Carrier grade VoIP systems with Kamailio

Welcome!

Henning Westerholt

Kamailio project

1&1 Internet AG

Linuxtag 2009, 24.06.2009

Outline

1. **1&1 VoIP backend**

   purpose and usage
   architecture

2. **Kamailio SIP server**

3. **High-availability and failure-tolerance**

   practical problems
   customer expectations and legal requirements
   monitoring, automation and procedures
   maintainability and system complexity

4. **Performance and scalability**

   scaling issues
   caching, partioning and data locality
   over-optimization

1&1 voice over IP backend

purpose

- provide telephony services for our DSL customers
- basic call routing and also supplemental services

some numbers

- over 1000 million minutes per month to the PSTN
- more than 2 Million customers on the platform

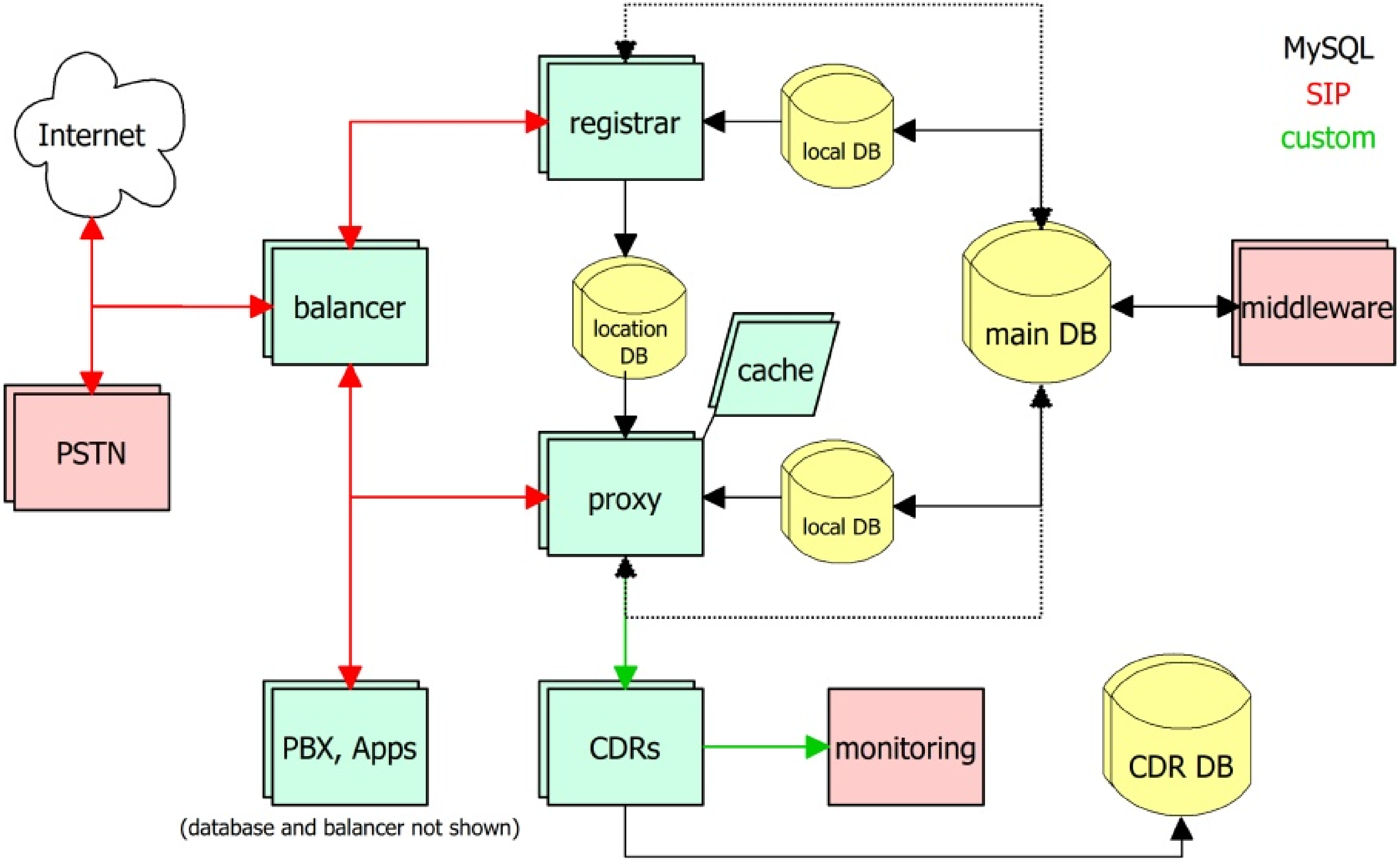
redundant infrastructure on several levels

clustering for applications and databases

interfacing to other carrier networks and internal systems

custom testing and monitoring systems

1&1 voice over IP backend



About Kamailio

- building block of VoIP infrastructures
- provides core services
  - proxy
  - registrar
  - balancer or router
  - application server
- no PBX, more like a router
- cares only about signaling, no RTP data
- foundation of custom high-performance SIP services

About Kamailio

an open source project

- licenced under GPL (version 2 or later)
- over 200,000 lines of C code
- frequent time-based releases
- managed from a board of core developers

community aspects

- over 20 developers provide support and contribute new features
- friendly and healthy user community
- regular meetings at international free and open source conferences

a mature product

- used from carriers like 1&1, QSC, Telefonica..
- several companies use it to provide turn-key solutions, also sold as appliance

merge with the SER project, Kamailio forked several years ago

Why Kamailio

Scalability

usable from small embedded systems to carrier grade systems
from a few hundreds user up to several millions

Performance

on a standard server several thousands calls per seconds throughput is no problem
a server with enough memory can manage 600.000 users

Flexibility

small core written in C
functionality can be added with modules, over 90 already available
configuration script allows access to any part of the SIP message
routing decisions can be derived from many different sources

further informations

extensive documentation available on kamailio.org, see also sip-router.org

High-availability and failure-tolerance

what happens if..

- the primary and secondary DNS of your carrier dies
- your carrier SBCs don't like your SIP anymore
- somehow IPs of some carrier GWs are firewalled

and don't forget your own infrastructure

- the call routing proxies suddendly starts to crash
- your database replication don't like to work anymore
- human error destroyes some critical databases

But:

- customers expect (of course) the same reliability as in the "normal" PSTN
- legal requirements (e.g. for emergency call routing) applies also to VoIP systems

Monitoring and redundancy

quality related parameter and utilization informations

- Jitter, Paket loss, ASR, NER, SIP status codes..
- minutes per minutes and ASR count
- incoming and outgoing traffic and its distribution

both human and automatic checking of important values

service level monitoring

- check if the service is still running, listen on the ports, writes logs..
- use tools like "sipp" to check if basic functions are available

redundancy solutions

- SRV load-balancing for front-end balancers
- automatic or manual failover for proxies and registrars
- automatic failure routing if one GW or carrier fails
- manual routing changes also necessary by quality problems

Automation and procedures

manual processes are risky because of human errors

try to automate as much as you can

example: least-cost-routing data update

- generate DB content from a description language that is managed in a repository
- push this automatically to a master DB in order to replicate it
- trigger cache re-load on all machines after finishing the upload

Establish procedures for common tasks

example: update to a new software release

- establish rules what and how much you change for every release
- have a pre-production test suite on a dedicated test system, and use it
- create release announcements with an update plan
- communicate and follow the plan

Quality assurance and maintainability

find bugs before they reach your production system

- by catching them in your internal test-suite
- by getting your code in the public repository
- by catching them in a upstream test-suite

prefer general solutions over custom implementations

- getting input from the community helps here a lot
- also good motivation for updates
- get your code in the public repository, or even better improve existing parts
- provide feedback that your requirements are heard

support the project

- for example with infrastructure, donations, organisational or development work..
- fix bugs directly in the upstream
- participate in discussions and events

Performance and scalability

Performance and scalability

storage and retrieving of location data

- difficult to scale because of frequent access and changes
- what works for two proxies don't work for more
- availability is critical for call setup

no stable and suitable clustering solution from MySQL in the past

- proprietary partioning solution implemented as Kamailio module
- provides also error-handling and automatic failover

try to minimize database access

- partioning also helps here to ensure data locality
- use modules that cache their content in RAM
- setup local read-only database slaves
- use local DNS caches
- use plenty of RAM

Performance and scalability

server tuning

increase PKG_MEM and SHM_MEM pool to a few times the default value
use more worker children for network connections
set TCP send and connect timeout to small values, to prevent blocking
make sure you use non-blocking syslog file writing
disable dynamic blacklisting and DNS search list usage, use the port for host names

configuration tuning

try to optimize the common path, special cases comes later
low-level tuning normally not necessary
use the benchmark module to find bottlenecks

data optimization

minimize least-cost-routing rules, e.g. by combining prefixes
try to get rid of obselete entries in DB based logic
think about how often data must be changed (registration interval, expire logic..)

Performance and maintainability

CPU power is not a problem

- given todays multi-core CPUs
- so try to find the bottleneck and optimize there
- most probably it will be something related to IO

prefer general solutions over custom implementations

- e.g. use script logic with PVs instead of coding a custom module
- implement DB queries with sqlops/ avpops and not in C
- use the perl module for quick hacks

design in a modular way

- hierachical sub-routes for certain areas (i.e. PSTN, presence..)
- use small routes as "functions" to provide common functionality (normalization, relaying)
- place the necessary SIP functional blocks on different hosts
- use only needed functionality, e.g. balancers can be stateless to save resources

Thanks for your attention!

More informations and contact:

henning.westerholt@1und1.de
kamailio user and sip-router developer mailing list
extensive documentation available at http://kamailio.org and http://sip-router.org

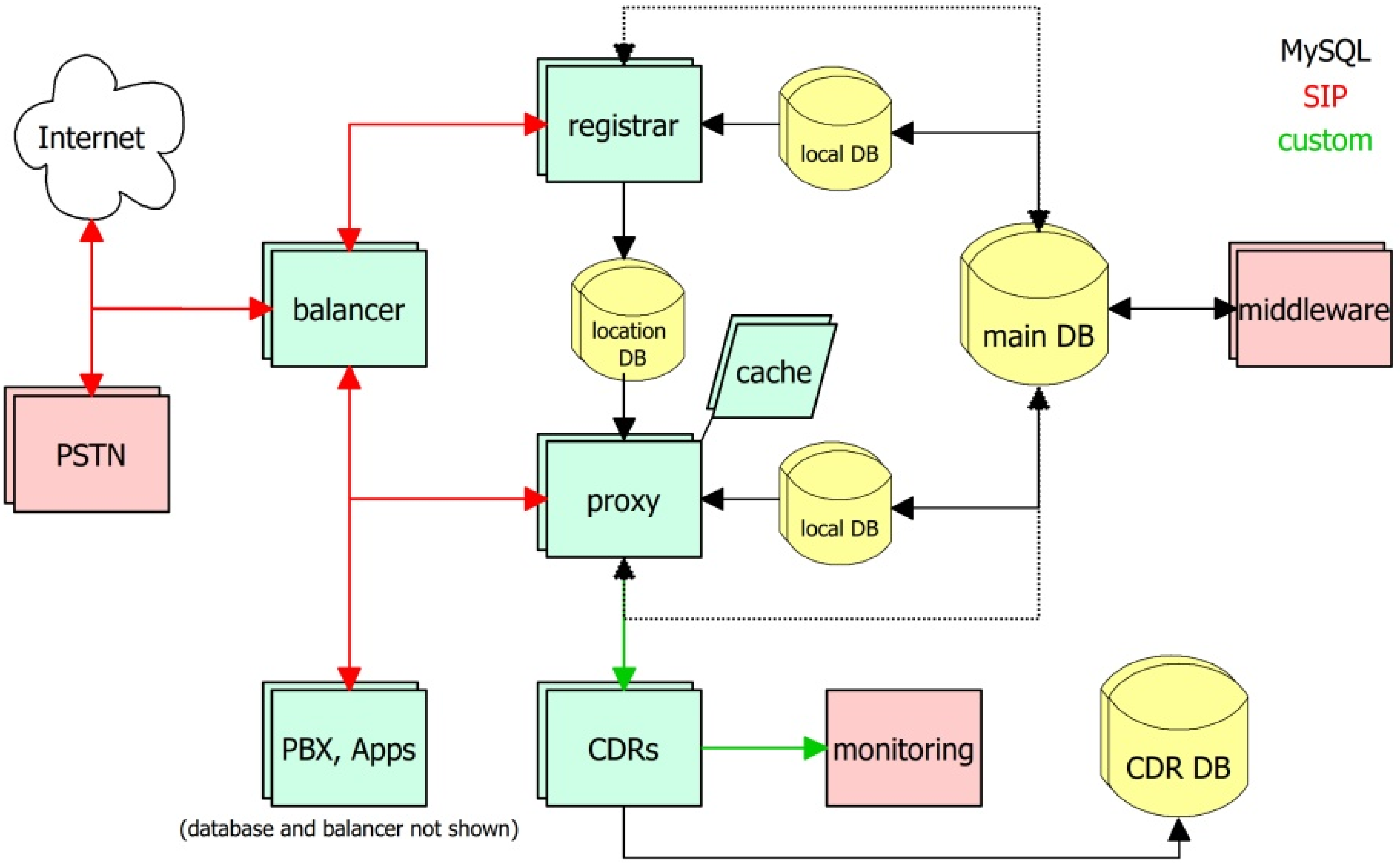
Pictures:

slide 1: spring flowers, central Turkey
slide 5: © Thorsten Wagner, http://www.flickr.com/people/thunderstar/
slide 8: © Randi Hausken, http://www.flickr.com/people/randihausken/
slide 12: © gato-gato-gato, http://www.flickr.com/people/gato-gato-gato/
slide 16: sunrise at Byala beach, Bulgaria

Licence of the slides: cc BY NC SA

http://creativecommons.org/licenses/by-nc-sa/2.0/

1&1 voice over IP backend



(database and balancer not shown)