Sniffer 5.3 manual

released 5.11.2012

Contents

Introduction

This manual describes installation and configuration of the VoIPmonitor C++ sniffer. If you are looking for WEB GUI manual, go to http://www.voipmonitor.org/download section. The upgrade procedure from versino 4.2 and 5.0 is described at the end of this manual. What's new is at the end of this manual.

VoIPmonitor is open source network packet sniffer for SIP and RTP VoIP protocol running on linux. VoIPmonitor was designed to analyze quality of SIP calls based on network parameters - delay variation and packet loss according to ITU-T G.107 E-model which predicts quality on MOS scale. Calls with all relevant statistics are saved to MySQL or ODBC enabled database. Each call can be optionaly saved to pcap file with either only SIP protocol or SIP/RTP/RTCP protocols. VoIPmonitor can also decode sound and play it over the commercial WEB GUI or save it to disk as WAV. Supported codecs are G.711 alaw/ulaw and commercial plugins supports G.729a/G.723/iLBC/Speex/GSM. VoIPmonitor uses jitterbuffer simulator to keep both direction of call synchronized.

How it works

VoIPmonitor is C++ program designed to handle thousands of simultaneouse calls. It listens on network interface and analyzes all SIP calls on defined SIP ports (default 5060). RTP streams which carries voice are analyzed for packet loss and variation delay (jitter). Each call is saved to MySQL or to any database supporting ODBC. SIP signalization and RTP packets can be saved to individual pcap file which can be opened with analyzers like wireshark and is also used by VoIPmonitor GUI.

How it scales

VoIPmonitor is able to use all available CPU cores but there are several bottlenecks which you should consider before deploying and configuring VoIPmonitor.

CPU bound

The top most consuming CPU is first thread which reads packets from kernel. If you have very large traffic above ~500 Mbit you should check if the first thread is not droping packets by checking syslog where the sniffer is reporting any drop occurences. If you have much more traffic and the CPU is not able to

handle, you can use special kernel modules and drivers which supports hardware acceleration for sniffing very large traffic – but this is only case when your traffic is very large (~5000 simulatenouse calls)

Second top most consuming CPU is threads processing jitterbuffer simulator. In case you do not have enough CPU cores (one or two only) you can turn off jitterbuffer simulator in configuration and keep enabled only one (f2) or turn it off completely. If you have enough CPU cores (at least 4) you should not worry about CPU.

RAM

VoIPmonitor uses several buffers and queues which can be tweaked to match system performance. The first important buffer is ring-buffer which is memory between kernel and libpcap preventing dropping packets due to load spikes. Default value is 20M which might be not enough for higher loads. Recommended value is at least 200M and on heavy loaded servers you should consider to set it to its maximum value of 2000M. The second important buffer is vmbuffer which is circular queue buffer between libpcap and voipmonitor main thread. This buffer can be as big as your available system memory where default is 20M and recommended value is at least 100M. In case the sniffer is saving data to disk and sniffer thread is blocked by the I/O load the sniffer buffers data in vmbuffer first and then in ring-buffer until all buffers are full – then packet drops occurs. Third buffer is used between reader thread and rtp threads (if not disabled). Defautl size is 20M for each thread and on heavier load 100M is recommended.

I/O

In case sniffer is saving RTP and graph data to disk it generates heavy random writes. For more than 100 - 200 concurrent calls you should consider tweaks on I/O level where the simplest is just using one dedicated disk formatted with special EXT4 options (check tuning voipmonitor section in this manual). If you do not have dedicated disk you should consider dedicate one partition for voipmonitor formated with special EXT4 options. If this is not an option too – you can use cachedir sniffer feature which transforms random writes to guaranteed serialized writing using RAM disk.

You should also realize that if you have MySQL database and sniffer data on the same disk / raid – the MySQL is set to sync each CDR by default which lowers throughput and can cause delaying CDR. This can be changed with slight tradeoff – check tuning voipmonitor section in this manual.

Common use cases

All in one

Typical setup is to have sniffer installed along with database and GUI on the same dedicated or PBX server. Web server can access pcap/graph files directly to voipmonitor folder.

Multiple sniffers 1 database/GUI

Next typical setup is to have several sniffers installed on various places and one central WEB server and database. In this case sniffer can store pcap/graph file to local disk and send CDR to central database. The central WEB GUI (if configured appropriatly) is able to download pcap/graph file on demand via sniffer TCP manager. To each sensor id_sniffer number is attached which is saved in cdr.id_sniffer SQL row.

How to deliver packets to sniffer

Sniffing on linux host

You can install or compile VoIPmonitor binary directly on linux PBX or SIP server. This does not requires additional hardware and changes in network topology. The only downside is that voipmonitor consumes hardware resources - RAM, CPU and I/O workload which can affect the whole system. If it is not acceptable to share hardware for voipmonitor the second common use case is doing port mirroring.

Hardware port mirroring

Port Mirroring is used on a network switch to send a copy of network packets seen on one switch port (or an entire VLAN) to a network monitoring connection on another switch port => voipmonitor dedicated linux box. Port mirroring on a Cisco Systems switch is generally referred to as Switched Port Analyzer (SPAN); some other vendors have other names for it, such as Roving Analysis Port (RAP) on 3Com switches.

IPTABLES mirroring

IPTABLES is able to mirror traffic to another IP address. This rules are not needed in case of hardware mirroring. Rules has to be defined on the SIP server (not on the voipmonitor sniffer).

```
iptables –I PREROUTING –t mangle –i eth0 –j TEE –gateway 10.0.0.2

iptables –I POSTROUTING –t mangle –j TEE –gateway 10.0.0.2
```

This is generic rules which will mirror ALL incoming traffic from eth0 and all outgoing traffic from server to VoIPmonitor dedicated box on IP address 10.0.0.2. It is better to mirror just UDP packets

```
iptables –I PREROUTING –t mangle –i eth0 -p udp –j TEE –gateway
10.0.0.2

iptables –I POSTROUTING –t mangle -p udp –j TEE –gateway 10.0.0.2
```

TCP socket traffic mirroring

VoIPmonitor can read data from standard TCP socket using netcat. The idea is to run TCP server and pipe all incoming data to voipmonitor. Then on the sniffer server run tcpdump and redirects to TCP server via IP. Here is example:

Run on voipmonitor web server

```
iptables -I INPUT -p tcp --dport 9999 -j DROP
iptables -I INPUT -p tcp --dport 9999 -s PBXserverIP -j ACCEPT

while [ 1 ] ; do tcpdump -s 0 -U -n -w - -i eth0 'not host remoteWEBserverIP' 2>&1 > /dev/null | nc
remoteWEBserverIP 9999; sleep 1; done &
```

Run on SIP server

```
while [ 1 ] ; do netcat -l -p 9999 > /dev/stdout | voipmonitor --config-file /etc/voipmonitor.conf -k -r /dev/stdin 2>&1
>/dev/null; sleep 1; done &l
```

Commands runs in while undefinit cycle because if the connection is interrupted the netcat is terminated.

SSH traffic mirroring

VoIPmonitor can read data from standard input which means that it can read traffic from any source. That source can be output from wireshark which can run on another host piping it over SSH. Here is example how to do it (the command run on voipmonitor server)

```
ssh root@yourSIPserverIP "tshark -i eth0 -R 'sip or rtp' -w -" | voipmonitor
-r /dev/stdin --config-file /etc/voipmonitor.conf -k
```

Offline pcap files reading

This is less common use case but also used in productions. VoIPmonitor is scripted to read pcap files which are saved on production systems to file with tcpdump or tshark or with any packet sniffer supporting pcap file format.

Installation

VoIP monitor sniffer can be installed in two ways - either as static binary which will run on any Linux distribution with kernels >= 2.6.18 or compiled from sources.

Install static binary

Static binary for 32bit or 64bit can be downloaded from http://www.voipmonitor.org/download pages. Step by step for 64bit linux procedure:

```
tar xzf voipmonitor-*-static.tar.gz
cd voipmonitor-*-static
./install-script.sh
cp voipmonitor.conf /etc/
mysqladmin create voipmonitor
cat cdrtable.sql | mysql voipmonitor
```

Now edit configuration file /etc/voipmonitor.conf and run voipmonitor

```
/etc/init.d/voipmonitor start
```

Compile shared binary

Debian 6 squeeze

```
apt-get install build-essential subversion libmysqlclient-dev
libvorbis-dev libpcap-dev apache2 php5-mysql php5-gd mysql-server
unixodbc-dev

cd /usr/src

svn co \
https://voipmonitor.svn.sourceforge.net/svnroot/voipmonitor/tags/voip
monitor-5 voipmonitor-svn

cd voipmonitor-svn

make clean

make

make install

mkdir /var/spool/voipmonitor

mysqladmin create voipmonitor
```

```
cat cdrtable.sql | mysql voipmonitor

cp config/voipmonitor.conf /etc/

#edit file /etc/voipmonitor.conf to your needs

cp config/init.d/voipmonitor /etc/init.d/

update-rc.d voipmonitor defaults

/etc/init.d/voipmonitor start
```

CentOS 6.3

```
yum groupinstall 'Development Tools'

yum install subversion libpcap-devel mysql-devel libogg libogg-devel
vorbis-tools libvorbis libvorbis-devel mysql-server unixODBC-devel

svn co \
https://voipmonitor.svn.sourceforge.net/svnroot/voipmonitor/tags/voip
monitor-5 voipmonitor-svn

cd voipmonitor-svn

./configure

make

make install

mkdir /var/spool/voipmonitor

/etc/init.d/mysqld start

mysqladmin create voipmonitor

cat cdrtable.sql | mysql voipmonitor

cp config/voipmonitor.conf /etc/

#edit file /etc/voipmonitor.conf to your needs

cp config/init.d/voipmonitor /etc/init.d/

chkconfig --add voipmonitor

chkconfig voipmonitor on

/etc/init.d/voipmonitor start
```

Database configuration

VoIPmonitor stores CDR data to MySQL or any ODBC enabled database. Installing and configuring database is covered in previous Installation chapter. This section explains it in detail.

MySQL

This step assumes that MySQL server is started.

Create database - default is voipmonitor

```
mysqladmin create voipmonitor
```

Create SQL schema – from versin 5.2 this is automatically done once the voipmonitor is started. This step is not neccessary anymore.

```
cat cdrtable.sql | mysql voipmonitor
```

Running voipmonitor

Voipmonitor starts via standard init.d script /etc/init.d/voipmonitor start|stop. This script tells voipmonitor to load configuration file from /etc/voipmonitor.conf. You can run voipmonitor also from command line

```
voipmonitor --config-file /etc/voipmonitor.conf
```

which will immediately fork and run as background. If you need to see what it does, run it like this

```
voipmonitor --config-file /etc/voipmonitor.conf -k -v 1
```

Almost all configuration directives can be also provided via command line (which takes precedence over the configuration file).

To show help run voipmonitor without any parameters.

Configuring voipmonitor

/etc/voipmonitor.conf

Configuration file has only one section named [general] where all configuration directives belongs. List of directives will now follow with their description and recommendation values. Name in [ ] brackets is equivalent for command line which takes precendence over configuration file.

interface = eth0 [-i]

This specifies on which interface will voipmonitor listen. It can listen on one interface or on all interfaces. To listen on all interfaces use interface = any

natalias = 1.1.1.1 10.0.0.3
natalias = 1.1.1.2 10.0.0.4

in case the SIP(media) server is behind public IP (1.1.1.1) NATed to private IP (10.0.0.3) to sniff all traffic correctly you can specify alias for this case. You can specify more netaliases duplicating rows. In most cases this is not necessary because voipmonitor is able to track both RTP streams based on the other side IP. But if the stream is incoming from another IP then SIP source signalization and also from another IP than the SIP device which is also behind NAT its impossible to track the correct IP. Please note that this is for case where the SIP server is behind NAT and also the client is behind NAT. If your SIP server has public IP do not bother with this.

managerport = 5029 [--manager-port <port number>]

This specifies TCP port which will voipmonitor listen for incoming connections which controls voipmonitor or for getting information about calls.

- reload configuration echo reload | nc localhost 5029

- get number of calls echo totalcalls | nc localhost 5029

- get list of calls in json format echo listcalls | nc localhost 5029

(listcalls is currently limited to max 200 calls)

managerip = 127.0.0.1

define bind address for manager interface. Default is 127.0.0.1 it is not recommended to change this unless really needed due to security. If you need it on some other IP make sure you set firewall and change the standard port for better security

sipport = 5060

define SIP ports wihch will voipmonitor listen. For each port make new line with sipport = port (multiple lines)

rtptimeout = 300

rtptimeout is important value which specifies how much seconds from the last SIP packet or RTP packet is call closed and writen to database. It means that if you need to monitor ONLY SIP you have to set this to at leat 2 hours = 7200 assuming your calls is not longer than 2 hours. Take in mind that seting this to very large value will cause to keep call in memory in case the call lost BYE and can consume all memory and slows down the sniffer - so do not set it to very high numbers. Default is 300 seconds.

ringbuffer = 20 [--ring-buffer]

This feature is the most essential parameter for high volume calls. The value is buffer size in MB allocated in kernel space. This feature will work only on kernels >= 2.6.32 and libpcap >= 1.0. The static version of voipmonitor contains libpcap 1.1.1. The ringbuffer is queued by packets from ethernet device and dequeued by voipmonitor. If the buffer is low and the system is overloaded (CPU or I/O) packets will be dropped. This situation will be logged to syslog.

Recommended value for high loads – more than 1000 (will take 1GB of RAM)

Notice: if you set this value over ~2000 libpcap will silently allocate nothing. Thus it is recommended to check with "ps axl|grep voipmonitor" if the voipmonitor actualy uses amount of RAM you specified.

vmbuffer = 50

vmbuffer is user space buffers in MB which is used in case there is more then 1 CPU and the sniffer run two threads - one for reading data from libpcap and writing to vmbuffer and second reads data from vmbuffer and process it. For very high network loads (more than 400 calls) set this to very high number (> 1000). Or in case the system is droping packets (which is logged to syslog) increase this value.

rtpthreads = 3

number of threads to process RTP packets. If not specified it will be number of available CPUs - 1. If equal to zero RTP threading is turned off. Each thread allocates default 20MB for buffers (increase to 100 on very high loads).

This buffer can be controlled with rtpthread-buffer. For < 150 concurrent calls you can turn it off.

rtpthread-buffer 100

size of rtp thread ring buffer queue in MB. Default is 20MB per thread - increase it at least to 100 for huge traffic (> 500 simultaneous calls)

jitterbuffer_f1 = yes
jitterbuffer_f2 = yes
jitterbuffer_adapt = yes

By default voipmonitor uses three types of jitterbuffer simulators to compute MOS score. First variant is saved into cdr.[ab]_f1 and represents MOS score for devices which has only fixed 50ms jitterbuffer. Second variant is same as first but for fixed 200ms and is saved to cdr.[ab]_f2 Third varinat is adaptive jitterbuffer simulator up to 500ms Jitterbuffer simulator is the most CPU intensive task which is voipmonitor doing. If you are hitting CPU 100% turn off some of the jitterbuffer simulator.

Recommended for higher loads is to use only fixed 200ms.

rtp-firstleg = no [--rtp-firstleg]

this is important option if voipmonitor is sniffing on SIP proxy like kamailio or openser and sees both RTP leg of CALL. In that case use this option. It will analyze RTP only for the first LEG and not each 4 RTP streams which will confuse voipmonitor. Drawback of this switch is that voipmonitor will analyze SDP only for SIP packets which have the same IP and port of the first INVITE source IP and port. It means it will not work in case where phone sends INVITE from a.b.c.d:1024 and SIP proxy replies to a.b.c.d:5060.

sipoverlap = yes

enable/disable updating called number from To: header from each caller INVITE. Default is enabled so it supports overlap dialing (RFC 3578)

if you want to disable this behaviour and see always number only from the first INVITE set sipoverlap = no

sip-register = no [-R]

Enable parsing of SIP REGISTER message. SQL register table stores active SIP registrations. Once it expires it is removed from the table to new sql register_state table. The register state table is used to store changes in registrations. SQL table register_failed is used to store all failed sip register. To not overload this table there is counter column which adds +1 for each failed register from the same source.

savesip = [--sip-register]

Store SIP packets to pcap file.

savertp = yes | header [-R]

Store RTP packets to pcap file or save only RTP headers and not payload (voice)

savertcp = yes [--save-rtcp]

Store RTCP packets to pcap file.

saveudptl = yes

save UDPTL packets (T.38). If savertp = yes the udptl packets are saved automatically. If savertp = no and you want to save only udptl packets enable saveudptl = yes and savertp = no

savegraph = plain [-G or --save-graph=[gzip|plain]]

This is usefull only if you have commercial WEB GUI which uses graph files for ploting graph

mos_g729 = no

enable MOS score for G.729 codec. If enabled, all cdr with 0 packet loss and stable delays will have maximum MOS of 3.92 and for loss and unstable delay MOS will be calculated according to ITU-T objective PESQ method for G.729 codec. if you want to use MOS as good search value which corellates loss and delay into single value leave it disabled (which is by default). If set to no, all calls will be calculated like it is G.711.

Recommended value = no

match_header = in-reply-to

enable saving content of custom header (typicaly in-reply-to) to cdr_next.match_header this header is used in related CDR GUI for matching legs to onen call

pcapcommand = gzip %pcap%

pcapcommand will run shell command after pcap file is closed (after call ends). %pcap% is substitution for real pcap file name. execution is guaranteed to run in serialized way (not in parallel). This example will gzip pcap file. Compressing the file to the same disk will overload disk I/O in high volume calls.

filter = udp [-f]

libpcap tcpdump style filter. Voipmonitor listens in default only for UDP packets. Unfortunatly filtering UDP packets will filter all VLAN tagged packets which means that you cannot filter only UDP if you want to listen to VLAN tagged packets.

spooldir = /var/spool/voipmonitor [-d]

This is directory where all pcap/graph/wav files are stored.

cachedir = /dev/shm/voipmonitor

store pcap and graph files to <cache/dir> and move it after call ends to spool directory. Moving all files are guaranteed to be serialized which solves slow random write I/O on magnetic or other media. Typical cache directory is /dev/shm/voipmonitor which is in RAM and grows automatically or /mnt/ssd/voipmonitor which is mounted to SSD disk.

promisc = yes [-n]

This option is only relevant if you are mirroring traffic to your network card/cards. This will not work if interface = any - in this case, use ifconfig to put your desired interfaces to promis mode.

Default value is yes and you want to turn it of on command line ues -n which will turn it off.

sqldriver = mysql
#sqldriver = odbc
#odbcdriver = mssql
#odbsdsn = voipmonitor
#odbcuser = root
#odbcpass =

voipmonitor can connect to mysql server or odbc driver. connecting voipmonitor to msssql please refer to README.mssql

mysqlhost = localhost [-h]

mysql server, default is localhost

mysqldb = voipmonitor [-b]

mysql database, default is voipmonitor

mysqltable = cdr [-t]

mysql table, default is cdr

sqlcdrtable_last30d = cdr_last30d
sqlcdrtable_last7d = cdr_last7d
sqlcdrtable_last1d = cdr_last1d

enable redundant tables which WEB GUI use for speedup searches on giant CDR tables with millions of records.

mysqlusername = root

mysql username, default is root

mysqlpassword =

mysql password, default is no password

Tuning VoIPmonitor

MySQL server

Edit /etc/mysql/my.cnf (debian/ubuntu) or /etc/my.cnf (centos) – [mysqld] section.

Compression

Mysql >= 5.1 can do table compression which greatly reduces size of the database by factor 2 without any slowdowns on CPU. Compressing CDR helps to keep whole table or its major part completely in innodb buffer (if set large) and thus minimizing time for reading database.

Enabling compression on MySQL 5.1

/etc/mysql/my.cnf – [mysqld] global innodb_file_per_table = 1

restart database

Enabling compression on MySQL > 5.1

MySQL> set global innodb_file_per_table = 1;
MySQL> set global innodb_file_format = barracuda;

Common procedure for both MySQL versions:

```
MySQL> ALTER TABLE cdr Engine=InnoDB ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
MySQL> ALTER TABLE cdr_next Engine=InnoDB ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
MySQL> ALTER TABLE cdr_ua Engine=InnoDB ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
```

If you choose KEY_BLOCK_SIZE=2 instead of 8 the compression will be twice better but with CPU penalty on read. We have tested differences between no compression, 8kb and 2kb block size compression on 700 000 CDR with this result (on single core system – we do not know how it behaves on multi core systems). Testing query is select with group by.

No compression – 1.6 seconds
8kb - 1.7 seconds
4kb - 8 seconds

innodb_buffer_pool_size = 4G (or more)

This is very important variable to tune if you're using Innodb tables. Innodb tables are much more sensitive to buffer size compared to MyISAM. MyISAM may work kind of OK with default key_buffer_size even with large data set but it will crawl with default innodb_buffer_pool_size. Also Innodb buffer pool caches both data and index pages so you do not need to leave space for OS cache so values up to 70-80% of memory often make sense for Innodb only installations.

innodb_flush_log_at_trx_commit = 2

Default value of 1 will mean each update transaction commit (or each statement outside of transaction) will need to flush log to the disk which is rather expensive, especially if you do not have Battery backed up cache. Many applications are OK with value 2 which means do not flush log to the disk but only flush it to OS cache. The log is still flushed to the disk each second so you normally would not loose more than 1-2 sec worth of updates. Value 0 is a bit faster but is a bit less secure as you can lose transactions even in case MySQL Server crashes. Value 2 only cause data loss with full OS crash.

If you are importing or altering cdr table it is strongly recommended to set temporarily innodb_flush_log_at_trx_commit = 0 and turn off binlog if you are importing CDR via inserts.

Hardware

If you have not enough memory for innodb_buffer_pool_size and you are doing a lot of searches in database (through web or whatever) the most dramatic speedup is storing mysql to SSD disk. We have very good experience with Intel 5XX SSD disk which has random write 40 000 IOPS (usual SATA disk have 300-600 IOPS for random write). On that SSD disk we were not able to produce workload which is I/O bound anymore even on inserting dumped database.

File system

The fastest filesystem for voipmonitor spool directory is EXT4 with turned off journaling and other tweaks. It is better to have system/mysql on system partition and use another partition with following tweaks (the best is to have voipmonitor spool directory on dedicated disk too). Assuming your partition is /dev/sda2:

```
mke2fs -t ext4 -O ^has_journal /dev/sda2

tune2fs -O ^has_journal /dev/sda2

tune2fs -o journal_data_writeback /dev/sda2

edit /etc/fstab

/dev/sda2       /var/spool/voipmonitor  ext4
errors = remount-
ro,noatime,nodiratime,data=writeback,barrier=0 0 0
```

Upgrade from 4.2 to 5.0

Database schema has changed in version 4 in way that it needs to alter table at least two times which is so inefficient that we had to write PHP script which transforms old CDR into new structure. On SATA disk upgrading 12 milions CDR takes ~24 hours which means that the table is also locked and no CDR is possible to write during the upgrade procedure. The procedure is:

1) create new database

```
mysqladmin create voipmonitor5
cat cdrtable.sql | mysql voipmonitor5
```

2) cd voipmonitor/scripts

edit mysql_copy_4.2to5.0.php and set appropriate constants:

```
define("HOST", "localhost");
define("USER", "root");
define("PASS", "");

define("SOURCE_DB", "voipmonitor");
define("DEST_DB", "voipmonitor5");
```

SOURCE_DB is the old database name, DEST_DB is the new database.

3) Run the script

```
php  mysql_copy_4.2to5.0.php
```

The script can run for very long time so its recommended to run it from “screen” (apt-get install screen | yum install screen).

The speed of conversion depends a lot on two factors

- if binlog is enabled it is better to disable it in /etc/mysql/my.cn (comment out log_bin)
- in /etc/mysql/my.cnf set innodb_flush_log_at_trx_commit = 0 (and after you finish the upgrade, set it to = 2 (more secure)
- if disk is raid5 or slow SATA disk, insertion would be very slow
- if the disk with mysql is shared with /var/spool/voipmonitor and voipmonitor sniffer is running. In this case stop mysql, mv /var/lib/mysql /mnt/dedicated/; mkdir /var/lib/mysql; chown mysql

/var/lib/mysql; mount -o bind /mnt/dedicated/mysql /var/lib/mysql – and start mysql. After you finish export you can move all files back (do not forget to keep right permission on the files - user mysql).

Upgrade from 5.0 to 5.1

Upgrading database from 5.0 do not changes cdr table and thus the upgrade is instant. Here is the procedure

Download voipmonitor sources and untar

```
wget
https://sourceforge.net/projects/voipmonitor/files/5.1/voipmonitor-5.1-src.tar.gz/download
```

Go to voipmonitor source directory and run this command

```
cat cdrtable.sql.5.0-5.1 | mysql voipmonitor
```

Where voipmonitor is name of the database. If you have password protected database, run

```
cat cdrtable.sql.5.0-5.1 | mysql -p voipmonitor
```

Upgrade from 5.1

Since version 5.1 database is upgraded and populated automatically during first run (if mysql credential is set properly – CREATE and ALTER is needed).

5.2 --> 5.3

- implement matchheader config in voipmonitor.conf which will save provided SIP header to cdr_next.match_header which is then used in WEB GUI to find all CDR legs. Typical is matchheader = in-reply-to. It is turned of by default and if enabled the database is automatically altered once the voipmonitor run which can take minutes - hours depends on number of rows in cdr_next and I/O speed.- fix cdrtable.sql.* typo for sensors table

- fix wav decode for GSM

- add rtptimeout (voipmonitor.conf), -m, --rtp-timeout it is important value which specifies how much seconds from the last SIP packet or RTP packet is call closed and writen to database. It means that if you need to monitor ONLY SIP you have to set this to at leat 2 hours = 7200 assuming your calls is not longer than 2 hours. Take in mind that seting this to very large value will cause to keep call in memory in case the call lost BYE and can consume all memory and slows down the sniffer - so do not set it to very high numbers. Default is 300 seconds. rtptimeout = 300

- remove custom_header1 from cdr table as it is not used (it is in cdr_next)- do not store last sip response to BYE but retain it for the invite.

Troubleshooting

voipmonitor does not sniff anything

- Always check if you actually see the SIP traffic. The easest way it to run

(apt-get install tshark | yum install wireshark)

```
tshark -i eth1 -R sip
```

if you do not see traffic, make sure that the interface is UP (ip link set up dev eth1)

if you use "-i any" and you are port-mirroring traffic, make sure you put interface to promisc mode (ifcofnig eth1 promisc; ifcofnig eth2 promis;) you can put this directly in /etc/init.d/voipmonitor

- Check /var/log/syslog or /var/log/messages for any problems releated to voipmonitor. Voipmonitor logs to syslog.

- Check if voipmonitor is runningn "ps axl |grep voipmonitor" and is using configuration file (--config-file /...)