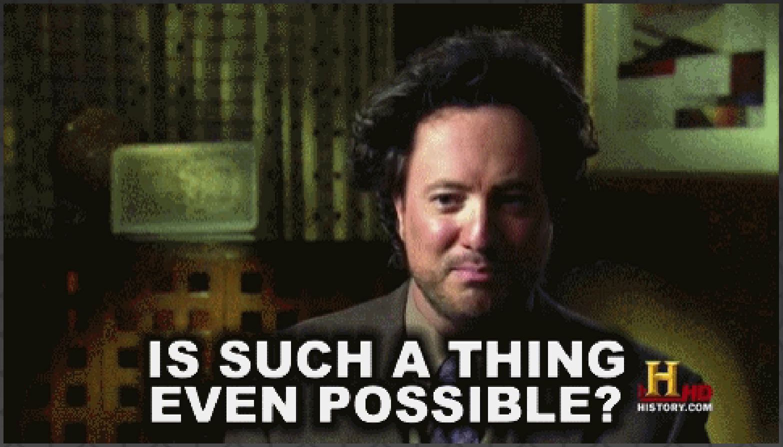
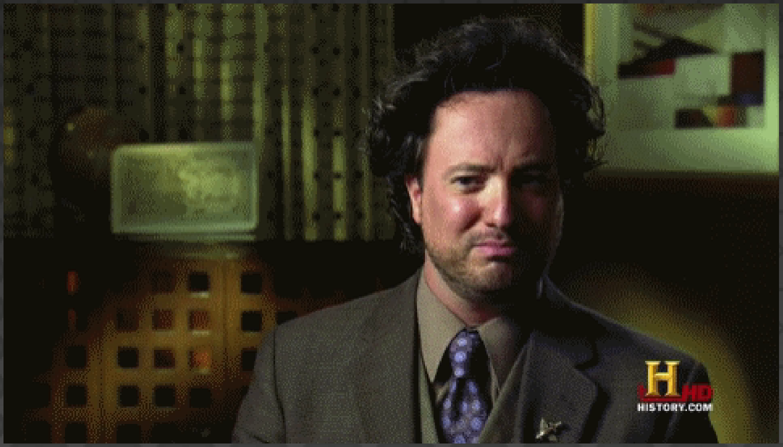
Erlang & Telephony

Presented by

Peter Lemenkov

Erlang & Telephony





Agenda

1. A very common VoIP issue
2. ...and a traditional approach.
3. How to solve it better with Erlang?

A common VoIP issue

- IPv4 + UDP + NAT + Lots of ports (RTP, RTCP, audio, video)
- IPv6 isn't going to fix that fully.
  - Transcoding
  - Lawful interception
  - Dumb (proprietary) hardware clients
  - Stats retrieval
  - Prepaid solutions

A traditional approach

- Setup some MiTM component
  - SIP Back-2-Back UA + RTP proxy = Session Border Controller
  - Just a single RTP proxy
    - With in-kernel processing (using netfilter, which is fast but feature-poor)
    - With processing in userspace (somewhat slow, but feature-rich)
- Rely on STUN/TURN/ICE which WON'T work reliably (compare Google Talk with Skype being behind the NAT)

Enter RTPproxy

- Simple (somewhat outdated) control protocol.
- Userspace RTP processing.
- Written in plain portable C (fast in terms of CPU usage per client).
- Reliable and proven.

Erlrtpproxy

- Userspace RTP/RTCP processing
- Written in Erlang (easily extensible)
- More than just a dumb proxy
  - Transcoding
  - Music-on-Hold and RTP injection
  - HTTP server for stats and fine tuning
  - SRTP/ZRTP using Erlang crypto library (w.i.p.)
  - RADIUS notifications
  - Events logging via syslog

What about performance?

- Somewhat slow (~10-15% slower) in terms of CPU per Client (it does more and it still not well optimized).
- A way too better in terms of scalability
  - No command reply penalty due to number of clients.
  - No additional latency after a few hundred of clients (“few hundred” is a practical limit for RTPproxy).
  - Faster replies (~ 10 times faster than RTPproxy)

Conclusion (a techie PoV)

- Just rewrite in Erlang and you'll get linear scalability for free.
- If you do “just rewrite in Erlang” you'll probably loose some CPU cycles. Ask Max Lapshin about possible optimizations (next talk).
- Much smaller and cleaner codebase (especially with regards to protocol parsing)
- Linear and predictable resource requirements – CPU, memory, NIC

Conclusion (an ISV view)

- No matter what your customer wants – you can implement it blazingly fast.
- Opensourcing was a good idea – I've got a lots of bugreports, use cases, and random ideas.
- Reliable and rock-solid – I rebooted it twice after the installation.
  - [petro@mediapro ~]$ uptime
  - 15:53:18 up 328 days, 16:53, 1 user, load average: 0.47, 0.49, 0.54

Links

- http://www.erlang.org/
- http://rtpproxy.org/
- http://mediaproxy.ag-projects.com/
- http://www.2p.cz/en/netfilter_rtp_proxy
- https://github.com/lemenkov/erlrtpproxy

Questions?

Contact:

lemenkov@gmail.com