Contents

Limit Resource Management

Limit the number of calls to or from an arbitrary resource.

When the limit is reached, the call is automatically transferred to "limit_exceeded" in the same context or the defined context.

Note that a limit is active within a given context - if you transfer an inbound call from the public dialplan to an extension in the default dialplan - any limit you just set in the public dialplan will be reset.

Listen to Rupa talk about limit on the FreeSWITCH Weekly_Conference_Call!

- MPEG
- Ogg
- Wave

Dialplan Apps

The limit dialplan apps are implemented by mod_dptools.

limit

```
limit <backend> <realm> <resource> <max[/interval]> [<transfer_destination_number> [<dialplan> [<
```

backend

The backend to use.

realm

Arbitrary name

resource

The resource on which to limit the number of calls.

max

The maximum number of concurrent calls allowed to pass or a call rate in calls/sec. If not set or set to a negative value the limit application only acts as a counter.

The interval argument is ONLY supported by the hash backend.

transfer_destination_number

Transfer to this extension in the dialplan. It's optional. If you don't give it, limit transfers to 'limit_exceeded' extension in the current dialplan and context.

limit_execute

Execute requested APP only if resource has not reached its limit

```
limit_execute <backend> <realm> <resource> <max[/interval]> <application> [application arguments]
```

backend

The backend to use.

realm

Arbitrary name

resource

The resource on which to limit application execution

max

The maximum number of concurrent executions or an interval in executions/sec. If set to zero (or below zero) then there is no limit; the limit system simply acts as a counter.

application

The application to execute

application arguments

The optional application arguments

API

The limit APIs are implemented by mod_commands. You can also use them via the dialplan like so:

```
<action application="set" data="api_result=${limit_usage(<backend> <realm> <id></pre>)}"/>
```

limit_reset

Resets a given limit backend.

- DB: deletes ALL entries for that hostname.
- Redis: finds all used keys and sets them to 0.
- Hash: not implemented.

```
API: limit_reset <backend>
```

limit_status

Retrieve current status of a given backend. Only supported in DB, not hash or redis.

```
API: limit_status <backend>
```

limit_usage

Retrieve current usage for a given resource, all backends.

```
limit_usage <backend> <realm> <id>
```

uuid_limit_release

Manually decrease the usage by one by removing the "usage" entry for that UUID.

If realm/resource is specified, it removes only that limit. Otherwise, it removes all limits help by the given UUID.

```
API: uuid_limit_release <uuid> <backend> [realm] [resource]
```

limit_interval_reset

Manually reset the interval counter to zero prior to the next interval starting: Only implemented in Hash, not DB or redis.

```
API: limit_interval_reset <backend> <realm> <resource>
```

hash_remote

You can access data from other FreeSWITCH systems with the **hash_remote** API. The hash_remote API uses the event socket. Configure your server names and credentials in conf/autoload_configs/hash.conf.xml.

It seems it queries each remote hash endpoint every 5 seconds for the entire hash list. It hen adds that to normal hash usage on the current server.

```
hash_remote <list>|<kill> [name]|<rescan>
```

Channel Variables

Variables set by limit

The following channel variables are set when limit is called.

- "limit_realm"
- "limit_id"
- "limit_max"

These channel variables are used at hang up to remove the record. More specifically, the delete is limited by uuid, hostname, realm and id.

Variables that affect limit

limit_ignore_transfer=true - causes the current call count to not be reset when the call is transferred. This is useful where calls that come into a gateway are transferred to an extension, but you want to preserve the call count.

limit_ignore_transfer=false - calls that are transferred cause the call count for that realm_id to decrement

Backends

db

You may choose to take advantage of the db backend mod_db to allow multiple servers to limit the number of calls and stay aware of how many calls <resource> has in session across all participating servers.

hash

mod_hash provides a hashtable backend for limit. It uses a hashtable as data structure (faster) and has some additional features. Here is the syntax for the main application:

```
<action application="limit" data="hash <realm> <id> [<max>[/<interval>] [number [dialplan [contex
```

Note that you can also do rate-limiting with this application by specifying an interval, 5/1 will limit the resource to 5 calls per second. The application will transfer the call to the specified number/dialplan/context if the resource is currently over-limit. You can also use tell limit_hash to automatically hangup the call when its over-limit, use a ! before the number to indicate that it is a hangup cause.

```
<action application="limit" data="hash inbound 15142223333 2 !USER_BUSY" />
```

If the maximum isn't specified, limit will count the number of active calls but won't limit anything.

redis

mod_redis provides a redis backend for limit.

Which backend do I use?

Backend	Speed	Persistence	Cluster-ability	Interval Support
Hash	fastest	no	see hash_remote	yes
DB	slow	yes	possible	no
Redis	fast	yes, configurable	yes	no

Examples

Limit access to an application

It is sometimes required to limit access to a single application, such as when trying outbound carriers, the following will do so:

```
<action application="limit_execute" data="hash <realm> <id> <max>[/<interval>] <application> <dat
```

Example: this will try 2 carriers, not sending more than 5 calls per carrier.

```
<extension name="outbound">
  <condition field="destination_number" expression="^1?[2-9]\d{2}[2-9]\d{6}$">
    <action application="limit_execute" data="hash outbound carrier1 5 bridge sofia/gateway/carri
    <action application="limit_execute" data="hash outbound carrier2 5 bridge sofia/gateway/carri
  </condition>
</extension>
```

Limit a user's concurrent calls

The following is a simple example of limiting a user in your domain to ${max_calls}.... Where max_calls is a user variable set in directory.conf or a global variable.

```
<extension name="limit_exceeded">
  <condition field="destination_number" expression="^limit_exceeded$">
    <action application="playback" data="/sounds/overthelimit.wav"/>
    <action application="hangup"/>
  </condition>
</extension>

<extension name="limit" continue="true">
  <condition>
    <action application="limit" data="db $${domain} ${sip_auth_username} ${max_calls}"/>
  </condition>
</extension>
```

NOTE that limit_exceeded comes before the limit extension as limit() uses transfer(), which will search from the beginning of the dialplan. You must do this or add regex to check the destination to avoid a transfer loop.

Rate Limiting calls / Anti SPIT

Limit the calls per second by source ip + destination number:

```
<action application="limit" data="hash ${sip_received_ip} ${destination_number} ${calls_per_secon
```

Limit calls to 5 calls every 10 minutes:

```
<action application="limit" data="hash ${sip_received_ip} ${destination_number} 5/600" />
```

User busy

This checks the current usage a limit counter before dialing a user, conditionally returning user_busy or putting the call through under a specific condition over the value of this counter. This can be useful if you wish to check against a counter managed by an external application or incremented by other events.

```
<action application="bridge" data="${cond(${limit_usage(db time_spent in_bed) <= 60} ? error/user
```

Note: The above action will not increment the limit counter.

Note: In recent versions, in order for this function to return a non-zero value you must call the limit application before. It should set some limit for this resource (even -1 which is unlimited) in order to enable counting.

Using limit with per-gateway or per-user channel limits

If you wish to set a limit bound to the b-leg part of a call (ex: outgoing counter), it is only possible using loopback channels. The following will do so:

```
<action application="set" data="destnum=${destination_number}" />
<action application="bridge" data="loopback/context/gw1,loopback/context/gw2" />
```

And inside the corresponding context:

```
<extension name="gw1">
  <condition field="destination_number" expression="gw1">
    <action application="limit" data="db outgoing gw1 10" />
    <action application="bridge" data="sofia/gateway/gw1/${destnum}" />
  </condition>
</extension>
<extension name="gw2">
  <condition field="destination_number" expression="gw2">
    <action application="limit" data="db outgoing gw2 5" />
    <action application="bridge" data="sofia/gateway/gw2/${destnum}" />
  </condition>
</extension>
```

The result of this example is that if the first gateway has too many channels open, then it cleans up the limit data for the first gateway before trying the next gateway.

If this was done within an extension as a series of limit and bridge apps, then the limit data wouldn't be cleaned up until the a-leg returned to the CS_ROUTING state. That would mean calls would continue holding a channel open on a gateway they had tried while they were still connected to another gateway.

Using limit with an outbound gateway

Example below, things to note:

- auto_hunt=true so that you can jump directly to extensions without going through the entire dialplan
- Replace PROVIDER1..3 with the relevant gateways
- Replace PROVIDER1..3_CHANNEL_LIMIT with the relevant channel limit
- transfer is done after the bridge so that failover works between the providers, ie, PROVIDER1 may be down
- This example is only for 10 digit US phone numbers, please adapt as needed to other dialplans

```
<extension name="Outbound calls">
        <!-- support calls to 10 digit US phone numbers directly -->
        <condition field="destination_number" expression="^(\d{10})$" break="on-true">
                <action application="set" data="continue_on_fail=true"/>
                <action application="set" data="hangup_after_bridge=true"/>
                <action application="enum" data="1$1 e164.arpa"/>
                <action application="bridge" data="${enum_auto_route}"/>
                <action application="enum" data="1$1 e164.org"/>
                <action application="bridge" data="${enum_auto_route}"/>
                <action application="enum" data="1$1 nrenum.net"/>
                <action application="bridge" data="${enum_auto_route}"/>
                <action application="set" data="auto_hunt=true"/>
                <action application="limit" data="$${domain} gw_PROVIDER1 PROVIDER1_CHANNEL_LIMIT
                <action application="bridge" data="sofia/gateway/PROVIDER1/1$1"/>
                <action application="transfer" data="usdirect2"/>
        </condition>
</extension>

<extension name="usdirect2">
        <condition field="destination_number" expression="^usdirect2$"/>
        <condition field="rdnis" expression="^(\d{10}$)">
                <action application="limit" data="db $${domain} gw_PROVIDER2 PROVIDER2_CHANNEL_LI
                <action application="bridge" data="sofia/gateway/PROVIDER2/1$1"/>
                <action application="transfer" data="usdirect3"/>
        </condition>
```

```
</extension>

<extension name="usdirect3">
        <condition field="destination_number" expression="^usdirect3$"/>
        <condition field="rdnis" expression="^(\d{10}$)">
                <action application="limit" data="db $${domain} gw_PROVIDER3 PROVIDER3_CHANNEL_LI
                <action application="bridge" data="sofia/gateway/PROVIDER2/1$1"/>
        </condition>
</extension>

<extension name="limit_exceeded">
        <condition field="destination_number" expression="^limit_exceeded$">
                <action application="playback" data="/sounds/overthelimit.wav"/>
                <action application="hangup"/>
        </condition>
</extension>
```

Using limit to monitor each gateway ip address

To get the number of concurrent calls per gateway ip address, you may issue a command as follows:

```
To monitor "inbound"
# fs_cli -x 'limit_usage db inbound 1.2.3.4'

To monitor "outbound
# fs_cli -x 'limit_usage db outbound 5.6.7.8'
```

Example Dialplan:

```
    <extension name="customer_a">
      <condition field="network_addr" expression="^1\.2\.3\.4$"/>
      <condition field="destination_number" expression="^(.*)$">
        <action application="limit" data="db inbound 1.2.3.4 10000" />
        <action application="limit_execute" data="db outbound 5.6.7.8 10000 bridge sofia/gateway/
      </condition>
    </extension>
```

Using limit to handle outbound calls example

If you have the default configs then locate the Local_Extension in conf/dialplan/default.xml. Add this line right after the condition:

```
<action application="limit" data="hash ${domain} $1 1 handle_over_limit XML over_limit_actions"/>
```

Then add this new file as "limits.xml" in conf/dialplan/ :

```
<include>
 <context name="over_limit_actions">
   <extension name="oops, too many calls for this one">
     <condition field="destination_number" expression="handle_over_limit">
       <action application="answer"/>
       <action application="playback" data="ivr/ivr-no_no_no.wav"/>
       <action application="playback" data="ivr/ivr-no_no_no.wav"/>
       <action application="playback" data="ivr/ivr-no_no_no.wav"/>
       <action application="hangup" data="USER_BUSY"/>
```

```
    </condition>
   </extension>
 </context>
</include>
```

Now when you call a local extension it won't allow more than one call. Note that you can change the value in the limit's data argument. For example, this would cause a limit of 4 concurrent calls, sending the 5th call into "oops, too many calls" extension:

```
<action application="limit" data="hash ${domain} $1 4 handle_over_limit XML over_limit_actions"/>
```

Here is another example that when this person makes a call to a 7-digit number or does 1+ 7 or more digits, it will add to his limit totals. However, if he just calls another 4-digit extension on the system then it won't add to his limit. Put it right after the "global" extension in default.xml:

```
    <!-- set outound caller limit -->
    <extension name="set outbound limit" continue="true">
      <condition field="destination_number" expression="^1?\d{7}" break="on-false"/>
      <condition field="caller_id_number" expression="^(10[01][0-9])">
        <action application="limit" data="hash ${domain} $1 4 handle_over_limit XML over_limit_ac
        <action application="log" data="INFO Added limit for caller $1"/>
      </condition>
    </extension>
```

More examples

Some examples of using limit to protect against toll fraud is here.

Here are some Dialplan Recipes using limit.

Invalid Application hash

You forgot to load the mod_hash module! [1]