SDN Internet Router (sir)

Disclaimer: sparkles not included

Spotify

Myself:
David Barroso

Network Engineer @Spotify

- +10 years in the network industry
- Python enthusiastic
- Automation junkie
- Swedish (slow)learner

Spotify

SDN Internet Router (sir)

- Cheap
- Scalable
- Extensible
- Intelligent
- Open

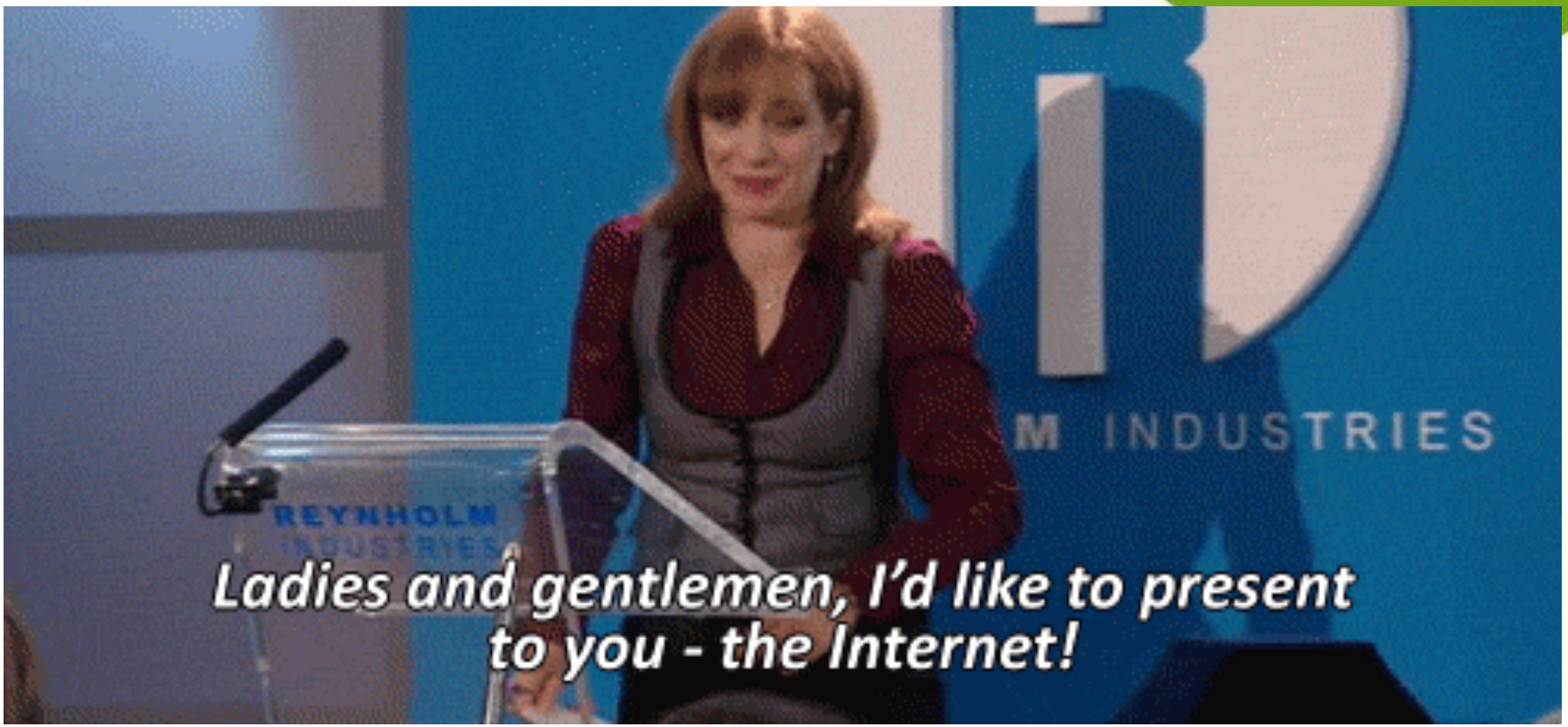
Internet Routers today

- Expensive
- Not very extensible
- Not very intelligent
- Thousands of features (that you probably don’t need)
- Did I say expensive?

Do you really need them?

- Maybe…
- Maybe not.

Why do we have them?



The Internet

- +500k prefixes
- Too many to fit them in devices with commodity ASICs (Trident 2 supports 32k and ARAD 64k prefixes)

When you travel...

- Do you carry an Atlas?
- Or do you carry a local map?

So...

- Why do I need all the prefixes?
- What if I only install the prefixes I really need?

Instead I am going...

- To use a cheap and programmable device
- Compute which prefixes I really need
- Install only those prefixes without hitting any hardware limits

Before we start...

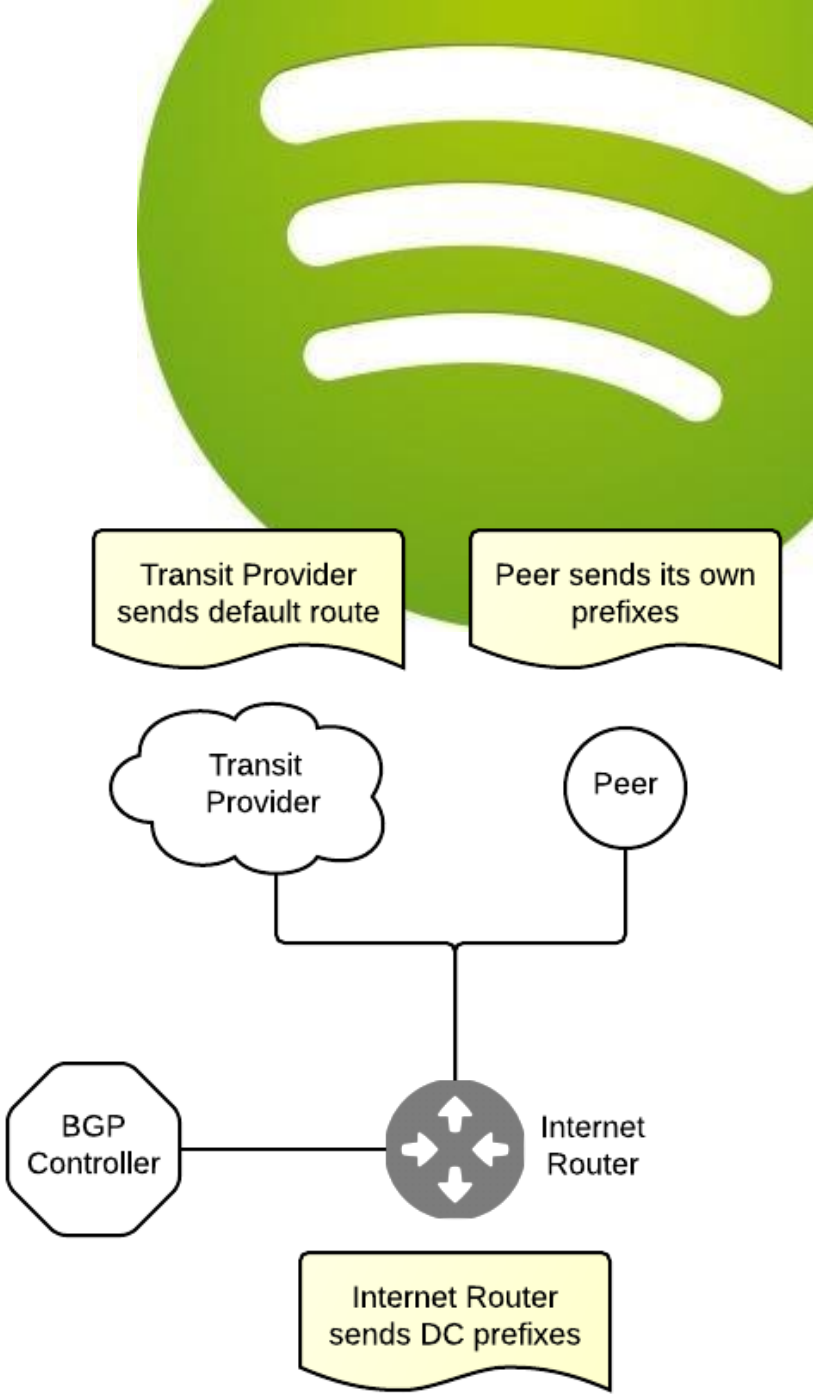
Two key components:

- **pmacct** - Flow collector that can aggregate flows by network, AS, BGP peer, etc... BGP information can be obtained by peering with other routers.
  http://www.pmacct.net/
- **Selective route download** - Feature that allows you to pick a subset of the routes on the RIB and install them on the FIB. Available in Cumulus (bird), Arista*, Juniper, some Cisco routers and probably others.

* If you plan to try on Arista contact your SE.

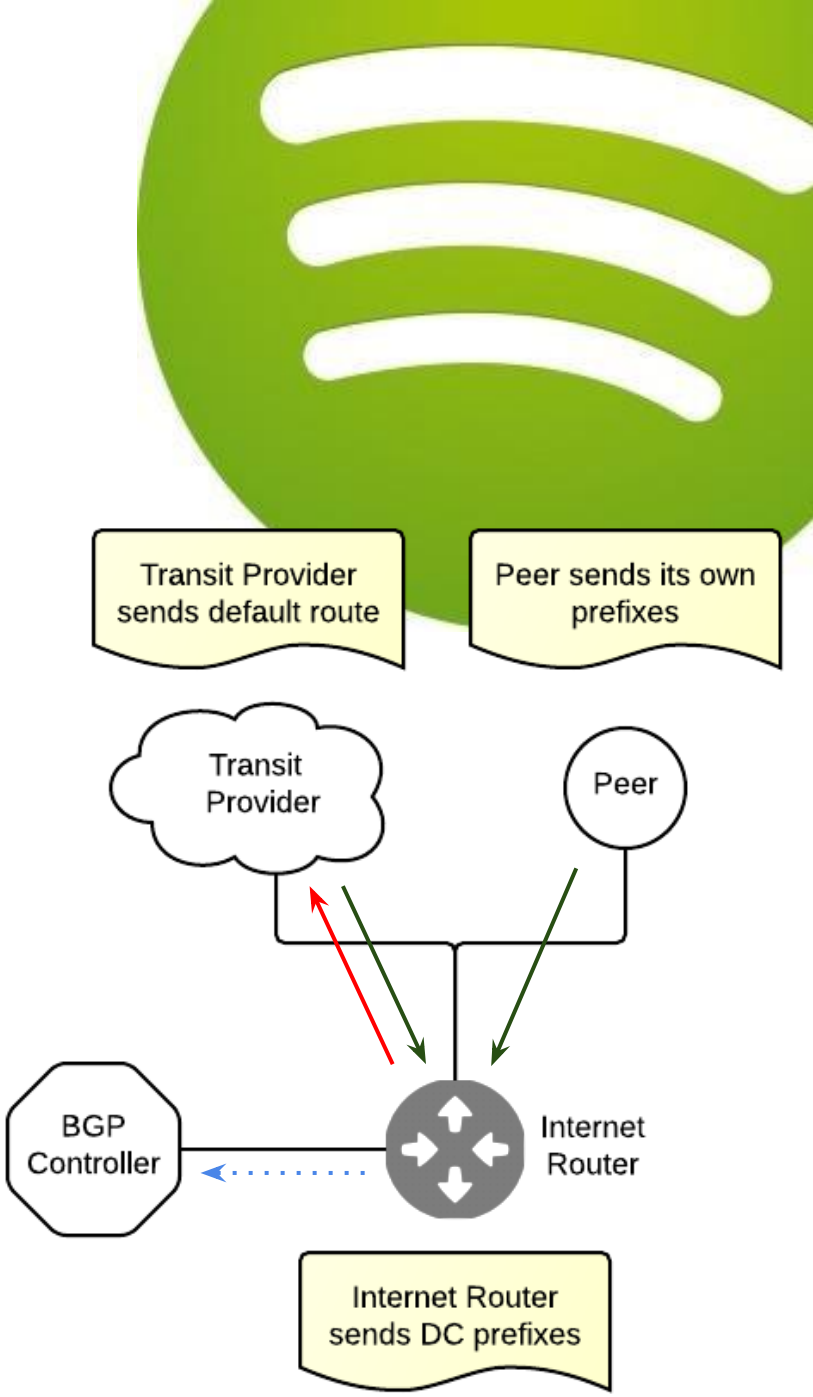
Overview

- Transit provider is my default exit. They announce the default route, which is installed on the FIB of the Internet Router.
- By peering I am able to offload traffic from my transit provider. They send their prefixes, which are not installed on the FIB by default. Instead they are sent to the BGP controller.
- sFlow information is sent from the Internet Router to the BGP controller.
- As traffic traverses my Internet Router the BGP controller instructs the Internet Router to install all necessary prefixes to offload traffic from the Transit Provider to the Peers.



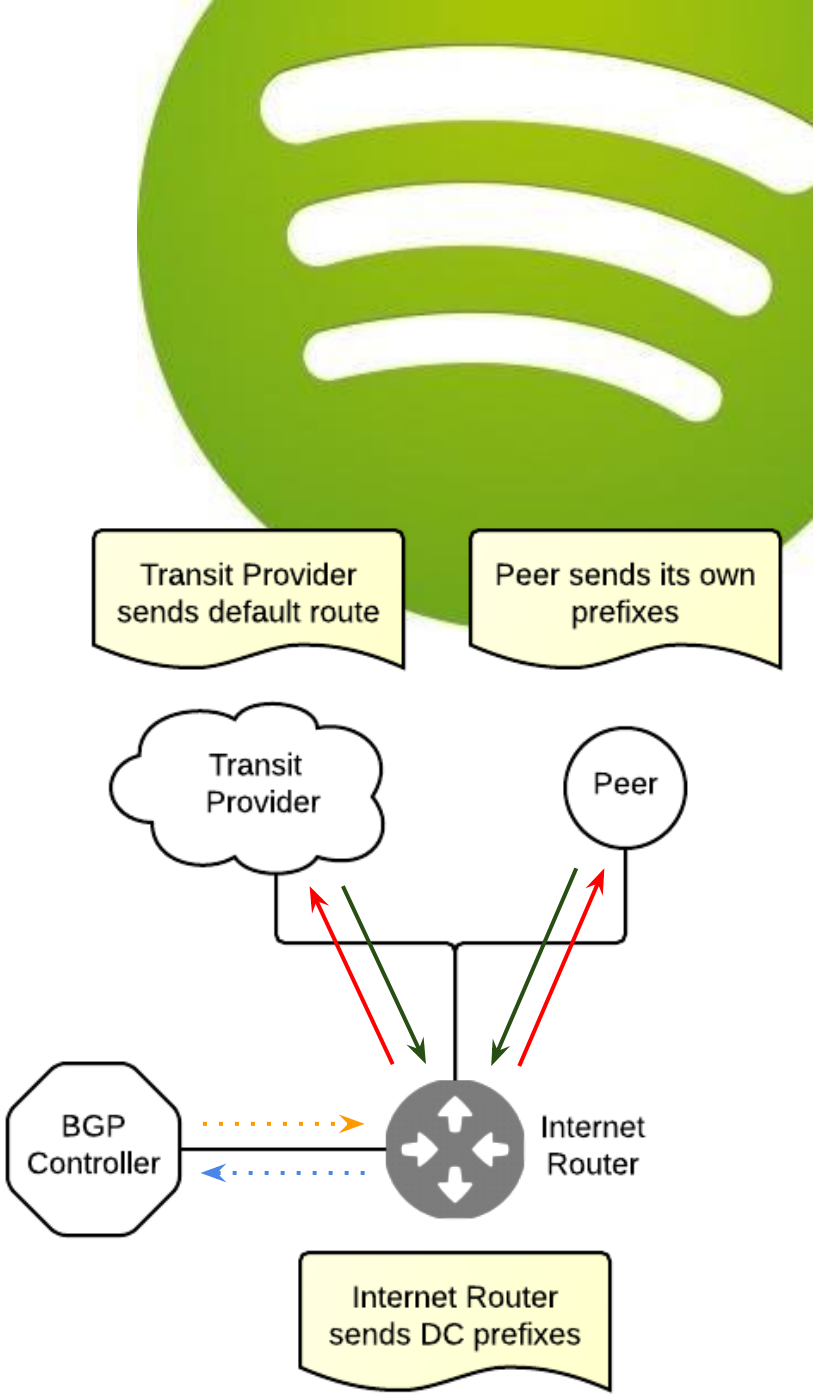
Time 0

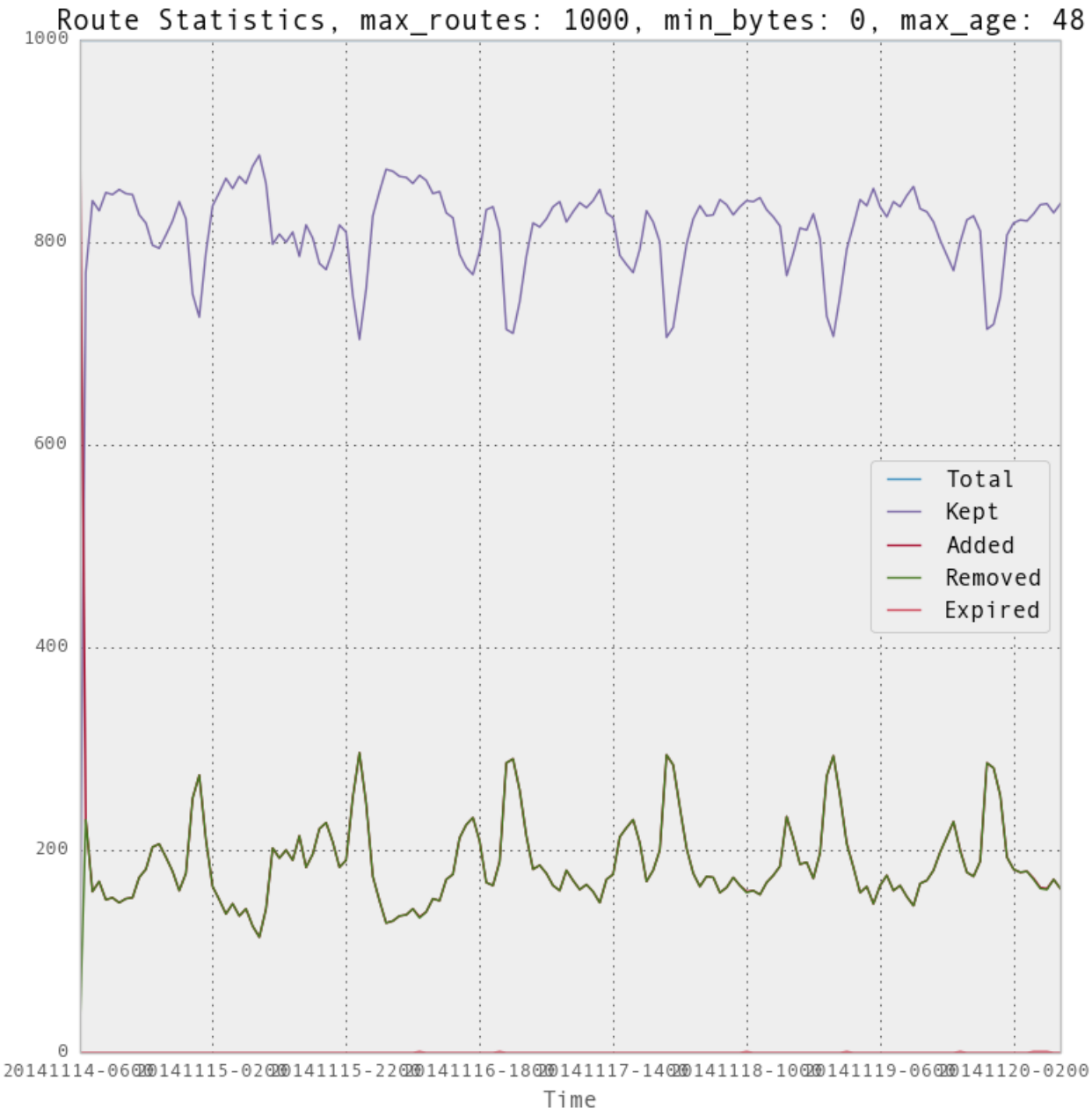
- The Internet Router only knows about the default route coming from the Transit provider, so all traffic exits the DC via this path.
- The Internet Router announces all the DC prefixes to everybody, so inbound traffic can come from any path.
- The Internet router starts sending sFlow and BGP information to the BGP controller

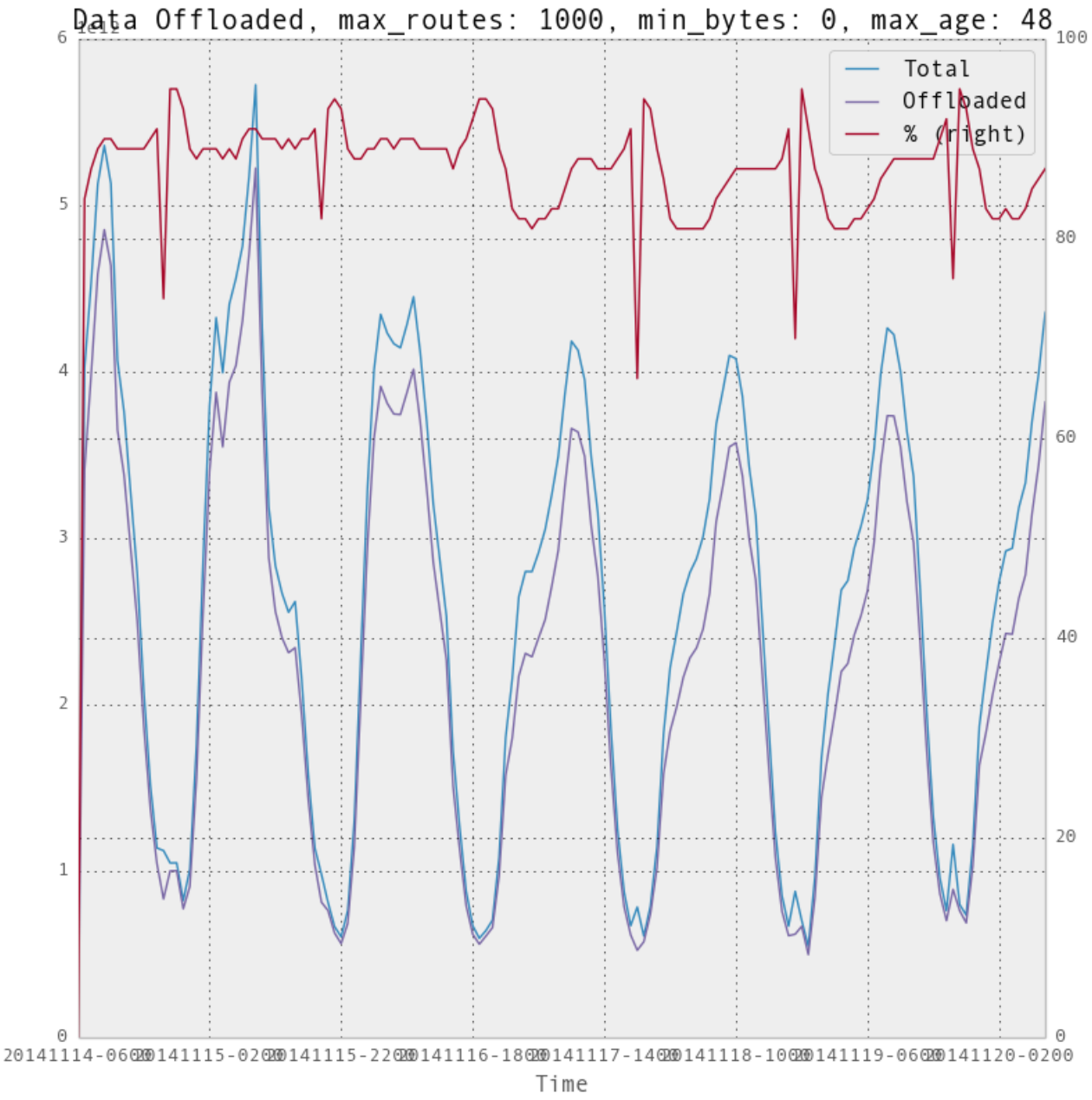


Time 1..N

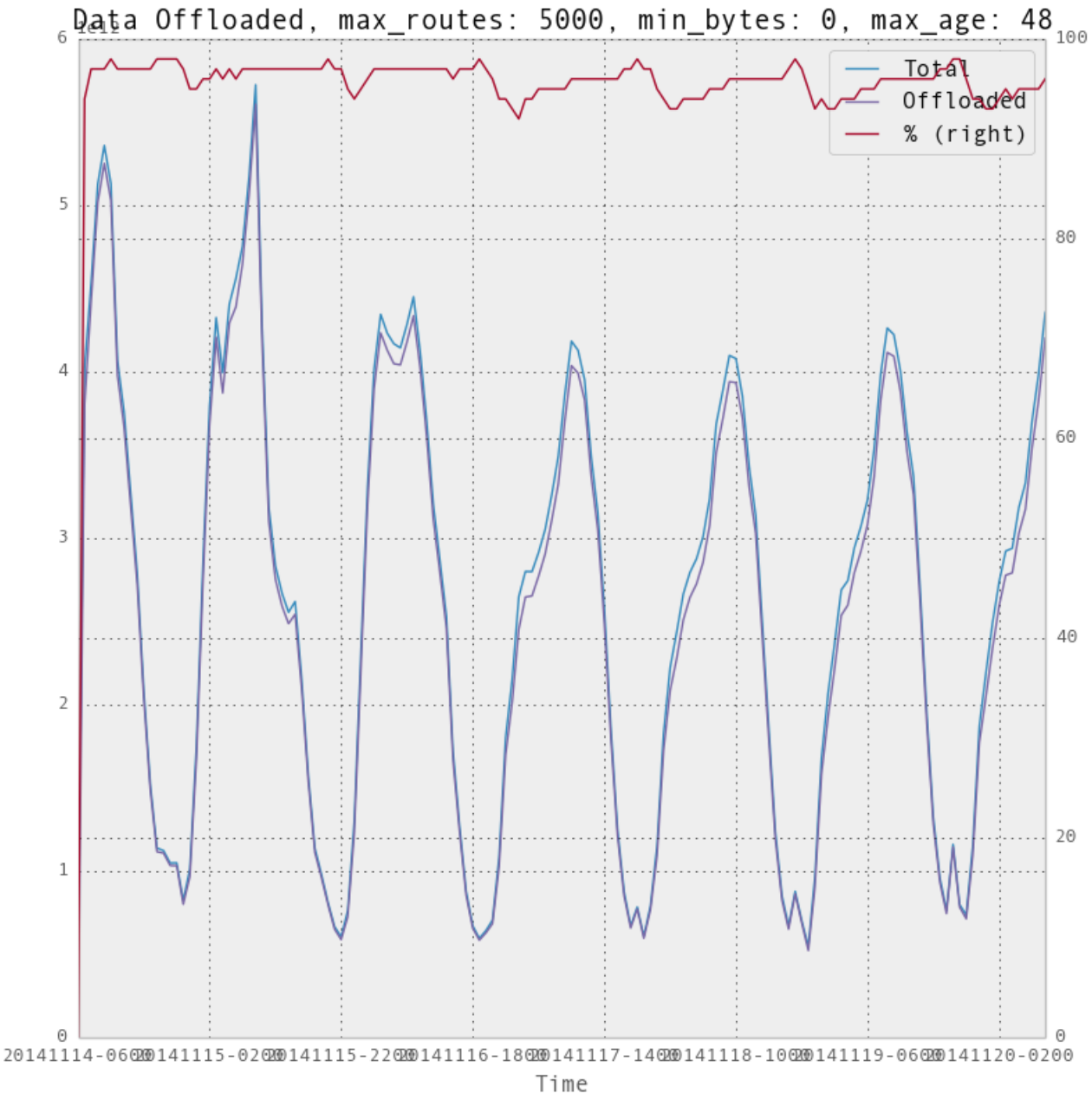
- The BGP controller computes the topN prefixes according to the BGP feed and the flow information and instructs the Internet Router to install those prefixes.
- The Internet Router now have more routes pointing to its peers so traffic start flowing on that direction.

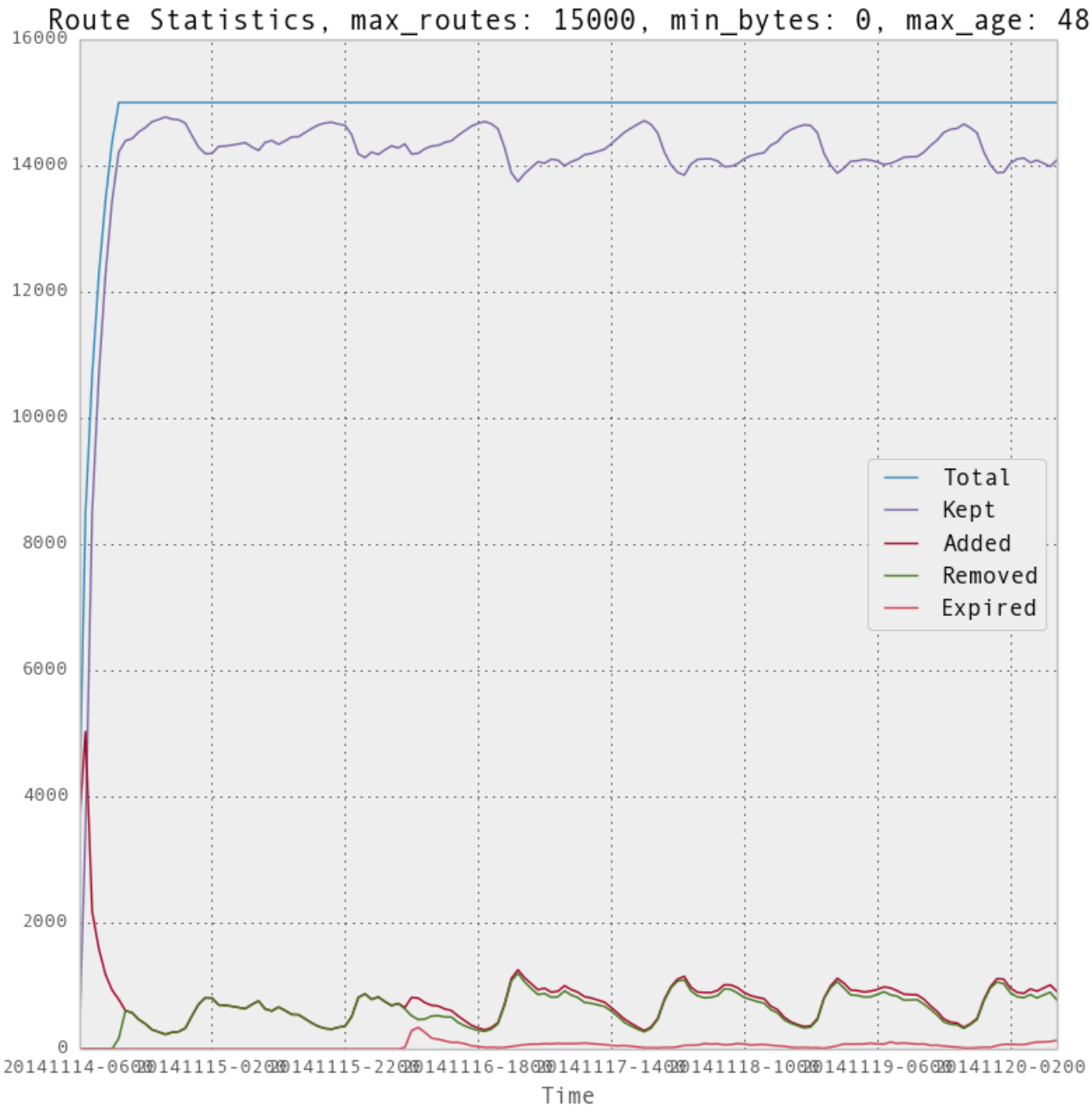


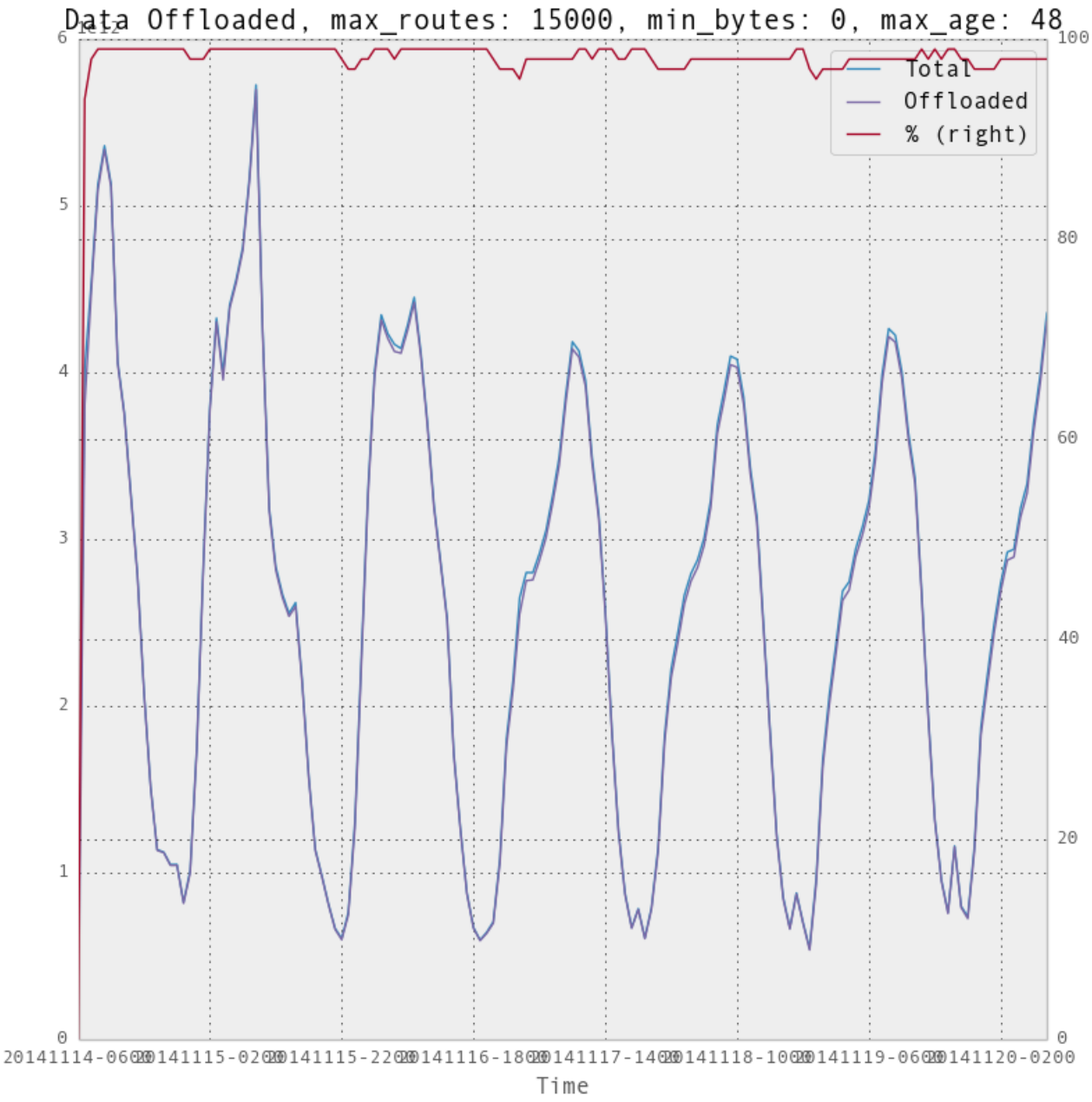

Route Statistics, max_routes: 1000, min_bytes: 0, max_age: 48
1000
800
600
400
200
0
Total
Kept
Added
Removed
Expired
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time

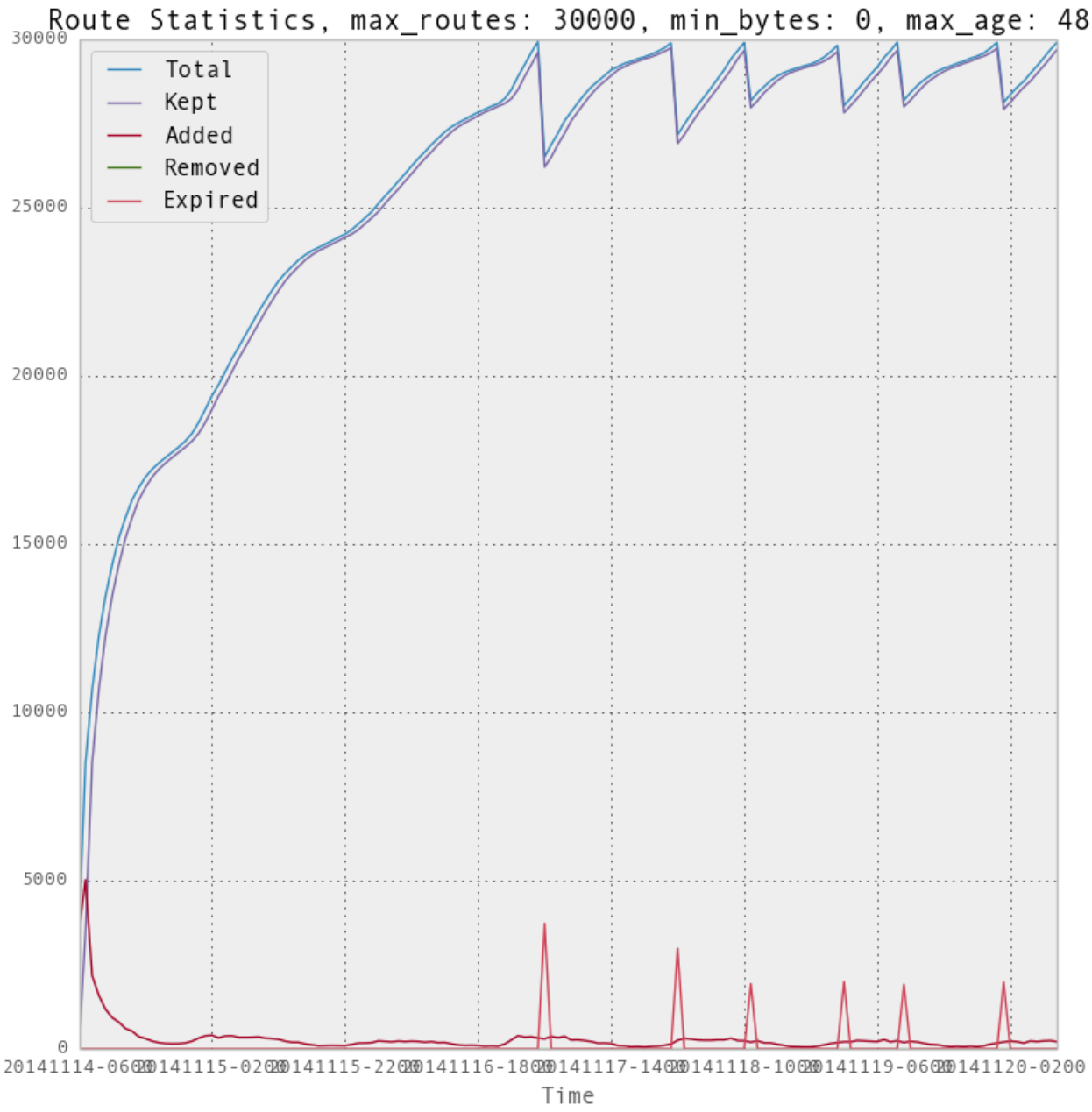

Data Offloaded, max_routes: 1000, min_bytes: 0, max_age: 48
Total
Offloaded
% (right)
0
1
2
3
4
5
6
0
20
40
60
80
100
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time

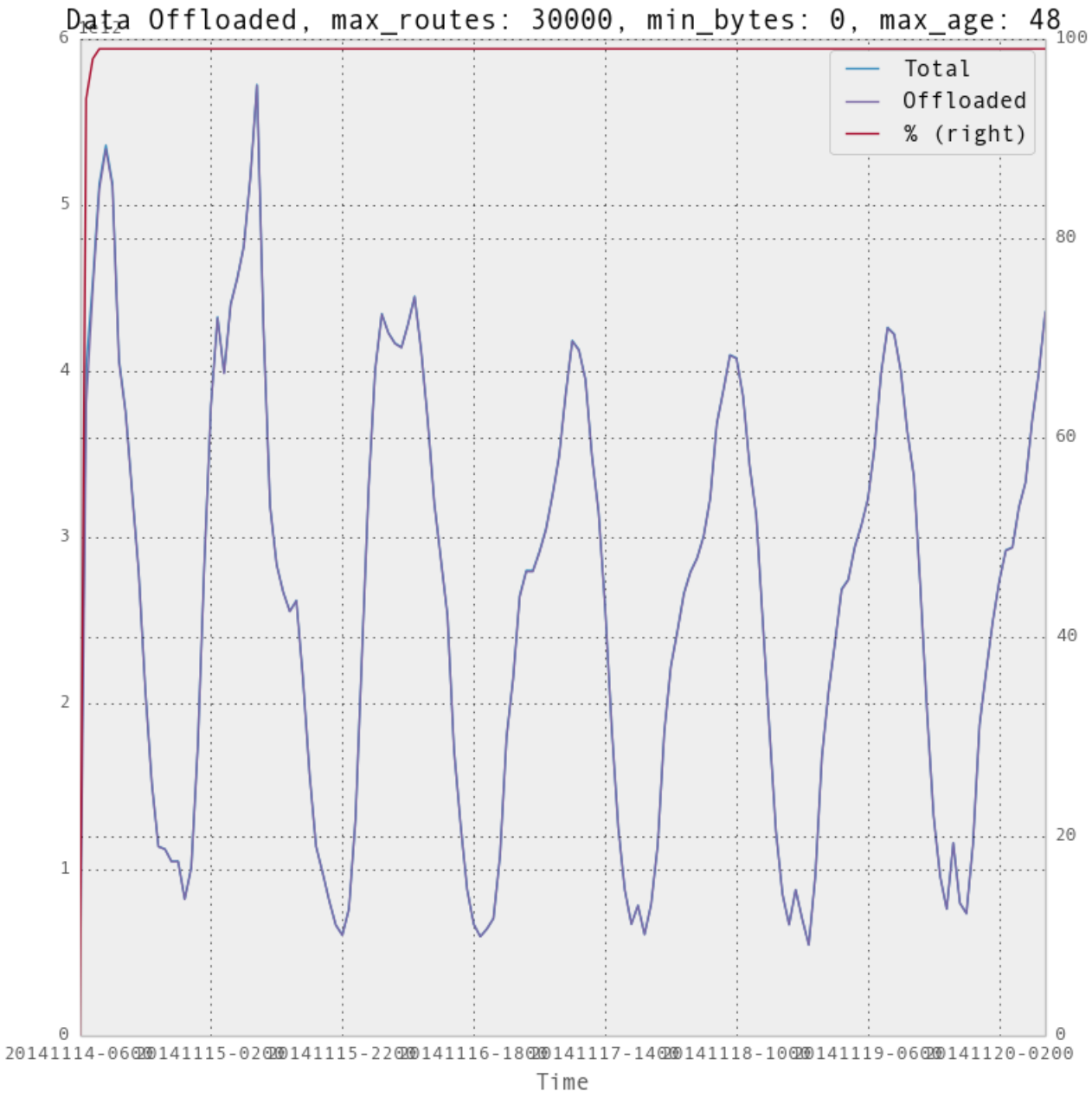

Route Statistics, max_routes: 5000, min_bytes: 0, max_age: 48
5000
4000
3000
2000
1000
0
Total
Kept
Added
Removed
Expired
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time


Data Offloaded, max_routes: 5000, min_bytes: 0, max_age: 48
Total
Offloaded
% (right)
Time

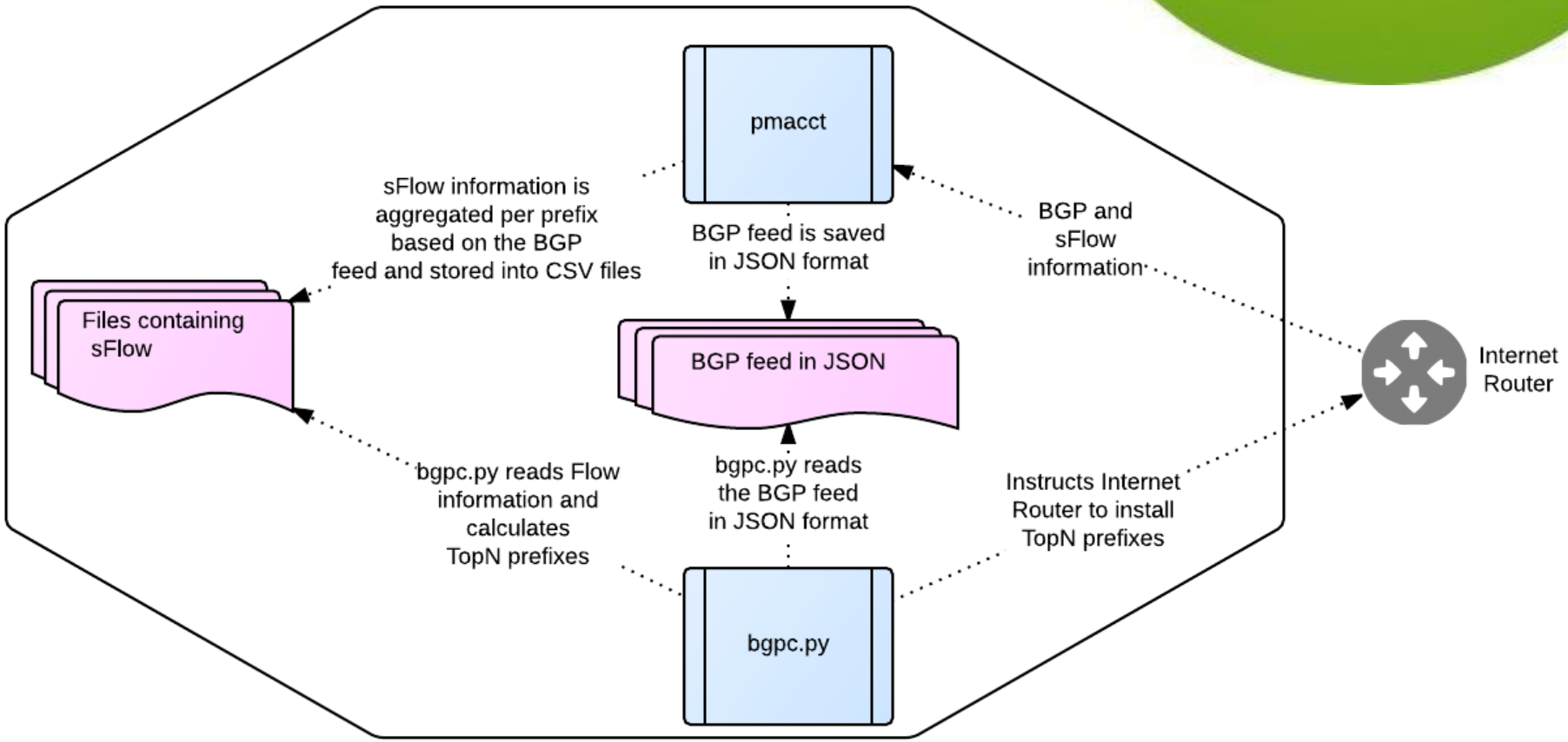

Route Statistics, max_routes: 15000, min_bytes: 0, max_age: 48
16000
14000
12000
10000
8000
6000
4000
2000
0
Total
Kept
Added
Removed
Expired
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time


Data Offloaded, max_routes: 15000, min_bytes: 0, max_age: 48
1e12
6
5
4
3
2
1
0
100
80
60
40
20
0
Total
Offloaded
% (right)
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time


Route Statistics, max_routes: 30000, min_bytes: 0, max_age: 48
Total
Kept
Added
Removed
Expired
30000
25000
20000
15000
10000
5000
0
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time


Data Offloaded, max_routes: 30000, min_bytes: 0, max_age: 48
1e12
Total
Offloaded
% (right)
0
1
2
3
4
5
6
0
20
40
60
80
100
20141114-0600
20141115-0200
20141115-2200
20141116-1800
20141117-1400
20141118-1000
20141119-0600
20141120-0200
Time

BGP Controller

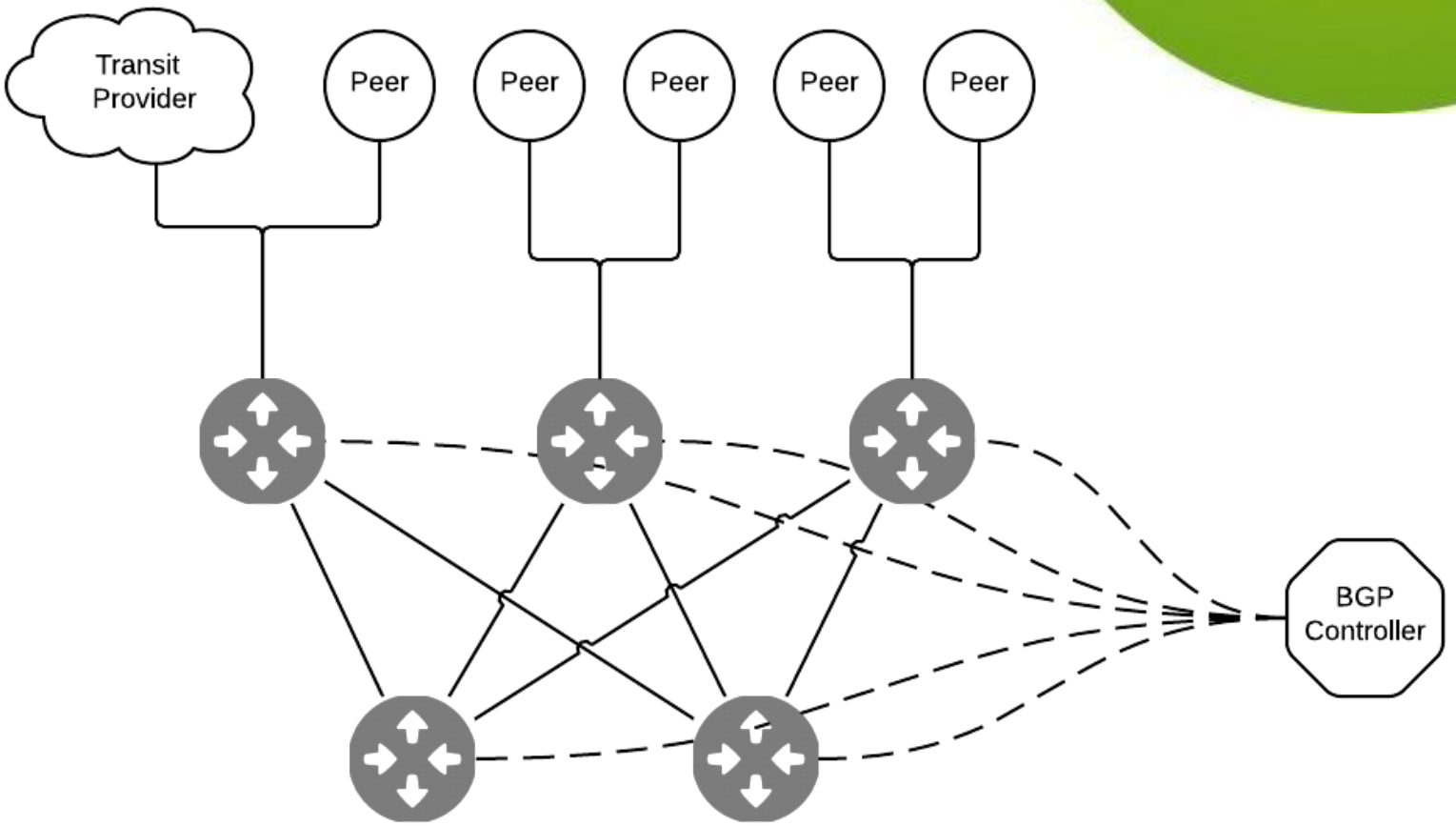


BGP Controller Extensibility

- The BGP controller by default only computes top prefixes and passes all the information used and the results to ‘plugins’.
- Plugins can do with this information whatever they want:
  - Build reports
  - Build a prefix list and send it to a router.
  - Compare possible next-hops, AS PATH... with a monitoring tool to choose peers based on reliability, latency, company policies, etc...

```
# cat etc/config.yaml

max_age: 48
csv_delimiter: ";"
max_routes: 30000
min_bytes: 0
packet_sampling: 10000

… (output omitted)

plugins:
  - 'prefix_data.SavePrefixData'
  - 'statistics.RouteStatistics'
  - 'statistics.OffloadedBytes'
  - 'bird.Bird'

… (output omitted)
```

BGP Controller Scalability



BGP Controller Reliability

- We are not inventing or re-inventing any protocol
- We are just modifying BGP automatically to match our traffic needs.
- If pmacct or the BGP controller fails, things will work as they were working before the failure.

Appendix: Bird Configuration Example

```
  # This file includes the
allow_prefixes() method, which will decide
which prefixes to install on the routing
table.
  include "/etc/allow_prefixes.bird";

  protocol kernel {
      export filter {
          if from = 10.0.0.1 then accept;
          if allow_prefixes() then accept;
          reject;
      };
  }
  # Transit Provider
  protocol bgp {
      local as 65010;
      neighbor 10.0.0.1 as 65001;
  }
```

```
# Peer
protocol bgp {
    local as 65010;
    neighbor 10.0.0.2 as 65002;
}
#pmacct/bgp_controller
protocol bgp {
    local as 65534;
    neighbor 192.168.231.1 as 65534;
    add paths tx;

    # We send only prefixes from our peers
    export filter {
        if from = 10.0.0.2 then accept;
        reject;
    };
}
```

Appendix: pmacct Configuration Example

```
daemonize: True

plugins: print[simpleoutput]

print_output_file[simpleoutput]: /spotify/pmacct-1.5.0/output/simpleoutput-%Y%m%d-%H%M.txt
print_latest_file[simpleoutput]: /spotify/pmacct-1.5.0/output/simpleoutput-latest.txt
files_umask: 022
print_output[simpleoutput]: csv
print_output_separator[simpleoutput]: ;
print_refresh_time[simpleoutput]: 3600
print_output_file_append[simpleoutput]: true
print_history[simpleoutput]: 1h
print_history_roundoff[simpleoutput]: h


… continues
```

Appendix: pmacct Configuration Example

```
… continues

aggregate: dst_net, dst_mask

bgp_daemon: true
bgp_daemon_ip: 192.168.231.2
bgp_daemon_max_peers: 2
bgp_agent_map: /spotify/pmacct-1.5.0/etc/agent_to_peer.map
bgp_table_dump_file: /spotify/pmacct-1.5.0/output/bgp-$peer_src_ip-%H%M.txt
bgp_table_dump_refresh_time: 60

sfacctd_as_new: bgp
sfacctd_net: bgp
sfacctd_port: 9999
sfacctd_ip: 192.168.231.2
```

Questions?

Email: dbarroso@spotify.com

URL: https://github.com/dbarrosop/sir

Spotify

Want to join the band?

Check out spotify.com/jobs or @Spotifyjobs for more information.

Spotify