E-Center Data Retrieval Service (DRS) 10/15/2010

Version 2.3

Previous Versions: 1.006a, 2.001a, 2.2b

Maxim Grigoriev

Introduction

The E-center consists of collection of the loosely coupled applications. It will be kept this way in order to allow independent development, easy maintenance and support. In order to make these applications to communicate we assume REST based web services approach.
How data is cached, retrieved from remote perfSONAR-PS(pS-PS) services, archived, pre-fetched behind the scenes is not covered in this document. In general it is valid to assume that all data is coming from the certain kind of the remote pS-PS service – Measurement Archive (MA).
Any resource representing the network measurement data will be easily addressed via simple URL and retrieved by sending HTTP request – for example to get pinger data for the end to end path with source and destination pair IP1=131.225.1.1 and IP2=130.224.3.3 one may send this query:

GET http://<ecenter server hostname>/data/pinger.json?src_ip=131.225.1.1&dst_ip=130.224.3.3&start=2010-05-02%20:01:02&end=2010-05-02%20:01:02

In addition, this document provides description of the more convenient data retrieval request targeted on the complete End-to-End path coverage.

Basic Concepts

The API uses HTTP GET request type and will return response encoded as JSON,YAML or XML data and HTTP status codes as:

- **200 OK**: Success, JSON encoded response is expected
- **400 Bad Request**: The request was invalid. An accompanying error message will explain why, see below for more details.
- **404 Not Found**: The URL requested is invalid or the resource requested, such as a metadata id, does not exists.
- **500 Internal Server Error**: Something is terribly broken or misconfigured. Report this error to the E-Center mailing list.
- **502 Bad Gateway**: E-Center service is down.

The E-Center DRS API follows REST principles as closely as possible. There is no default format for the response encoding. The service will send back **404 Error** code if format is not set in the request. One has to specify in what format the body of response is requested. For example by adding ***.json*** to the end of the requested resource path one may expect to receive **JSON** encoded response. The client 's API should utilize corresponded **JSON** parsing facility available for the language of choice. Or it may request **XML** or **YAML** formatted data structure by adding ***.xml*** or ***.yaml*** correspondently. Please note all examples of the DRS responses in this document are formatted for the case of **JSON** encoding. The **JSON** encoding is used by the E-Center front-end.
The easiest way to get data from DRS is to use ***curl*** command line tool available on any UNIX platform. Or utilize any HTTP protocol API library for particular language of choice – perl, python, java…etc.

Data Retrieval Protocol

Each request is constructed as a specially formed URI with supplied string of parameters. The most basic resource is **data** time series.
As was noted above, in case of error the corresponded HTTP status will be returned (500 for example) and instead of normal response the JSON encoded error message could be utilized by client application:
Next results are for the example where ***src_ip*** and ***dst_ip*** are supplied as and the possible traceroute command would result in this trace:

Traceroute:

from - 192.12.15.26(lhcprefmon.bnl.gov)

to - 131.243.24.11(nettest.bnl.gov)

```
traceroute to 131.243.24.11 (131.243.24.11), 30 hops max, 140 byte packets
 1  mutt.usatlas.bnl.gov (192.12.15.224)  0.538 ms  0.633 ms  0.628 ms
 2  amon.bnl.gov (130.199.3.124)  0.476 ms  0.469 ms  0.593 ms
 3  bnlmr2-bnlsite.es.net (198.124.216.189)  0.424 ms  0.424 ms  0.416 ms
 4  aofacr2-ip-bnlmr2.es.net (134.55.217.142)  2.566 ms  2.548 ms  2.533 ms
 5  bostcr1-ip-aofacr2.es.net (134.55.41.121)  7.460 ms  7.459 ms  7.444 ms
 6  clevcr1-ip-bostcr1.es.net (134.55.41.145)  20.577 ms  20.609 ms  20.524 ms
 7  chiccr1-ip-clevcr1.es.net (134.55.217.54)  29.848 ms  29.827 ms  29.827 ms
 8  kanscr1-ip-chiccr1.es.net (134.55.221.57)  40.235 ms  40.205 ms  40.208 ms
 9  denvcr2-ip-kanscr1.es.net (134.55.209.45)  53.373 ms  53.345 ms  53.350 ms
10  sunncr1-denvcr2.es.net (134.55.220.50)  80.493 ms  80.503 ms  80.479 ms
11  sunnsdn2-sunncr1.es.net (134.55.209.97)  80.458 ms  80.415 ms  80.457 ms
12  slacmr2-ip-sunnsdn2.es.net (134.55.217.1)  80.856 ms  80.831 ms  80.820 ms
13  lblmr2-ip-slacmr2.es.net (134.55.219.9)  81.723 ms  81.674 ms  81.559 ms
14  lbnl-ge-lblmr2.es.net (198.129.224.1)  82.135 ms  82.091 ms  82.090 ms
15  ir2gw.lbl.gov (131.243.128.12)  82.245 ms  82.058 ms  82.042 ms
16  nettest.lbl.gov (131.243.24.11)  81.734 ms  81.728 ms  81.722 ms
```

And partial results (truncated for this document to the first 3 network segments in the trace):


```
{direct_traceroute: { 92: [ {
      longitude: "-72.8805",
```

```
        nodename: "bnl-mr1-bnl-pt1.es.net",
        hop_delay: "0.093",
        hub: "bnl-mr2",
        hop_id: "778",
        hop_num: "1",
        trace_id: "92",
        latitude: "40.8703",
        hop_ip: "198.124.238.37",
        updated: "1286570606",
        netmask: "30",
        hops: "12",
        },{
        longitude: "-74.005",
        nodename: "aofacr2-ip-bnlmr2.es.net",
        hop_delay: "2.224",
        hub: "aofa-cr2",
        hop_id: "779",
        hop_num: "2",
        trace_id: "92",
        latitude: "40.72",
        hop_ip: "134.55.217.142",
        updated: "1286570606",
        netmask: "30",
        hops: "12",
        },{
        longitude: "-71.0868",
        nodename: "bostcr1-ip-aofacr2.es.net",
        hop_delay: "7.089",
        hub: "bost-cr1",
        hop_id: "780",
        hop_num: "3",
        trace_id: "92",
        latitude: "42.3673",
        hop_ip: "134.55.41.121",
        updated: "1286570606",
        netmask: "30",
        hops: "12",
    } ] },
reverse_traceroute: {143: [ {
            longitude: "-122.252",
            nodename: "lblmr2-lblpt1.es.net",
            hop_delay: "0.093",
            hub: "lbl-mr2",
            hop_id: "1324",
            hop_num: "1",
            trace_id: "143",
            latitude: "37.8769",
            hop_ip: "198.129.254.29",
            updated: "1286570736",
            netmask: "30",
            hops: "12",
            },{
            longitude: "-122.204",
            nodename: "slacmr2-ip-lblmr2.es.net",
            hop_delay: "1.128",
            hub: "slac-mr2",
            hop_id: "1325",
            hop_num: "2",
            trace_id: "143",
            latitude: "37.4211",
            hop_ip: "134.55.219.10",
            updated: "1286570736",
            netmask: "30",
            hops: "12",
            },{
            longitude: "-121.991",
            nodename: "sunnsdn2-ip-slacmr2.es.net",
            hop_delay: "1.535",
            hub: "sunn-sdn2",
```

```
            hop_id: "1326",
            hop_num: "3",
            trace_id: "143",
            latitude: "37.3739",
            hop_ip: "134.55.217.2",
            updated: "1286570736",
            netmask: "30",
            hops: "12",
 }]},
 snmp:  {  1325: [
                    [ 1286570430, {  utilization: 211551400,
                                     capacity: 10000000000,
                                   }
                  ], [1286570580, { utilization: 199693000
                                    capacity: 10000000000
                                  }
                     ]
            ],
            778: [
                  [1286570430, {utilization: 493038181.33334
                                 capacity: 10000000000
                               }
                  ], [1286570580, {utilization: 2084.47994
                                    capacity: 10000000000
                                  }
                  ]
       }
}
```

Notice that the ***hop_id=1325*** from the traceroute data fragment is used as the key for the **SNMP** time series data. If reverse traceroute results are available then it will be present in the Response data set under the ***reverse_traceroute*** key and DRSwill return SNMP data for the additional ***hop_ids*** as well.

Data Request

The request URI construction for data time series retrieval must follow these rules:

ecenter_server_hostname := <hostname of the E-Center data service>;
ipv4 := <ipv4 address, dotted>;
datetime := < YYYY-MM-DD%20hh:mm:ss >;
data_type :='/', ['pinger'|'snmp'|'bwctl'|'owamp'];
response_format := ['.json'|'.xml'|'.yaml'];
data_request := ' GET http://', ecenter_server_hostname , '/data', data_type?, response_format, '?', ['src_ip=', ipv4|'src_hub=',string],'&', ['dst_ip=', ipv4|'dst_hub=',string], ('&start=', datetime)?, ('&end=', datetime)?,('&resolution=',integer)?;

Where ***pinger*** for example should be used for requesting data from the PingER service. The ***start*** and ***end*** parameters should be provided to specify time period for the time series. The default value for the ***start*** is *12 hours* ago from *the* "**end**" value and for the ***end*** is *now – current time in UTC*.
When ***data_type*** is omitted then DRS will return all available data from all services.
The resolution parameter is optional as well and used to specify number of datapoints to be returned for each type of data requested. The default value is **100**. The DRS will return average values for the aggregated data points.

As one can see there are two distinctive types of the end-point used for the data requests. One is based on the provided IPv4 address and another one is HUB based. Where **HUB** is the identifier used for the aggregated collection of the end-site pS-PS monitoring services along with ESnet pS-PS services located at the border of the end-site. The naming scheme for **HUB** names is following the ESnet Layer2/3 topology scheme.

Data Response

The response for the data request will follow the next format (in case of **JSON** response format, but the data flow is the same for **XML** or **YAML**):

```
data_id :=  integer;
trace_id := integer;
hop_id := integer;
timestamp := integer;
metric_name :=
['minRtt'|'maxRtt'|'medianRtt'|'minIpd'|'maxIpd'|'meanIpd'|'duplicates'|'clp'|iqrIpd'|
'lossPercent'|'throughput'|'jitter'|'lost'|'sent'|'min'|'max'|minttl'|'maxttl'|'dups'|'maxerr'|
'max_delay'|'min_delay'|'timestamp'];
snmp_metric := ['capacity'|'utilization'];
hop_names  := ['hop_id'|'hop_ip'|'hop_num'|'hop_delay'|'number_hops'|
              'updated'|'longitude'|'latitude'|'nodename'|'hub'|'trace_id'|'netmask'|'hops'];
data_result := [integer|float|'NULL'];
hop_result=[data_result|ipv4|timestamp];
metric_stat := [metric_name|snmp_metric], ':', data_result;
hop_stat := hop_names, ':',   hop_result;
data_item := '{ direct_traceroute:  {  ',    trace_id, ':[ ',
                                    {  '{',  hop_stat, { ',', hop_stat }*,  '}'  }*,
                                    '] }, ',
              'reverse_traceroute:  {  ',    trace_id, ':[ ',
                                     {  '{',  hop_stat, { ',', hop_stat }*,  '}'  }*,
                                    '] }, ',

               {  'snmp:  { ', hop_id , ': [' ,
                                 {  timestamp , ': {',  metric_stat, { ',', metric_stat }*, '}'    }*,
                                          '] },'
              }*,
             {  ['owamp'|'bwctl'|'pinger'],  ': {',
                'src_ip:  { ',
                        'dst_ip : { ',
                                    {  timestamp , ':  { ',  metric_stat, { ',', metric_stat }*,  '}'     }*,
                                 '}
                           }
                   },'  }*

              '}';
```

Example of request:

GET http://<ecenter server hostname>/data/snmp.json?src_ip=131.243.24.11&dst_ip=198.32.44.130&start=2010-06-05 06:01:02&end=2010-06-05 07:02:01

Please note that only utilization (SNMP) data is returned per **hop_id**. The End-to-End metrics are returned per source or destination pair (IPv4 address or HUB) and for the whole network path.

Service Informational Requests

First service request is the status one. One can check if service is running by sending this:

response_format := ['.json'|'.xml'|'.yaml'];
hub_request := ' GET http://', ecenter_server_hostname , '/status', response_format;

If everything is fine then it responds with **{ status : 'ok' }** in case of requested JSON encoding.
Or your API will receive a HTTP error code.
There is more complete request/response where several metrics about the DRS health could be requested. It called a ***health*** request and it represented as:

health_request := ' GET http://', ecenter_server_hostname , '/health', response_format,], ('&start=', datetime)?,('&end=', datetime)?;

and response is formatted as:
domain_name := <domain part of the hostname – to aggregate service hosts>
health_response := '{ time_period: {', { ['owamp'|'bwctl'|'pinger'|'traceroute'], ': {',
'start: ',timestamp, ', end: ',timestamp,'},' }*, '}, ',
'metadata: {', { domain_name, ': {', {
['owamp'|'bwctl'|'pinger'|'traceroute'], ': {',
'cached_data_count: ',integer,
'metadata_count: ',integer,'},' }*, ' },', }*
' } }';

Here is the example of the ***"health"*** response:

```
{ time_period: { bwctl: {   end: "1287069738",
                            start: "1287026538"},
                 traceroute: {end: "1287069738",
                            start: "1287026538"},
                   owamp: {end: "1287069738",
                            start: "1287026538"},
                   pinger: {end: "1287069738",
```

```
                                start: "1287026538"}
                        },
                metadata: {es.net: {bwctl: {cached_data_count: "4",
                                            metadata_count: “780” },
                                   traceroute: {cached_data_count: "2",
                                                metadata_count: “879”},
                                    owamp: {cached_data_count: "406",
                                             metadata_count: “1146” },
                                    pinger: {cached_data_count: "0",
                                             metadata_count: “1435” }
                           }
        }
```

One ca notice the **cached_data_count** indicate how many data entries were cached already for some specific domain and **metadata_count** is the number of the cached metadata. The later number is important in understanding if E-Center data collection service were able to acquire any metadata at all. If counter is 0 then E-Center will not be able to send any data requests to the remote pS-PS services.

Next is the basic request for the list of the hubs and could be expressed as:

hub_request := ‘ GET http://’, ecenter_server_hostname , ‘/hub’, response_format;

with response formatted according to:

hub_name := <ESnet provided topology id for the HUB – 4 letters short and uppercase>;
hub_response := ‘{‘ , { hub_name, ‘ : { hub_name:‘, hub_name, ‘, latitude:’, float, ‘ longitude: ‘, float, ‘},’ }+, ‘}’;

Example of the hub request and response:

http://xenmon.fnal.gov:8098/hub.json

_____

```
{  KANS: {     longitude: "-94.5822",
               hub_name: "KANS",
               latitude: "39.1008",
          },
   ALBU: {
               longitude: "-106.647",
               hub_name: "ALBU",
               latitude: "35.0822",
         }
}
```

Next request is used to obtain list of the nodes with available traceroute metadata where node is the originating node for the traceroute – source node:

source_request := ‘ GET http://’, ecenter_server_hostname , ‘/source’, response_format;

and response is formed as:

string := < any character string >;
source_response := '[' , { '{ hub:' , string, ', hub_name:', string, ', nodename:', string, ', ip_noted:', ipv4 ', latitude:', float, ' longitude: ', float, ', netmask:', integer, '},' }+, ']';

To get all available destinations for any of the IPs from the previous response one may use a ***destination*** request:

destination_ip_request := ' GET http://', ecenter _server_hostname , '/destination/', ipv4, response_format;

and response is formed as:

string := < any character string >;
destination_ip_response := '[' , { '{ hub:' , string, ', hub_name:', string, ', nodename:', string, ', ip_noted:', ipv4 ', latitude:', float, ' longitude: ', float, ', netmask:', integer, '},' }+, ']';

Example of the ***destination*** request and response (the same example could be shown for the ***source*** request above):

http://xenmon.fnal.gov:8098/destination/134.79.104.208.json

```
[
    {
    longitude: "-72.8805",
    hub: "bnl-mr2",
    hub_name: "BNL",
    nodename: "bnl-owamp.es.net",
    ip_noted: "198.124.238.49",
    latitude: "40.8703",
    netmask: "24"
    }
]
```


The ***service*** request is utilized to get information about all or some pS-PS services cached by the ECenter.

condition := '/', ['id'|'ip'|'url'|name']
service_request := ' GET http://', ecenter _server_hostname , '/service', condition?, response_format;

Without any conditions it will return all services. And with any condition supplied it will try to match this condition with particular attribute of the service. In case of successful match it will return the same data structure but for the shorten list of services.

service_type := ['pinger'|'traceroute'|'snmp'|'owamp'|'bwctl'|'hLS'];
datetime := < YYYY-MM-DD%20hh:mm:ss >;

service_data := '{ service:', integer, ', name:', string, ', url:', string, ', type:', service_type, ', comments:', string, ', created: ', datetime, ', updated:', datetime, ', ip_noted: ', ipv4, ', is_alive: ', ['0'|'1'], '}';
service_response := '[', service_data, { ',', service_data }* ']';

Example of the service request and response:

http://xenmon.fnal.gov:8098/service.json

_____

```
[
   {
      name: "ESnet Home Lookup Service",
      service: "1",
      comments: "ESnet Home Lookup Service (Berkeley, CA. USA)",
      created: "2010-07-06 15:55:17",
      is_alive: "1",
      ip_noted: "198.129.254.242",
      url: http://ps1.es.net:8095/perfSONAR_PS/services/hLS,
      updated: "2010-07-06 15:55:36",
      type: "hLS",
   }
]
```

Appendix

E-Center DRS Database scheme

```
--
-
--  Schema for  ECenter data
--
--   author: Maxim Grigoriev, 2010, maxim_at_fnal_dot_gov
--
--
drop  database if exists ecenter_data;
create database ecenter_data;
grant all privileges on ecenter_data.* to ecenter@localhost identified by 'ecenter2010';
grant select on ecenter_data.* to www@localhost identified by 'www_user';
flush privileges;
use ecenter_data;

--
--  topology hub ( something with coordinates, name )
--
--
drop table if exists hub;
CREATE TABLE  hub (
hub  varchar(32) NOT NULL,
hub_name  varchar(32) NOT NULL,
description varchar(100)   NOT NULL,
longitude float NULL,
latitude float NULL,
PRIMARY KEY  (hub)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='topology hub (esnet)';

--
```

```
--  topology layer2 info
--
--
drop table if exists l2_port;
CREATE TABLE  l2_port (
l2_urn varchar(512)   NOT NULL,
description varchar(100)   NOT NULL,
capacity   bigint  unsigned   NOT NULL,
hub varchar(32) NOT NULL,
PRIMARY KEY  (l2_urn ),
FOREIGN KEY ( hub ) REFERENCES hub ( hub )
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2 info';
--
--  topology layer2 linkage
--
--
drop table if exists l2_link;
CREATE TABLE  l2_link (
l2_link  bigint  unsigned AUTO_INCREMENT  NOT NULL,
l2_src_urn varchar(512)   NOT NULL,
l2_dst_urn varchar(512)   NOT NULL,
PRIMARY KEY  (l2_link),
FOREIGN KEY ( l2_src_urn ) REFERENCES l2_port ( l2_urn),
FOREIGN KEY ( l2_dst_urn ) REFERENCES l2_port ( l2_urn )
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2  links';
--
--   nodes - all of them
--   ip_addr supports   ipv4 and ipv6 addresses in binary form
--
--    it holds topology info as well via layer2 id ( as http://ogf.org/schema/network/topology/base/20070828/ schema describes)
--
--    it set as INET6_PTON('131.225.1.1') and essentially a 16 byte representation of the IP
--    it allows indexing and netblock search, to get original IP address - INET6_NTOP(ip_addr)
--    to see dotted form of ipv4 or ipv6 - select ip_noted
--
--
drop  table if exists node;
CREATE TABLE  node (
 ip_addr  varbinary(16)  NOT NULL,
 nodename  varchar(255)  NULL,
 ip_noted  varchar(40)  NOT NULL,
 netmask  smallint(3) NOT NULL default '32',
 PRIMARY KEY  (ip_addr),
 KEY (nodename),
 KEY (ip_noted),
 KEY (netmask)
)  ENGINE=InnoDB CHARSET=latin1  COMMENT='nodes';
--
--  topology layer3 mapping to layer2,
--  use netmask from the node table to get all addresses from the block
--
--
drop table if exists l2_l3_map;
CREATE TABLE  l2_l3_map (
l2_l3_map  bigint  unsigned AUTO_INCREMENT  NOT NULL,
ip_addr   varbinary(16)  NOT NULL,
l2_urn    varchar(512)    NOT NULL,
PRIMARY KEY  (l2_l3_map),
FOREIGN KEY (l2_urn) REFERENCES l2_port ( l2_urn ),
FOREIGN KEY ( ip_addr ) REFERENCES node ( ip_addr )
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps topology layer2-layer3 mapping';

--
--    list of keywords, stores most recent regexp used to obtain that keyword as well
--
drop  table if exists keyword;
CREATE TABLE  keyword (
 keyword  varchar(255) NOT NULL,
 pattern  varchar(255)  NULL,
```

```
 PRIMARY KEY  (keyword)
 )  ENGINE=InnoDB CHARSET=latin1  COMMENT='project keywords';
--
--  operational table for the most recent status of the service
--  all services are here
--
drop  table if exists service;
CREATE TABLE   service (
 service  bigint  unsigned AUTO_INCREMENT NOT NULL,
 name varchar(255) NOT NULL,
 ip_addr varbinary(16) NOT NULL,
 url varchar(255) NOT NULL,
 type varchar(32) NOT NULL DEFAULT 'hLS',
 comments  varchar(255) NULL,
 is_alive boolean,
 created  TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
 updated  TIMESTAMP NOT NULL DEFAULT 0,
 PRIMARY KEY  (service),
 UNIQUE KEY url (url),
 KEY is_alive (is_alive),
 FOREIGN KEY (ip_addr) REFERENCES  node (ip_addr)
 ) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps services operational table';
--
--    list of eventtypes
--
drop  table if exists eventtype;
CREATE TABLE   eventtype  (
 ref_id   bigint  unsigned AUTO_INCREMENT NOT NULL,
 eventtype  varchar(255)  NULL,
 service bigint unsigned  NOT NULL,
 PRIMARY KEY  (ref_id),
 UNIQUE KEY eventtype_service (eventtype, service),
 FOREIGN KEY  (service) REFERENCES  service (service)  on DELETE CASCADE ON UPDATE CASCADE
 )  ENGINE=InnoDB CHARSET=latin1 COMMENT='project keywords';
--
--  keywords_service for  many - many rel
--
--
drop table if exists keywords_service;
CREATE TABLE keywords_service (
ref_id bigint  unsigned AUTO_INCREMENT NOT NULL,
keyword varchar(255) NOT NULL,
service bigint  unsigned NOT NULL,
PRIMARY KEY  (ref_id),
UNIQUE KEY key_service (keyword, service),
FOREIGN KEY (keyword) REFERENCES keyword  (keyword) on DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (service) REFERENCES  service (service)  on DELETE CASCADE ON UPDATE CASCADE
)ENGINE=InnoDB CHARSET=latin1  COMMENT='many to many for keywords_services';
--
--  metadata for the service
--  keeping XML in the subject and parameters but extracting end-to-end data
--
--   src - source or interface address depending on type of the service
--   dst - destination address
--   rtr - router address in caseof snmp
--
--
drop  table if exists  metadata;
CREATE TABLE   metadata (
metaid  bigint  unsigned AUTO_INCREMENT NOT NULL,
src_ip  varbinary(16)  NOT NULL,
dst_ip  varbinary(16)  NOT  NULL DEFAULT '0',
direction   enum('in','out') NOT NULL default 'in',
service bigint  unsigned  NOT NULL,
subject varchar(1023) NOT NULL  DEFAULT '',
parameters varchar(1023) NULL,
PRIMARY KEY  (metaid),
KEY  (metaid),
FOREIGN KEY (src_ip) REFERENCES  node (ip_addr),
```

```
FOREIGN KEY (service) REFERENCES  service (service)  on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps metaid provided by each service';
--
--  NEXT TABLES FOR THE  DATA CACHE
--  we are going to store pinger, owamp, bwctl, snmp data
--  based on source -> destination pairs
--
--    generic data table - all non-identified data can go here
--
--
drop  table if exists  data;
CREATE TABLE   data (
data  bigint  unsigned AUTO_INCREMENT NOT NULL,
metaid bigint  unsigned  NOT NULL,
param     varchar(255) NOT NULL,
value   float NOT NULL,
PRIMARY KEY  (data),
KEY  (param),
FOREIGN KEY  (metaid) REFERENCES  metadata (metaid)  on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps  data  cache';
--
--    pinger data storage
--
--
drop  table if exists  pinger_data;
CREATE TABLE   pinger_data  (
 pinger_data bigint  unsigned AUTO_INCREMENT NOT NULL,
 metaid   BIGINT  unsigned   NOT NULL,
 minRtt float  NOT NULL DEFAULT  '0.0',
 meanRtt float NOT NULL DEFAULT  '0.0',
 medianRtt float NOT NULL DEFAULT  '0.0',
 maxRtt float NOT NULL DEFAULT  '0.0',
 timestamp   bigint(12) unsigned NOT NULL,
 minIpd float  NOT NULL DEFAULT  '0.0',
 meanIpd float NOT NULL DEFAULT  '0.0',
 maxIpd float NOT NULL DEFAULT  '0.0',
 duplicates tinyint(1) NOT NULL DEFAULT '0',
 outOfOrder  tinyint(1) NOT NULL DEFAULT '0',
 clp float NOT NULL DEFAULT '0.0',
 iqrIpd float  NOT NULL DEFAULT '0.0',
 lossPercent  float NOT NULL DEFAULT '0.0',
 PRIMARY KEY (pinger_data),
 KEY (timestamp),
 INDEX (meanRtt, medianRtt, lossPercent, meanIpd, clp),
 FOREIGN KEY (metaid) references  metadata  (metaid) on DELETE CASCADE ON UPDATE CASCADE
 ) ENGINE=InnoDB CHARSET=latin1 COMMENT='ps-ps  pinger data  cache';
--
--    BWCTL data  storage
--
--
--
drop  table if exists  bwctl_data;
CREATE TABLE  bwctl_data (
   bwctl_data  bigint  unsigned AUTO_INCREMENT NOT NULL,
   metaid    BIGINT  unsigned NOT NULL,
   timestamp   bigint(12) unsigned NOT NULL,
   throughput  float default NULL,
   jitter      float default NULL,
   lost  int unsigned default NULL,
   sent  int unsigned default NULL,
   PRIMARY KEY  (bwctl_data),
   KEY (timestamp),
   FOREIGN KEY (metaid) REFERENCES   metadata  (metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps  bwctl data  cache';
--
--          OWAMP data  storage
--
--
drop  table if exists owamp_data;
```

```
CREATE TABLE  owamp_data (
  owamp_data  bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid   BIGINT  unsigned  NOT NULL,
  timestamp  bigint(12) unsigned NOT NULL,
  min float  NOT NULL DEFAULT  '0.0',
  max float  NOT NULL DEFAULT  '0.0',
  minttl tinyint(3) unsigned NOT NULL DEFAULT  '0',
  maxttl tinyint(3) unsigned NOT NULL DEFAULT  '0',
  sent   int unsigned NOT NULL DEFAULT  '0',
  lost   int unsigned NOT NULL DEFAULT  '0',
  dups   int unsigned NOT NULL DEFAULT  '0',
  maxerr float  NOT NULL DEFAULT  '0.0',
  PRIMARY KEY  (owamp_data),
  KEY  (timestamp),
  FOREIGN KEY  (metaid) REFERENCES   metadata  (metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps  owamp  data  cache';
--
--          SNMP data  storage
--          knows about layer2 topologies
--
drop  table if exists snmp_data;
CREATE TABLE  snmp_data (
  snmp_data  bigint unsigned AUTO_INCREMENT NOT NULL,
  metaid   BIGINT  unsigned  NOT NULL,
  timestamp  bigint(12) unsigned NOT NULL,
  utilization float  NOT NULL DEFAULT  '0.0',
  errors int unsigned NOT NULL DEFAULT  '0',
  drops int unsigned  NOT NULL DEFAULT  '0',
  PRIMARY KEY  (snmp_data),
  KEY  (timestamp),\
  UNIQUE KEY meta_time (metaid, timestamp),
  FOREIGN KEY (metaid) REFERENCES metadata (metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps snmp data  cache';
--
--  traceroute data table - update only timestamp if nothing changed ( delay < 10% )
--
--
--
drop table if exists  traceroute_data;
CREATE TABLE traceroute_data (
trace_id   bigint  unsigned AUTO_INCREMENT NOT NULL,
metaid   BIGINT  unsigned  NOT NULL,
number_hops tinyint(3) NOT NULL DEFAULT '1',
updated bigint(12) unsigned NOT NULL,
PRIMARY KEY (trace_id),
UNIQUE KEY updated_metaid (metaid, updated),
FOREIGN KEY (metaid) references  metadata (metaid) on DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps traceroute';
--
--  per hop info
--
--
--
drop table if exists hop;
CREATE TABLE hop (
hop_id  bigint  unsigned AUTO_INCREMENT  NOT NULL,
trace_id bigint unsigned NOT NULL,
hop_ip  varbinary(16) NOT NULL,
hop_num tinyint(3) NOT NULL DEFAULT '1',
hop_delay  float NOT NULL DEFAULT '0.0',
PRIMARY KEY (hop_id),
FOREIGN KEY (trace_id) REFERENCES traceroute_data(trace_id),
FOREIGN KEY (hop_ip) REFERENCES  node(ip_addr)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='ps-ps traceroute hops';
```