THE ERICSSON SGSN-MME - OVER A DECADE OF ERLANG SUCCESS

Urban Boquist
Ericsson AB
OUTLINE

› Mobile Telecommunications Networks
› SGSN-MME
› Erlang
› Fault Tolerance
› Capacity & Overload
› Multicore & Scalability
› Large scale software development
MOBILE TELEPHONY

Radio Network
+
Core Network

Circuit Switched
Old
Voice
SMS

Packet Switched
New
IP
WWW, Email, etc.
Voice-over-IP
3GPP MOBILE SYSTEMS – GSM, W-CDMA & LTE

Telephony Network

GSM:
- MS
- BTS
- BSC
- MSC
- HLR
- G-MSC

GSM+GPRS:
- MS
- NB
- RNC
- SGSN
- MME
- GGSN
- HSS

W-CDMA:
- MS
- NB
- RNC
- SGSN
- MME
- GGSN
- HSS

LTE:
- UE
- eNB
- S1-MME
- Serving GW
- PDN GW

IP Network

GSM: 1991
GSM+GPRS: 2000
W-CDMA: 2002
LTE: 2009
SGSN-MME HARDWARE

› 3 magazine cabinet
› Each general board:
  - recent Intel Xeon multicore
  - lots of RAM
› Special purpose HW:
  - switches, routing HW
  - FPGAs
  - physical interfaces
› Everything redundant

› Price: high!
SGSN-MME capacity over 12 years

Capacity

R1.0
R2.0
R2.1
R2.2
R3.0
R5.0
R5.5
R6.0
R7.0
R8.0
R2008B
R2009
R2010
R2011
R2012

MSAU
Requirements

› Control Signalling
  - Between network and Mobile Phone (MS)
  - Invisible to user
  - Called “Signalling”

› User Traffic
  - Normal IP packets between MS and Internet
  - Requested and seen by user
  - Called “Payload”
ARCHITECTURE

Control Plane

Switch

Payload Plane

soft real time

hard real time

MS

Internet
WHY ERLANG?

› High level language
› Built-in concurrency
› Built-in distribution
› Built-in fault tolerance
› Runtime code replacement

Exactly what is needed to build a robust control plane!
Fault Tolerance

ISP – In Service Performance
SGSN-MME must never be out of service! \(\rightarrow 99.9999\%\)

Hardware fault tolerance ("easy")
- Detect faulty HW
- Take it out of service

Software fault tolerance ("hard"!)
- Many more degrees of freedom
- Not so easy to take SW out of service
EXAMPLE SW FAULT TOLERANCE

› System principle: one Erlang process serves one MS
› SW error in SGSN-MME (“MS handling code”) leads to:
  − restart of process
  − all data stored for MS removed from SGSN-MME
  − MS is forced to restart signalling from the beginning
  − ISP effect: short service outage for this MS
  − no other MS:es affected
Do not try to “handle errors”
Crash instead!
Offensive programming
Error could be in MS or in SGSN-MME:
- failure to follow standard
- internal state messed up
- packet corrupt
SW RECOVERY STRATEGY

- Restart Levels
- Escalation Hierarchy
- Kill more and more processes
- Remove more and more stored data
- Time vs. effect?

- very small restart
- small restart
- large restart
- very large restart
BUGS IN ERLANG

› If the SGSN-MME fails our customers do not care who introduced the bug
› We must be able to handle Erlang/OTP bugs

› Same basic recovery mechanisms are used!
› Special rule for this case: “kill entire Erlang BEAM”
› SGSN-MME includes lots of “monitoring” of internal state
› Try to identify Erlang BEAMs that misbehave
OVERLOAD PROTECTION

› The SGSN-MME must never “stop to respond”
› CPU load must be kept below 100% (unreliable otherwise)
› High load can be:
  - user initiated
  - network faults leading to excessive signalling
  - denial of service attacks

› Solution: drop some packets (selectively)
› Natural in Erlang message passing paradigm!
› Difficult in practice: takes years of experience from live networks to get right
Multicore & Scalability

- Erlang in theory: “scalability for free”
- In practice: not for free, but quite good
- SGSN-MME workload “one process per MS” is almost the perfect fit!
- But very hard to avoid system level bottlenecks
  - dispatcher processes
  - ETS tables
  - lock contention
  - communication
- Multicore profiling at high load is very hard!
OTP R14 → R15

› HW is Intel Xeon, 8 schedulers
› Test is “SGSN-MME traffic model”
   - simulating a number of MS doing “normal things”

› multicore scheduler improvements
› half word machine
› ASN.1 decoding NIF
› “nospin” patch

→

› CPU load R14: ~30%
› CPU load R15: ~20%
RUNTIME CODE CHANGE

› Live patching is a must
› The less disturbance the better
› Erlang built in support is good but far from enough

› A whole system level strategy needs to be built on top
› Must include “operational and usability aspects”
› Procedure should be automatic – humans make mistakes!

› A single failed patching means it will be harder to convince customer to install next patch!
FUNCTIONAL PROGRAMMING?

- SGSN-MME technical standards (GPRS) are extremely complex
- We invented lots of abstractions and design patterns
- Let programmer concentrate on GPRS – not on programming details
- Functional parts of Erlang make this easier
- Result is a kind of “Telecom/GPRS domain specific language” embedded within Erlang

- Works very well!
- Hard for some programmers to accept that they are not in full control
LARGE SCALE DEVELOPMENT

› Several hundred people – almost 15 years
› In the beginning many different sites – all over the world
› Now mainly on two sites

› Difficulties:
  ¬ manage the source code: lots of parallel activities
  ¬ merging and integration activities take much resources
  ¬ how to keep good quality of “very old code”?  
  ¬ hard to do some fundamental changes – too much code depends
  ¬ ways of working constantly improving
  ¬ from RUP to cross functional teams and lean
CONCLUSIONS

› Erlang is more or less “perfect” for the control plane in a system like this
› Erlang/OTP is very good now – many bugs historically
› Tools can be improved, eg high load profiling
› Many telecom nodes have similar requirements – few use Erlang

› Final words:
  − Erlang is fun to work with!
  − How long can this amazing system continue to evolve?