Psyche Database Description

Sponsored By:

12 Oct 2008

Revision History

Date	Author(s)	Summary of Changes
12 Oct 2008	Ponte Technologies	Initial release; corresponds to Psyche v0.2

Table of Contents

Introduction

Psyche uses a specialized, non-relational PostgreSQL database (DB) to meet its near real-time archival, aggregation, and retrieval needs. The SQL file that defines the DB's schema, as well as its permissions and stored procedures, is named init_db.sql and is located in the distribution package.

The following sections are meant to supplement the comments and explanations found in init_db.sql. That initialization file explains the structure of each table and contains the actual stored procedure code, while this document describes the overall approach and GUI-to-DB application programming interface (API). Note that database installation and configuration instructions are located in the distribution package's INSTALL file.

Design and Approach

With Psyche being a security analysis tool, speed and intelligent preprocessing were two primary goals of the DB's design; the tool should provide relevant, aggregated data to an analyst in a timely fashion. To meet these front-end needs, several different aggregation approaches were implemented, including:

- Source traffic by host/protocol/port
- Destination traffic by host/protocol/port
- Sessions by protocol (i.e., TCP, UDP, and Other)

The DB's tables and stored procedures, described in the following sections, were engineered specifically for these aggregation and retrieval purposes.

Tables

The following table descriptions briefly explain each table's purpose and its use within the Psyche architecture; for granular details of the tables themselves, please refer to the init_db.sql file mentioned above.

flow

The *flow* table stores all the raw flows received by the Psyche Collector, pfcapd. This table, which currently grows indefinitely, will be controlled by an aging process in a future development phase.

Flows will never be deleted once they have been aggregated into the specialized, front-end tables; this allows these tables to be reconstructed in the event of integrity issues or corruption. In addition, this approach easily allows future aggregate tables to encapsulate the flows archived during previous development phases.

flow
id
router
src_addr
dst_addr
start_time
end_time
tcp_flags
prot
input_if
output_if
src_as
dst_as
src_port
dst_port
src_mask
dst_mask
next_hop
pkts
bytes

The Psyche Collector is the only process to currently access the *flow* table; it directly inserts flows into the table and subsequently selects flows from the table via aggregation stored procedures.

src_traffic

The *src_traffic* table contains aggregate source traffic data (by interval) for each unique IP/port/protocol triple seen per router/exporter. It is populated by an aggregation stored procedure, *aggregate_src_traffic()*, invoked by the Psyche Collector several times per second and is leveraged by various front-end stored procedures.

src_traffic
id
interval_start
router
addr
port
prot
bytes
pkts
last_update

dst_traffic

The *dst_traffic* table contains aggregate destination traffic data (by interval) for each unique IP/port/protocol triple seen per router/exporter. It is populated by an aggregation stored procedure, *aggregate_dst_traffic()*, invoked by the Psyche Collector several times per second and is leveraged by various front-end stored procedures.

dst_traffic
id
interval_start
router
addr
port
prot
bytes
pkts
last_update

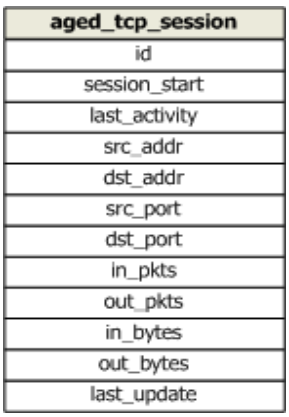
tcp_session

The *tcp_session* table holds reconstructed TCP sessions and is populated by an aggregation stored procedure, *aggregate_tcp_session()*, invoked by the Psyche Collector several times per second. This table is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures.

Note that the *age_tcp_session()* stored procedure moves expired sessions to the *aged_tcp_session* table. By maintaining two separate tables for TCP sessions rather than one containing all TCP sessions, the TCP session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

tcp_session
id
session_start
last_activity
src_addr
dst_addr
src_port
dst_port
in_pkts
out_pkts
in_bytes
out_bytes
last_update

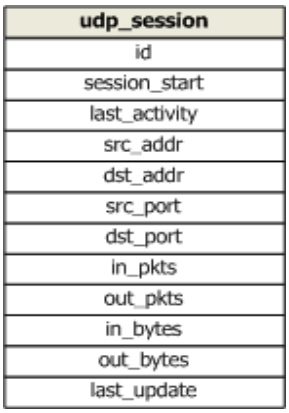
aged_tcp_session



The *aged_tcp_session* table, which holds reconstructed TCP sessions that have been aged out of the *tcp_session* table, is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures. The *age_tcp_session()* stored procedure, invoked by the Psyche Collector several times per second, is responsible for moving old sessions from the *tcp_session* table to the *aged_tcp_session* table.

Again, by maintaining two separate tables for TCP sessions rather than one containing all TCP sessions, the TCP session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

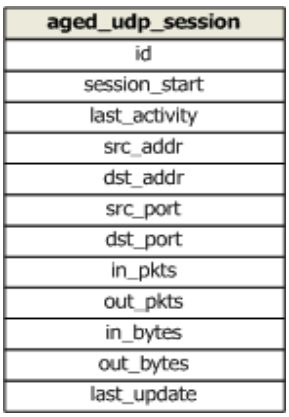
udp_session



The *udp_session* table holds reconstructed UDP sessions and is populated by an aggregation stored procedure, *aggregate_udp_session()*, invoked by the Psyche Collector several times per second. This table is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures.

Note that the *age_udp_session()* stored procedure moves expired sessions to the *aged_udp_session* table. By maintaining two separate tables for UDP sessions rather than one containing all UDP sessions, the UDP session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

aged_udp_session



The *aged_udp_session* table, which holds reconstructed UDP sessions that have been aged out of the *udp_session* table, is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures. The *age_udp_session()* stored procedure, invoked by the Psyche Collector several times per second, is responsible for moving old sessions from the *udp_session* table to the *aged_udp_session* table.

Again, by maintaining two separate tables for UDP sessions rather than one containing all UDP sessions, the UDP session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

other_session

other_session
id
session_start
last_activity
src_addr
dst_addr
src_port
dst_port
in_pkts
out_pkts
in_bytes
out_bytes
last_update

The *other_session* table holds reconstructed 'other' sessions (non-TCP and non-UDP) and is populated by an aggregation stored procedure, *aggregate_other_session()*, invoked by the Psyche Collector several times per second. This table is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures.

Note that the *age_other_session()* stored procedure moves expired 'other' sessions to the *aged_other_session* table. By maintaining two separate tables for these sessions rather than one containing all 'other' sessions, the 'other' session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

aged_other_session

aged_other_session
id
session_start
last_activity
src_addr
dst_addr
src_port
dst_port
in_pkts
out_pkts
in_bytes
out_bytes
last_update

The *aged_other_session* table, which holds reconstructed 'other' sessions that have been aged out of the *other_session* table, is leveraged by the *get_sessions()* and *bw_ratio()* front-end stored procedures. The *age_other_session()* stored procedure, invoked by the Psyche Collector several times per second, is responsible for moving old sessions from the *other_session* table to the *aged_other_session* table.

Again, by maintaining two separate tables for these sessions rather than one containing all 'other' sessions, the 'other' session aggregation stored procedure is able to search a smaller number of rows and minimize its processing time.

Note: There are also various internal state tables that maintain the state information the Psyche Collector needs for aggregation purposes. Holding this data in the DB enables the Collector to remain stateless, which improves its performance and helps protect data integrity if system issues arise.

The Psyche Collector is the only process to currently access and update these state tables, which are solely used to hold internal information and are not intended for front-end use.

Stored Procedures

Stored procedures provide the mechanism by which flow aggregation and front-end data retrieval occur. From an aggregation perspective, these procedures serve as functions invoked by the Psyche Collector to update tables and preserve state within the DB itself. From the front-end, these procedures are used to provide highly configurable views into the database. The following stored procedure descriptions briefly explain the procedures and their roles within Psyche; please refer to the GUI-to-DB API section and the init_db.sql file for code details.

aggregate_src_traffic()

The *aggregate_src_traffic()* stored procedure is used by the Psyche Collector to aggregate flows from the *flow* table into the *src_traffic* table. It determines which flows need to be aggregated by querying the *src_traffic_state* table for the current *last_fid* value, which corresponds to the last flow processed by this aggregation routine. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *src_traffic_state* table for the duration of each transaction.

For new flows being aggregated, this procedure groups data by interval, src_addr, src_port, and protocol into a temporary table, then compares these values to entries in the *src_traffic* table. If a match is found, then the appropriate row in the *src_traffic* table is updated to include the new data. However, if the new flow corresponds to the first time a tuple has been seen, a new row is inserted into the *src_traffic* table. The procedure concludes by updating the *last_fid* value in the *src_traffic_state* table with the id of last flow processed.

aggregate_dst_traffic()

The *aggregate_dst_traffic()* stored procedure is used by the Psyche Collector to aggregate flows from the *flow* table into the *dst_traffic* table. It determines which flows need to be aggregated by querying the *dst_traffic_state* table for the current *last_fid* value, which corresponds to the last flow processed by this aggregation routine. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *dst_traffic_state* table for the duration of each transaction.

For new flows being aggregated, this procedure groups data by interval, dst_addr, dst_port, and protocol into a temporary table, then compares these values to entries in the *dst_traffic* table. If a match is found, then the appropriate row in the *dst_traffic* table is updated to include the new data. However, if the new flow corresponds to the first time a tuple has been seen, a new row is inserted into the *dst_traffic* table. The procedure concludes by updating the *last_fid* value in the *dst_traffic_state* table with the id of last flow processed.

aggregate_tcp_session()

The *aggregate_tcp_session()* stored procedure is used by the Psyche Collector to aggregate flows from the *flow* table into the *tcp_session* table. It determines which flows need to be aggregated by querying the *tcp_session_state* table for the current *last_fid* value, which corresponds to the last flow processed by this aggregation routine. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *tcp_session_state* table for the duration of each transaction.

For each flow being aggregated, this procedure checks to see if the flow belongs to an existing TCP session in the *tcp_session* table. If so, it updates the appropriate fields of the matched row. However, if the new flow belongs to a new TCP session, a new row is inserted into the *tcp_session* table. The procedure concludes by updating the *last_fid* value in the *tcp_session_state* table with the id of last flow processed.

age_tcp_session()

The *age_tcp_session()* stored procedure is used by the Psyche Collector to move old TCP sessions from the *tcp_session* table to the *aged_tcp_session* table. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *aged_tcp_session_state* table for the duration of each transaction.

This function checks the last activity time stamps of sessions in the *tcp_session* table to determine whether or not they should be aged; this technique will be updated in a future development phase to leverage flows' TCP flags.

aggregate_udp_session()

The *aggregate_udp_session()* stored procedure is used by the Psyche Collector to aggregate flows from the *flow* table into the *udp_session* table. It determines which flows need to be aggregated by querying the *udp_session_state* table for the current *last_fid* value, which corresponds to the last flow processed by this aggregation routine. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *udp_session_state* table for the duration of each transaction.

For each flow being aggregated, this procedure checks to see if the flow belongs to an existing UDP session in the *udp_session* table. If so, it updates the appropriate fields of the matched row. However, if the new flow belongs to a new UDP session, a new row is inserted into the *udp_session* table. The procedure concludes by updating the *last_fid* value in the *udp_session_state* table with the id of last flow processed.

age_udp_session()

The *age_udp_session()* stored procedure is used by the Psyche Collector to move old UDP sessions from the *udp_session* table to the *aged_udp_session* table. To prevent potential

concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *aged_udp_session_state* table for the duration of each transaction.

aggregate_other_session()

The *aggregate_other_session()* stored procedure is used by the Psyche Collector to aggregate flows from the *flow* table into the *other_session* table. It determines which flows need to be aggregated by querying the *other_session_state* table for the current *last_fid* value, which corresponds to the last flow processed by this aggregation routine. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *other_session_state* table for the duration of each transaction.

For each flow being aggregated, this procedure checks to see if the flow belongs to an existing session in the *other_session* table. If so, it updates the appropriate fields of the matched row. However, if the new flow belongs to a new session, a new row is inserted into the *other_session* table. The procedure concludes by updating the *last_fid* value in the *other_session_state* table with the id of last flow processed.

age_other_session()

The *age_other_session()* stored procedure is used by the Psyche Collector to move old non-TCP and non-UDP sessions from the *other_session* table to the *aged_other_session* table. To prevent potential concurrency issues between simultaneous invocations of this function, an exclusive lock is placed on the *aged_other_session_state* table for the duration of each transaction.

get_flows()

The *get_flows()* stored procedure is used by the Psyche GUI to populate the grid view on the 'Query Raw Flows' web page. This function leverages the *flow* table to obtain its results.

get_traffic()

The *get_traffic()* stored procedure is used by the Psyche GUI to populate the grid view on the 'Top Talkers' web page. This function leverages the *src_traffic* and *dst_traffic* tables to obtain its results.

get_sessions()

The *get_sessions()* stored procedure is used by the Psyche GUI to populate the grid view on the 'Histogram' web page. This function leverages the various *session* tables to obtain its results.

bw_by_port()

The *bw_by_port()* stored procedure is used by the Psyche GUI when creating the graph located on the 'Top Talkers' web page. This function leverages the *src_traffic* and *dst_traffic* tables to obtain the list of protocol/port pairs that have consumed the most bandwidth over a specified timeframe.

bw_ratio()

The *bw_ratio()* stored procedure is used by the Psyche GUI when creating the graph located on the 'Histogram' web page. This function leverages the various *session* tables to obtain the data points, which represent the distribution of (in_bytes / out_bytes) values of relevant sessions, for the histogram.

Note: The various front-end stored procedures were designed to be flexible, building their final queries dynamically based on their numerous input parameters. Please refer to the GUI-to-DB API section for details.

In addition, these front-end stored procedures leverage various internal 'helper' functions not listed here. Please see the init_db.sql file mentioned above for code details.

GUI-to-DB API

This section documents the API for the various stored procedures accessible by the Psyche GUI. For each stored procedure below, details are provided for input parameters, output parameters, and example invocation syntax.

get_flows()

Parameters

The following function prototype illustrates the stored procedure name and sequence of parameters. Following the prototype is a detailed list of the input parameters.

get_flows(network(varchar(18)), router(varchar(16)), inc_prot_ranges(integer[][]),
inc_prots(integer[]), exc_prot_ranges(integer[][]), exc_prots(integer[]),
inc_port_ranges(integer[][]), inc_ports(integer[]), exc_port_ranges(integer[][]),
exc_ports(integer[]), start_time(timestamp with time zone),
end_time(timestamp with time zone), OUT count(bigint), OUT results(refcursor));

network [varchar(18)] // network (in CIDR notation) to match; a value of 'none' applies no filter
router [varchar(16)] // router name or IP to match; a value of 'none' applies no filter
inc_prot_ranges [integer[][]] // array of protocol ranges to match; a value of null applies no filter
inc_prots [integer[]] // array of specific protocols to match; a value of null applies no filter
exc_prot_ranges [integer[][]] // array of protocol ranges to filter out; a value of null applies no filter
exc_prots [integer[]] // array of specific protocols to filter out; a value of null applies no filter
inc_port_ranges [integer[][]] // array of port ranges to match; a value of null applies no filter
inc_ports [integer[]] // array of specific ports to match; a value of null applies no filter
exc_port_ranges [integer[][]] // array of port ranges to filter out; a value of null applies no filter
exc_ports [integer[]] // array of specific ports to filter out; a value of null applies no filter
start_time [timestamp with time zone] // time from which to search
end_time [timestamp with time zone] // time to end search

Output Parameters

The results of the *get_flows()* stored procedure are accessed via output parameters. The two output parameters for this function are:

count [bigint] // number of rows in result set
results [refcursor] // result set cursor

The results refcursor will point into a result set with the following columns:

id | *router* | *src_addr* | *dst_addr* | *start_time* | *end_time* | *tcp_flags* | *prot* | *tos* |
input_if | *output_if* | *src_as* | *dst_as* | *src_port* | *dst_port* | *src_mask* | *dst_mask* | *next_hop* | *pkts* | *bytes*

Note that these columns reflect the contents of the *flow* table. Please see init_db.sql file for details of these fields.

Invocation

Note that each cursor exists only as long as the current transaction. Therefore, the GUI must explicitly begin and end the transaction that encompasses this stored procedure call. Below is an example invocation of the *get_flows()* stored procedure that obtains the flows seen by the 68.34.49.254 exporter/router between '2008-09-06 00:00:00-04' and the current time. Note that it includes all protocols and ports with the exception of protocol 1 and ports 22 and 25. The fetch command retrieves the first 10 results.

```
begin;
select get_flows('none','68.34.49.254',null,null,null,array[1],null,null,null,array[22,25],
                '2008-09-06 00:00:00-4',now());
fetch 10 from "<unnamed portal 2>";
end;
```

get_traffic()

Parameters

The following function prototype illustrates the stored procedure name and sequence of parameters. Following the prototype is a detailed list of the input parameters.

```
get_traffic( network(varchar(18)), router(varchar(16)), direction(varchar(3)),
            inc_prot_ranges(integer[][]),  inc_prots(integer[]), exc_prot_ranges(integer[][]),
            exc_prots(integer[]), inc_port_ranges(integer[][]), inc_ports(integer[]),
            exc_port_ranges(integer[][]), exc_ports(integer[]), start_time(timestamp with time zone),
            end_time(timestamp with time zone), OUT count(bigint), OUT results(refcursor) );
```

```
network [varchar(18)]    // network (in CIDR notation) to match; a value of 'none' applies no filter
router [varchar(16)]    // router name or IP to match; a value of 'none' applies no filter
direction [varchar(3)]  // perspective or orientation of traffic; acceptable values are 'src' and 'dst'
inc_prot_ranges [integer[][]]   // array of protocol ranges to match; a value of null applies no filter
inc_prots [integer[]]    // array of specific protocols to match; a value of null applies no filter
exc_prot_ranges [integer[][]]   // array of protocol ranges to filter out; a value of null applies no filter
exc_prots [integer[]]    // array of specific protocols to filter out; a value of null applies no filter
inc_port_ranges [integer[][]]   // array of port ranges to match; a value of null applies no filter
inc_ports [integer[]]    // array of specific ports to match; a value of null applies no filter
exc_port_ranges [integer[][]]    // array of port ranges to filter out; a value of null applies no filter
exc_ports [integer[]]    // array of specific ports to filter out; a value of null applies no filter
start_time [timestamp with time zone]    // time from which to search
end_time [timestamp with time zone]    // time to end search
```

Output Parameters

The results of the *get_traffic()* stored procedure are accessed via output parameters. The two output parameters for this function are:

```
count [bigint]    // number of rows in result set
results [refcursor]    // result set cursor
```

The results refcursor will point into a result set with the following columns:

interval_start | router | addr | prot | port | pkts |bytes

interval_start // beginning timestamp of this data's interval
router // name or IP of exporting device
addr // IP address of host
prot // protocol number
port // port number
pkts // number of packets for this router/addr/prot/port over the interval
bytes // number of bytes for this router/addr/prot/port over the interval

Invocation

Note that each cursor exists only as long as the current transaction. Therefore, the GUI must explicitly begin and end the transaction that encompasses this stored procedure call. Below is an example invocation of the *get_traffic()* stored procedure that obtains the source traffic seen by the 68.34.49.254 exporter/router between '2008-08-05 00:00:00-04' and the current time. Note that it includes all protocols and ports with the exception of protocol 1 and ports 22 and 25. The fetch command retrieves the first 10 results.

```
begin;
select get_traffic('none','68.34.49.254','src',null,null,null,array[1],null,null,null,array[22,25],
                    '2008-08-05 00:00:00-4',now());
fetch 10 from "<unnamed portal 2>";
end;
```

get_sessions()

Parameters

The following function prototype illustrates the stored procedure name and sequence of parameters. Following the prototype is a detailed list of the input parameters.

```
get_sessions( network(varchar(18)), router(varchar(16)), inc_prot_ranges(integer[][]),
              inc_prots(integer[]), exc_prot_ranges(integer[][]), exc_prots(integer[]),
              inc_port_ranges(integer[][]), inc_ports(integer[]), exc_port_ranges(integer[][]),
              exc_ports(integer[]), min_ratio(real), max_ratio(real),
              start_time(timestamp with time zone), end_time(timestamp with time zone),
             OUT count(bigint), OUT results(refcursor) );
```

network [varchar(18)] // network (in CIDR notation) to match; a value of 'none' applies no filter
router [varchar(16)] // router name or IP to match; a value of 'none' applies no filter
inc_prot_ranges [integer[][]] // array of protocol ranges to match; a value of null applies no filter
inc_prots [integer[]] // array of specific protocols to match; a value of null applies no filter
exc_prot_ranges [integer[][]] // array of protocol ranges to filter out; a value of null applies no filter
exc_prots [integer[]] // array of specific protocols to filter out; a value of null applies no filter
inc_port_ranges [integer[][]] // array of port ranges to match; a value of null applies no filter
inc_ports [integer[]] // array of specific ports to match; a value of null applies no filter
exc_port_ranges [integer[][]] // array of port ranges to filter out; a value of null applies no filter
exc_ports [integer[]] // array of specific ports to filter out; a value of null applies no filter
min_ratio [real] // minimum (in_bytes / out_bytes) value to match
max_ratio [real] // maximum (in_bytes / out_bytes) value to match

start_time [timestamp with time zone] // time from which to search
end_time [timestamp with time zone] // time to end search

Output Parameters

The results of the *get_sessions()* stored procedure are accessed via output parameters. The two output parameters for this function are:

count [bigint] // number of rows in result set
results [refcursor] // result set cursor

The results refcursor will point into a result set with the following columns:

session_start | last_activity | router | src_addr | dst_addr | src_port | dst_port | prot | in_bytes | out_bytes | ratio

session_start // timestamp of the beginning of this session
last_activity // timestamp of the last activity seen for this session
router // name or IP of exporting device
src_addr // IP address of source
dst_addr // IP address of destination
src_port // source port number
dst_port // destination port number
prot // protocol number
in_bytes // bytes entering the source in this session
out_bytes // bytes leaving the source in this session
ratio // (in_bytes / out_bytes)

Invocation

Note that each cursor exists only as long as the current transaction. Therefore, the GUI must explicitly begin and end the transaction that encompasses this stored procedure call. Below is an example invocation of the *get_sessions()* stored procedure that obtains the relevant sessions seen by the 68.34.49.254 exporter/router between '2008-08-05 00:00:00-04' and the current time. Note that it includes all protocols and ports with the exception of protocol 1 and ports 22 and 25. Further, it only requests sessions with ratios between .5 and 2. The fetch command retrieves the first 10 results.

```
begin;
select get_sessions('none','68.34.49.254',null,null,null,array[1],null,null,null,array[22,25],.5, 2,
                    '2008-08-05 00:00:00-4',now());
fetch 10 from "<unnamed portal 2>";
end;
```

bw_by_port()

Parameters

The following function prototype illustrates the stored procedure name and sequence of parameters. Following the prototype is a detailed list of the input parameters.

```
bw_by_port( network(varchar(18)), router(varchar(16)), direction(varchar(3)),
            inc_prot_ranges(integer[][]), inc_prots(integer[]), exc_prot_ranges(integer[][]),
            exc_prots(integer[]), inc_port_ranges(integer[][]), inc_ports(integer[]),
            exc_port_ranges(integer[][]), exc_ports(integer[]), num_ports(integer),
            start_time(timestamp with time zone),  end_time(timestamp with time zone),
           OUT count(bigint), OUT results(refcursor) );
```

network [varchar(18)] // network (in CIDR notation) to match; a value of 'none' applies no filter
router [varchar(16)] // router name or IP to match; a value of 'none' applies no filter
direction [varchar(3)] // perspective or orientation of traffic; acceptable values are 'src' and 'dst'
inc_prot_ranges [integer[][]] // array of protocol ranges to match; a value of null applies no filter
inc_prots [integer[]] // array of specific protocols to match; a value of null applies no filter
exc_prot_ranges [integer[][]] // array of protocol ranges to filter out; a value of null applies no filter
exc_prots [integer[]] // array of specific protocols to filter out; a value of null applies no filter
inc_port_ranges [integer[][]] // array of port ranges to match; a value of null applies no filter
inc_ports [integer[]] // array of specific ports to match; a value of null applies no filter
exc_port_ranges [integer[][]] // array of port ranges to filter out; a value of null applies no filter
exc_ports [integer[]] // array of specific ports to filter out; a value of null applies no filter
num_ports [integer] // number of ports to obtain; note that providing number x will return the top x-1 port values
// and then one 'all other' value per interval
start_time [timestamp with time zone] // time from which to search
end_time [timestamp with time zone] // time to end search

Output Parameters

The results of the *bw_by_port()* stored procedure are accessed via output parameters. The two output parameters for this function are:

count [bigint] // number of rows in result set
results [refcursor] // result set cursor

The results refcursor will point into a result set with the following columns:

interval_start | prot | port | bw

interval_start // interval boundary (5 minute granularity)
prot // protocol number
port // port number
bw // bandwidth = sum(bytes)

Invocation

Note that each cursor exists only as long as the current transaction. Therefore, the GUI must explicitly begin and end the transaction that encompasses this stored procedure call. Below is an example invocation of the *bw_by_port()* stored procedure that obtains the bandwidth consumed (i.e., was the destination of the traffic) by the top 3 ports (per interval) over the last 7 days and

seen by the 208.74.181.139 exporter/router. Note that it includes all protocols and ports, but excludes protocol 1 and ports 80 and 443. The fetch command retrieves the first 10 results.

```
begin;
select * from bw_by_port('none', '208.74.181.139', 'dst', null, null, null, array[1], null, null, null, array[80,443], 3,
                         now(), '7 days');
fetch 10 from "<unnamed portal 2>";
end;
```

bw_ratio()

Parameters

The following function prototype illustrates the stored procedure name and sequence of parameters. Following the prototype is a detailed list of the input parameters.

```
bw_ratio( network(varchar(18)), router(varchar(16)), inc_prot_ranges(integer[][]),
          inc_prots(integer[]), exc_prot_ranges(integer[][]), exc_prots(integer[]),
          inc_port_ranges(integer[][]), inc_ports(integer[]),  exc_port_ranges(integer[][]),
          exc_ports(integer[]), num_buckets(integer), min_ratio(real), max_ratio(real),
          start_time(timestamp with time zone), end_time(timestamp with time zone),
          OUT count(bigint), OUT results(refcursor) );
```

```
network [varchar(18)]    // network (in CIDR notation) to match; a value of 'none' applies no filter
router [varchar(16)]    // router name or IP to match; a value of 'none' applies no filter
inc_prot_ranges [integer[][]]    // array of protocol ranges to match; a value of null applies no filter
inc_prots [integer[]]    // array of specific protocols to match; a value of null applies no filter
exc_prot_ranges [integer[][]]    // array of protocol ranges to filter out; a value of null applies no filter
exc_prots [integer[]]    // array of specific protocols to filter out; a value of null applies no filter
inc_port_ranges [integer[][]]    // array of port ranges to match; a value of null applies no filter
inc_ports [integer[]]    // array of specific ports to match; a value of null applies no filter
exc_port_ranges [integer[][]]    // array of port ranges to filter out; a value of null applies no filter
exc_ports [integer[]]    // array of specific ports to filter out; a value of null applies no filter
num_buckets [integer]    // number of data buckets for histogram; corresponds to graph width
min_ratio [real]    // minimum (in_bytes / out_bytes) value to match
max_ratio [real]    // maximum (in_bytes / out_bytes) value to match
start_time [timestamp with time zone]    // time from which to search
end_time [timestamp with time zone]    // time to end search
```

Output Parameters

The results of the *bw_ratio()* stored procedure are accessed via output parameters. The two output parameters for this function are:

```
count [bigint]    // number of rows in result set
results [refcursor]    // result set cursor
```

The results refcursor will point into a result set with the following columns:

```
index | ratio_count
```

```
index      // index of data bucket
ratio_count  // number of sessions whose (in_bytes/out_bytes) values fell into this bucket
```

Invocation

Note that each cursor exists only as long as the current transaction. Therefore, the GUI must explicitly begin and end the transaction that encompasses this stored procedure call. Below is an example invocation of the *bw_ratio()* stored procedure that obtains the histogram values for the relevant sessions seen by the 68.34.49.254 exporter/router between '2008-08-05 00:00:00-04' and the current time. Note that it includes all protocols and ports with the exception of protocol 1 and ports 22 and 25. Further, it specifies a graph width (i.e., num_buckets) value of 256. The fetch command retrieves the first 10 results.

```
begin;
select bw_ratio('none','68.34.49.254',null,null,null,array[1],null,null,null,array[22,25],256,
                    '2008-08-05 00:00:00-4',now());
fetch 10 from "<unnamed portal 2>";
end;
```