YETI SBC Manual

January 10, 2014

Contents

1 Introduction

1.1 What is YETI SBC ?

YETI SBC is project based on sbc module of SEMS project. The main goal is to provide ability to control routing logic and entire softwitch behaviour from database. This approach provides flexibility, ability to quickly add a non-trivial features and simple integraton with other systems. It is possible to completely change the business logic without having to change any code except of database stored procedures.

Project consist of few separate parts:

- **Core**
  modified SEMS and our module binaries
- **Database**
  set of tables and appropriate PL/pgSQL stored procedures. As a database system we have chosen PostgreSQL.
- **Web**
  web-interface written in Active Admin

All parts may be located at the same physical server or on separate servers to increase performance.

We recommends using of Debian GNU/Linux 7 as operating system for all software parts but you can use any Linux which satisfy to the following conditions:

- Core:
- Database:
- Web:

YETI SBC is distributed under GPL license and we can provide source code of any part on demand

1.2 Features

Basic set of features is based on current SEMS implementation so it would be reasonable to first familiarize yourself with documentation for SEMS SBC. We often commit merges with master branch of SEMS.

YETI SBC is optimized for routing of transit traffic.

- Very detailed CDRs, possibility to configure custom variables from database
- Well thought out web-interface
- Useful xmlrpc interface for control and monitoring
- Current database part implementation supports LCR/ASR/ACD routing. And we are still adding new features to control routing (trying to do this without sacrificing performance)

2 Quick start

2.1 Requirements

2.2 Installation

2.2.1 Database

2.2.2 Web-interface

2.2.3 Core

2.3 Configuration

2.3.1 Database

2.3.2 Web-interface

see section 7.2

2.3.3 Core

3 Overview

3.1 Common architecture

3.2 Interaction of the parts

4 Configuration

4.1 Database

4.2 Web-interface

4.3 Core

Currently all parameters specific to YETI module are presented in yeti.conf (default location: /etc/sems/etc/yeti.conf)
Keep in mind that there are many parameters which loaded from database during application startup

4.3.1 Common

node_id (integer,id) each application instance has it own node_id to distinguish from each other in the database.
default: mandatory parameter

pop_id (integer,id) point of presence id. used to group nodes with certain propeties (eg. geographical location).
default: mandatory parameter

msg_logger_dir (string,path) path to the directory where will be written calls pcap dumps
default: mandatory parameter

calls_show_limit (integer) upper limit for getCalls [getCalls] xmlrpc request

4.3.2 Registrations

reg_check_interval(integer, milliseconds) SIP registration client works as separate thread. List of required registrations is loaded from database. This parameter determines interval between checks for configured registration entries state in milliseconds.
default: 5000

4.3.3 Routing database

db_schema(string) database scheme. used during routing and loading of configuration of resources control and codes translations
default: mandatory parameter

getprofile_function(string) SQL query to get callprofiles defined as db_schema.get_profile_function(...)
default: mandatory parameter

master_host, port,user,name,pass parameters is the same as libpq connection string params
default: mandatory parameters

master_pool_size(integer) database connections pool size
default: 10

master_check_interval(integer,seconds) interval between database connection checks in case when no active load.
default: 25

master_max_exceptions(integer) the number of non-critical connection exceptions after which consider it a bad.
zero means infinity
default: 0

master_max_wait(integer,milliseconds) timeout of connection obtaining from connections pool in milliseconds
default: 125

failover_to_slave(boolean) enable using slave database for failover
default: 0

slave_host,port,user,name,pass master_host, port,user,name,pass

slave_pool_size(integer) master_pool_size(integer)

slave_check_interval(integer,milliseconds) master_check_interval(integer,seconds)

slave_max_exceptions(integer) master_max_exceptions(integer)

slave_max_wait(integer,milliseconds) master_max_wait(integer,milliseconds)

4.3.4 Profiles cache

profiles_cache_enabled(boolean) enable using of callprofiles local cache
default: 0

profiles_cache_check_interval(integer,seconds) interval between checks for obsolete profiles in seconds (also check is made before returning of found entry)
default: 30

profiles_cache_buckets(integer) number of buckets in callprofiles local cache hash
default: 65000

4.3.5 CDRs

cdr_check_interval(integer,milliseconds) interval between database connection checks in milliseconds
default: 5000

mastercdr_host,port,name,user,pass master_host, port,user,name,pass

slavecdr_host,port,name,user,pass master_host, port,user,name,pass

writecdr_schema(string) schema used during CDR write
default: mandatory

writecdr_function(string) SQL query is writecdr_schema.writecdr_function(...)
default: mandatory

failover_to_file(boolean) enable using slave database for failover
default: 1

cdr_dir(string) full path for temporary csv files with CDRs
default: mandatory if failover_to_file

cdr_completed_dir(string) full path for completed csv files with CDRs
default: mandatory if failover_to_file

4.3.6 Resource control

write_redis_host,port redis connection settings
default: 127.0.0.1:6739

write_redis_size(integer) redis pool size
default: mandatory

write_redis_timeout(integer,seconds) TTL for resource entry in seconds
default: mandatory

read_redis_host,port redis connection settings
default: 127.0.0.1:6739

read_redis_size(integer) redis pool size
default: mandatory

read_redis_timeout(integer,seconds) TTL for resource entry in seconds
default: mandatory

reject_on_cache_error(boolean) pretend that resource check is sucessful on checking internal errors
default: 0

5 Core

YETI Core is essentially a module which is loaded with SBC module which is loaded with SEMS.

5.1 Routing & Call params

The central notion of calls processing is CallProfile. This is a structure that defines the parameters and behaviors for each call.
For the full list of possible parameters please refer to the documentation of SEMS SBC and source code of SEMS SBC,YETI SBC.
Specific for YETI SBC CallProfile parameters:

disconnect_code_id (integer) used instead of **refuse_with** parameter. means internal disconnect code which is described in database and resolved by Translations [5.3] class

cache_time (integer) determines whether to cache the database response and his TTL [5.5]

time_limit (integer) determines maximum call duration

resources (string) resources list for call. [5.2]

Also we changed all string constants (eg. FilterType, RefreshMethod) to appropriate id values. This values partly described in database and partly hardcoded.
Actions which being performed on initial-INVITE:

- retreive appropriate set of CallProfiles (from ProfliesCache [5.5] or from database [6.1.3])
- check resources availability and try to grab them (if resources are busy iterate over available CallProfiles till reached profile with available resources)
- try to establish connection with terminator (perform rerouting in case when we have non-succ response and it stop_hunting [5.3] flag equal to false in early-state dialog. write CDR for each rerouting attempt)

All the dirty work about routing process is performed by database stored procedures. We plan create middleware software which will be responsible for routing.

5.2 Resources

Resource is abstract concept. It is something that is required to make a call (eg. channels, simultaneous calls, etc)

Each resource has following properties:

id unique id within type space

type values which determines behavior when resource is busy (6.1.9)

takes amount of abstract units needed for one call

limit upper value limit for resource

We use Redis DSS as storage for current resources state. One Redis server can be shared between many YETI SBC instances thus we can provide for example control of the max number of calls of the logical client even if he has the calls over different nodes.

To ensure the effective use of Redis replication we are working with two separated logical Redis servers (it may be one physical server). One of them is used only for read-only queries (checking of resources availability) other server is used for resources grabbing and releasing. Of course, if this is separate servers, between them must be configured replication.

Each resource has it own key on DSS. Currently this is combination of id and type separated with symbol ':'. You can check the status of resources with your own utilites or software directly on Redis server.

Each returned from database CallProfile has **resources** string field. It's formatted list of resources needed for this call.

Resources string has following format:

`r1_type:r1_id:r1_limit:r1_takes;r2_type:r2_id:r2_limit:r2_takes[;...]`

ie: ';' is separator for resources and ':' is separator for fields within one resource

For details about checking, grabbing and releasing implementation refer to source code.

5.3 Translations

Translations class serves the following purposes:

- Decide whether continue rerouting on certain response code [6.1.5]
- Resolve internal codes into corresponding combinations of code and reason [6.1.7]
- Rewrite terminator response codes [6.1.6]

5.4 Registrations

Sometimes we need to be able to perform SIP registration on foreign servers.

YETI SBC Registrations class uses SEMS **registrar_client** module functionality. It simply obtains list of required registrations from database [6.1.8], transmits them to the registrar_client for handling and periodically checks their state.

5.5 ProfilesCache

Implemented to improve performance and to reduce the load on database.

Just memorises responses for getprofile [6.1.3] query from database into hash

Can be fully disabled from local configuration file. When enabled can be controlled with database [5.1]

5.6 CDRs

CDRs is the key for billing. To ensure reliability SEMS SBC supports failover with slave CDR database and (in worst cases) CSV files.

You can upload CDRs from CSV file into desired database using script yeti-csv2sql which is provided with core package.

CDRs are written asyncronously by multiple threads, each of which has its own connection to the primary and (on enabled failover) to the slave database.

For details about CDRs fields, refer to 6.1.4

5.7 XmlRPC interface

YETI module has its own xmlrpc interface which allows to control and monitor it. All responses formatted in JSON.

5.7.1 showVersion

build git commit hash of current binary

compiled_at build date

compiled_by build user

5.7.2 getConfig

Shows current settings

calls_show_limit (integer) upper limit for getCalls [5.7.8] request

node_id (integer) current node id [4.3.1]

pop_id (integer) curent pop id [4.3.1]

resources_control (node)

- **cache (node)** redis cache parameters
  - **read_pool (node)** readonly Redis server parameters
    - **pool_size (integer)** number of simultaneous connections
    - **server (string)** server connect string
  - **write_pool (node)** master Redis server parameters
    - **pool_size (integer)** number of simultaneous connections
    - **server (string)** server connect string
- **db_config (string)** database
- **db_schema (string)** database schema

router (node) routing engine parameters

cache_buckets (integer) ProfilesCache [5.5] buckets

cache_check_interval (double) ProfilesCache [5.5] grabage collector check interval

cache_enabled (boolean) ProfilesCache [5.5] enabled

config_db (string) database connection string which used to obtain configuration

dyn_fields (string list) current dynamic fields [6.1.1] list

failover_to_slave (boolean) use slave database for failover on getprofile

getprofile_call (string) sql query used in getprofile procedure

sipreq_header_fields (string list) considered in getprofile SIP headers list [6.1.2]

writecdr_call (string) sql query used in writecdr procedure

master_pool (node) master database pool parameters

- **check_interval (integer)** connection status checks interval
- **db (string)** used database connection string
- **max_exceptions (integer)** max non-fatal exceptions count before consider connection as bad
- **max_wait (integer)** timeout for connection obtaining from connections pool
- **size (integer)** pool size

slave_pool (node) slave database pool parameters

- **check_interval (integer)** connection status checks interval
- **db (string)** used database connection string
- **max_exceptions (integer)** max non-fatal exceptions count before consider connection as bad

max_wait (integer) timeout for connection obtaining from connections pool

size (integer) pool size

cdrwriter (node) cdrwriter parameters

failover_to_file (boolean) failover to file enabled

failover_file_completed_dir (string) directory for completed CSV files

failover_file_dir (string) directory for temporary CSV files

failover_to_slave (boolean) failover to slave database enabled

master_db (string) master database connection string

slave_db (string) master database connection string

query_args (string list) getprofile query function arguments list (very useful for debug purposes)

translator (node) CodesTranslation [5.3] parameters

config_db (string) used database connection string

db_schema (string) used database schema

hunting (node list) contain pairs of (node with name equal to code, parameters related to this code)

is_stop_hunting (boolean) continue hunting [6.1.5]

internal_translations (node list) contain pairs of (node with name equal to code, parameters related to this code)

internal code (integer), internal_reason (string), response_code (integer), response_reason (string) [6.1.7]

response_translations (node list) contain pairs of (node with name equal to code, parameters related to this code)

pass_reason_to_originator (boolean), rewrite_code (integer), rewrite_reason (string) [6.1.6]

5.7.3 getStats

Shows runtime stats

active_routers_count (integer) router instances count

calls_show_limit (integer) upper limit for getCalls [5.7.8] request

localtime (integer) YETI SBC node localtime timestamp

uptime (double) uptime of node in seconds

AmSession (node) AmSession stats

AvgCPS (integer), MaxCPS (integer) calls per second counters

AvgSessionNum (integer), MaxSessionNum (integer) SessionNum (integer) sessions counters (keep in mind that the number of calls equal half of the sessions count)

AmSessionContainer (node) AmSessionContainer stats

dead_sessions_count (integer) sessions which are await for cleanup

unclean_shutdown_enabled (boolean) determines whether send shutdown event for each session on SEMS shutdown

routers (node list) always contain one **'active'** node (router instance which will be used for all new calls) and optionally set of **'old'** nodes (obsolete router instances which still referenced by uncompleted calls). all of this nodes has similar parameters within.

uptime (double) uptime in second of this router instance

refs (integer) how many legs refer to this router instance

hits (integer) overall initial INVITE requests count

cache_hits (integer) ProfilesCache matched requests count

db_hits (integer) queries to database count

gps_avg (double), gps_max (double) queries per second (close to calls per second)

gt_max (double), gt min (double) how long takes one CallProflies set obtaining (both from database and cache)

cdr_writer (node) CdrWriter stats

name (string) class instance name

poolsize (integer) database connection pool size

threads (array list) list of arrays each of which mean one thread with it own stats

db_exceptions (integer) database exceptions count

queue_len (integer) how many CDRs currently in queue of this thread (if it grows something must be changed. most likely, database doesn't have time to handle requests)

queue_run (boolean) queue processing flag

stopped (boolean) thread scheduled to stop

tried_cdrs (integer) how many CDRs was passed to this thread

writed_cdrs (integer) how many CDRs was successfully written (must be equal or near of **tried_cdrs)**

master_pool (node), slave_pool (node) database connection pools stats

check_transactions (integer) number of connections checks

transactions (integer) number of completed transactions

failed_connections (integer) number of failed connections

total_connections (integer) how many connections in pool

tps_avg (double), tps_max (double) transactions per second for entire pool

tt_max (double), tt_min (double) transaction time

connections (array list) contain per connection stats

exceptions (integer) number of exceptions in this database connection

proflies_cache (node) ProfilesCache stats

entries (integer) current entries number in cache

resource_control (node) resource control stats

hits (integer) overall checks number

errors (integer) errors number

nextroute (integer), overloaded (integer), rejected (integer) number of responses for each type

translator (node) CodesTranslator stats

missed_response_configs (integer) count of unconfigured reactions for rerouting

unknown code_resolves (integer) count of unresolved responses translations

unknown_internal_codes (integer) count of unresolved internal codes translations

5.7.4 clearStats

Clears runtime statistics

5.7.5 clearCache

Clears ProfliesCache immediatly

5.7.6 dropCall

Allows to drop certain active call using him local_tag (of course, mandatory argument is local_tag)

5.7.7 getCall

Get information about active call by local_tag (local_tag is mandatory argument)
Returning values in common equal to CDR fields [6.1.4] except of:

local_time (double) YETI SBC localtime timestamp (the fractional part is responsible for a milliseconds)

5.7.8 getCalls

Returns set of active calls (count of entries is limited [calls_show_limit (integer)])
Each element of array equal to one getCall [getCall] response

5.7.9 getCallsCount

Returns count of active calls

5.7.10 getRegistrations

Returns list of configured registrations
For fields meaning refer to Registrations

5.7.11 reload

Without any parameters shows all available reload actions:

resources reload action for [Resources]. reload resources configs (no reconnect to Redis servers)

translations reload action [Translations]. reload entire translations configuration from database

registrations reload action for [Registrations]. remove current registrations from registart_client and load new list from database.

router intented for zero downtime database configuration upgrade. creates new router instance and configure it to work with new database or/and database scheme. Old router instances will continue processing of already established calls and will be gracefully turned off when this calls completed. (count of references to old router can be obtained with getStats)

5.7.12 closeCdrFiles

Close opened CSV files immediatly

6 Database

6.1 mandatory interface functions

6.1.1 load_interface_out

We have posibility to pass directly arbitrary custom parameters specific to calls from routing database into CDR. They will be obtained from **getprofile** query response and passed at the end of parameters list of **cdrwrite** query in the order of they appearance in **load_interface_out** response.

query:

```
SELECT * FROM switch.load_interface_out()
```

response fields:

varname (varchar) custom variable name

vartype (varchar) custom vaiable type (see PostgreSQL datatypes)

forcdr (boolean) determines should we use this field or not (only fields with forcdr=true will be processed)

6.1.2 load_interface_in

There are tasks when we need to take into account for routing and billing some of the SIP headers from INVITE request. For this purpose we have list of custom SIP request headers which will be passed to the getprofile function after static parameters in the order of they appearance in load_interface_in response.

query:

```
SELECT * FROM switch.load_interface_in()
```

response fields:

varname (varchar) custom variable name

vartype (varchar) custom vaiable type (see PostgreSQL datatypes)

6.1.3 getprofile

Function that is responsible, in fact, for routing. It takes static parameters (which are described below) and dynamic after them (see load_interface_in).

Function returns set of callprofiles [5.1] (one callprofile on the row)

query:

```
SELECT * FROM switch.getprofile_f(i_node_id integer, i_pop_id integer, i_remote_ip inet,
    i_remote_port integer, i_local_ip inet, i_local_port integer, i_from_dsp character
    varying, i_from_name character varying, i_from_domain character varying, i_from_port
    integer, i_to_name character varying, i_to_domain character varying, i_to_port integer,
    i_contact_name character varying, i_contact_domain character varying, i_contact_port
    integer, i_user character varying, ...)
```

request parameters:

i_node_id (integer), i_pop_id (integer) see node_id (integer,id), pop_id (integer,id),

i_remote_ip (inet), i_remote_port (integer) originator ip:port

i_local_ip (inet), i_local_port (integer) switch ip:port

i_from_dsp (varchar) display name (rfc3261 8.1.1.3)

i_from_name (varchar), i_from_domain (varchar), i_from_port (integer) parsed From field parts

i_to_name (varchar), i_to_domain (varchar), i_to_port (integer) parsed To field parts

i_contact_name (varchar), i_contact_domain (varchar), i_contact_port (integer) parsed Contact field parts

i_user (varchar) user part of RURI field

response:

set of CallProfiles [5.1]

6.1.4 cdrwrite

Is called in order to write final CDR. As well as for getprofile has static input parameters and customizable after them (see load_interface_out)

query:

```
SELECT * FROM switch.writecdr(i_node_id integer, i_pop_id integer, i_routing_attempt integer
    , i_is_last_cdr boolean, i_time_limit integer, i_lega_local_ip character varying,
    i_lega_local_port integer, i_lega_remote_ip character varying, i_lega_remote_port integer
    , i_legb_local_ip character varying, i_legb_local_port integer, i_legb_remote_ip
    character varying, i_legb_remote_port integer, i_time_start bigint, i_time_connect bigint
    , i_time_end bigint, i_disconnect_code integer, i_disconnect_reason character varying,
    i_disconnect_initiator integer, i_lega_disconnect_code integer, i_lega_disconnect_reason
    character varying, i_orig_call_id character varying, i_term_call_id character varying,
    i_local_tag character varying, i_msg_logger_path character varying, i_log_rtp boolean,
    i_log_sip boolean, i_lega_rx_payloads character varying, i_lega_tx_payloads character
    varying, i_legb_rx_payloads character varying, i_legb_tx_payloads character varying, ...)
```

request parameters:

i_node_id (integer), i_pop_id (integer) see node_id (integer,id), pop_id (integer,id),

i_routing_attempt (integer) determines the routing attempt number of this CDR within call

i_is_last_cdr (boolean) added when it was implemented the rerouting (serial-fork) feature. We write separate CDR for each rerouting attempt, therefore a good idea to know whether it is the last attempt for this call. Means final call result if flag equal to true

i_time_limit (integer) call duration limit in seconds

i_lega_local_ip (varchar), i_lega_local_port (integer) local ip:port which were used to communicate with originator

i_lega_remote_ip(varchar), i_lega_remote_port (integer) originator ip:port

i_legb_local_ip(varchar), i_legb_local_port (integer) local ip:port which were used to communicate with terminator

i_legb_remote_ip (varchar), i_legb_remote_port (integer) terminator ip:port

i_time_start (bigint) initial INVITE request timestamp of arrival. (real call duration is **time_end-time_connect)**

i_time_connect (bigint) connect time timestamp

i_time_end (bigint) termination time timestamp

i_disconnect_code (integer), i_disconnect_reason (varchar) depending on **disconnect_initiator** means original (non rewrited) terminator code/response or internal code/response

i_disconnect_initiator (integer) disconnect initiator id. current possible values:

- **0 (by database)** database returned particular type of exception or refused the request due to some reasons
- **1 (by traffic switch)** internal switch error, mailformed request, resources validation failed, etc
- **2 (by destination)** terminator request/response was the cause of call termination
- **3 (by originator)** originator request/response was the cause of call termination

i_lega_disconnect_code (integer), i_lega_disconnect_reason (varhchar) code and reason which was passed to originator

i_orig_call_id (varchar) originator callid

i_term_call_id (varchar) terminator callid

i_local_tag (varchar) internal call local_tag

i_msg_logger_path(varchar) path to pcap file with call dump

i_log_rtp (boolean), i_log_sip (boolean) logging parameters was used for pcap file

i_lega_rx_payloads (varchar), i_lega_tx_payloads (varchar), i_legb_rx_payloads (varchar), i_legb_tx_payloads each variable contains comma-separated list of codecs which was used during call (we collect this directly from each RTP packet payload type)

6.1.5 load_disconnect_code_rerouting

Determines YETI SBC reaction for different terminator response codes.

query:

```
SELECT * from switch.load_disconnect_code_rerouting()
```

response fields:

received_code (integer) terminator response code

stop_rerouting (boolean) stop rerouting if true

6.1.6 load_disconnect_code_rewrite

We able to rewrite terminator and switch response codes before they will be sent to the originator.

query:

```
SELECT * FROM switch.load_disconnect_code_rewrite()
```

response fields:

o_code (integer) response code

o_reason (varchar) response code description (not used in processing)

o_pass_reason_to_originator (boolean) pass the original response reason to originator

o_rewrited_code (integer) code which will be passed to originator (be careful, you risk to break the correct SIP sequence here)

o_rewrited_reason (varchar) reason which will pe passed to originator. if empty, then will be transferred original response reason

6.1.7 load_disconnect_code_refuse

Responsible for resolving of internal error codes into response code and response reason separately for CDR and response to originator .

Currently we have such hardcoded internal error codes:

```
#define FC_PARSE_FROM_FAILED          114
#define FC_PARSE_TO_FAILED            115
#define FC_PARSE_CONTACT_FAILED       116
#define FC_NOT_PREPARED               117
#define FC_DB_EMPTY_RESPONSE          118
#define FC_READ_FROM_TUPLE_FAILED     119
#define FC_EVALUATION_FAILED          120
#define FC_GET_ACTIVE_CONNECTION      121
#define FC_DB_BROKEN_EXCEPTION        122
#define FC_DB_CONVERSION_EXCEPTION    123
#define FC_DB_BASE_EXCEPTION          124
#define DC_RTP_TIMEOUT                125
```

query:

```
SELECT * FROM  switch.load_disconnect_code_refuse()
```

response fields:

o_id (integer) internal code

o_code (integer) code which will be written into CDR

o_reason (varchar) reason which will be written into CDR

o_rewrited_code (integer) code which will be passed to originator. (use **o_code** if empty)

o_rewrited_reason (varchar) reason which will be passed to originator. (use **o_reason** if empty)

6.1.8 load_registrations_out

Provide desired registrations list

query:

```
SELECT * FROM switch.load_registrations_out(IN i_pop_id integer, IN i_node_id integer)
```

request parameters:

i_node_id (integer) see node_id (integer,id)

i_pop_id (integer) see pop_id (integer,id)

response fields:

o_id (integer) internal id

o_domain (varchar)

o_user (varchar)

o_display_name (varchar)

o_auth_user (varchar)

o_passwd (varchar)

o_proxy (varchar)

o_contact (varchar)

To understand fields meanings is useful to know about SEMS registrar client. Anyway existing web-interface allows set it intuitively.

6.1.9 load_resource_types

Provide for each Resource (5.2) type

query:

```
SELECT * FROM switch.load_resource_types()
```

response fields:

name (string) resource name (used for clarity in reports)

reject_code (integer), reject_reason (string) code will be passed to originator

action_id behavior on resource overload. possible values:

- **1 (reject)** reject call with specified code and reason
- **2 (next route)** choose next CallProfile (5.1) (equal to reject if there are no routes more)
- **3 (accept)** just ignore overload of resource. pretend that all right.

6.2 event-based changes propagation

There was a problem automatically applying the necessary changes for the YETI SBC nodes after the configuration changes in the database without node restart.

To solve it we implemented set of xmlrpc commands which force to reload configuration for each subsystem of node. Then we use simple table in database for events of changes which need to be processed.

There is small script written on Python [yeti-process-events] which loads events from that table and sends appropriate xmlrpc commands to the desired nodes.

Script gets list of nodes and database connection settings from local configuration file.

When it starts, it receives events from the event table:

```
SELECT id, command, retries FROM switch.events FOR UPDATE NOWAIT
```

for each event after successfull xmlrpc request we do something like this:

```
DELETE FROM switch.events WHERE id={event_id}
```

and otherwise:

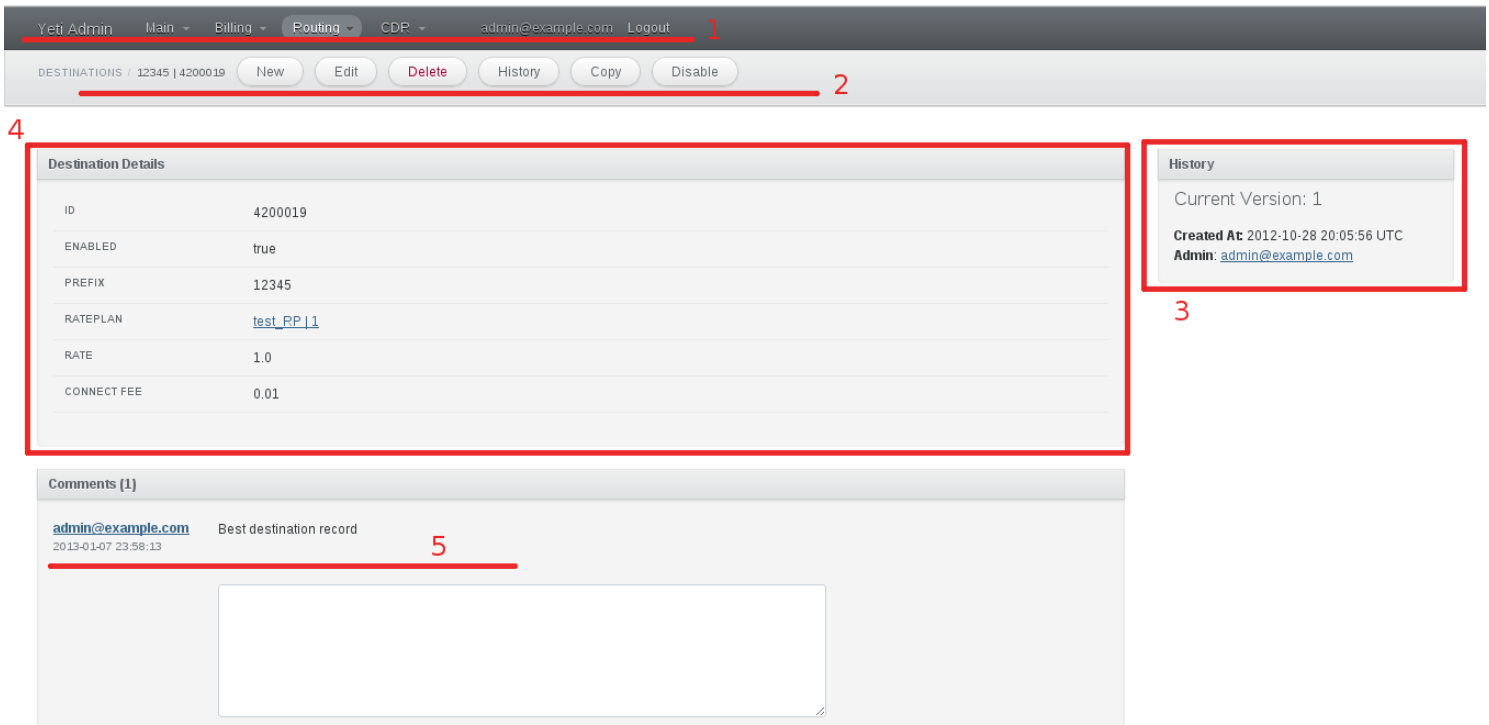
```
UPDATE switch.events SET retries = retries+1 WHERE id={event_id}
```

If you want to take advantage of this decision it would be a good idea to add this script into any external scheduler like cron

7 Web

Web interface written using Active Admin technology. The basic objects of interface is tables and records. In common case each table has at least several actions as creation, removal, review of records. There are two basic modes: a table and specific record view.

7.1 What you see on the screen



1. Navigation panel
2. Current record control panel
   - New,Edit,Delete - speaks for itself
   - History - changes history of current record. You can see the previous versions of the record and infomation about who is editing them
   - Copy - create new record using data from current record

- Disable - toggle 'Enable' flag

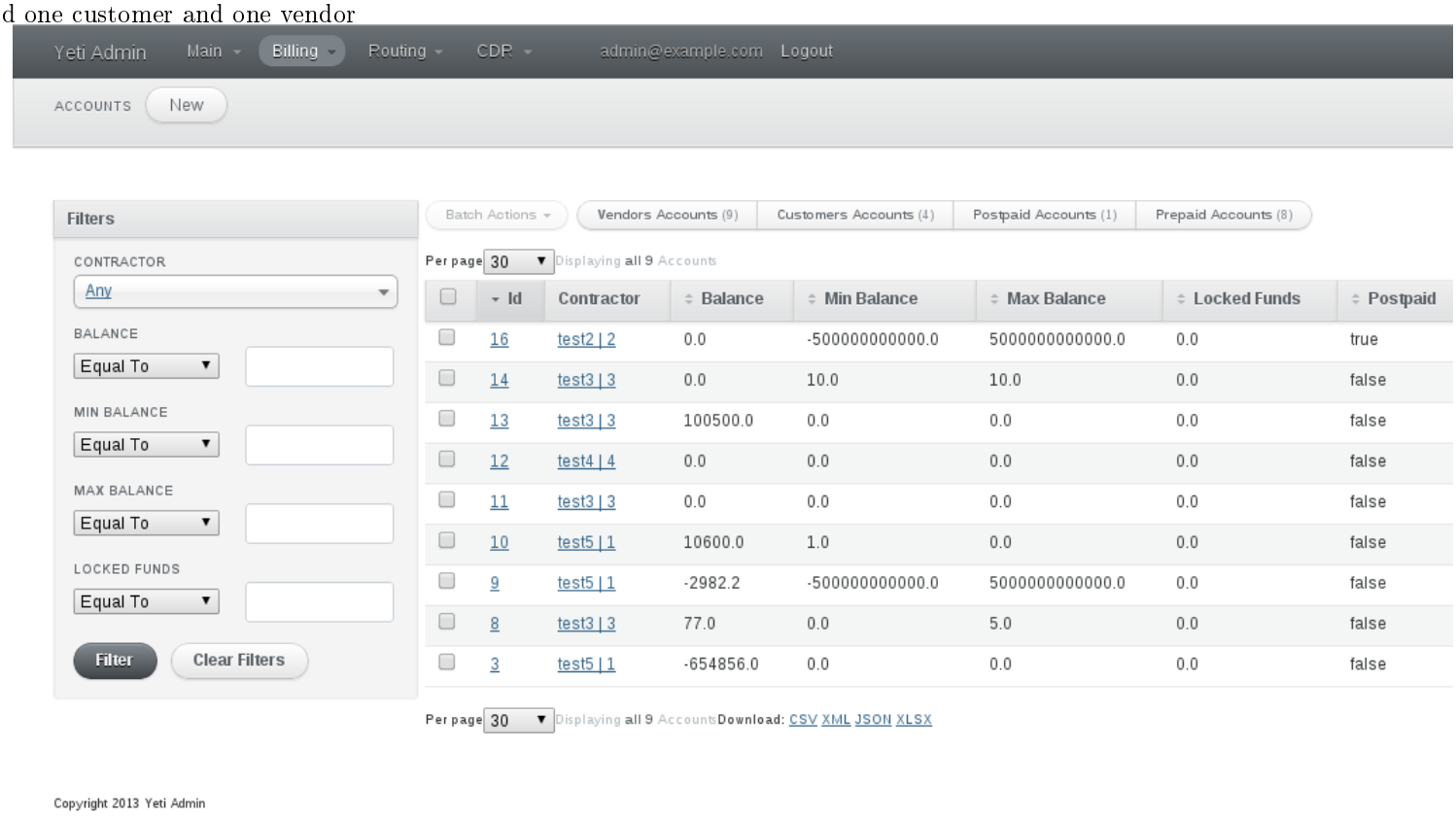
3. Inromation about current record
4. Detailed information about current record
5. Comments. Each record can have own comments except of some read-only data (eg. CDR or Active Calls)

7.2 Quick start

step by step quick configuration

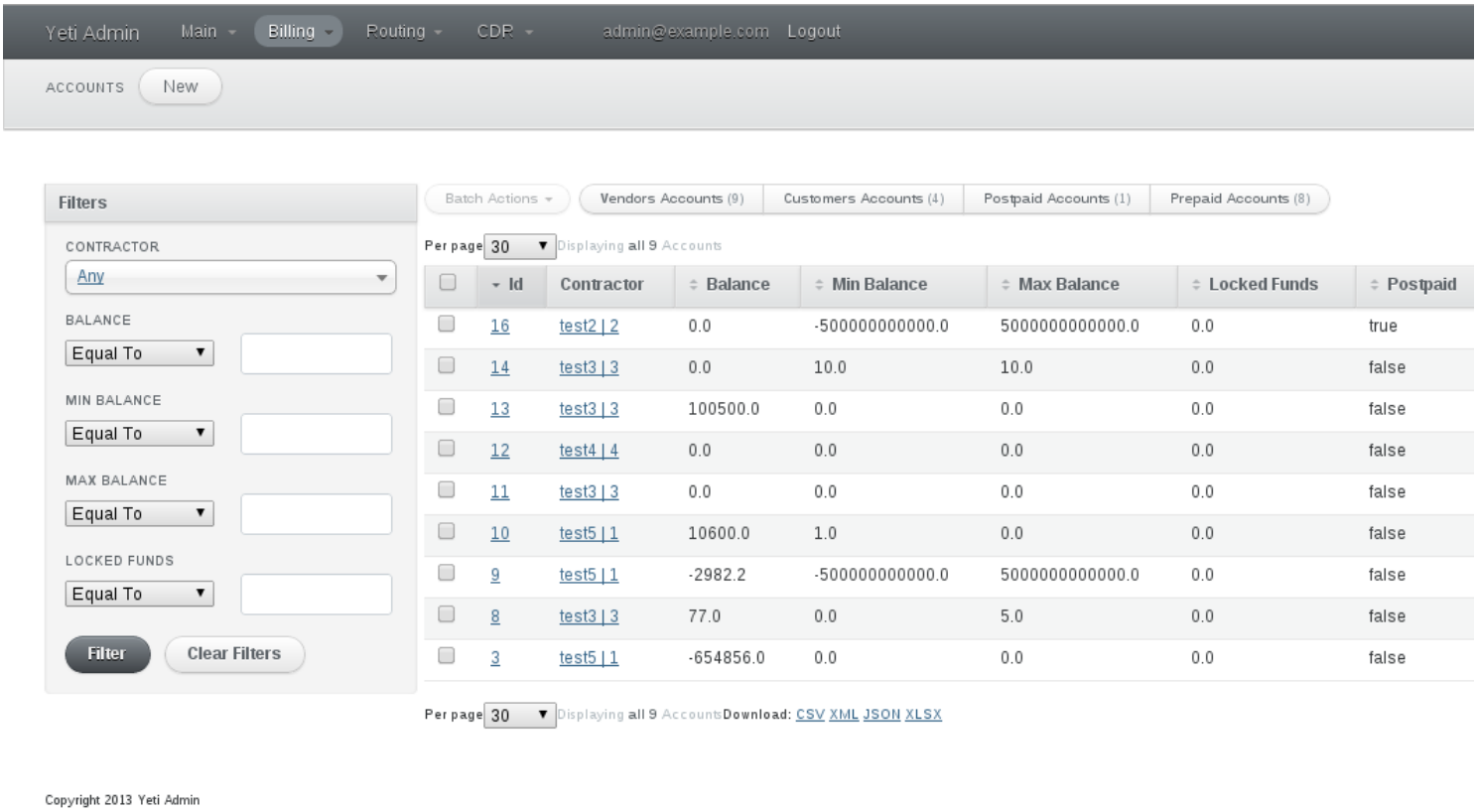
7.2.1 Main->Contractors

add one customer and one vendor



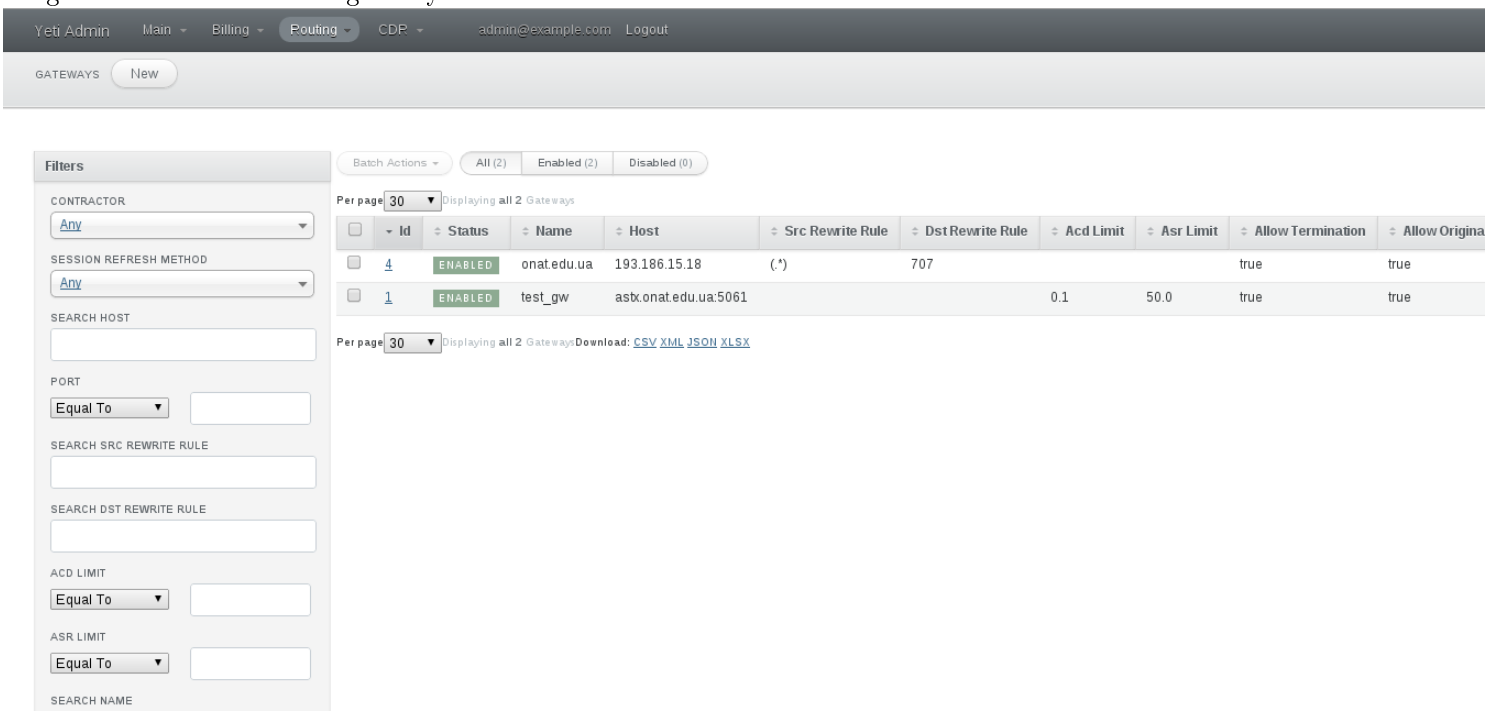
7.2.2 Billing->Accounts

add account for each contractor



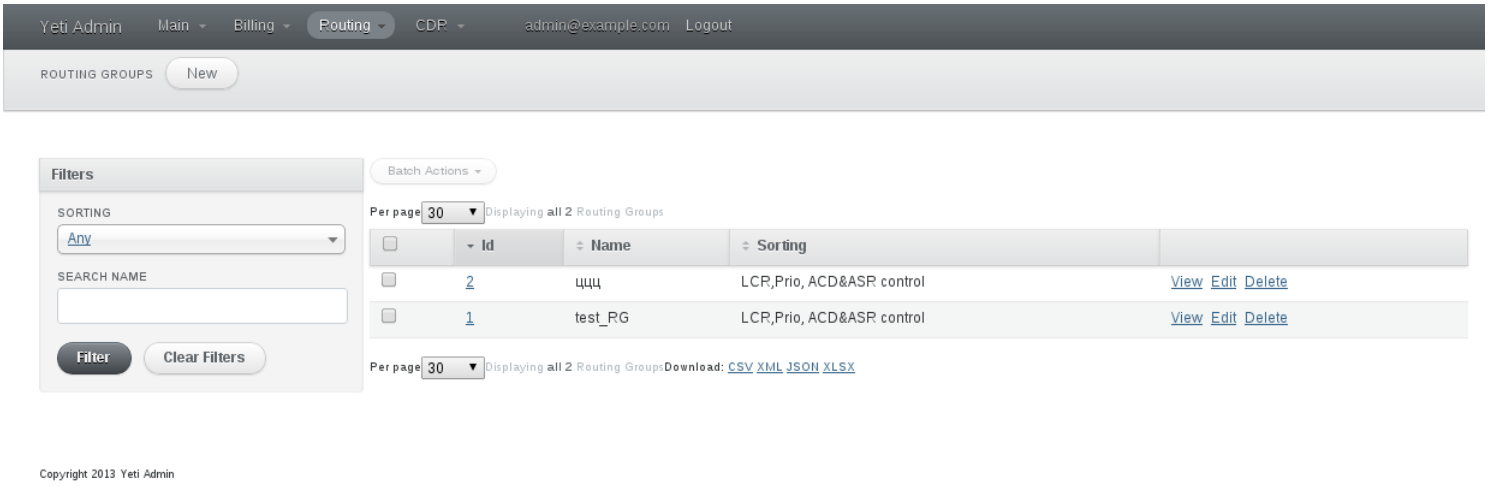
7.2.3 Routing->Gateways

add origination and termination gateways



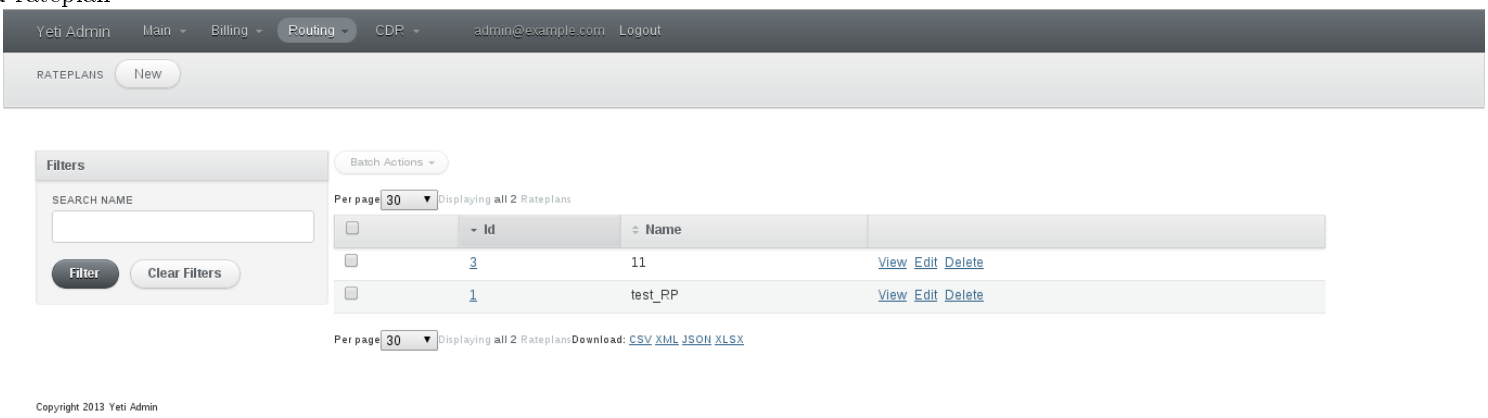
7.2.4 Routing->Routing groups

add routegroup



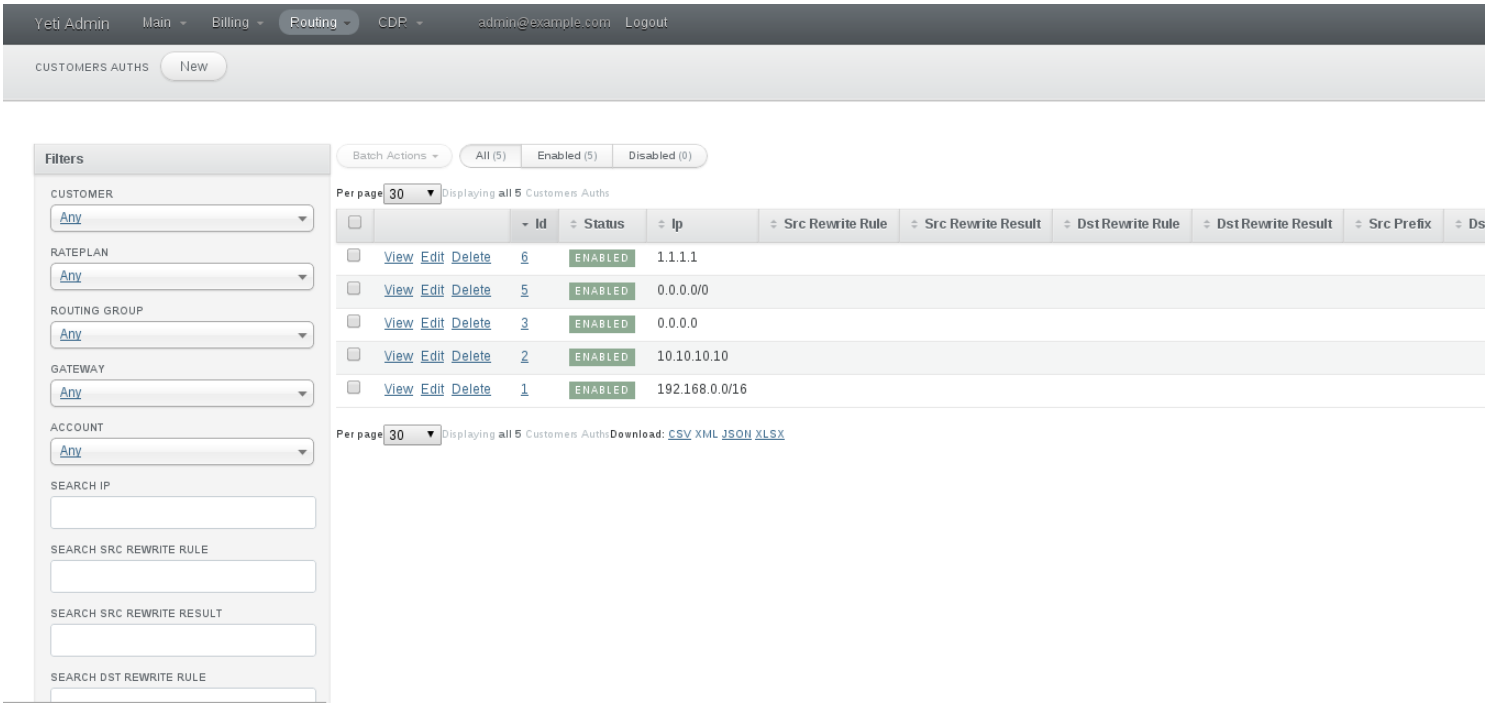
7.2.5 Routing->Rateplans

add rateplan



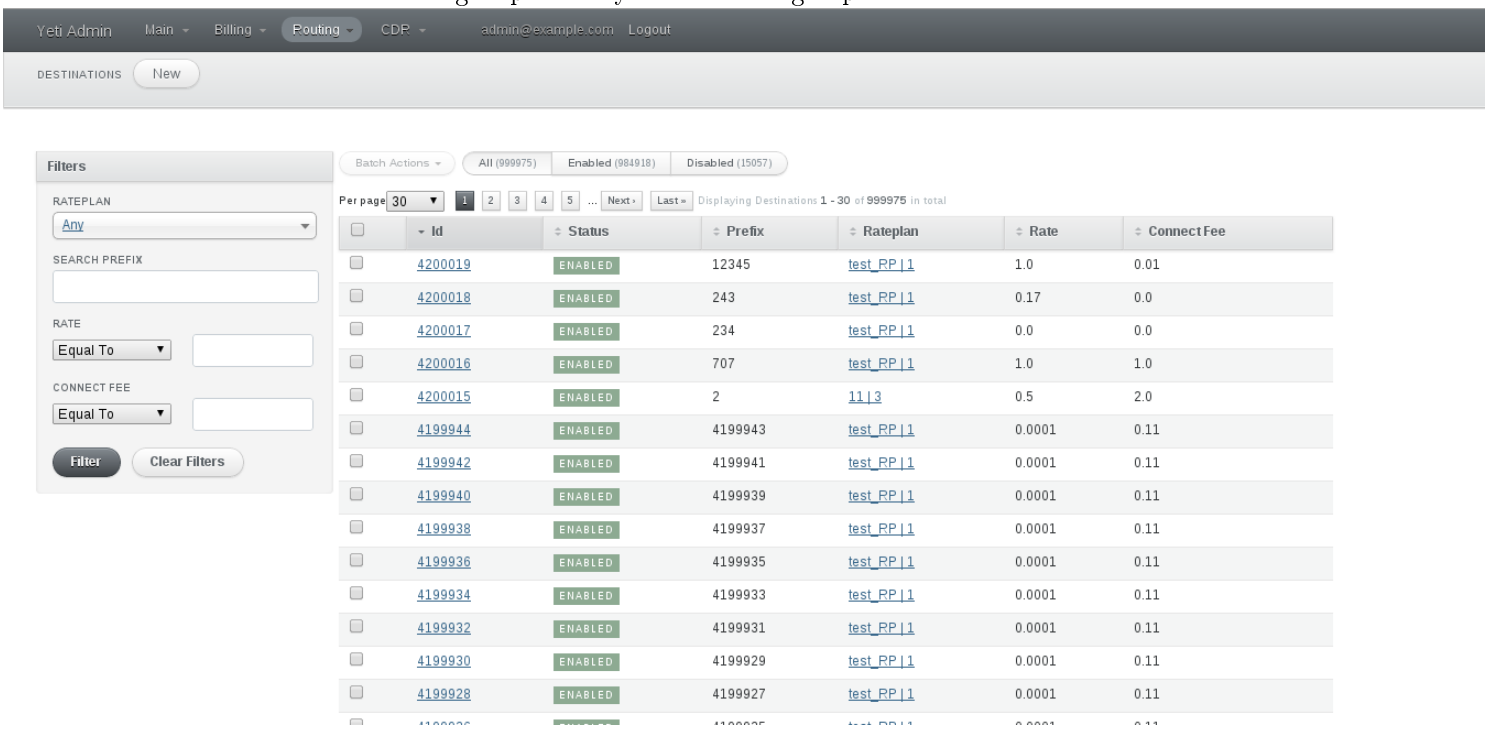
7.2.6 Routing->Customer auth

add authorization record (customer auth)



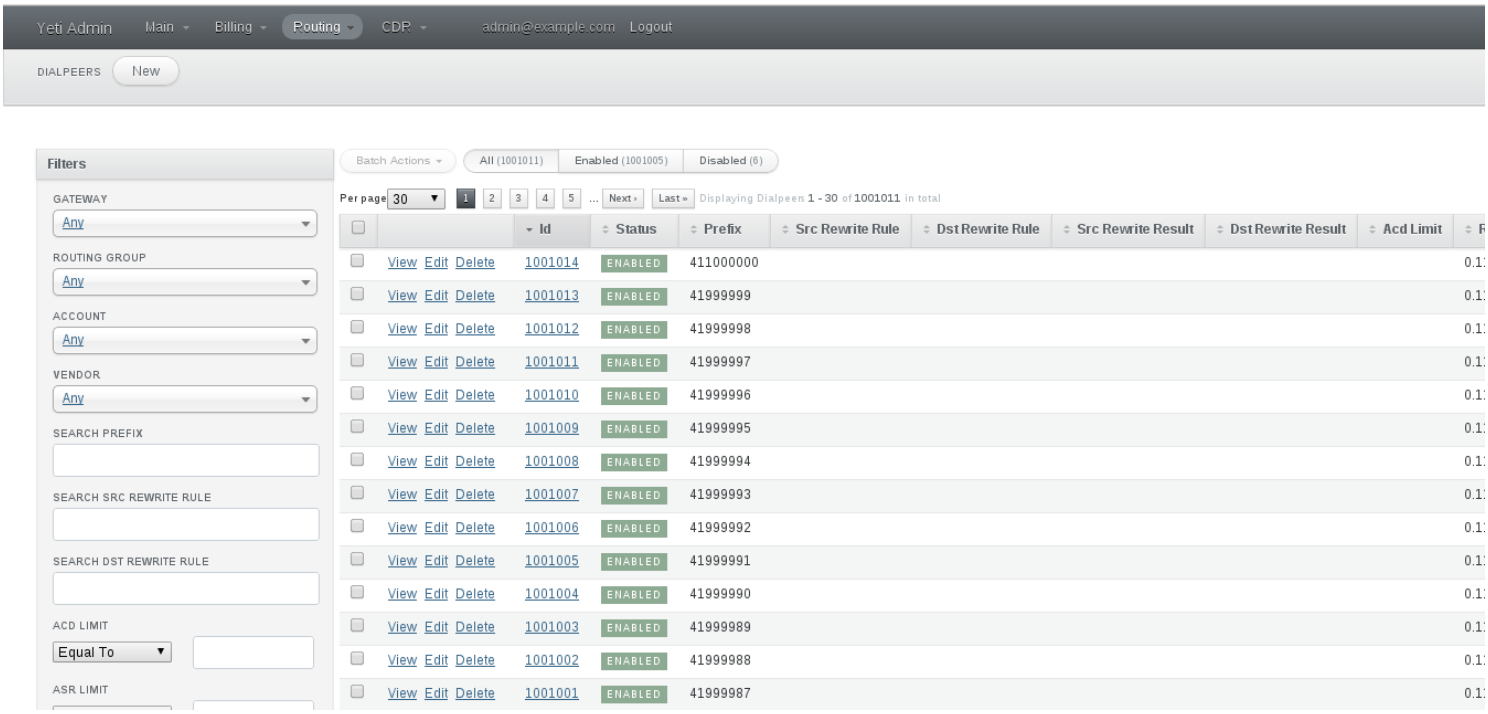
7.2.7 Routing->Destinations

add destinations. Destinations must belong to previously created routegroup



7.2.8 Routing->Dialpeers

add dialpeers



7.3 Standart procedures

7.3.1 Main

Admin Users You can configure authentification of users which will control your system.

Currently there are two privilege levels: Admin and Superadmin. The only difference is Superadmin may also add new users into the system.

User admin@example.com is the SuperAdmin initially.

In current implementation there is no ability to remove users, but you can block desired user.

id unique id. generated automatically

email email used as login

encrypted password user password hash

sign in count successful login count

current sign in IP current user IP if it logged in

last sign in IP IP of last login

created at user creation date

updated at update user date

enabled toggle access to the web-interface for this user

Contractors Contractor is partner to which there is any interaction. System must have at least two contractor. One with the flag customer = true, the other with a vendor = true.

id unique id. generated automatically

name contractor name

enabled state

vendor contractor is provider can be used for the calls termination

customer contractor is client and we are awaiting calls from him

The same contractor can be both vendor and customer at the same time

Audit

7.3.2 Routing

Customers Auths

Name Unique name of authorization rule

Enabled If false then rule will be ignored in the process of authorization

Customer Contractor who will pay for this call

Rateplay Rateplan, according to which will be searched destination

Routing Group Routing group, according to which will be searched routes for call termination

Gateway Incoming gateway. Parameters of this gateway will determine commutation system behaviour on legA

Account Account, which belongs to previously selected contractor. Funds for call will be debited from this account

Src rewrite rule, Src rewrite result translation rules for source number of incoming call
TODO: add link to regexp translations mechanism explanation

Dst rewrite rule, Dst rewrite result translation rules for destination number of incoming call
TODO: add link to regexp translations mechanism explanation

Dump level Logging level for calls. (Attention: recording may significally increase load on system and quickly fill the hard drive. One should use it only as temporary tool for diagnosing of protocol problems)

Capture all traffic record sip signalling and media streams

Capture nothing record disabled

Capture rtp traffic record media streams only

Capture signalling traffic record sip signalling only

Ip Addresses of originator. There is possible format x.x.x.x/z, where x.x.x.x - network address, z - netmask length

Pop Point of presence for which this rule is valid. If not choosen rule valid anywhere.

Src prefix Source number prefix of incoming call. If filled, rule will be triggered only for calls which source number begins from this prefix

Dst prefix Destination number prefix of incoming call. If filled, rule will be triggered only for calls which destination number begins from this prefix

Uri domain Domain part of request URI of incoming call. If empty any domain considered as valid

X-Yeti-Auth Value of header field X-Yeti-Auth in INVITE request of incoming call. If filled, rule will be triggered only for calls which contains appropriate header field value in INVITE request

TODO: explain auth select algo on incoming call

Destinations

Status If equal to false, this entry will be ignored

Prefix Destination number prefix. Used in search for the most appropriate destination using longest prefix match algorithm

Rateplan Each destination belongs to rateplan. Destinations search always performed within certain rateplan

Reject calls If true, all calls to this destination will be rejected

Rate Policy method of determining of the price for the destination for consumer

Fixed Fixed price. Price will be determined by parameters Initial Rate, Next Rate, Connect Fee

Based on used dialpeer Parameter Next Rate will be determined as:
$NextRate = Dp\,next\,rate * (1 + Dp\,margin\,percent) + Dp\,margin\,fixed$,
where "Dp next rate" is "Next rate" parameter from used dialpeer

Min(Fixed,Based on used dialpeer) Will be choosen minimal Next Rate from two first cases

Max(Fixed,Based on used dialpeer) Will be choosen maximal Next Rate from two first cases

Initial Interval First billing interval length

Next Interval Length of each billing interval following after first

Use dp intervals If true then Initial Interval and Next Interval will be obtained from dialpeer instead of using own

Initial rate Price of one minute of conversation during first billing interval (In case when Rate Policy=Fixed)

Next rate Price of one minute of conversation during following (after first) billing intervals (In case when Rate Policy=Fixed)

Connect Fee Fee for the connection. On successfull connection establishment this value will be debited from customer account

Dp margin fixed Fixed margin from dialpeer rates (In case when Rate Policy=Fixed,Based on used dialpeer)

Dp margin percent Relative margin from dialpeer rates (In case when Rate Policy=Fixed,Based on used dialpeer)

TODO: explain destination select algo on incoming call

Dialpeers

Enabled if false entry will be always skipped during processing

Prefix Destination number prefix. Used for search of appropriate route (longest prefix match algo)

Priority Dialpeer priority. Determines position in set of dialpeers after sorting (highest priority value means highest position in set)

Initial interval First billing interval length (supplier billing)

Initial rate Price of one minute of conversation during first billing interval (supplier billing)

Next interval Length of each billing interval following after first (supplier billing)

Next rate Price of one minute of conversation during following (after first) billing intervals (supplier billing)

Lcr rate multiplier Next Rate multiplier which is used on dialpeers sorting

Connect fee Fee for the connection. On successfull connection establishment this value will be added to the supplier balance

Vendor Name of vendor that owns this dialpeer

Gateway Name of gateway which will be used as terminator for call

Account Supplier acount. Call price will be added to this account

Routing group Routing group that owns this dialpeer

Valid from Begin of validity interval. dialpeer will be ignored if time of call less than this value

Valid_till End of validity interval. dialpeer will be ignored if time of call greater than this value

Capacity Route capacity. If capacity overloaded we skip this route and choose another appropriate route if exist

Src rewrite rule, Src rewrite result translation rules for source number after routing
TODO: add link to regexp translations mechanism explanation

Dst rewrite rule, Dst rewrite result translation rules for destination number after routing
TODO: add link to regexp translations mechanism explanation

Acd limit Lower ACD limit for this dialpeer. Dialpeer will be blocked if this limit reached

Asr limit Lower ASR limit for this dialpeer. Dialpeer will be blocked if this limit reached

Rateplans

Name unique rate plan name

Routing Groups

Gateways

Registrations

7.3.3 CDR

Cdr Tables

CDR History

7.3.4 Billing

Accounts

Name Unique account name

Contractor Contractor which owns account. Each contractor may have any number of accounts. Billing is done independently for each account.

Min balance Minimal value of account balance. When this value is reached the incoming calls for this account will not be served (if the contractor acts as consumer)

Max balance Minimal value of account balance. When this value is reached, dialpeers associated with this account will be excluded from the process of calls routing. (if the contractor acts as provider)

Origination capacity The maximum number of concurrent incoming calls that can be authorized for this account (if the contractor acts as consumer)

Termination capacity The maximum number of concurrent outgoing calls that can be routed to dial peers belonging to this account (if the contractor acts as provider)

Payments

Account Account name which belongs to payment

Amount Payment amount. Can be both positive or negative.

Notes Comments to payment

Invoices

7.3.5 RT Data

Acive Calls

Switch Stats

7.3.6 System

Background Tasks

Disconnect codes

Logic Logs

Debug Call

Nodes

Gui Configs

RS Replication

Events

8 Roadmap