unicorn productions.

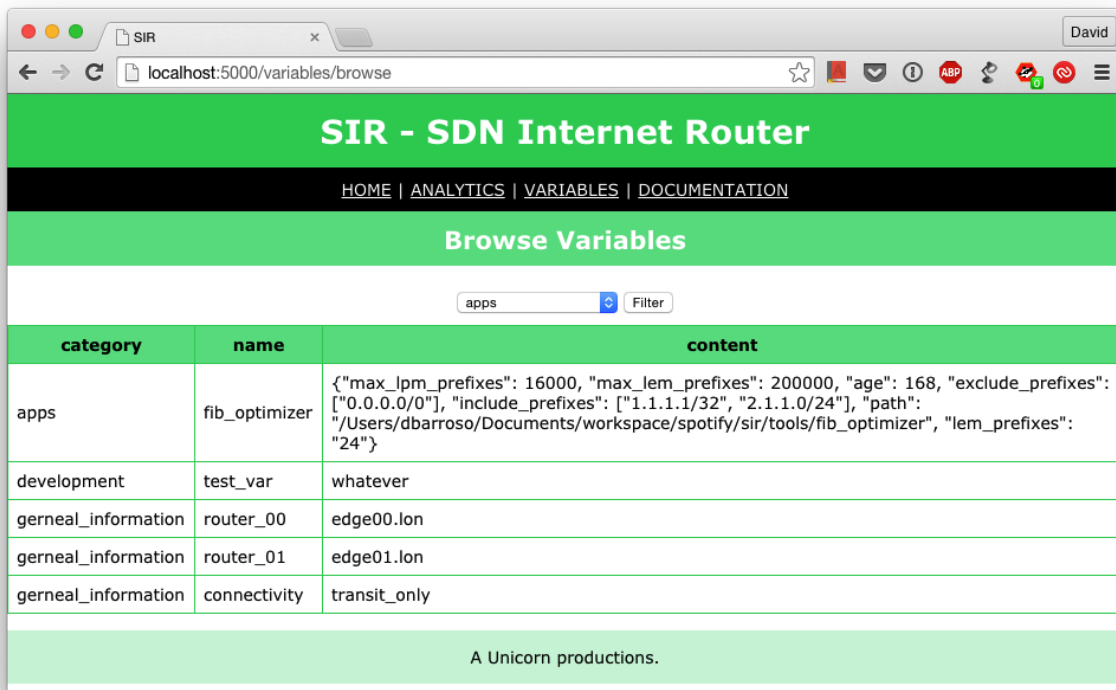
1.1.6 Simulate FIB Optimization

For a given period of time and a number of prefixes N, simulates how the switch/router would perform if you had only the top N prefixes installed in the FIB vs the Total Traffic.



1.2 Variables

You can store in the agent and retrieve later any arbitrary data that you want. This is useful if you plan to write applications that consumes data via the API to perform some actions. For example, in the example below you can see a variable called `fib_optimizer`. The data in the content is the actual configuration for that application. The application will retrieve the Top N prefixes via the API and install them in a commodity switch. Thanks to these feature we can tweak the behavior of that application from a centralized management tool while allowing the device to run independently.

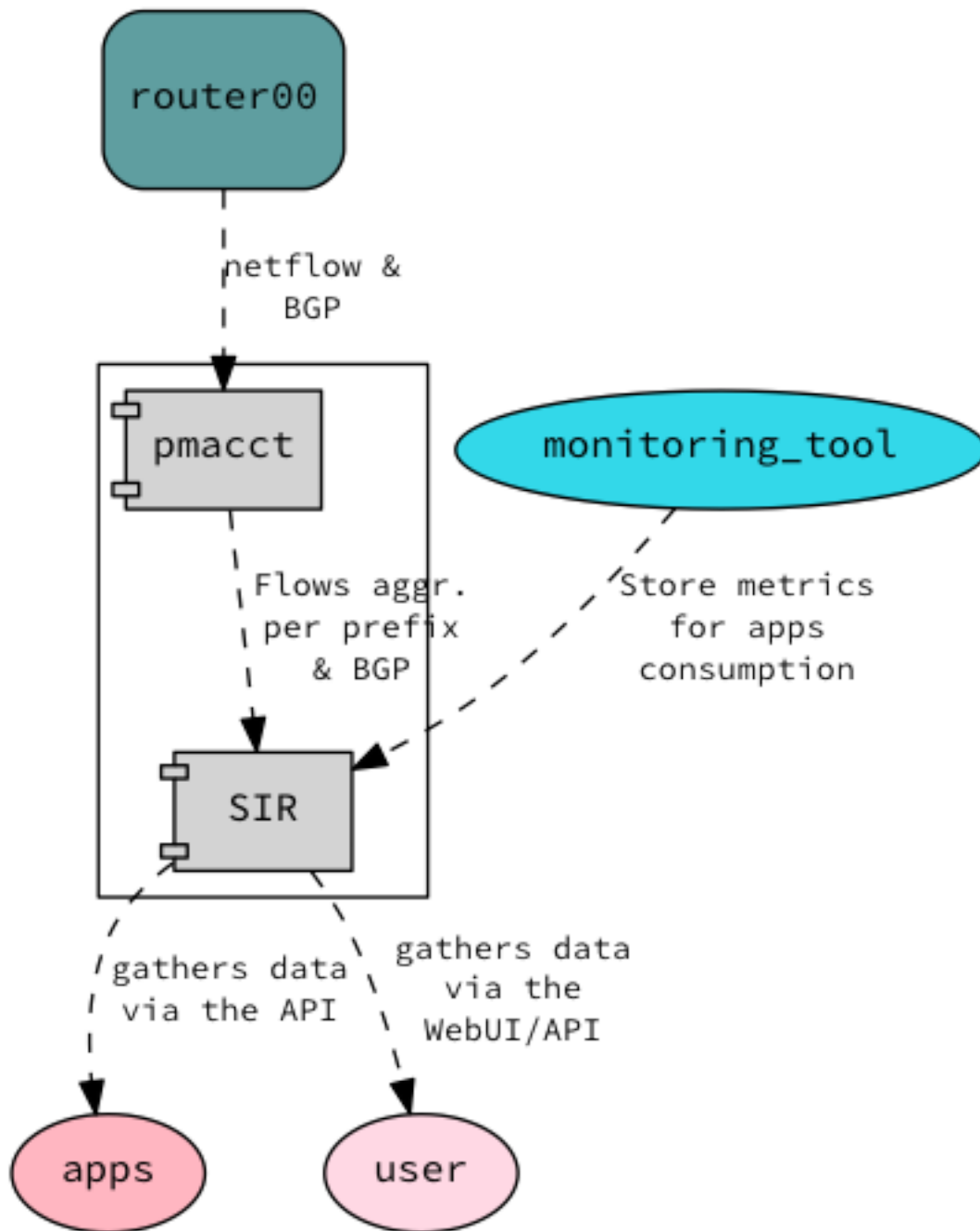


Use this in any way you want. This will not influence how the agent performs. It's just a convenient way for you to store/retrieve some data for some applications that you might be running using the SIR API.

Architecture

2.1 Local Architecture

The local architecture of the SIR agent is quite simple:



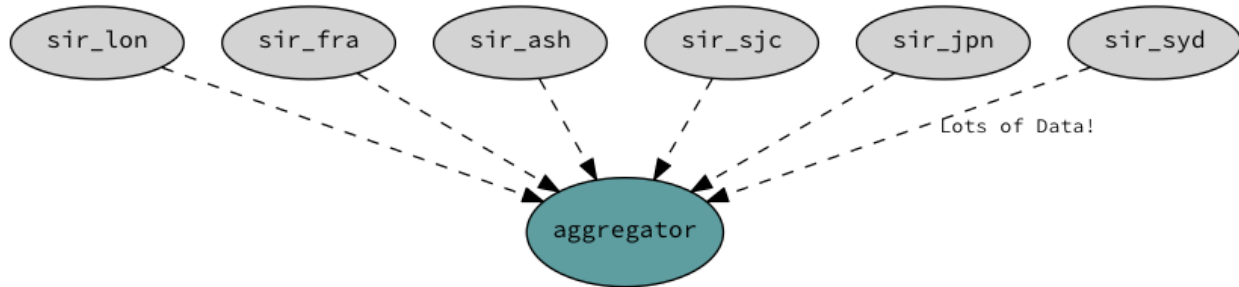
There are two main components:

- **pmacct** - This is a flow collector that is going to get BGP and flow information from your routers.
- **SIR** - This code is going to do some analytics with the data collected by pmacct and expose it to apps/users via the WebUI and the API.

Users and applications can consume the data exposed by SIR and take peering or traffic engineering decisions. Applications and users can also store data in the agent. This is useful if you want to influence the behavior of some other applications later on.

2.2 Global Architecture

Although each agent is meant to run independently the API allows you to combine/aggregate data from multiple locations and take global decisions.



Once you have a global view of your global network and your traffic patterns you can do things like:

- Send users to a location where you know you are peering with the network that user belongs to.
- Redirect users to another POP where his/her network is reachable.
- Redirect users based on cost, latency, bandwidth or any other metric.
- Predict your global traffic patterns and optimize your capacity.
- Build reports to see who you should be peering with.

API v1.0 Documentation

3.1 API

3.1.1 Variables

When reading this documentation you will find variables in two forms:

- **<variable>**: Variables that are between <> have to be replaced by their values in the URL. For example, `/api/v1.0/variables/categories/<category>` will turn into `/api/v1.0/variables/categories/my_category`.
- **variable**: Variables that are NOT enclosed by <>:
- If the method is a GET variables that are between <> have to be replaced by their values in the URL. For example, `/api/v1.0/variables/categories/<category>` will turn into `/api/v1.0/variables/categories/my_category`.
- If the method is a POST or a PUT variables that are between <> have to be sent as a JSON object.

3.1.2 Responses

All the responses from the agent will be in JSON format and will include three sections:

- **meta**: Meta information about the response. For example, *request_time*, *length* of the response or if there was any *error*.
- **parameters**: The parameters used for the call.
- **result**: The result of the call or a description of the error if there was any.

For example, for the following call:

```
/api/v1.0/analytics/top_prefixes?limit_prefixes=10&start_time=2015-07-13T14:00&end_time=2015-07-14T14:00
```

You will get the following response:

```
1 {
2   "meta": {
3     "error": false,
4     "length": 10,
5     "request_time": 11.99163
6   },
7   "parameters": {
8     "end_time": "2015-07-14T14:00",
```

```
9      "exclude_net_masks": false,
10      "limit_prefixes": 10,
11      "net_masks": "20,24",
12      "start_time": "2015-07-13T14:00"
13  },
14  "result": [
15      {
16          "as_dst": 43650,
17          "key": "194.14.177.0/24",
18          "sum_bytes": 650537594
19      },
20      ...
21      {
22          "as_dst": 197301,
23          "key": "80.71.128.0/20",
24          "sum_bytes": 5106731
25      }
26  ]
27  }
```

3.2 Endpoints

3.2.1 Analytics Endpoint

/api/v1.0/analytics/top_prefixes

GET

Description Retrieves TOP prefixes sorted by the amount of bytes that they consumed during the specified period of time.

Arguments

- **start_time**: Mandatory. Datetime in unicode string following the format %Y-%m-%dT%H:%M:%S. Starting time of the range.
- **end_time**: Mandatory. Datetime in unicode string following the format %Y-%m-%dT%H:%M:%S. Ending time of the range.
- **limit_prefixes**: Optional. Number of top prefixes to retrieve.
- **net_masks**: Optional. List of prefix lengths to filter in or out.
- **exclude_net_masks**: Optional. If set to any value it will return prefixes with a prefix length not included in net_masks. If set to 0 it will return only prefixes with a prefix length included in net_masks. Default is 0.
- **filter_proto**: Optional. If you don't set it you will get both IPv4 and IPv6 prefixes. If you set it to 4 you will get only IPv4 prefixes. Otherwise, if you set it to 6 you will get IPv6 prefixes.

Returns A list of prefixes sorted by sum_bytes. The attribute sum_bytes is the amount of bytes consumed during the specified time.

Examples

```
http://127.0.0.1:5000/api/v1.0/analytics/top_prefixes?limit_prefixes=10&start_time=2015-07-13T14:00&
http://127.0.0.1:5000/api/v1.0/analytics/top_prefixes?limit_prefixes=10&start_time=2015-07-13T14:00&
http://127.0.0.1:5000/api/v1.0/analytics/top_prefixes?limit_prefixes=10&start_time=2015-07-13T14:00&
```

/api/v1.0/analytics/top_asns

GET

Description Retrieves TOP ASN's sorted by the amount of bytes that they consumed during the specified period of time.

Arguments

- **start_time**: Mandatory. Datetime in unicode string following the format %Y-%m-%dT%H:%M:%S. Starting time of the range.
- **end_time**: Mandatory. Datetime in unicode string following the format %Y-%m-%dT%H:%M:%S. Ending time of the range.

Returns A list of ASN's sorted by sum_bytes. The attribute sum_bytes is the amount of bytes consumed during the specified time.

Examples

```
http://127.0.0.1:5000/api/v1.0/analytics/top_asns?start_time=2015-07-13T14:00&end_time=2015-07-14T14:00&
```

/api/v1.0/analytics/find_prefix/<prefix>/<prefix_length>

GET

Description Finds all prefixes in the system that contains the prefix <prefix>/<prefix_length>

Arguments

- **<<prefix>>**: Mandatory. IP prefix of the network you want to find.
- **<<prefix_length>>**: Mandatory. Prefix length of the network you want to find.
- **date**: Mandatory. Datetime in unicode string following the format '%Y-%m-%dT%H:%M:%S'.

Returns It will return a dictionary where keys are the IP's of the BGP peers peering with SIR. Each one will have a list of prefixes that contain the prefix queried.

Examples

```
$ curl http://127.0.0.1:5000/api/v1.0/analytics/find_prefix/192.2.3.1/32\?date\=2015-07-22T05:00:01
{
  "meta": {
    "error": false,
    "length": 2,
    "request_time": 1.88076
  },
  "parameters": {
    "date": "2015-07-22T05:00:01",
    "prefix": "192.2.3.1/32"
  },
  "result": {
    "193.182.244.0": [
      {
        "as_path": "1299 3356",
        "bgp nexthop": "62.115.48.29",
        "comms": "1299:20000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "192.2.0.0/16",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.0"
      }
    ],
    "193.182.244.64": [
      {
        "as_path": "1299 3356",
        "bgp nexthop": "80.239.132.249",
        "comms": "1299:20000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "192.2.0.0/16",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.64"
      }
    ]
  }
}
```

/api/v1.0/analytics/find_prefixes_asn/<asn>

GET

Description Finds all prefixes in the system that traverses and/or originates in <asn>

Arguments

- **<<asn>>**: Mandatory. ASN you want to query.
- **date**: Mandatory. Datetime in unicode string following the format ' %Y-%m-%dT%H:%M:%S'.
- **origin_only**: Optional. If set to any value it will return only prefixes that originate in <asn>.

Returns It will return a dictionary where keys are the IP's of the BGP peers peering with SIR. Each one will have a list of prefixes that traverses and/or originates in <asn>

Examples

```
curl http://127.0.0.1:5000/api/v1.0/analytics/find_prefixes_asn/345?date\=2015-07-22T05:00:01\&orig
{
  "meta": {
    "error": false,
    "length": 2,
    "request_time": 1.15757
  },
  "parameters": {
    "asn": "345",
    "origin_only": "1",
    "date": "2015-07-22T05:00:01"
  },
  "result": {
    "193.182.244.0": [
      {
        "as_path": "1299 209 721 27064 575 306 345",
        "bgp nexthop": "62.115.48.29",
        "comms": "1299:25000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "55.3.0.0/16",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.0"
      },
      {
        "as_path": "1299 209 721 27065 6025 345",
        "bgp nexthop": "62.115.48.29",
        "comms": "1299:20000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "156.112.250.0/24",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.0"
      }
    ],
    "193.182.244.64": [
      {
        "as_path": "1299 209 721 27064 575 306 345",
        "bgp nexthop": "80.239.132.249",
        "comms": "1299:25000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "55.3.0.0/16",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.64"
      },
      {
        "as_path": "1299 209 721 27065 6025 345",
        "bgp nexthop": "80.239.132.249",
        "comms": "1299:20000 8403:100 8403:2001",
        "event_type": "dump",
        "ip_prefix": "156.112.250.0/24",
        "local_pref": 100,
        "origin": 0,
        "peer_ip_src": "193.182.244.64"
      }
    ]
  }
}
```

```
}  
}  
}
```

3.2.2 Variables Endpoint

/api/v1.0/variables

GET

Description Retrieves all the variables in the system.

Arguments

Returns A list of all the variables.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables
```

POST

Description You can create a variable from the CLI with curl like this:

```
curl -i -H "Content-Type: application/json" -X POST -d '{"name": "test_var", "content": "whatever", '}
```

Arguments

- **content:** Content of the variable.
- **category:** Category of the variable.
- **name:** Name of the variable.
- **extra_vars:** Use this field to add extra data to your variable. It is recommended to use a JSON string.

Returns The variable that was just created.

Examples

/api/v1.0/variables/categories

GET

Description Retrieves all the categories in the system.

Arguments

Returns A list of all the categories.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables/categories
```

/api/v1.0/variables/categories/<category>

GET

Description Retrieves all the variables the belong to <category> in the system.

Arguments

- **<category>**: Category you want to query.

Returns A list of variables belonging to <category>.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables/categories/<category>
```

/api/v1.0/variables/categories/<category>/<name>

GET

Description Retrieves the variable with <name> and <category>.

Arguments

- **<category>**: Category of the variable you want to retrieve.
- **<name>**: Name of the variable you want to retrieve.

Returns A list of variables belonging to <category>.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables/categories/<category>/<name>
```

PUT

Description This API call allows you to modify all of some of the values of a variable. For example, you can update the name and the extra_vars of a variable with the following command:

```
1 curl -i -H "Content-Type: application/json" -X PUT -d '{"name": "test_varc", "extra_vars": "{\'my_pa
2 {
3   "meta": {
4     "error": false,
5     "length": 1,
```

```
6     "request_time": 0.0055
7   },
8   "parameters": {
9     "categories": "development",
10    "name": "test_vara"
11  },
12  "result": [
13    {
14      "category": "development",
15      "content": "whatever",
16      "extra_vars": "{my_param1: my_value1, my_param2: my_value2}",
17      "name": "test_varc"
18    }
19  ]
20 }
```

Arguments

- **category**: Optional. New category.
- **content**: Optional. New content.
- **name**: Optional. New name.
- **<name>**: Name of the variable you want to modify.
- **<category>**: Category of the variable you want to modify.
- **extra_vars**: Optional. New extra_vars.

Returns The variable with the new data.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables/categories/<category>/<name>
```

DELETE

Description Deletes a variable. For example:

```
1  curl -i -X DELETE http://127.0.0.1:5000/api/v1.0/variables/categories/deveopment/test_vara HTTP/1.0
2  {
3    "meta": {
4      "error": false,
5      "length": 0,
6      "request_time": 0.0016
7    },
8    "parameters": {
9      "categories": "deveopment",
10     "name": "test_vara"
11   },
12   "result": []
13 }
```

Arguments

- **<category>**: Category of the variable you want to delete.
- **<name>**: Name of the variable you want to delete.

Returns An empty list.

Examples

```
http://127.0.0.1:5000/api/v1.0/variables/categories/<category>/<name>
```

3.2.3 Pmacct Endpoint

/api/v1.0/pmacct/dates

GET

Description Retrieves all the available dates in the system.

Arguments

Returns A list of all the available dates in the system.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/dates
```

/api/v1.0/pmacct/flows

GET

Description Retrieves all the available dates in the system.

Arguments

- **start_time**: Mandatory. Datetime in unicode string following the format '`%Y-%m-%dT%H:%M:%S`'. Starting time of the range.
- **end_time**: Mandatory. Datetime in unicode string following the format '`%Y-%m-%dT%H:%M:%S`'. Ending time of the range.

Returns A list of all the available dates in the system.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/flows?limit_prefixes=10&start_time=2015-07-14T14:00&end_time=2015-07-14T14:00
http://127.0.0.1:5000/api/v1.0/pmacct/flows?limit_prefixes=10&start_time=2015-07-13T14:00&end_time=2015-07-14T14:00
```

/api/v1.0/pmacct/bgp_prefixes

GET

Description Retrieves all the BGP prefixes in the system.

Arguments

- **date:** Mandatory. Datetime in unicode string following the format ' %Y-%m-%dT%H:%M:%S' .

Returns A list of all the available BGP prefixes in the system.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/bgp_prefixes?date=2015-07-16T11:00:01
```

/api/v1.0/pmacct/raw_bgp

GET

Description Retrieves the BGP raw data from pmacct. That includes AS PATH, local-pref, communities, etc....

Warning: Do it only if need it. If you have the full feed this can return hundreds of MB of data.

Arguments

- **date:** Mandatory. Datetime in unicode string following the format ' %Y-%m-%dT%H:%M:%S' .

Returns The raw data from pmacct.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/raw_bgp?date=2015-07-16T11:00:01
```

/api/v1.0/pmacct/purge_bgp

GET

Description Deletes all the BGP data that is older than `older_than`.

Arguments

- **older_than:** Mandatory. Datetime in unicode string following the format ' %Y-%m-%dT%H:%M:%S' .

Returns The list of files containing BGP data that was deleted.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/purge_bgp?older_than=2015-07-29T13:00:01
```

/api/v1.0/pmacct/purge_flows**GET**

Description Deletes all the flows that are older than `older_than`.

Arguments

- **older_than**: Mandatory. Datetime in unicode string following the format `'%Y-%m-%dT%H:%M:%S'`.

Returns The flows that were deleted.

Examples

```
http://127.0.0.1:5000/api/v1.0/pmacct/purge_flows?older_than=2015-07-29T13:00:01
```

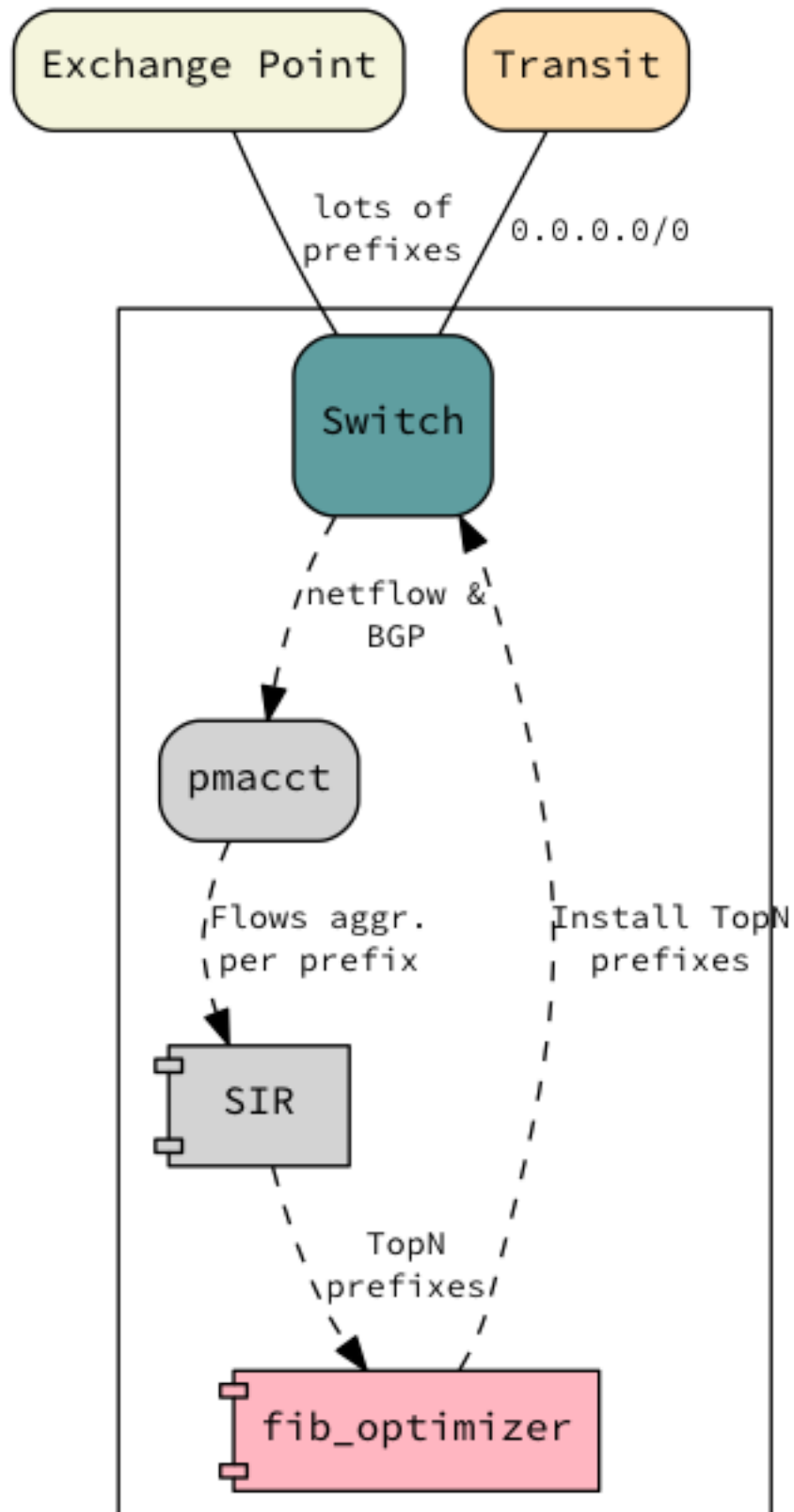
Use Cases

Here are some use cases for SIR. I will elaborate more on this use cases and add new ones soon enough. If you have any don't hesitate to let me know.

4.1 A commodity switch as a Peering Router

This is a simple and interesting use case. Once you start collecting data in your peering routers you will quickly realize that you don't use more than 30.000 prefixes daily. So why do you need a big and expensive router to hold the full routing table? Wouldn't be better and easier (and most probably cheaper) to hold the full routing table in the RIB and just install the routes you need in the FIB?

With SIR you can easily see how many routes you need to offload your traffic and which routes you will need. Once you have this information it's just a matter of instructing your switch to accept those prefixes and keep the rest in memory.



You can do that in different ways, being SRD the most stable and simplest way. Soon I will share an app that leverages

on SIR to convert a cheap Arista 7280 switch into a peering router.

This was the original purpose when I started SIR. Although the scope of SIR has changed (before it was a monolithic app and now it's an agent that provides information via an API) you can see a presentation and a podcast about this topic in the following links:

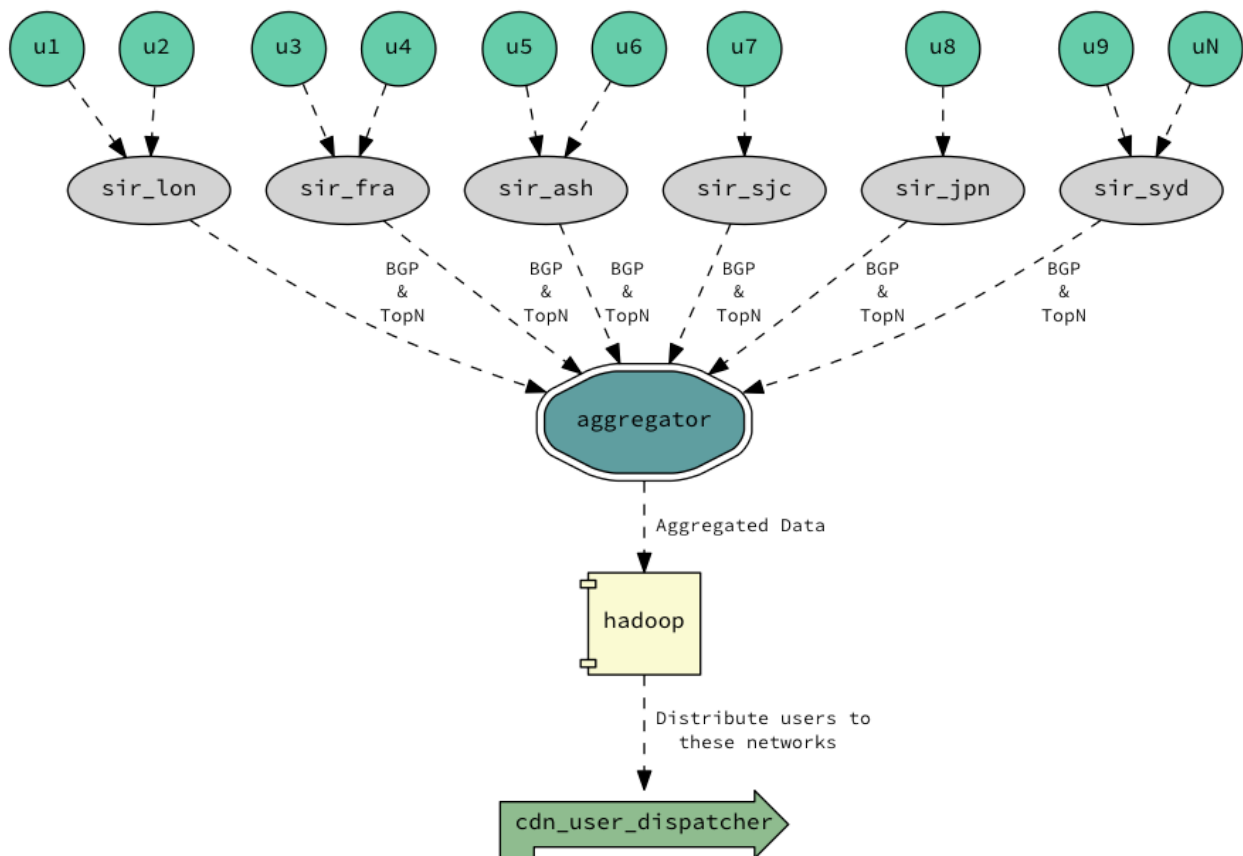
- <http://youtu.be/o1njanXhQqM>
- <http://blog.ipospace.net/2015/01/sdn-router-spotify-on-software-gone-wild.html>

4.2 A Software Defined Content Delivery Network

As mentioned in the global architecture, you can aggregate the data exposed by SIR. This aggregated data can tell you many things:

- Which networks are connecting to yours at what times.
- How much throughput they need.
- From where you can deliver your content to those networks.

This aggregated data could be sent to hadoop to decide things like from which POP to serve your content to your users:



You could also add metrics from other sources. Metrics like:

- Cost of each link.
- Latency.
- Load of each site.

- Reliability.

Once all the data is in Hadoop you could try to analyze your global traffic pattern and metrics and redistribute users to:

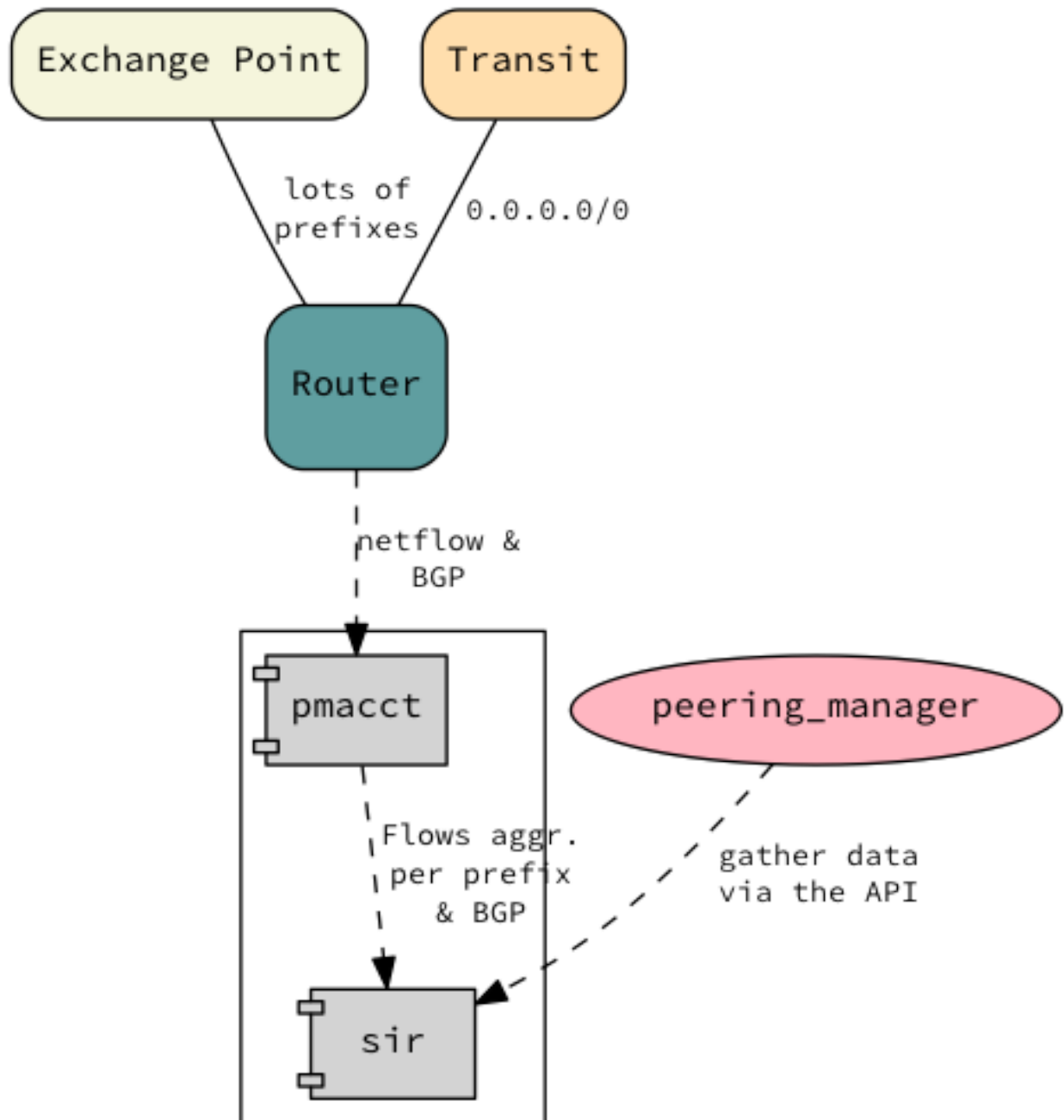
- Minimize transit costs.
- Maximize capacity usage.
- Improve user experience.

How To: Generic Router

5.1 Scenario

We want to deploy the SIR agent connecting with a generic router to add some visibility to our edge router. This visibility will allow us to understand who we are sending traffic to and who to peer with.

We can't run code in the generic router so we are going to deploy the agent on a linux machine in the Datacenter. This shouldn't be an issue as both BGP and netflow/sflow can be sent to a device that is not directly connected to the router.



Per se this is not very interesting, however, if you have a large network with several POPs, having this agent on each location would give you a nice way of gathering and aggregating data.

5.2 Enabling pmacct

First, we have to install `pmacct` with JSON, IPv6 and SQLite3 support. Compiling `pmacct` for your specific distro is out of the scope of this document but you can find below some instructions on how to do it for a debian based distro.

5.2.1 Compiling pmacct

These are some of the dependencies that you might need:

```
apt-get install libpcap-dev libsqlite3-dev libjansson-dev zlib1g-dev
```

And this is how to compile pmacct:

```
$ wget http://www.pmacct.net/pmacct-1.5.1.tar.gz
--2015-07-23 09:25:46-- http://www.pmacct.net/pmacct-1.5.1.tar.gz
Connecting to 127.0.0.1:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 874563 (854K) [application/x-gzip]
Saving to: 'pmacct-1.5.1.tar.gz'

100%[=====>] 874,563

2015-07-23 09:25:46 (9.80 MB/s) - 'pmacct-1.5.1.tar.gz' saved [874563/874563]

$ tar xzf pmacct-1.5.1.tar.gz
$ cd pmacct-1.5.1
$ ./configure --enable-sqlite3 --enable-jansson --enable-ipv6 --prefix=/pmacct-1.5.1
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal-1.4... missing
checking for working autoconf... missing
checking for working automake-1.4... missing
checking for working autoheader... missing
checking for working makeinfo... missing
checking for gcc... gcc
checking whether the C compiler (gcc ) works... yes
checking whether the C compiler (gcc ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking OS... Linux
checking hardware... x86_64
checking for ranlib... ranlib
checking whether to enable debugging compiler options... no
checking whether to relax compiler optimizations... no
checking whether to disable linking against shared objects... no
checking for dlopen... no
checking for dlopen in -ldl... yes
checking for gmake... no
checking for make... make
checking whether make sets ${MAKE}... (cached) yes
checking for __progname... yes
checking for extra flags needed to export symbols... --export-dynamic
checking for static inline... yes
checking endianness... little
checking unaligned accesses... ok
checking whether to enable L2 features... yes
checking whether to enable IPv6 code... yes
checking for inet_pton... yes
checking for inet_ntop... yes
checking whether to enable IPv4-mapped IPv6 sockets ... yes
checking whether to enable IP prefix labels... checking default locations for pcap.h... found in /usr
checking default locations for libpcap... no
```

```

checking for pcap_dispatch in -lpcap... yes
checking for pcap_setnonblock in -lpcap... yes
checking packet capture type... linux
checking whether to enable MySQL support... checking how to run the C preprocessor... gcc -E
no
checking whether to enable PostgreSQL support... no
checking whether to enable MongoDB support... no
checking whether to enable SQLite3 support... yes
checking default locations for libsqlite3... not found
checking for sqlite3_open in -lsqlite3... yes
checking default locations for sqlite3.h... found in /usr/include
checking whether to enable RabbitMQ/AMQP support... no
checking whether to enable GeoIP support... no
checking whether to enable Jansson support... yes
checking default locations for Jansson library... not found
checking for json_object in -ljansson... yes
checking default locations for jansson.h... found in /usr/include
checking for ANSI C header files... no
checking for sys/wait.h that is POSIX.1 compatible... yes
checking for getopt.h... yes
checking for sys/select.h... yes
checking for sys/time.h... yes
checking for u_int64_t in sys/types.h... yes
checking for u_int32_t in sys/types.h... yes
checking for u_int16_t in sys/types.h... yes
checking for u_int8_t in sys/types.h... yes
checking for uint64_t in sys/types.h... no
checking for uint32_t in sys/types.h... no
checking for uint16_t in sys/types.h... no
checking for uint8_t in sys/types.h... no
checking whether to enable 64bit counters... yes
checking whether to enable multithreading in pmacct... yes
checking whether to enable ULOG support... no
checking return type of signal handlers... void
checking for strlcpy... no
checking for vsnprintf... no
checking for setproctitle... no
checking for mallopt... no

PLATFORM ..... : x86_64
OS ..... : Linux 3.13.0-34-generic
COMPILER ..... : gcc
CFLAGS ..... : -O2 -g -O2
LIBS ..... : -ljansson -lsqlite3 -lpcap -ldl -lm -lz -lpthread
SERVER_LIBS ... : -lnfprobe_plugin -Lnfprobe_plugin/ -lsfprobe_plugin -Lsfprobe_plugin/ -lbgp -Lbgp/ -
LDFLAGS ..... : -Wl,--export-dynamic

Now type 'make' to compile the source code.

Are you willing to get in touch with other pmacct users?
Join the pmacct mailing-list by sending a message to pmacct-discussion-subscribe@pmacct.net

Need for documentation and examples?
Read the README file or go to http://wiki.pmacct.net/

updating cache ./config.cache
creating ./config.status

```

```

creating Makefile
creating src/Makefile
creating src/nfprobe_plugin/Makefile
creating src/sfprobe_plugin/Makefile
creating src/bgp/Makefile
creating src/tee_plugin/Makefile
creating src/isis/Makefile
creating src/bmp/Makefile
$ make
... (output omitted for clarity)
$ sudo make install
... (output omitted for clarity)

```

5.2.2 Configuring pmacct

To configure pmacct you will need to know the IP the router will use as source IP for both netflow and BGP. Once you know, paste the following configuration in the file `/pmacct-1.5.1/etc/pmacct.conf`:

```

$ cd /pmacct-1.5.1
$ sudo mkdir etc
$ sudo vi etc/pmacct.conf
daemonize: true

plugins: sqlite3[simple]

sql_db[simple]: /pmacct-1.5.1/output/pmacct.db
sql_refresh_time[simple]: 3600
sql_history[simple]: 60m
sql_history_roundoff[simple]: h
sql_table[simple]: acct
sql_table_version[simple]: 9

aggregate: dst_net, dst_mask, dst_as

bgp_daemon: true
bgp_daemon_ip: $ROUTER_SRC_IP
bgp_daemon_max_peers: 1
bgp_table_dump_file: /pmacct-1.5.1/output/bgp-$peer_src_ip-%Y_%m_%dT%H_%M_%S.txt
bgp_table_dump_refresh_time: 3600

nfacctd_as_new: bgp
nfacctd_net: bgp
nfacctd_port: 9999
nfacctd_ip: $ROUTER_SRC_IP
nfacctd_time_new: true

```

Warning: Don't forget to replace `$ROUTER_SRC_IP` with the IP of your router will use for both netflow and BGP.

Now it's time to setup the database:

```

$ sudo mkdir output
$ sudo sqlite3 output/pmacct.db << EOF
CREATE TABLE 'acct' (
  'tag'      INT(8) NOT NULL DEFAULT 0,
  'class_id' CHAR(16) NOT NULL DEFAULT ' ',

```

```

'mac_src'          CHAR(17) NOT NULL DEFAULT '0:0:0:0:0:0',
'mac_dst'          CHAR(17) NOT NULL DEFAULT '0:0:0:0:0:0',
'vlan' INT(4) NOT NULL DEFAULT 0,
'as_src'           INT(8) NOT NULL DEFAULT 0,
'as_dst'           INT(8) NOT NULL DEFAULT 0,
'ip_src'           CHAR(15) NOT NULL DEFAULT '0.0.0.0',
'ip_dst'           CHAR(15) NOT NULL DEFAULT '0.0.0.0',
'mask_dst'         INTEGER(1) NOT NULL DEFAULT 0,
'port_src'         INT(4) NOT NULL DEFAULT 0,
'port_dst'         INT(4) NOT NULL DEFAULT 0,
'tcp_flags'        INT(4) NOT NULL DEFAULT 0,
'ip_proto'         CHAR(6) NOT NULL DEFAULT 0,
'tos' INT(4) NOT NULL DEFAULT 0,
'packets'          INT NOT NULL,
'bytes' BIGINT NOT NULL,
'flows' INT NOT NULL DEFAULT 0,
'stamp_inserted'    DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
'stamp_updated'    DATETIME,
'column'           peer_as_srcINT(8) NOT NULL DEFAULT 0,
'peer_as_dst'      INT(8) NOT NULL DEFAULT 0,
'peer_as_src'      INT(8) NOT NULL DEFAULT 0,
'peer_dst_ip'      TEXT NOT NULL DEFAULT '0.0.0.0',
PRIMARY KEY(tag,class_id,mac_src,mac_dst,vlan,as_src,as_dst,ip_src,ip_dst,mask_dst,port_src,port_dst);
);
CREATE TABLE 'variables' (
  'name' TEXT,
  'content' TEXT,
  'category' TEXT,
  PRIMARY KEY(name,category)
);
);

CREATE INDEX acct_idx1 ON acct(stamp_updated);
CREATE INDEX acct_idx2 ON acct(stamp_updated, as_dst);
CREATE INDEX acct_idx3 ON acct(stamp_updated, ip_dst, mask_dst);

CREATE INDEX variables_idx1 ON variables(category);
EOF

```

Finally, it's just a matter of starting pmacct:

```
$ sudo /pmacct-1.5.1/sbin/nfacctd -f /pmacct-1.5.1/etc/pmacct.conf
```

5.2.3 Configuring the Router

For the example we are going to use an ASR. Adapt the config for your network and your device. Configuring the ASR is relatively easy, you only have to configure netflow to send the flows that you want to process and BGP to send the prefixes you want to use for the aggregation. Here is an example:

```

flow exporter-map SIR
version v9
  options interface-table timeout 60
  template data timeout 60
!
transport udp 9999
source Loopback0
destination $PMACCT_IP
!

```



```

flow monitor-map SIR-FMM
  record ipv4
  exporter SIR
  cache timeout active 60
  cache timeout inactive 15
!
sampler-map SIR
  random 1 out-of 10000

interface HundredGigE0/0/0/1
  flow ipv4 monitor SIR-FMM sampler SIR egress

route-policy PASS
  pass
end-policy

route-policy BLOCK
  drop
end-policy

router bgp $AS
  neighbor $PMACCT_IP
  remote-as $AS
  description SIR
  update-source Loopback0
  address-family ipv4 unicast
    route-policy BLOCK in
    route-policy PASS out

```

Warning: Don't forget to replace \$PMACCT_IP with the IP of the server where you are running pmacct and \$AS with your own AS.

Note: If you want you can configure several routers pointing to the same SIR agent. The only thing you have to change is in pmacct's configuration the parameter `bgp_daemon_max_peers` to the number of routers that you are going to configure.

5.3 Enabling SIR agent

This is super easy to do. First install using pip:

```

$ sudo pip install SIR
You are using pip version 6.0.8, however version 7.1.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting SIR
  Downloading SIR-0.10.tar.gz (79kB)
    100% |#####| 81kB 1.2MB/s
Collecting flask (from SIR)
  Using cached Flask-0.10.1.tar.gz
Collecting ipaddress (from SIR)
  Using cached ipaddress-1.0.14-py27-none-any.whl
Collecting PyYAML (from SIR)
  Using cached PyYAML-3.11.tar.gz
Collecting Werkzeug>=0.7 (from flask->SIR)

```

```
Using cached Werkzeug-0.10.4-py2.py3-none-any.whl
Collecting Jinja2>=2.4 (from flask->SIR)
  Downloading Jinja2-2.8-py2.py3-none-any.whl (263kB)
    100% |#####| 266kB 1.1MB/s
Collecting itsdangerous>=0.21 (from flask->SIR)
  Using cached itsdangerous-0.24.tar.gz
Collecting MarkupSafe (from Jinja2>=2.4->flask->SIR)
  Using cached MarkupSafe-0.23.tar.gz
Installing collected packages: MarkupSafe, itsdangerous, Jinja2, Werkzeug, PyYAML, ipaddress, flask,
Running setup.py install for MarkupSafe
  building 'markupsafe._speedups' extension
  clang -fno-strict-aliasing -fno-common -dynamic -I/usr/local/include -I/usr/local/opt/sqlite/inc
  clang -bundle -undefined dynamic_lookup -L/usr/local/lib -L/usr/local/opt/sqlite/lib build/temp.m
Running setup.py install for itsdangerous

Running setup.py install for PyYAML
  checking if libyaml is compilable
  clang -fno-strict-aliasing -fno-common -dynamic -I/usr/local/include -I/usr/local/opt/sqlite/inc
  build/temp.macosx-10.10-x86_64-2.7/check_libyaml.c:2:10: fatal error: 'yaml.h' file not found
  #include <yaml.h>
           ^
  1 error generated.
  libyaml is not found or a compiler error: forcing --without-libyaml
  (if libyaml is installed correctly, you may need to
   specify the option --include-dirs or uncomment and
   modify the parameter include_dirs in setup.cfg)

Running setup.py install for flask
Running setup.py install for SIR
Successfully installed Jinja2-2.8 MarkupSafe-0.23 PyYAML-3.11 SIR-0.10 Werkzeug-0.10.4 flask-0.10.1
```

Now configure SIR. Edit file settings.py:

```
DATABASE = '/pmacct-1.5.1/output/pmacct.db'           # Path to the db
BGP_FOLDER = '/pmacct-1.5.1/output'                   # Path to the folder where BGP info is
DEBUG = False                                         # Set to True only if you are trying to develop
SECRET_KEY = 'My_super_duper_secret_key'             # Secret key. Keep it secret.
BIND_IP = '0.0.0.0'                                  # IP you want to bind the service to
PORT= 8080                                           # Port you want to bind the service to
```

Note: You can place the settings.py file anywhere you like. However, you will have to set the environment variable `SIR_SETTINGS` pointing towards it so it can be found. For example, export `SIR_SETTINGS=/path/to/settings.py`.

Now you need an application server like UWSGI or gunicorn to run the application. There is tons of documentation out there on how to run flask application. I suggest you to start [here](#). For the sake of testing we can use gunicorn like this:

```
$ sudo pip install gunicorn
You are using pip version 6.0.8, however version 7.1.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting gunicorn
  Using cached gunicorn-19.3.0-py2.py3-none-any.whl
Installing collected packages: gunicorn
  Compiling /private/var/folders/d9/9h8nsvsd2_vgt9y1l73jhhvr0000gn/T/pip-build-FcSaUJ/gunicorn/gunic
```

```
Successfully installed gunicorn-19.3.0
$ sudo gunicorn sir.agent:app
[2015-07-30 13:21:30 +0200] [46008] [INFO] Starting gunicorn 19.3.0
[2015-07-30 13:21:30 +0200] [46008] [INFO] Listening at: http://127.0.0.1:8000 (46008)
[2015-07-30 13:21:30 +0200] [46008] [INFO] Using worker: sync
[2015-07-30 13:21:30 +0200] [46017] [INFO] Booting worker with pid: 46017
```

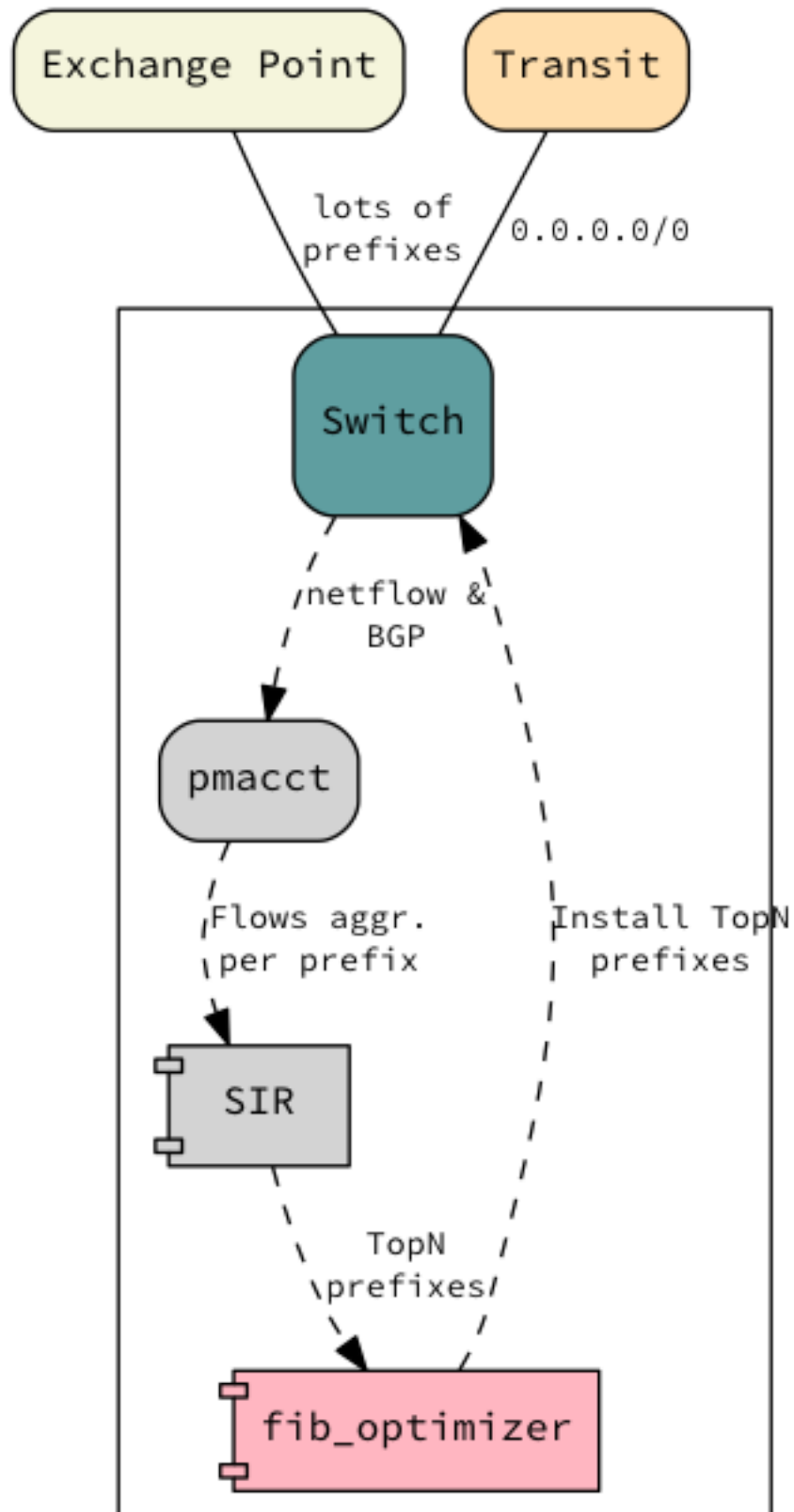
At this point you should be able to access SIR at <http://127.0.0.1:8000>.

How To: EOS

6.1 Scenario

We want to deploy the SIR agent inside an Arista switch. All the code will be run inside the same switch (no external server will be needed). To run the application and present it to the user we will use the same application/HTTP server that the eAPI is using.

This will give us extra visibility on how we are using the Arista switch and do smart things like optimizing the fib or take peering decisions.



6.2 Deploying SIR Manually

If you want, you can deploy SIR manually. If you prefer to do it automatically go to the section *Deploying SIR Automatically*.

Warning: Note that rebooting the switch will uninstall pmacct and SIR. The data will still be there but the software has to be installed after every reboot. That's why the automatic deployment is recommended.

6.2.1 pmacct

For simplicity I am providing both pmacct and the dependencies (jansson) precompiled. If you prefer you can compile both yourself but that is outside the scope of this document. There are no changes from the original sources which you can get in the following links:

- [pmacct](#)
- [jansson](#)

Getting pmacct

First you have to get all the related configuration files and copy them to the correct location:

```
lab#bash

Arista Networks EOS shell

[dbarroso@lab ~]$ cd /tmp/
[dbarroso@lab tmp]$ sudo ip netns exec ns-mgmtVRF wget https://sdn-internet-router-sir.readthedocs.io/en/latest/_static/eos_files.tar.gz
--2015-07-28 13:30:35-- https://sdn-internet-router-sir.readthedocs.io/en/latest/_static/eos_files.tar.gz
Resolving sdn-internet-router-sir.readthedocs.org... 162.209.114.75
Connecting to sdn-internet-router-sir.readthedocs.org|162.209.114.75|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1341140 (1.3M) [application/octet-stream]
Saving to: `eos_files.tar.gz'

100%[=====] 1341140 833 KB/s

2015-07-28 13:30:37 (833 KB/s) - `eos_files.tar.gz' saved [1341140/1341140]

[dbarroso@lab tmp]$ tar xvzf eos_files.tar.gz
eos_files/
eos_files/._DS_Store
eos_files/.DS_Store
eos_files/pmacct/
eos_files/sir_nginx.conf
eos_files/sir_settings.py
eos_files/sir_uwsgi.ini
eos_files/pmacct/._DS_Store
eos_files/pmacct/.DS_Store
eos_files/pmacct/libjansson.so.4
eos_files/pmacct/pmacct
eos_files/pmacct/pmacct.conf
eos_files/pmacct/pmacct.db
eos_files/pmacct/sfacctd
```

```
[dbarroso@lab tmp]$ sudo cp eos_files/pmacct/pmacct.conf /etc/
[dbarroso@lab tmp]$ sudo cp eos_files/pmacct/pmacct /usr/bin/
[dbarroso@lab tmp]$ sudo cp eos_files/pmacct/sfacctd /usr/sbin/
[dbarroso@spine01 tmp]$ sudo cp eos_files/pmacct/libjansson.so.4 /usr/lib/
[dbarroso@lab tmp]$ sudo mkdir -p /mnt/drive/sir/output/bgp/
```

If the database doesn't exist yet, now you can just copy the empty database:

```
[dbarroso@lab tmp]$ sudo cp eos_files/pmacct/pmacct.db /mnt/drive/sir/output/
```

And finally, start pmacct:

```
[dbarroso@lab tmp]$ sudo immortalize --log=/var/log/pmacct.log --daemonize /usr/sbin/sfacctd -f /etc,
```

6.2.2 SIR

Installing SIR

Installing SIR is very easy. You can install it by using PIP as any other python package:

```
lab#bash

Arista Networks EOS shell

[dbarroso@lab ~]$ cd /tmp
[dbarroso@lab tmp]$ sudo ip netns exec ns-mgmtVRF pip install SIR
Downloading/unpacking SIR
  Downloading SIR-0.10.tar.gz (79kB): 79kB downloaded
  Running setup.py (path:/tmp/pip_build_root/SIR/setup.py) egg_info for package SIR
...
Successfully installed SIR flask ipaddress Werkzeug Jinja2 itsdangerous MarkupSafe
Cleaning up...
```

Enabling SIR

Instead of running our own HTTP and/or application server we are just going to use the services that EOS is already using for their eAPI. For that you just have to copy a couple of configuration files that were provided in the same tar file you downloaded before:

```
[dbarroso@lab tmp]$ sudo cp eos_files/sir_uwsgi.ini /etc/uwsgi/
[dbarroso@lab tmp]$ sudo cp eos_files/sir_nginx.conf /etc/nginx/external_conf/
```

Now, copy the configuration file for SIR:

```
[dbarroso@lab tmp]$ sudo cp eos_files/sir_settings.py /mnt/drive/sir/settings.py
```

Starting the application server and SIR

To start SIR you just have to start application server:

```
[dbarroso@lab tmp]$ sudo SIR_SETTINGS='/mnt/drive/sir/settings.py' immortalize --daemonize --log=/var
```

And finally you have to restart the HTTP server back in the EOS CLI. Assuming you already have the eAPI enabled:


```
[dbarroso@lab tmp]$ exit
logout
lab#conf
lab(config)#management api http-commands
lab(config-mgmt-api-http-cmds)#shut
lab(config-mgmt-api-http-cmds)#no shut
```

6.3 Deploying SIR Automatically

If you want, you can deploy SIR automatically. If you prefer to do it manually go to the section [Deploying SIR Manually](#).

6.3.1 Requirements

SIR will use the same HTTP and application server than EOS' eAPI so make sure it's running.

6.3.2 Extensions

To install SIR and all of its dependencies automatically we are going to use EOS extensions (SWIX packages). This will allow us to deploy all the files and do some post-install operations like reconfigure the HTTP/application servers, start pmacct, initialize the database, etc. Everything is done automatically so sit back and enjoy :)

First, let's get the extensions (make sure you are getting the latest versions, check the releases in the [github page](#)):

```
# Get into the management VRF if needed
peer00.lab#routing-context vrf mgmtVRF
peer00.lab(vrf:mgmtVRF)#copy https://github.com/dbarrosop/sir/releases/download/v0.17/sir-0.17-1.noarch.swix
Copy completed successfully.
peer00.lab(vrf:mgmtVRF)#copy https://github.com/dbarrosop/sir/releases/download/v0.17/pmacct_sir-0.1-1.noarch.swix
Copy completed successfully.
```

Now we just have to install the extensions:

```
peer00.lab(vrf:mgmtVRF)#extension sir-0.17-1.noarch.swix
peer00.lab(vrf:mgmtVRF)#extension pmacct_sir-0.1-1.noarch.swix
peer00.lab(vrf:mgmtVRF)#show extensions
```

Name	Version/Release	Status	extension
pmacct_sir-0.1-1.noarch.swix	0.1/1	A, I	1
sir-0.17-1.noarch.swix	0.17/1	A, I	1

A: available | NA: not available | I: installed | NI: not installed | F: forced

And that's it!

Warning: If you reboot your switch the changes will be lost. To make them permanent execute `copy installed-extensions boot-extensions`

6.4 Configuring EOS

No matter if you did the manual or the automated deployment, you have to configure EOS to peer with pmacct and send sFlow data:

```
[dbarroso@lab tmp]$ exit
logout
lab# conf

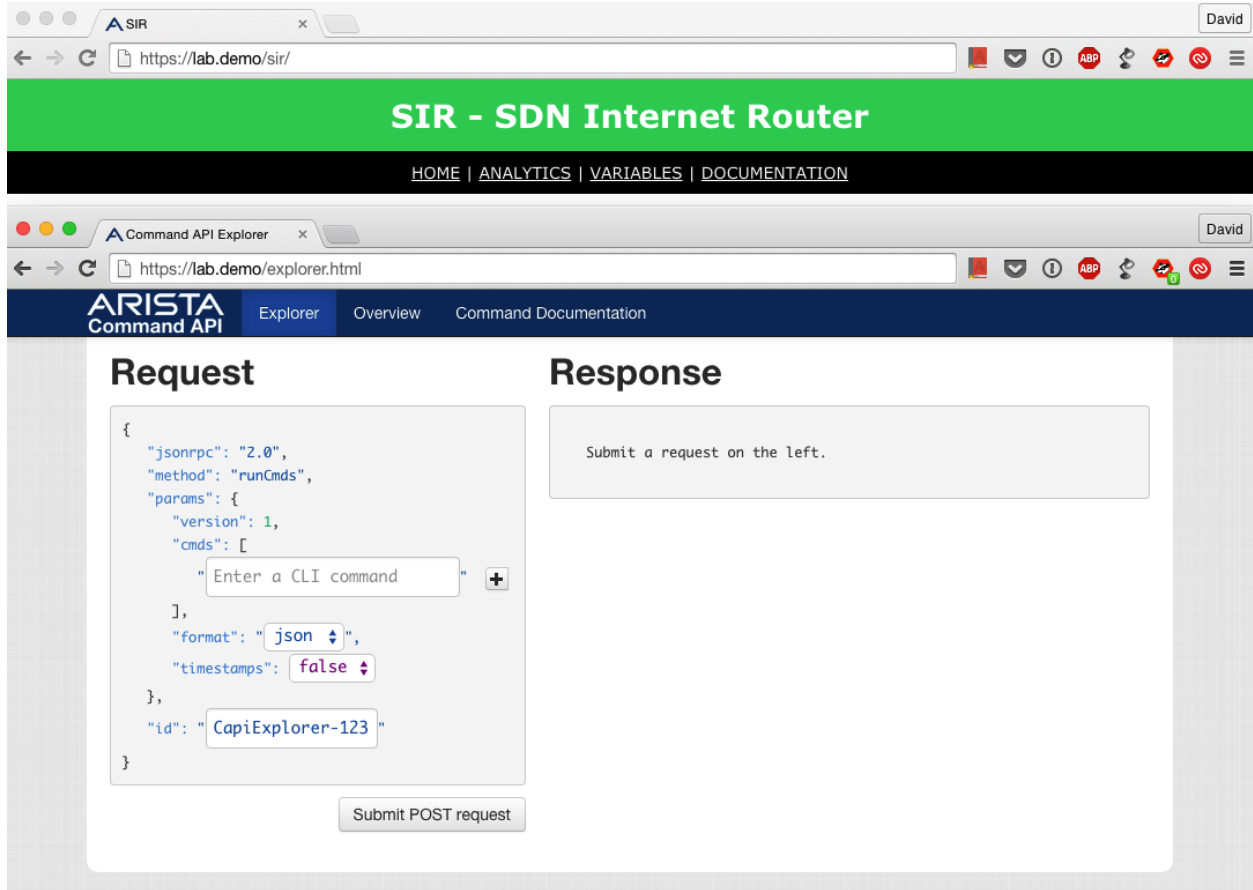
sflow sample dangerous 10000
sflow polling-interval 1
sflow destination 127.0.0.9 9999
sflow source-interface $SOURCE_INTERFACE
sflow run

router bgp $AS
  neighbor 127.0.0.9 transport remote-port 1179
  neighbor 127.0.0.9 update-source $SOURCE_INTERFACE
  neighbor 127.0.0.9 remote-as $AS
  neighbor 127.0.0.9 description "SIR/pmacct"
  neighbor 127.0.0.9 maximum-routes 12000
```

Warning: Don't forget to replace `$SOURCE_INTERFACE` with the source-interface you want to use to connect from your device to the agent and `$AS` with your own AS.

6.5 Accessing SIR

To access SIR you just have to go the same URL you would access the eAPI and add `/sir/`. For example, if the URL to access the eAPI is `https://lab.demo/` you can access SIR with `https://lab.demo/sir/`.



6.6 Logging

Both SIR and pmacct write logs into files:

- `pmacct - /var/log/pmacct.log`
- `sir - /var/log/sir.uwsgi.log`

Just access the files using common linux tools like `tail` or `less`:

```
lab#bash sudo tail /var/log/sir.uwsgi.log
*** Operational MODE: single process ***
added /usr/lib/python2.7/site-packages/ to pythonpath.
WSGI app 0 (mountpoint='') ready in 1 seconds on interpreter 0xf8463420 pid: 5702 (default app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 5702, cores: 1)
[pid: 5702|app: 0|req: 1/1] ::ffff:10.210.20.240 () {44 vars in 713 bytes} [Wed Jul 29 12:25:20 2015]
[pid: 5702|app: 0|req: 2/2] ::ffff:10.210.20.240 () {44 vars in 695 bytes} [Wed Jul 29 12:25:20 2015]
[pid: 5702|app: 0|req: 3/3] ::ffff:10.210.20.240 () {44 vars in 733 bytes} [Wed Jul 29 12:25:20 2015]
[pid: 5702|app: 0|req: 4/4] ::ffff:10.210.20.240 () {44 vars in 696 bytes} [Wed Jul 29 12:25:20 2015]
[pid: 5702|app: 0|req: 5/5] ::ffff:10.210.20.240 () {44 vars in 694 bytes} [Wed Jul 29 12:25:20 2015]
```

6.7 fib_optimizer

One of the main use cases of SIR is using a switch for peering purposes. Traditionally routers has been used for that purpose as switches had limited routing table. However, with SIR and the `fib_optimizer` app you can use instead a switch and bring down costs. For more information about this use case go to the section *A commodity switch as a Peering Router*.

For more details about the `fib_optimizer` go to the github repo where the `fib_optimizer` lives. For instructions on how to easily deploy it within EOS keep reading.

Warning: Disclaimer: Use the following instructions as reference. Adapt the configuration for your network.

6.7.1 Installing the fib_optimizer

To install the `fib_optimizer` we are going to use SWIX packages to simplify the operations:

```
# Go to the management VRF if needed
peer00.lab#routing-context vrf mgmtVRF
peer00.lab(vrf:mgmtVRF)#copy https://github.com/dbarrosop/pySIR/releases/download/v0.45/pySIR-0.45-1.noarch.swix
Copy completed successfully.
peer00.lab(vrf:mgmtVRF)#copy https://github.com/dbarrosop/sir_apps/releases/download/v0.1/fib_optimizer-0.1-1.noarch.swix
Copy completed successfully.
peer00.lab(vrf:mgmtVRF)#extension pySIR-0.45-1.noarch.swix
peer00.lab(vrf:mgmtVRF)#extension fib_optimizer-0.1-1.noarch.swix
peer00.lab(vrf:mgmtVRF)#show extensions
```

Name	Version/Release	Status	extension
fib_optimizer-0.1-1.noarch.swix	0.1/1	A, I	1
pmacct_sir-0.1-1.noarch.swix	0.1/1	A, I	1
pySIR-0.45-1.noarch.swix	0.45/1	A, I	1
sir-0.17-1.noarch.swix	0.17/1	A, I	1

A: available | NA: not available | I: installed | NI: not installed | F: forced

6.7.2 Configuring the fib_optimizer

To configure the `fib_optimizer` you will have to run some python code from some maching with access to the HTTPS port of the switch.

First, install `pySIR` if you don't have it already:

```
pip install pySIR
```

Now, execute inside your python shell the following code (modify the configuration parameters and `base_url` to meet your needs):

```
from pySIR.pySIR import pySIR
import json

base_url = 'https://peer00.lab/sir'
configuration = {
    'lem_prefixes': '24',
    'max_lem_prefixes': 20000,
    'max_lpm_prefixes': 16000,
```

```
'path': '/tmp/',
'age': 168,
'purge_older_than': 336,
}
sir = pySIR(base_url, verify_ssl=False)

sir.post_variables(category='apps', name='fib_optimizer', content = json.dumps(configuration))
```

If you want to modify any variable later on you can run the following code in your python shell:

```
from pySIR.pySIR import pySIR
import json

base_url = 'https://peer00.lab/sir'
configuration = {
    'lem_prefixes': '24',
    'max_lem_prefixes': 40000,
    'max_lpm_prefixes': 16000,
    'path': '/tmp/',
    'age': 120,
    'purge_older_than': 240,
}
sir = pySIR(base_url, verify_ssl=False)

sir.put_variables_by_category_and_name(category='apps', name='fib_optimizer', content = json.dumps(configuration))
```

6.7.3 Scheduling fib_optimizer

In order to run the fib_optimizer hourly you will need to add the following line to your switch's configuration:

```
schedule fib_optimizer at 09:05:00 08/17/2015 interval 60 max-log-files 48 command bash sudo ip netns exec $NS ns-sd
```

Note: If you get a comment saying ! Schedule a command starting in past you can just ignore it.

Note: Replace ns-mgmtVRF with ns-\$MGMT_VRF or default if you don't have any.

This command is going to schedule the fib_optimizer to run every hour. You can run the fib_optimizer outside the switch if you want, maybe in some server. In that case change the last argument to match the URL of SIR.

On every run, the fib_optimizer is going to create a few prefix-lists that we are going to use on a route-map to control SRD (Selective Route Download).

6.7.4 Configuring SRD

SRD is a feature of some BGP implementations that allows you to pick some routers from the RIB and install them in the FIB. The routes not installed will still be processed as usual. This means that, if other policies permit it, they will be processed and forwarded to other BGP neighbors.

To enable SRD with EOS you only need to create a route-map (called SRD in our example) and execute:

```
router bgp $YOUR_ASN
  bgp route install-map SRD
```

The content of the route-map can be anything, however, I recommend that you have at least:

```
route-map SRD permit 10
  match as-path ASN_DC
!
route-map SRD permit 20
  match as-path ASN_TRANSIT
!
route-map SRD permit 30
  match ip address prefix-list fib_optimizer_lem_v4
!
route-map SRD permit 40
  match ip address prefix-list fib_optimizer_lpm_v4
!
```

The first block is going to match the prefix-lists coming from your internal network, the second block is going to match the prefixes coming from your transit provider:

```
ip as-path access-list ASN_DC permit ^$YOUR_INTERNAL_ASN$ any
ip as-path access-list ASN_TRANSIT permit ^$YOUR_TRANSIT_ASN$ any
```

These two blocks are going to ensure that all prefixes coming from your DC are going to be accepted and that the prefix coming from your transit provider (who is sending me the default route) is always installed. This will ensure that even if SIR or the fib_optimizer fails, I will still be able to route traffic.

The third and fourth block will be the ones controlled by the fib_optimizer.

Warning: Disclaimer: Take this as what it is, an example. Adapt the configuration for your network. This works for my network, it might not work for you. Even if you decide that this will work for you, you will still have to change the as-path list to match your own ASN's.