

# Collecting NetFlow with pmacct

Paolo Lucente

pmacct

# Collecting NetFlow with pmacct

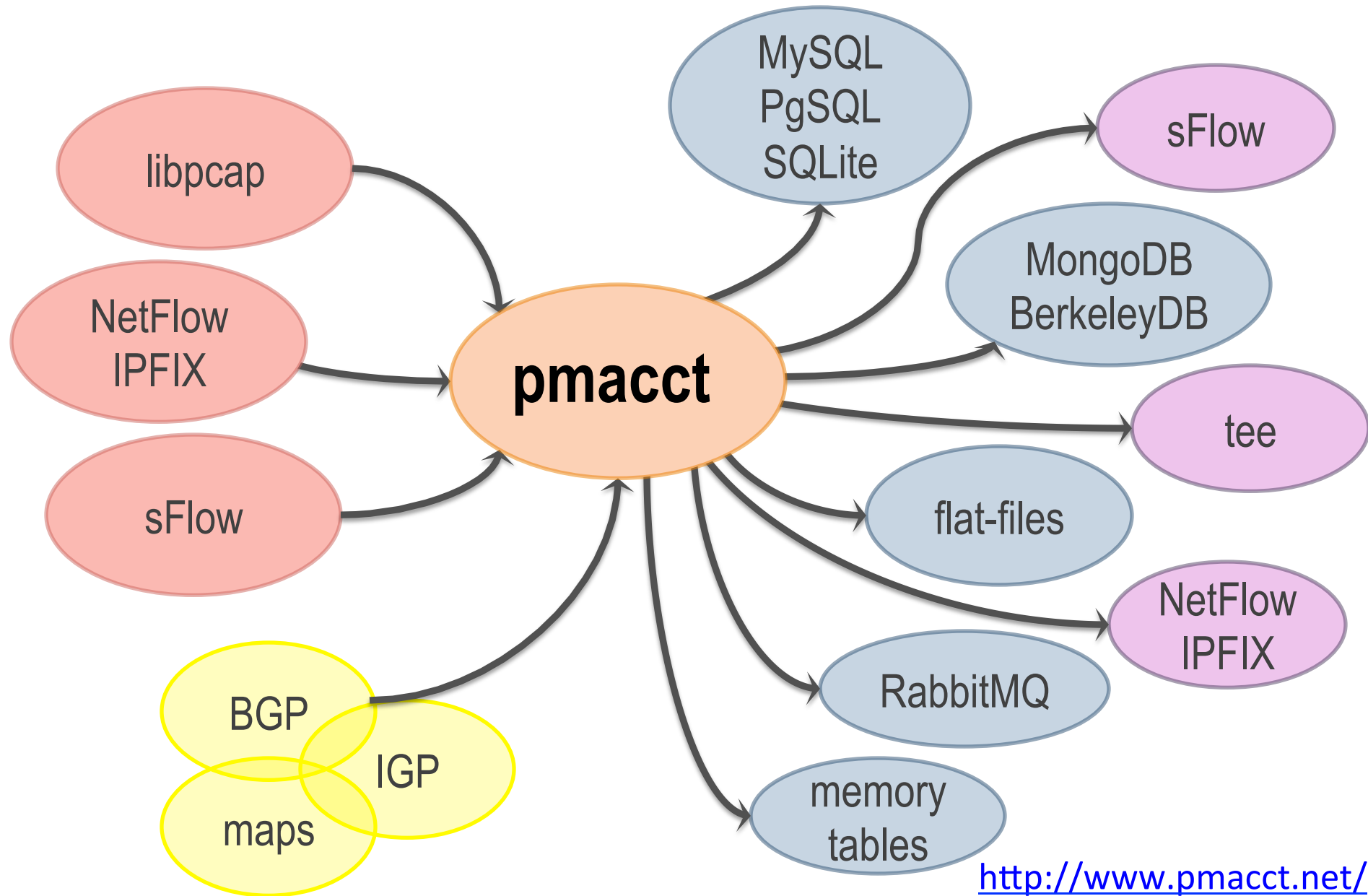
## Agenda

- **Introduction**
- pmacct architecture & benefits
- example, data aggregation: traffic matrices
- example, logging micro-flows or events
- tee: briefly on horizontal scalability

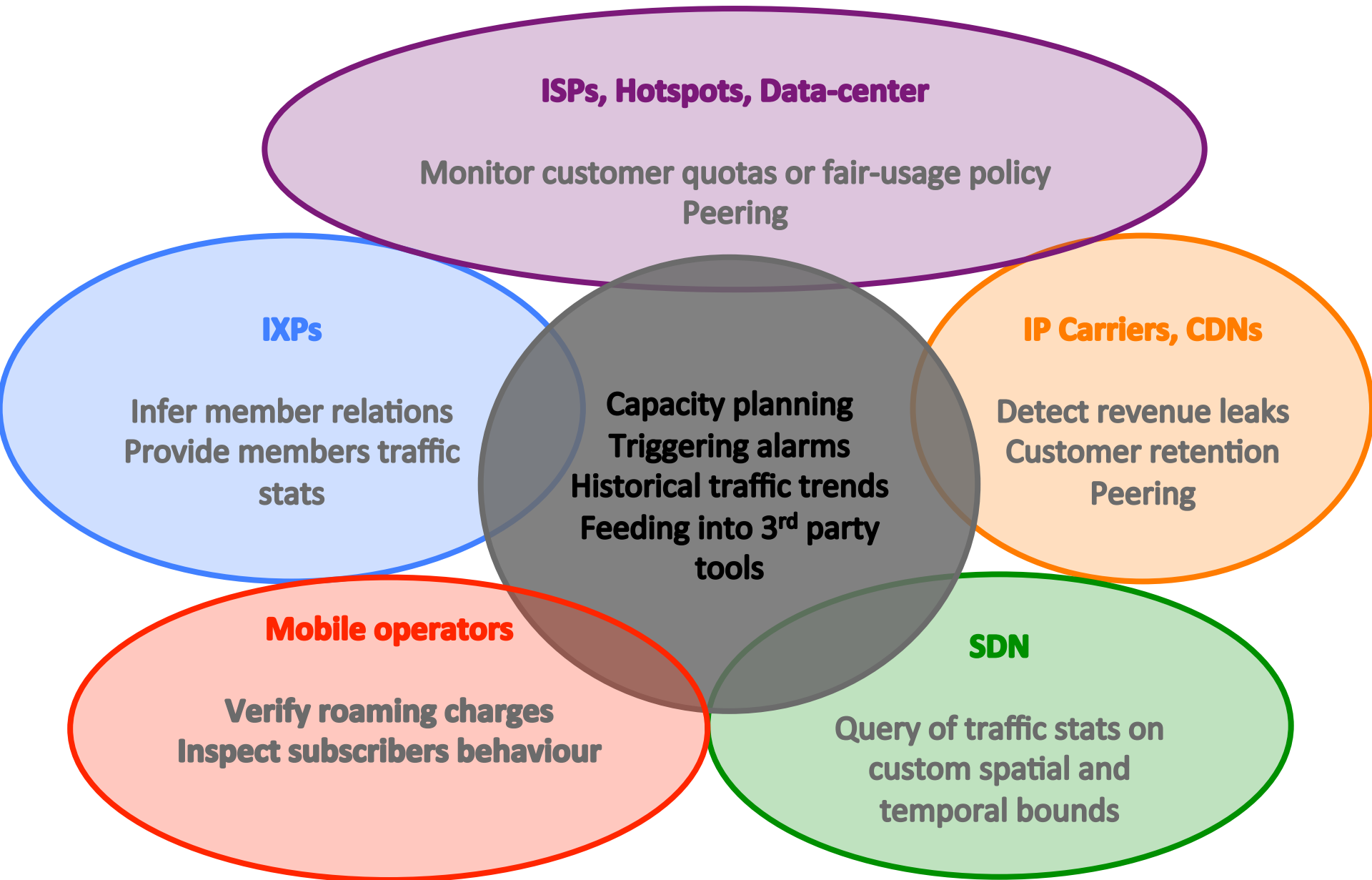
# whoami: Paolo

- Been originally working for operators for a while
- Been working for vendors for a little while after that
- Been involved with IP accounting for a while
  - Hence I stumbled upon NetFlow in the 90's 😊
- Within operators, network traffic telemetry is beneficial in several contexts, ie.:
  - Traffic engineering
  - Capacity planning
  - Peering
  - ...
  - and also (ie. not only) security

# pmacct is open-source, free, GPL'ed software



# Usage scenarios



# Key pmacct non-technical facts

- 10 years old project
- Free, open-source, independent
- Under active development
- Innovation being introduced
- Well deployed around, also large SPs
- Aims to be the traffic accounting tool closer to the SP community needs

# Collecting NetFlow with pmacct

## Agenda

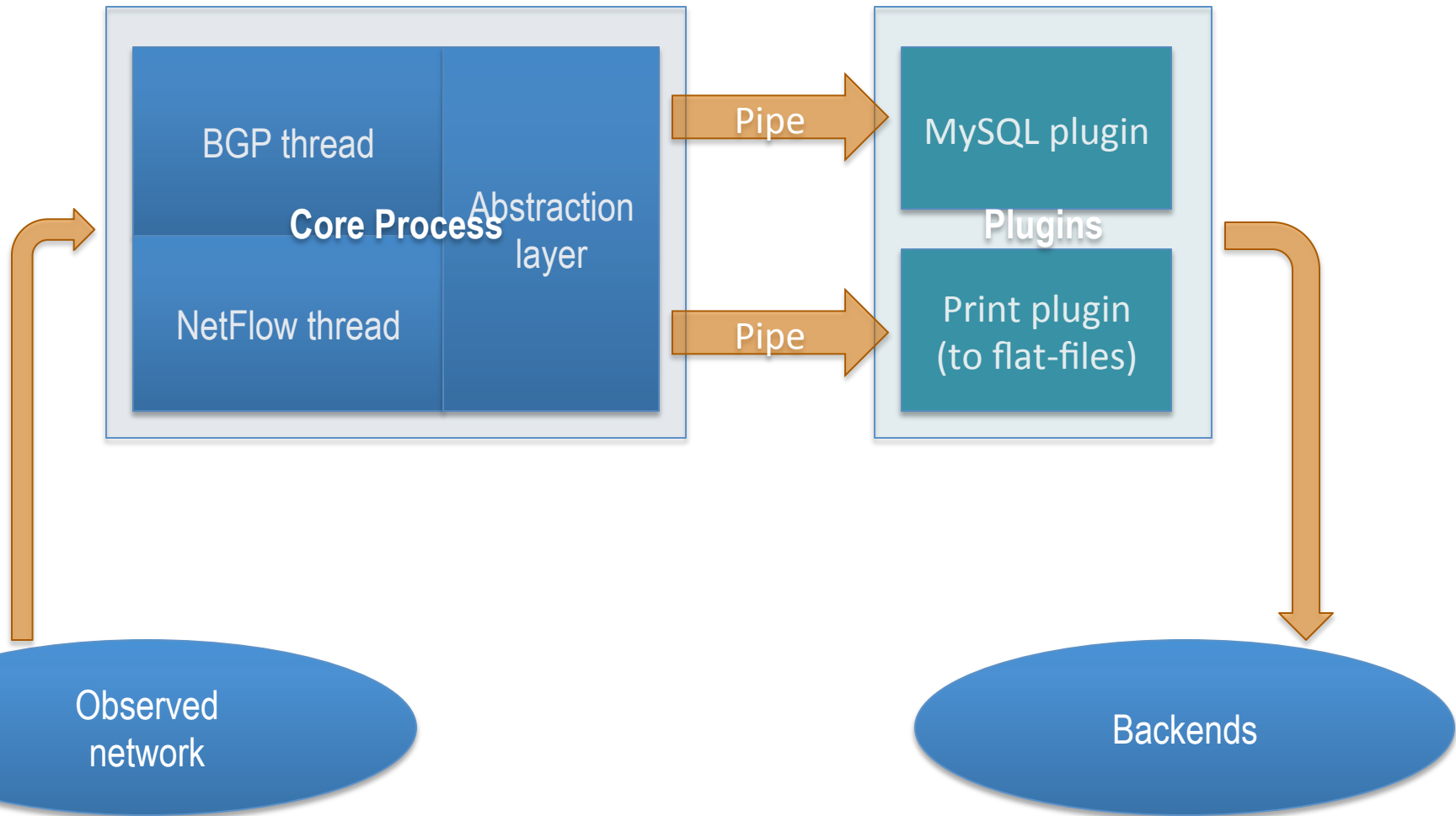
- o Introduction
- o **pmacct architecture & benefits**
- o example, data aggregation: traffic matrices
- o example, logging micro-flows or events
- o tee: briefly on horizontal scalability

# Some technical facts (1/3)

- Pluggable architecture
  - Straightforward to add support for new collection methods or backends
- An abstraction layer allows out-of-the-box any collection method to interact with any backend
- Both multi-process and (coarse) multi-threading
  - Multiple plugins (of same or different type) can be instantiated at runtime, each with own config



# Some technical facts (2/3)



# Some technical facts (3/3)

- Pervasive data-reduction techniques, ie.:
  - Data aggregation
  - Tagging and filtering
  - Sampling
- Ability to build multiple views out of the very same collected network traffic dataset , ie.:
  - Unaggregated to flat-files for security and forensic purposes
  - Aggregated as [ <ingress router>, <ingress interface>, <BGP next-hop>, <peer destination ASN> ] to build an internal traffic matrix for capacity planning purposes

# Touching ground: a config snippet for both aggregated and unaggregated views

```
nfacctd_ip: 10.0.0.1  
nfacctd_port: 2100
```

Basic daemon  
config

```
!
```

```
plugins: print[forensics], mysql[int_traffic_matrix]
```

```
!
```

```
aggregate[forensics]: src_host, dst_host, \  
peer_src_ip, peer_dst_ip, in_iface, out_iface, \  
timestamp_start, timestamp_end, src_port, \  
dst_port, proto, tos, src_mask, dst_mask, src_as, \  
dst_as, tcpflags
```

Instantiating  
plugins

```
!
```

```
aggregate[int_traffic_matrix]: in_iface, peer_src_ip, \  
peer_dst_ip, peer_dst_as
```

```
!
```

```
! <rest of config skipped>
```

Defining plugin  
aggregation  
methods

# Touching ground: data aggregation & custom-defined primitives

## ■ Config file snippet:

```
! ...  
aggregate[int_traffic_matrix]: peer_src_ip, \  
  mplsTopLabelIPv4Address  
!  
aggregate_primitives: /path/to/primitives.lst  
! ...
```

Node-to-node internal TM  
(egress NetFlow)

mplsTopLabelIPv4Address  
not supported natively, let's  
define it!

## ■ Custom primitive definition:

```
! ...  
name=mplsTopLabelIPv4Address field_type=47 len=4 semantics=ip  
! ...
```

Primitive name,  
will be used for  
everything

NetFlow field type,  
IPFIX Information Element

NetFlow/IPFIX  
field length

Data presentation: [ u\_int,  
hex, ip, mac, str ]

# Introducing BGP natively into a NetFlow/sFlow collector

- pmacct introduced a Quagga-based BGP daemon
  - Implemented as a parallel thread within the collector
  - Doesn't send UPDATES and WITHDRAWs whatsoever
  - Behaves as a passive BGP neighbor
  - Maintains per-peer BGP RIBs
  - Supports 32-bit ASNs; IPv4 and IPv6 families
- Why BGP at the collector?
  - Telemetry reports on forwarding-plane
  - Telemetry should not move control-plane information over and over

# IGP (IS-IS) integration

- A Quagga-based IS-IS daemon was introduced:
  - Implemented as a parallel thread within the collector
  - IS-IS neighborship over a GRE tunnel
  - Currently limited to single neighborship, single level, single topology
  - Useful to look up non BGP-routed networks
  - Promising, many plans:
    - Implement a non real-time, IGP-protocol agnostic version
    - Introduce IGP-specific primitives (ie. IGP distance)
    - Collecting feedback on interest around TRILL

# Storing data persistently

- Data need to be aggregated both in spatial and temporal dimensions before being written down:
  - Optimal usage of system resources
  - Avoids expensive consolidation of micro-flows
  - Suitable for project-driven data-sets
- Open-source RDBMS appear a natural choice
  - Able to handle large data-sets
  - Flexible and standardized query language
  - Solid and scalable (clustering, partitioning)
- noSQL DBs picking up: between myth and bust ..

# Storing data persistently: MongoDB

- pmacct opening to noSQL databases
- noSQL landscape difficult to move through, ie. fragmented and lacks of standardization
- MongoDB seemed interesting for:
  - Its native grouping operation (more performing and less complex than map/reduce)
  - Its horizontal scaling concept (called sharding)
- Picking up among operators and being constantly improved (ie. batching inserts, native support for indexing) for the past 9 months!



# Brokering data around: RabbitMQ message exchanges

- pmacct opening to AMQP protocol
- noSQL landscape difficult to move through, ie. fragmented and lacks of standardization, am i repeating myself? 😊
- Data can be picked up at the message exchange in the preferred programming/scripting language
- Data can be then easily inserted in the preferred backend, ie. not natively supported by pmacct

# Collecting NetFlow with pmacct

## Agenda

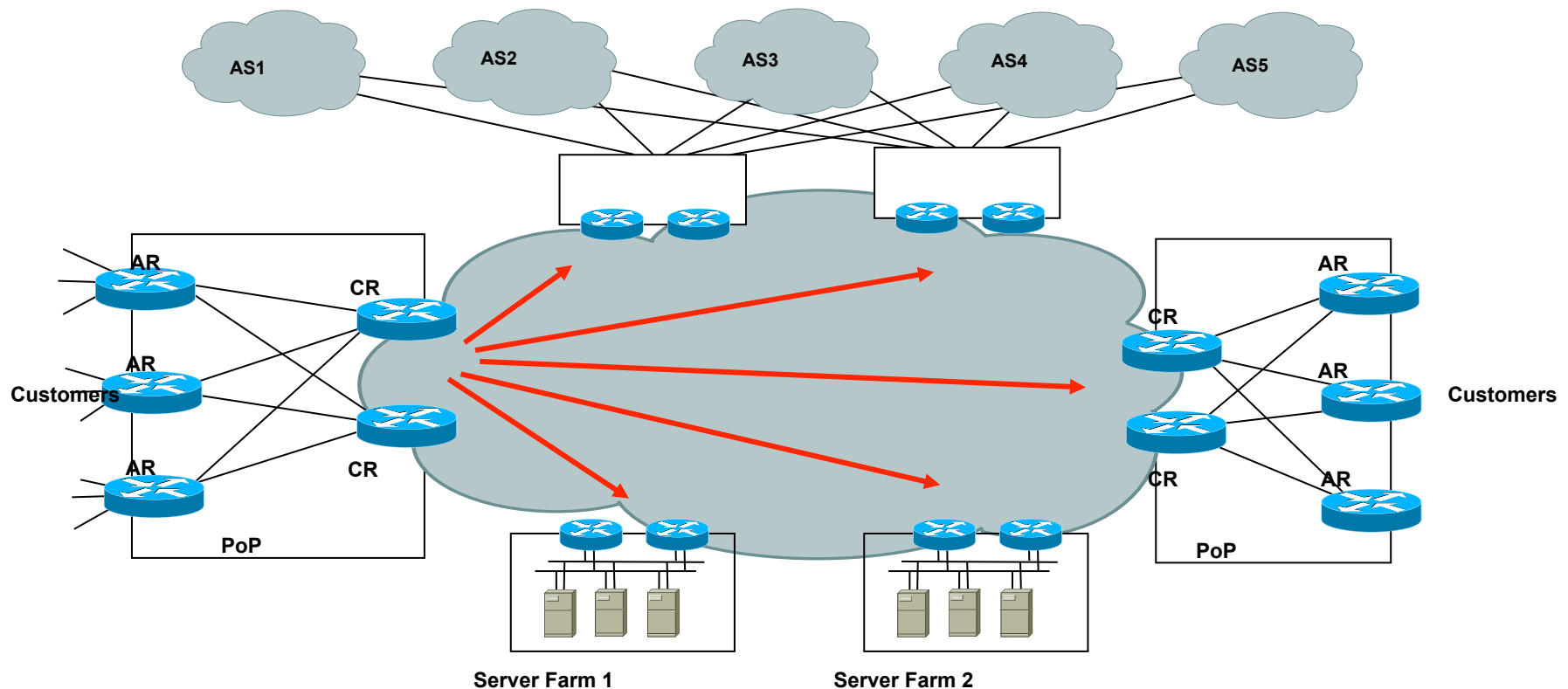
- o Introduction
- o pmacct architecture & benefits
- o **example, data aggregation: traffic matrices**
- o example, logging micro-flows or events
- o tee: briefly on horizontal scalability

# Why speaking of traffic matrices?

- So, after all, are traffic matrices useful to a network operator in the first place? Yes ...
  - Capacity planning (build capacity where needed)
  - Traffic Engineering (steer traffic where capacity is available)
  - Better understand traffic patterns (what to expect, without a crystal ball)
  - Support peering decisions (traffic insight, traffic engineering at the border, support what if scenarios)

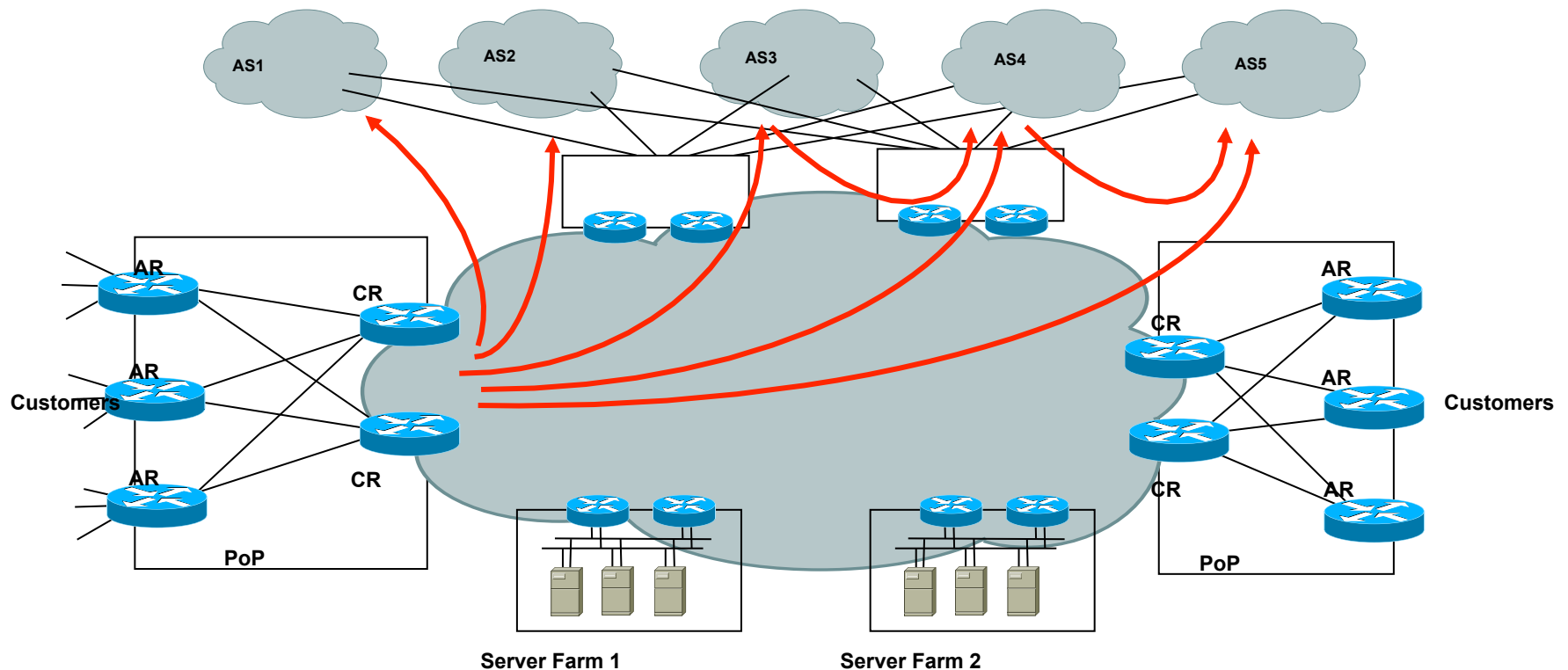
# Review of traffic matrices: internal

- POP to POP, AR to AR, CR to CR



# Review of traffic matrices: external

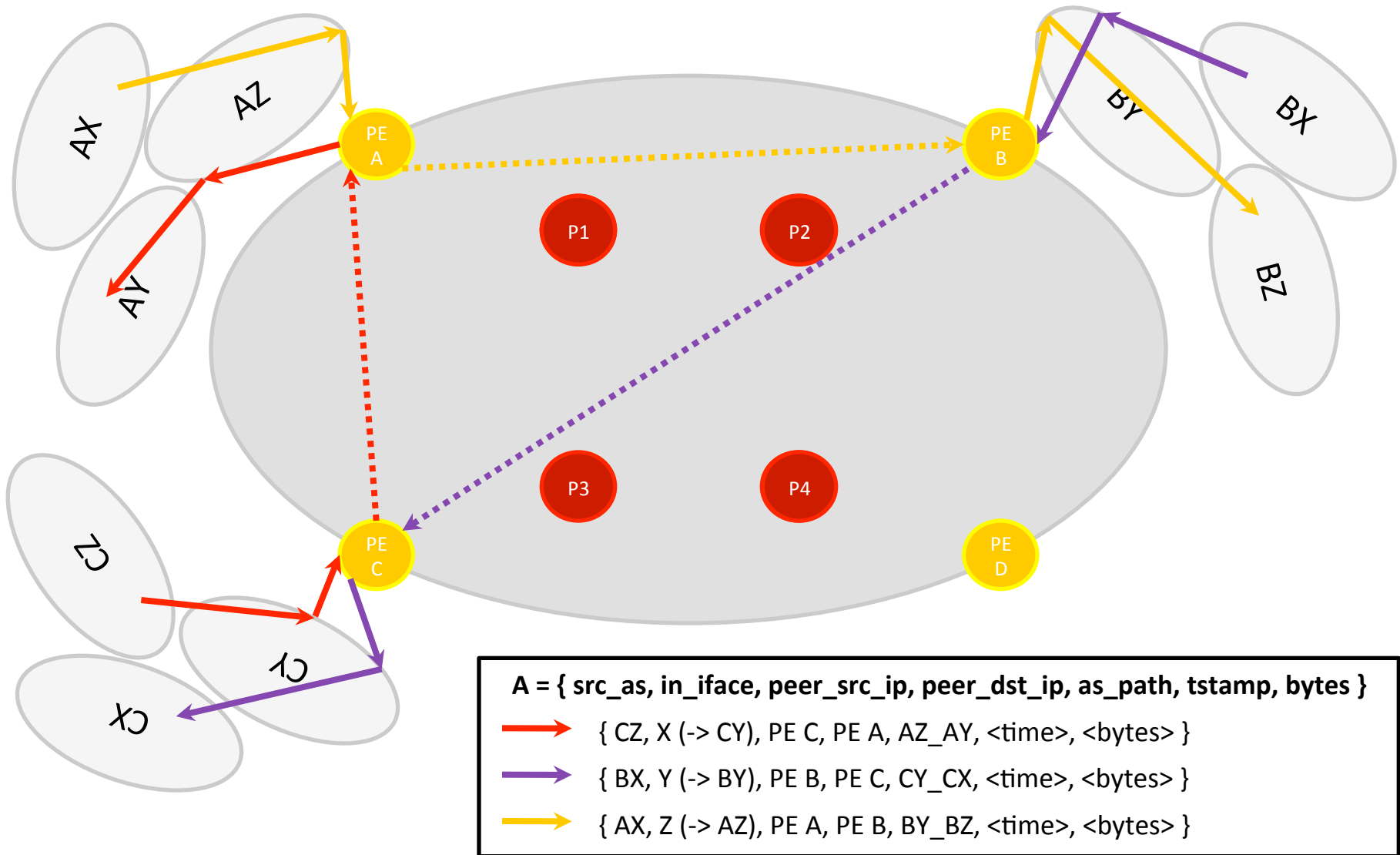
- Router (AR or CR) to external AS or external AS to external AS (for IP transit providers)



# Let's focus on an external traffic matrix to support peering decisions

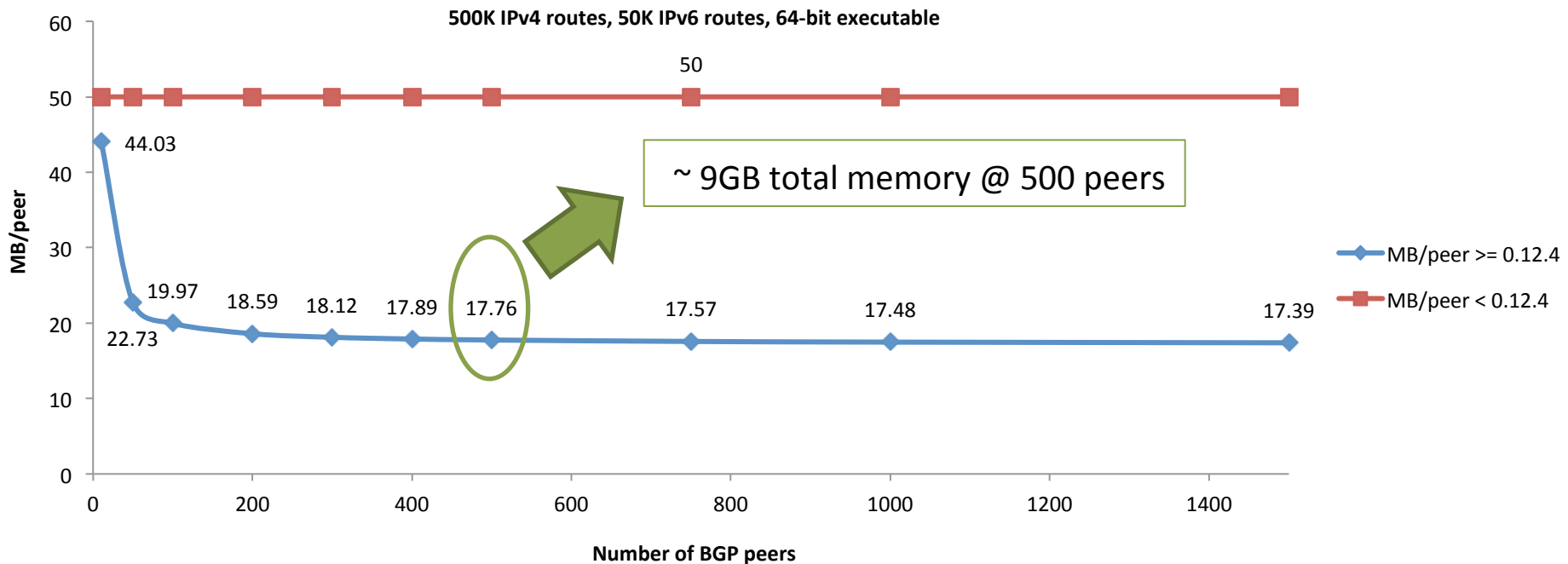
- Analysis of existing peers and interconnects:
  - Support policy and routing changes
  - Fine-grained accounting of traffic volumes and ratios
  - Determine backbone costs associated to peering
  - Determine revenue leaks
- Planning of new peers and interconnects:
  - Who to peer next
  - Where to place next interconnect
  - Modeling and forecasting

# Our traffic matrix visualized



# Getting BGP to the collector (1/2)

- Let the collector BGP peer with all PE devices: facing peers, transit and customers.
- Determine memory footprint (below in MB/peer)





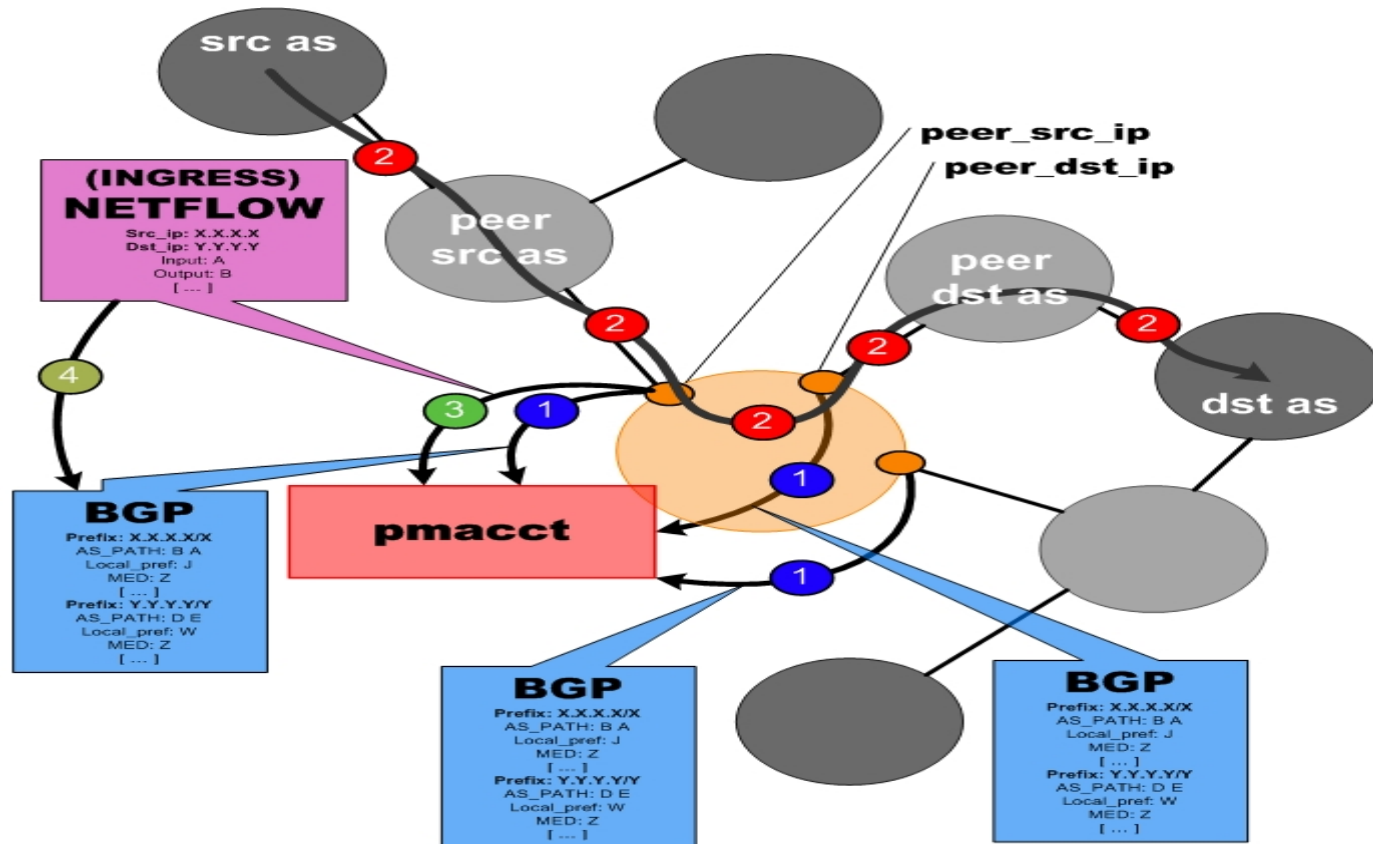
# Getting BGP to the collector (2/2)

- Set the collector as iBGP peer at the PE devices:
  - Configure it as a RR client for best results
  - Collector acts as iBGP peer across (sub-)ASes
- BGP next-hop has to represent the remote edge of the network model:
  - Typical scenario for MPLS networks
  - Can be followed up to cover specific scenarios like:
    - BGP confederations
      - Optionally polish the AS-Path up from sub-ASNs
    - default gateway defined due to partial or default-only routing tables

# Getting telemetry to the collector

- Export ingress-only measurements at all PE devices: facing peers, transit and customers.
  - Traffic is routed to destination, so plenty of information on where it is going to
  - It is crucial instead to get as much as possible about where traffic is coming from
- Leverage data reduction techniques at the PE:
  - Sampling
  - Aggregation (but be sure to carry IP prefixes!)

# Telemetry data/BGP correlation



- 1 Edge routers send full BGP tables to pmacct
- 2 Traffic flows
- 3 NetFlow records are sent to pmacct
- 4 pmacct looks up BGP information: NF src addr == BGP src addr

# Touching ground: a config snippet for traffic matrices

```
plugins: mysql[int_tm], mysql[ext_tm]
```

```
!
```

```
aggregate[int_tm]: in_iface, peer_src_ip, \  
  peer_dst_ip, peer_dst_as
```

```
!
```

```
aggregate[ext_tm]: src_as, in_iface, peer_src_ip, \  
  peer_dst_ip, peer_dst_as, as_path, dst_as
```

```
!
```

```
sql_table[int_tm]: int_tm-%Y%m%d %H%M
```

```
!
```

```
sql_table[ext_tm]: ext_tm-%Y%m%d %H%M
```

```
sql_cache_entries[ext_tm]: 99991
```

```
!
```

```
sql_refresh_time: 300
```

```
sql_history: 5m
```

```
!
```

```
! <rest of config skipped>
```

Port-to-port internal TM

AS-to-AS external TM

Dynamic SQL table names,  
ie.: XXX\_tm-20130803\_1400

Insert data every 5 mins

Build time bins of 5 mins

# Touching ground: how data would look like: internal traffic matrix example (1/2)

```
mysql> SHOW TABLES FROM pmacct;
```

```
+-----+
| Tables_in_pmacct |
+-----+
| ...              |
| int_tm-20130803_1400 |
| int_tm-20130803_1405 |
| int_tm-20130803_1410 |
| ...              |
| ext_tm-20130803_1400 |
| ext_tm-20130803_1405 |
| ext_tm-20130803_1410 |
| ...              |
+-----+
```



Set of internal traffic matrices



Set of external traffic matrices

NOTE: sub-aggregation is expensive: we could also have had our traffic matrices over multiple temporal aggregations in parallel, ie. 5 mins (as above) but also hourly and daily.

# Touching ground: how data would look like: internal traffic matrix example (2/2)

```
mysql> SELECT * FROM int_tm-20130803_1400 LIMIT 10;
```

iface_in	peer_ip_src	peer_ip_dst	peer_dst_as	stamp_inserted	bytes
212	10.0.0.107	10.0.0.3	65000	03-08-2013 14:00	859
212	10.0.0.107	10.0.0.253	65001	03-08-2013 14:00	5358
212	10.0.0.107	10.0.0.234	65002	03-08-2013 14:00	6181
212	10.0.0.107	10.0.0.251	65003	03-08-2013 14:00	27002
205	10.0.0.107	10.0.0.233	65004	03-08-2013 14:00	1200
258	10.0.0.107	10.0.0.240	65005	03-08-2013 14:00	560
212	10.0.0.107	10.0.0.252	65006	03-08-2013 14:00	62682
212	10.0.0.107	10.0.0.234	65007	03-08-2013 14:00	3843
212	10.0.0.107	10.0.0.17	65008	03-08-2013 14:00	21074
205	10.0.0.107	10.0.0.254	65009	03-08-2013 14:00	2023

10 rows in set (0.000 sec)

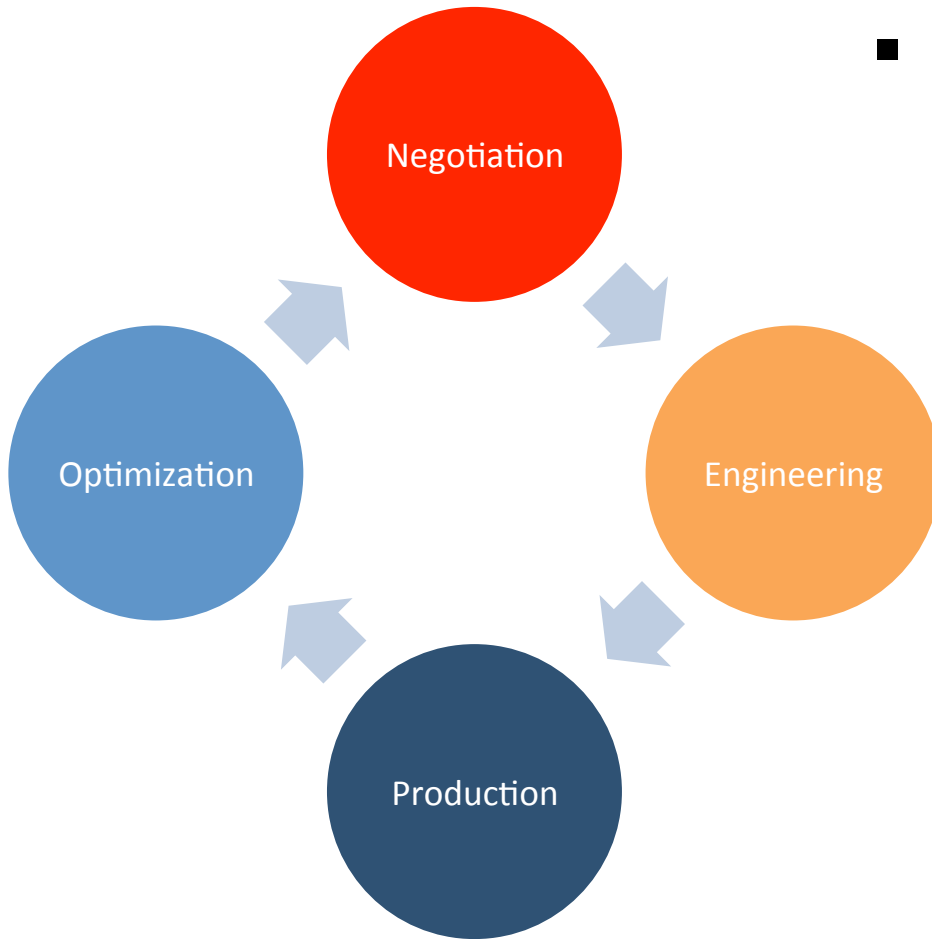
Here is our matrix

Getting ingress NetFlow from 10.0.0.7

Time reference

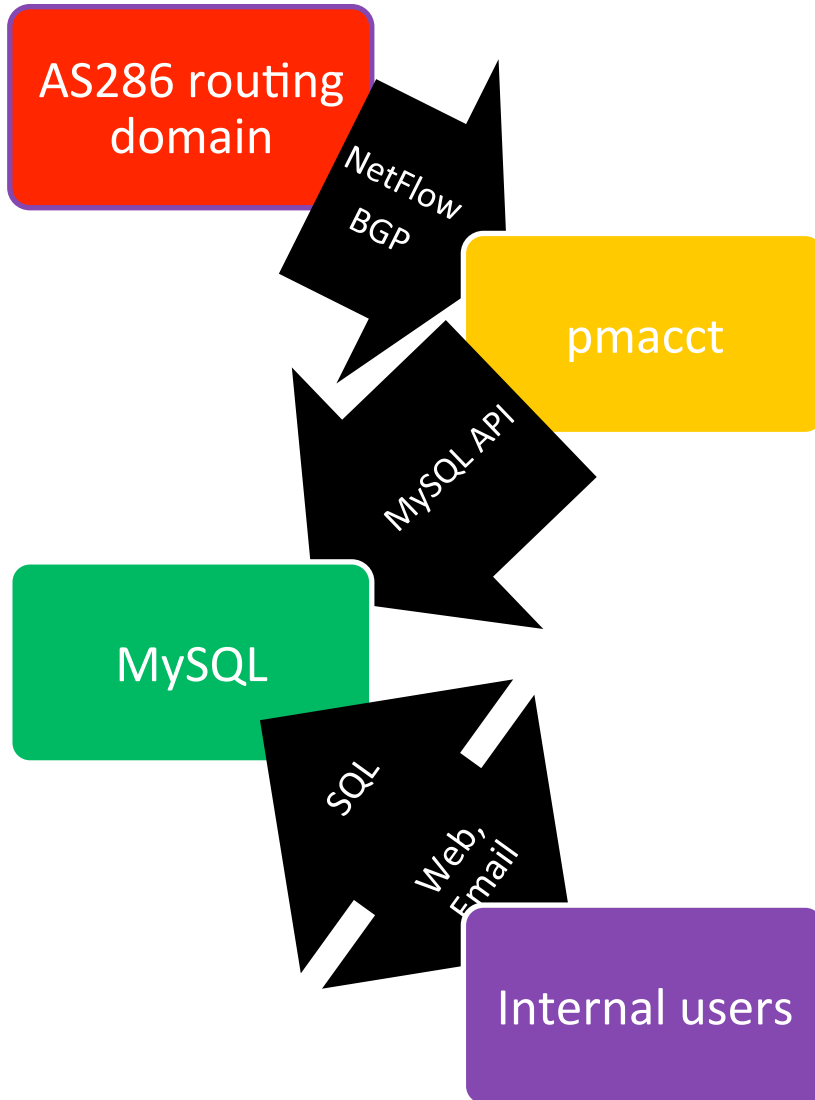
Amount of traffic sent in the time window 14:00 – 14:05

# Case-study: peering at AS286



- Peering as a cycle
- NetFlow + BGP traffic matrix steers peering optimization:
  - Identify new and “old” peers
  - Traffic analysis: backbone costs, 95<sup>th</sup> percentiles, ratios
  - Analysis of interconnection density and traffic dispersion
  - Forecasting and trending
  - Ad-hoc queries from Design & Engineering and indeed ... the IPT Product Manager

# Case-study: peering at AS286



- 250+ Gbps routing-domain
- 100+ high-end routers around the globe:
  - Export sampled NetFlow
  - Advertise full routing table
  - Mix of Juniper and Cisco
- Collector environment:
  - Runs on 2 Solaris/SPARC zones
  - pmacct: dual-core, 4GB RAM
  - MySQL: quad-core, 24GB RAM, 500 GB disk
- Data retention: 6 months



# Case-study: peering at AS286

- AS286 backbone routers are first configured from templates:
  - NetFlow + BGP collector IP address defined over there
  - Enabler for auto-discovery of new devices
- Edge interfaces are provisioned following service delivery manuals:
  - Relevant manuals and TSDs include NetFlow activation
  - Periodic checks NetFlow is active where it should
- Maps, ie. source peer-AS, are re-built periodically

# Collecting NetFlow with pmacct

## Agenda

- o Introduction
- o pmacct architecture & benefits
- o example, data aggregation: traffic matrices
- o **example, logging micro-flows or events**
- o tee: briefly on horizontal scalability

# pmacct, logging & flat-files: brief history (1/2)

- Originally pmacct was about memory tables and RDBMS (no flat-files).
- It was also about data aggregation (no logging micro-flows).
- This is because there were already tools doing this – so why spending time in reinventing the wheel?
- Much better invest time in filling the gaps and give room to innovative ideas

# pmacct, logging & flat-files: brief history (2/2)

- Recently the landscape is changing and NetFlow and IPFIX protocols are being generalized
- Noticeably, they are entering in the Event Logging space, ie.:
  - Cisco NEL, ie. (CG)NAT events
  - Cisco NSEL, ie. security events
- This is great time to review the strategy in this sense introducing support for both flat-files and logging (of events and, as a consequence, of micro-flows)

# Logging use-cases

- Micro-flows:
  - R&D, lab activities
  - Security, DDoS detection, forensics
  - Related to data aggregation:
    - Analysis, temporary: elaborate aggregation methods
    - Back-up, permanent: keep raw data in case useful
- Events:
  - NAT events: compliancy with local authorities
  - FW events: security reports

# Logging feature status (1/2)

- Split data basing on time of the day:
  - Ie. 5 mins, hourly, daily files (fully customizable)
  - Not (yet?) possible to split content basing on file size
- Can build a structure of directories basing on time of the day and (some) primitive values (ie. IP address of the telemetry exporter):
  - To facilitate archiving (files can be also further archived right away via triggers, `print_trigger_exec`)
  - To control files/directory ratio more fine-grained
- Can append to an existing file: ie. hourly files but refreshed with data every 5 mins.

# Logging feature status (2/2)

- Files in text-format only: tab-spaced, CSV and JSON formats:
  - no binary format (yet?)
  - Ideas behind it so far:
    - Why constraining users to capabilities of a client tool?
    - If binary is really wanted, insert in a noSQL database
  - When both compressed, the file size ratio, in CSV format, to an equivalent file from flow-tools is 4:1 (so four times bigger). Worth direct access to data?
- Will appreciate your feedback!

# Touching ground: a config snippet for logging micro-flows

```
plugins: print[forensics]
```

```
!
```

```
aggregate[forensics]: src_host, dst_host, \  
  peer_src_ip, peer_dst_ip, in_iface, out_iface, \  
  timestamp_start, timestamp_end, src_port, \  
  dst_port, proto, tos, src_mask, dst_mask, src_as, \  
  dst_as, tcpflags
```

```
!
```

```
print_output_file[forensics]: /path/to/forensics-%Y%m%d_%H%M.txt
```

```
print_output[forensics]: csv
```

```
print_refresh_time[forensics]: 300
```

```
print_output_file_append[forensics]: true
```

```
!
```

```
! from pmacct 1.5.0rc2
```

```
print_latest_file[forensics]: /path/to/forensics-latest
```

```
! <rest of config skipped>
```

Micro-flow (de)aggregation

Dynamic file names, ie.:  
forensics-20130803\_1400

[ formatted, csv,  
json ]

Insert data every 5 mins,  
append to file if exists

Pointer to latest file to  
become optional and  
explicitly configured



# Touching ground: how data would look like: logging micro-flows (1/2)

```
shell> ls -la
...
-rw----- 1 pmacct pmacct <size> Aug 02 13:50 forensics-20130802-1345.txt
-rw----- 1 pmacct pmacct <size> Aug 02 13:55 forensics-20130802-1350.txt
-rw----- 1 pmacct pmacct <size> Aug 02 14:00 forensics-20130802-1355.txt
-rw----- 1 pmacct pmacct <size> Aug 02 14:05 forensics-20130802-1400.txt
lrwxrwxrwx 1 pmacct pmacct 10 Aug 02 14:05 forensics-latest ->
forensics-20130802-1400.txt
...
```



Configurable ownership



Pointer to latest finalized file

# Touching ground: how data would look like: logging micro-flows (2/2)

```
shell> cat forensics-latest
```

```
SRC_AS,DST_AS,PEER_SRC_IP,PEER_DST_IP,IN_IFACE,OUT_IFACE,SRC_IP,DST_IP,  
SRC_MASK,DST_MASK,SRC_PORT,DST_PORT,TCP_FLAGS,PROTOCOL,TOS,TIMESTAMP_ST  
ART,TIMESTAMP_END,PACKETS,BYTES
```

```
65001,65002,10.0.0.1,10.0.0.100,101,8,192.168.158.133,192.168.126.141,2  
4,24,61912,22,24,tcp,16,2013-08-04 17:40:12.167216,2013-08-04  
17:41:36.140279,21,1407
```

```
[ .. ]
```

# Touching ground: a config snippet for logging NAT events

```
plugins: print[cgn]
!
aggregate[cgn]: src_host, post_nat_src_host, src_port, \
  post_nat_src_port, proto, nat_event, timestamp_start
!
print_output_file[cgn]: /path/to/cgn-%Y%m%d_%H%M.txt
print_output[cgn]: json
print_refresh_time[cgn]: 300
print_cache_entries[cgn]: 9999991
print_output_file_append[cgn]: true
! <rest of config skipped>
```

Well-defined NAT event

Let's increase readability 😊

Bigger cache size to cope with increased number of records

NOTE 1: see config snippet for micro-flows (a few slides back) for additional comments on configuration directives listed above.

NOTE 2: a bigger cache is beneficial to limit scattering of writes to the backend. If the configured cache is unable to contain all records, a purge of data to the backend is triggered and cache content is flushed so to make room to new data.

# Touching ground: how data would look like: logging NAT events (1/2)

```
shell> ls -la
...
-rw----- 1 pmacct pmacct <size> Aug 02 13:50 cgn-20130802-1345.txt
-rw----- 1 pmacct pmacct <size> Aug 02 13:55 cgn-20130802-1350.txt
-rw----- 1 pmacct pmacct <size> Aug 02 14:00 cgn-20130802-1355.txt
-rw----- 1 pmacct pmacct <size> Aug 02 14:05 cgn-20130802-1400.txt
lrwxrwxrwx 1 pmacct pmacct 10 Aug 02 14:05 cgn-latest ->
cgn-20130802-1400.txt
...
```

# Touching ground: how data would look like: logging NAT events (2/2)

```
shell> cat cgn-latest
```

```
{"timestamp_start": "2013-02-21 16:56:33.518000000", "ip_proto": "tcp",  
"post_nat_ip_src": "1.2.179.16", "ip_src": "192.168.37.51", "port_src":  
61020, "nat_event": 1, "post_nat_port_src": 31392}  
[ .. ]
```

# A single (set of) collector(s) for both micro-flows and events logging?

- Yes, possible:
  - All NetFlow, regardless, pointed to the same place
  - Makes sense on small-medium deployments
  - On larger ones potentially pressuring the collector with DDoS's and CGN blades rebooting is not a good idea. Go for splitting.
- pmacct able to tag and channelize data (ie. send data selectively to plugins) basing on a number of clauses, ie. NAT event information element.

# Touching ground: a config snippet for both micro-flows and event logging (1/2)

```
plugins: print[forensics], print[cgn]
!  
pre_tag_filter[forensics]: 10  
aggregate[forensics]: <micro-flows aggregation method>  
print_output_file[forensics]: /path/to/forensics-%Y%m%d_%H%M.txt  
!  
pre_tag_filter[cgn]: 20  
aggregate[cgn]: <NAT events aggregation method>  
print_output_file[cgn]: /path/to/cgn-%Y%m%d_%H%M.txt  
print_cache_entries[cgn]: 9999991  
!  
print_output: csv  
print_refresh_time: 300  
print_output_file_append: true  
!  
pre_tag_map: /path/to/pretag.map  
! <rest of config skipped>
```

Allowing only flows through  
(see next slide)

Allowing only events  
through (see next slide)

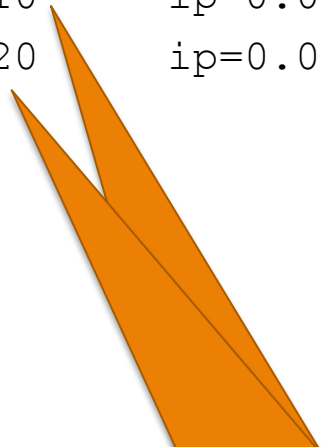
Enabler for channelizing  
data to the correct plugin  
instance: map to assign tags  
to records

NOTE: This configuration merely merges the micro-flows and event logging configurations seen before. Check them out (a few slides back) for additional comments on configuration directives listed above.

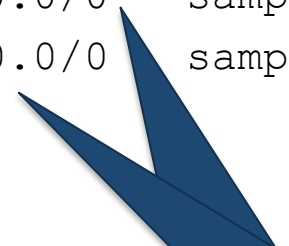
# Touching ground: a config snippet for both micro-flows and event logging (2/2)

```
shell> cat pretag.map
```

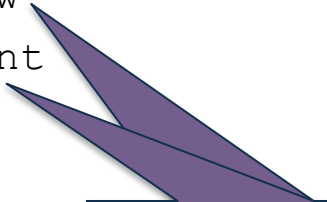
```
set_tag=10      ip=0.0.0.0/0   sample_type=flow
set_tag=20      ip=0.0.0.0/0   sample_type=event
```



Tags are assigned in pretag.map and recalled in the configuration by pre\_tag\_filter



Apply to all NetFlow/IPFIX exporters



Apply heuristics to classify records among flows and events



# Collecting NetFlow with pmacct

## Agenda

- o Introduction
- o pmacct architecture & benefits
- o example, data aggregation: traffic matrices
- o example, logging micro-flows or events
- o **tee: briefly on horizontal scalability**

# Briefly on scalability

- A single collector might not fit it all:
  - Memory: can't store all BGP full routing tables
  - CPU: can't cope with the pace of telemetry export
  - Divide-et-impera approach is valid:
    - Assign PEs (both telemetry and BGP) to collectors
    - Assign collectors to databases; or cluster the database.
- A traffic matrix can get big, but can be reduced:
  - Keep smaller routers out of the equation
  - Filter out specific services/customers on dense routers
  - Focus on relevant traffic direction (ie. upstream if CDN, downstream if ISP)
  - Sample or put thresholds on traffic relevance

# Need for horizontal scalability in a telemetry collector

- Cope with increasing data rates:
  - 10G to 100G but depending on the application sampling rates might stay the same
  - Events logging: ie. NetFlow now challenges Syslog to log Carrier Grade NAT (CGN) and firewall events
- Scale without super-computing powers
- Hashing can help coupling information, ie.
  - NetFlow exporter IP address for traffic matrices
  - Destination IP address for security

# pmacct & horizontal scaling

- Supports a 'tee' plugin
  - Receivers can be added/changed/removed on the fly
  - Load-balanced tee'ing (hashed or round-robin)
  - Selective tee'ing (based on telemetry exporter)
- Multiple pmacct instances can run in parallel
  - Coupling telemetry and routing data from same PE
- Supports multiple backend plugins
  - Build multiple views of the same traffic hence creating project-oriented data-set

# Touching ground: a config snippet for transparent hashing balancer (1/2)

```
plugins: tee[balancer]
!  
! In Linux: adjust /proc/sys/net/core/rmem_max to cope with this value  
! plugin_pipe_size[default]: 1024000000  
!  
! plugin_pipe_size[balancer]: 2048000000  
! plugin_buffer_size[balancer]: 10240000  
!  
tee_receivers[balancer]: /path/to/tee_receivers.lst  
tee_transparent: true
```

Instantiating a tee plugin

Buffering: to handle sustained packet rates. Here commented out.

File containing receivers definitions

Transparent balancing enabled. Disabling it acts as a proxy

# Touching ground: a config snippet for transparent hashing balancer (2/2)

```
shell> cat tee_receivers.lst
```


```
id=1 \
```

```
ip=192.168.5.1:2100,192.168.5.2:2100,192.168.5.3:2100 \
```

```
balance-alg=hash-agent
```

# Touching ground: a config snippet for transparent selective balancer (1/2)

```
plugins: tee[balancer]
!
! In Linux: adjust /proc/sys/net/core/rmem_max to cope with this value
! plugin_pipe_size[default]: 1024000000
!
! plugin_pipe_size[balancer]: 2048000000
! plugin_buffer_size[balancer]: 10240000
!
tee_receivers[balancer]: /path/to/tee_receivers.lst
tee_transparent: true
!
pre_tag_map: /path/to/pretag.map
```



Enabler for selective balancing: map to assign tags to NetFlow/IPFIX exporters

NOTE: see config snippet for transparent hashing balancer (a few slides back) for additional comments on configuration directives listed above.

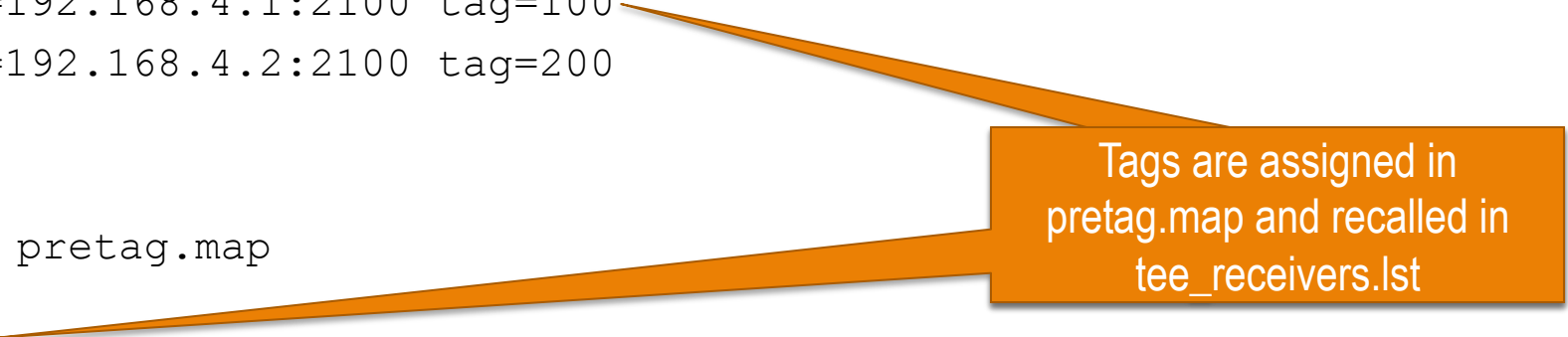
# Touching ground: a config snippet for transparent selective balancer (2/2)

```
shell> cat tee_receivers.lst
```

```
id=2    ip=192.168.4.1:2100 tag=100  
id=3    ip=192.168.4.2:2100 tag=200
```

```
shell> cat pretag.map
```

```
set_tag=100    ip=10.0.0.0/25  
set_tag=200    ip=10.0.0.128/25
```



Tags are assigned in  
pretag.map and recalled in  
tee\_receivers.lst



# Further information

- [http://www.pmacct.net/lucente\\_pmacct\\_uknof14.pdf](http://www.pmacct.net/lucente_pmacct_uknof14.pdf)
  - More about coupling telemetry and BGP
- <http://ripe61.ripe.net/presentations/156-ripe61-bcp-planning-and-te.pdf>
  - More about traffic matrices, capacity planning & TE
- <http://wiki.pmacct.net/OfficialExamples>
  - Compiling instructions and quick-start guides
- <http://wiki.pmacct.net/ImplementationNotes>
  - Implementation notes (RDBMS, maintenance, etc.)

# Collecting NetFlow with pmacct

Thanks for your attention!  
Questions? Now or later ..

Paolo Lucente  
pmacct

<paolo at pmacct dot net>  
Keep in touch via LinkedIn

# Collecting NetFlow with pmacct

Backup slides

# Post-processing RDBMS and reporting (1/2)

- Traffic delivered to a BGP peer, per location:

```
mysql> SELECT peer_as_dst, peer_ip_dst, SUM(bytes), stamp_inserted
        FROM acct_bgp
        WHERE peer_as_dst = <peer | customer | IP transit> AND
              stamp_inserted = < today | last hour | last 5 mins >
        GROUP BY peer_as_dst, peer_ip_dst;
```

- Aggregate AS PATHs to the second hop:

```
mysql> SELECT SUBSTRING_INDEX(as_path, '.', 2) AS as_path, bytes
        FROM acct_bgp
        WHERE local_pref = < IP transit pref> AND
              stamp_inserted = < today | yesterday | last week >
        GROUP BY SUBSTRING_INDEX(as_path, '.', 2)
        ORDER BY SUM(bytes);
```

- Focus peak hour (say, 8pm) data:

```
mysql> SELECT ... FROM ... WHERE stamp_inserted LIKE '2010-02-% 20:00:00'
...

```

# Post-processing RDBMS and reporting (2/2)

- Traffic breakdown, ie. top N grouping BGP peers of the same kind (ie. peers, customers, transit):

```
mysql> SELECT ... FROM ... WHERE ...  
        local_pref = <<peer | customer | IP transit> pref>  
        ...
```

- Download traffic matrix (or a subset of it) to 3<sup>rd</sup> party backbone planning/traffic engineering application (ie. Cariden, Wandl, etc.):

```
mysql> SELECT peer_ip_src, peer_ip_dst, bytes, stamp_inserted  
        FROM acct_bgp  
        WHERE [ peer_ip_src = <location A> AND  
                peer_ip_dst = <location Z> AND ... ]  
                stamp_inserted = < today | last hour | last 5 mins >  
        GROUP BY peer_ip_src, peer_ip_dst;
```