Michał Ptaszek
@michalptaszek
michal.ptaszek@erlang-solutions.com

Erlang Solutions Ltd.

Refactoring and testing ejabberd XMPP server

ejabberd: what is it?

- XMPP application server, written in Erlang
- scalability, fault tolerance and performance included!
- extensible and modular
- used widely in the field

ejabberd: users

- LiveJournal
- Facebook
- Tuenti
- Nokia Ovi
- ooVoo
- nk.pl
- Nimbuzz
- BBC

motivation: why to bother?

- experience: tens of projects already done
  - changing the same things over and over again
- better product for everyone
- great fun
- good karma

ejabberd: the good

- works great out of the box
  - supports most the the features we dream of
- scales extremely well for the needs of 90% of companies in the world
- relatively small codebase to deal with
  - 45k LOC
  - excluding Pub/Sub
- so many XEPs supported!

ejabberd: the bad

- problems with large clusters
- strings! strings! strings everywhere!
- backward compatibility with ancient R10
  - avoiding new Erlang/OTP goodies
    - binaries, re, NIFs
- limited visibility in what is going on inside of the node

ejabberd: the ugly

- sometimes not so efficient implementation

```
length(ets:tab2list(session))

vs.

ets:info(session, size)
```

- no test suites

ejabberd: the new beginning

- git@github.com:esl/ejabberd.git
- no automatic tests == manual testing and intuition
- at first - non-functional changes
  - core of the system
  - millions of orthogonal parameters
  - hundreds of use cases
- regression tests
- stress tests

escalus: regression tests

- git@github.com:esl/escalus.git
  - Krzysztof Goj
- Common Test-powered framework for convenient testing XMPP servers
- story-based test scenarios
- layer of abstraction on top of exmpp library
- Nagios healthchecks

escalus: regression tests

- automated testing
  - registering/unregistering users
  - making friends
  - convenient helpers
- white- and black-box approach
  - testing any XMPP-compatible server
  - Erlang RPC calls
- Common Test reports

escalus: presence test

```
escalus:story(Config, [1],
  fun(Alice) ->
    escalus_client:send(Alice,
       escalus_stanza:presence(available)),

  Stanza = escalus_client:wait_for_stanza(Alice),
  escalus_assert:is_presence_stanza(Stanza)
end).
```

escalus: message test

```
escalus:story(Config, [1, 1],
  fun(Alice, Bob) ->
    escalus_client:send(Alice,
       escalus_stanza:chat_to(Bob, "Hi!")),

  Stanza = escalus_client:wait_for_stanza(Bob),
  escalus_assert:is_chat_message("Hi!", Stanza)
end).
```

escalus: ejabberd test coverage

- covered core of the system
  - registration + login
  - presence + roster handling
  - privacy controls
- TDD

ejabberd: -ifications

- team consisting of Aleksandra Lipiec, Radosław Szymczyszyn and Michał Ptaszek
- OTPification
- rebarification
  - available in ejd_otp_clean branch
- binarification
  - available in ejd_binaries branch
- SNMPification
  - available in mod_snmp branch

ejabberd: OTPification

- starting Erlang applications from the code
- detached processes
  - random string generator
  - stringprep
- single, massive application
- directory structure
- release unfriendly
  - hot code upgrade!

ejabberd: rebarification

- rebar – most popular build tool for Erlang applications (Dizzy!)
- in Erlang, for Erlang
- goodbye autoconf
- pluggability in other projects
  - dependencies
- integration with agner
- release handling

ejabberd: binarification

- internal XML representation

```
xmltype() :: xmlelement() | xmlcdata()
xmlelement() :: {xmlelement,
                 Name :: string(),
                 Attrs :: [{string(), string()],
                 Body :: [xmltype()]}
xmlcdata() :: {xmlcdata, iolist()}
```

- internal JID representation

ejabberd: binarification

- strings as syntactic sugar in Erlang
- memory overhead of strings
  - strings: 8B for each character
  - binaries: 4B for internal structure + 1B for each character
- faster pattern matching
- reference counting for binaries larger than 64B
- binaries accepted by most of the stdlib modules
- consistent way of handing the data in the code

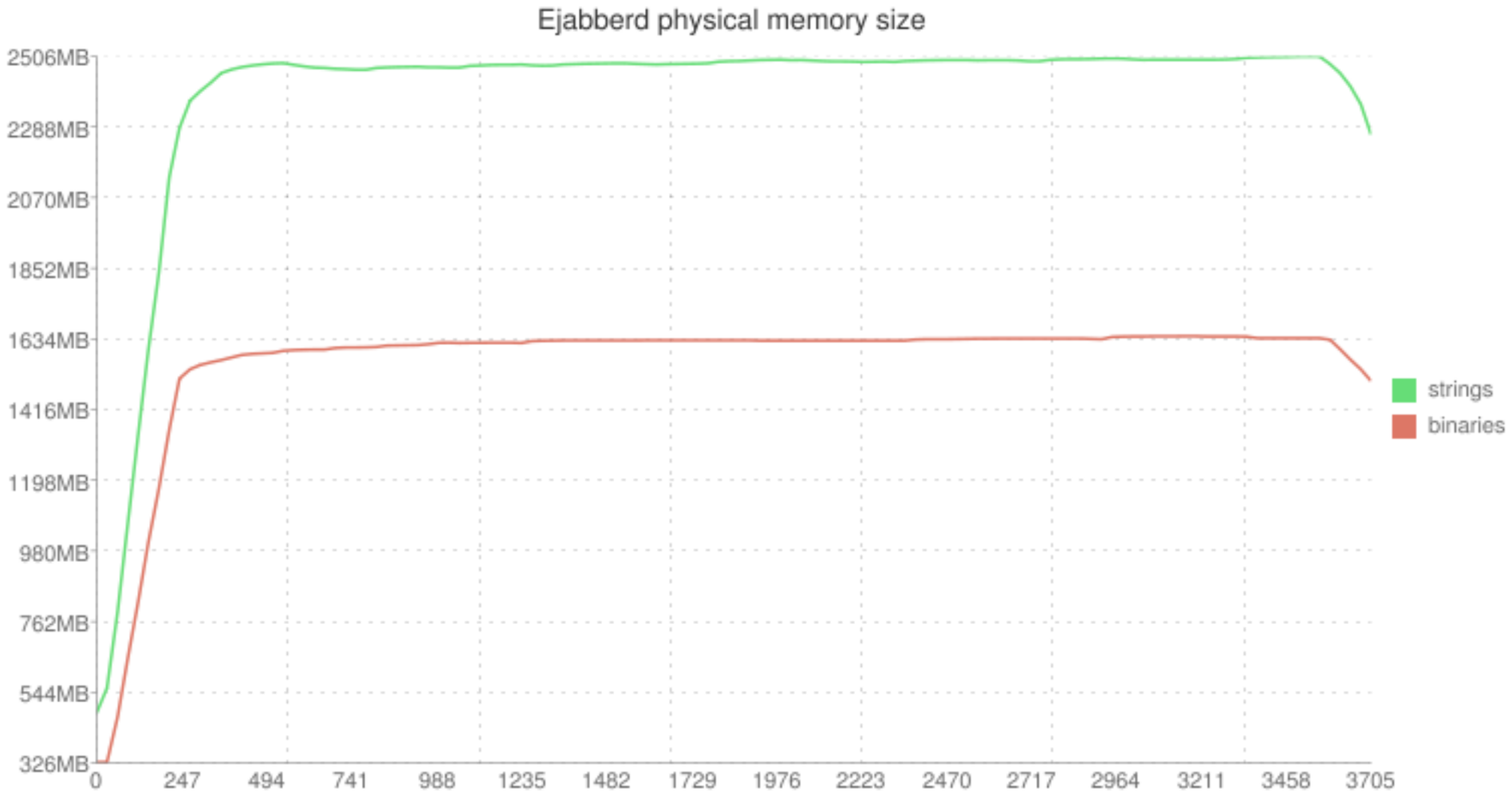
ejabberd: binarification

- core of the system refactored
  - ejabberd_c2s
  - ejabberd_sm
  - ejabberd_auth
- binarified version support for RFC6120 and RFC6121
- PoC for now
  - more modules will be supported as time goes by

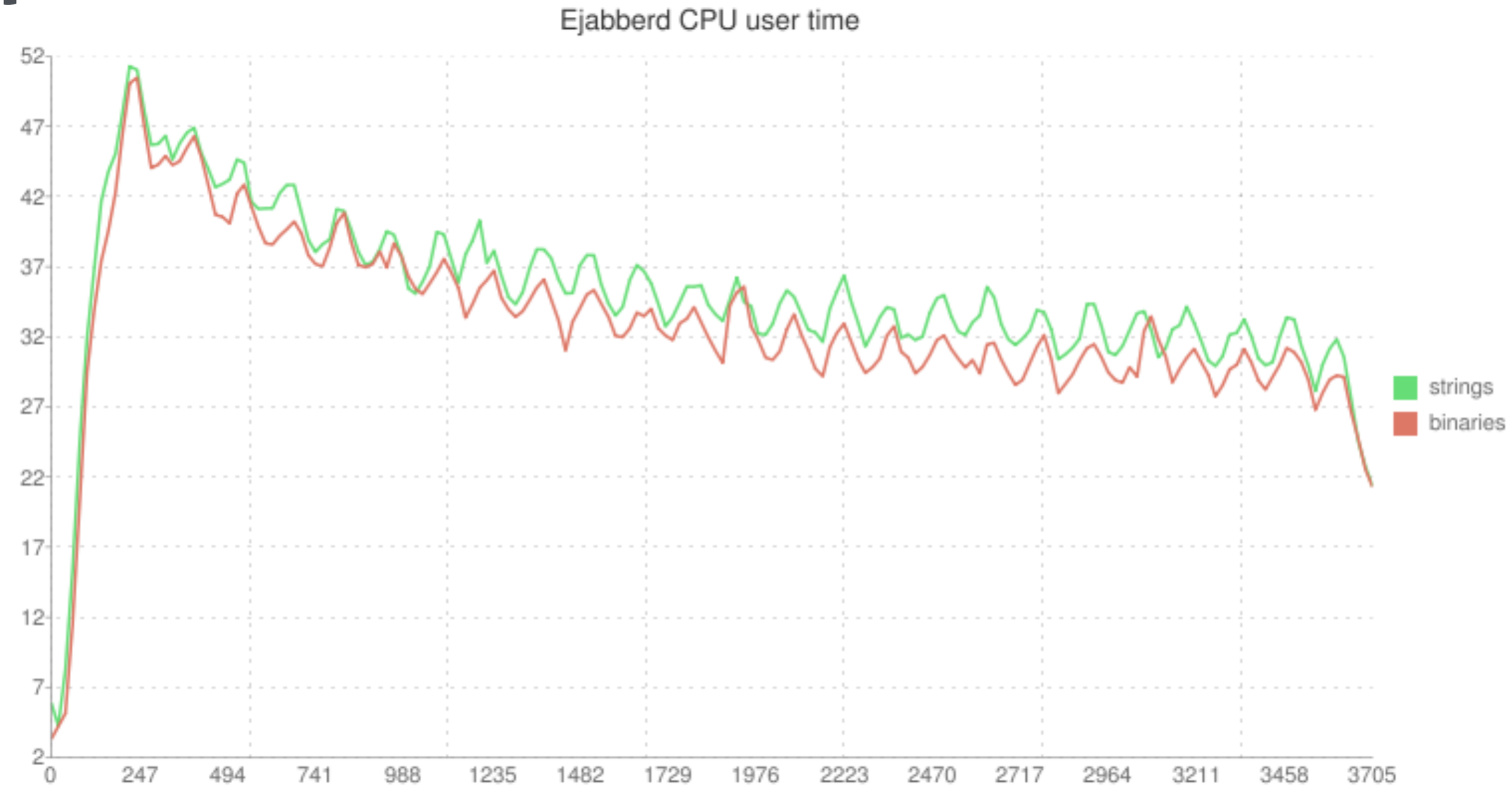
ejabberd: binarification

online users	20 000
roster size	5
message sending frequency	1/10s
frequency of presence updates	1/2min

ejabberd: binarification

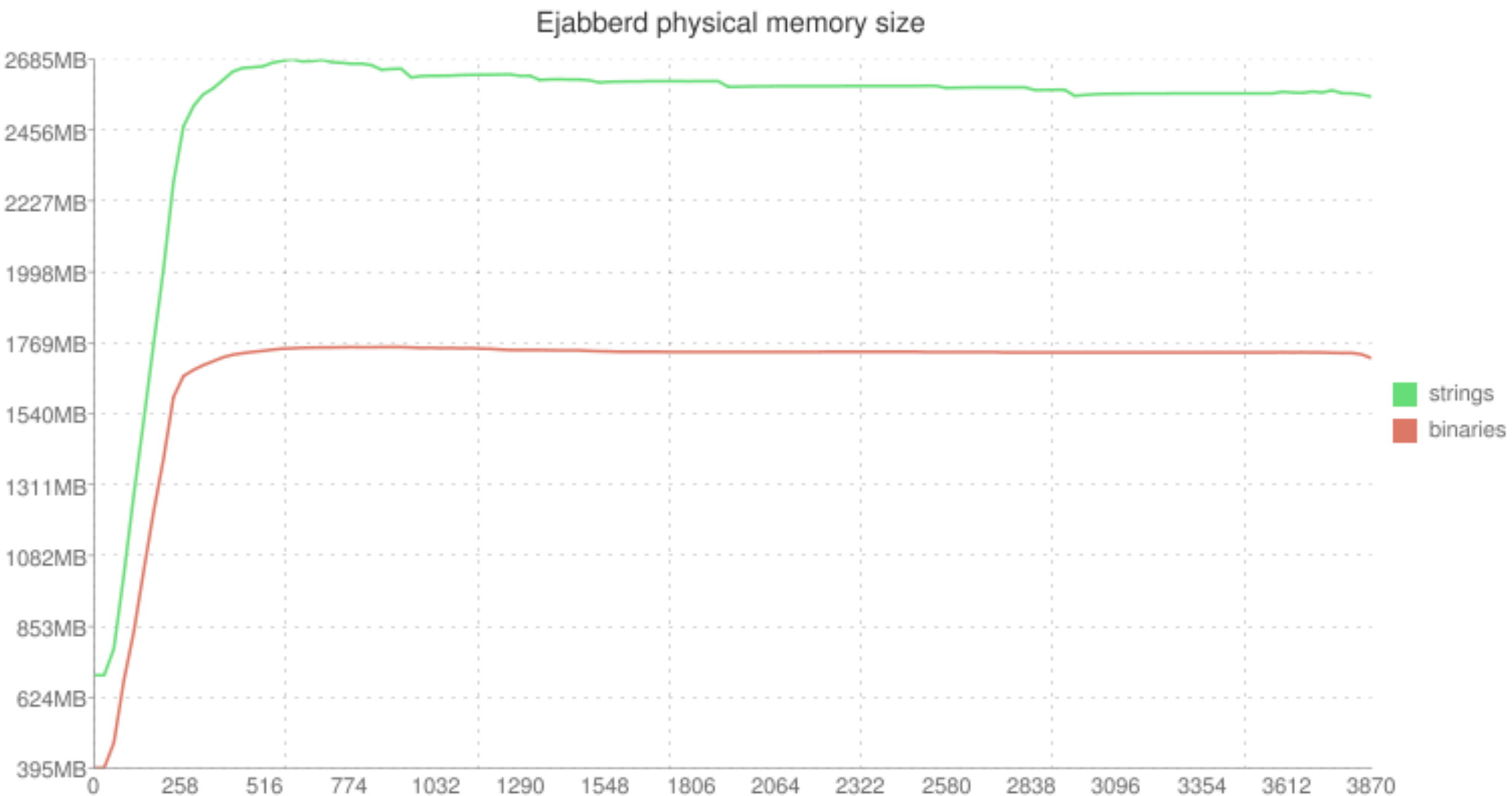


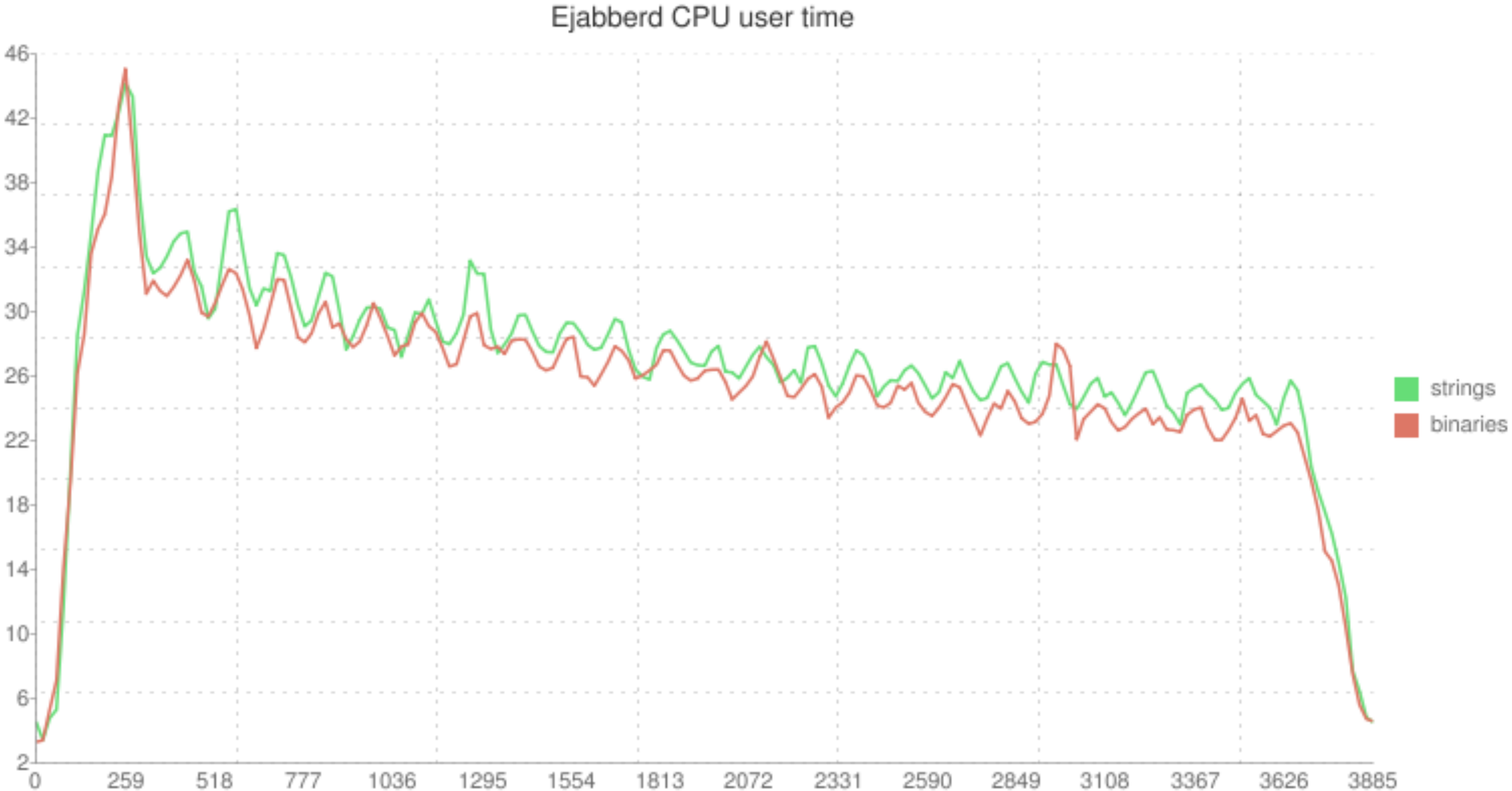
ejabberd: binarification



ejabberd: binarification

online users	20 000
roster size	5
privacy list size	20
privacy list number	2
message sending frequency	1/10s
frequency of presence updates	1/2min





ejabberd: binarification

- Mnesia memory usage (120k users)

	strings	binaries
passwd	60 MB	21 MB
roster	800 MB	301 MB
privacy	817 MB	423 MB

ejabberd: SNMPification

- SNMP – Simple Network Management Protocol
- standard, widely used
- compatible with most of the monitoring tools, such as Nagios, Zabbix, Cacti
- SNMP application included in the Erlang/OTP release

ejabberd: SNMPification

- benefits:
  - increased visibility
  - monitoring trends, collecting statistics
  - alarming
  - forensics
- problems:
  - scalability
  - additional overhead to the system

ejabberd: SNMPification

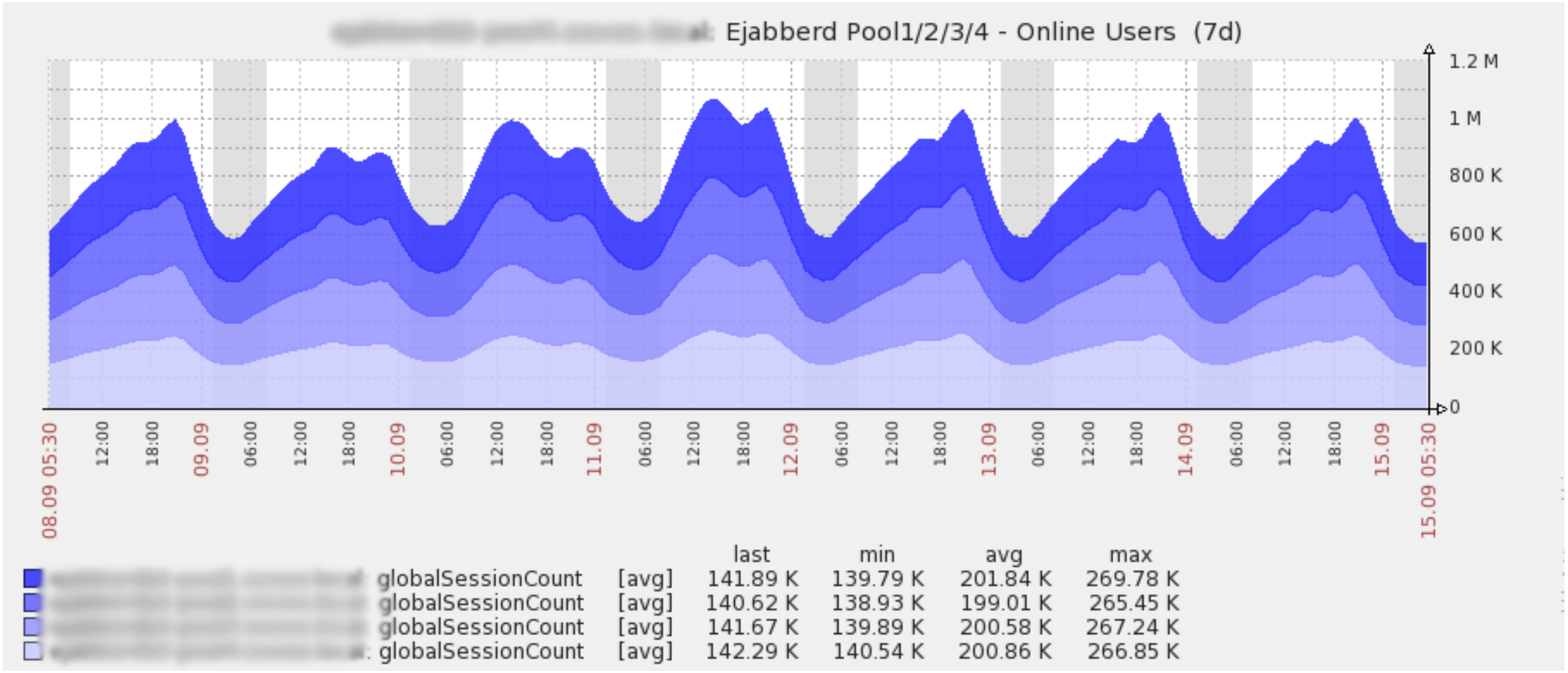
- 70 counters implemented
  - logins/logouts
  - messages, presences, IQs sent/received
  - erroneous situations: timeouts, IQ errors, privacy violations
  - accumulative and sliding window approach
- adding new counters childly easy
- thinking about traps

ejabberd: SNMPification

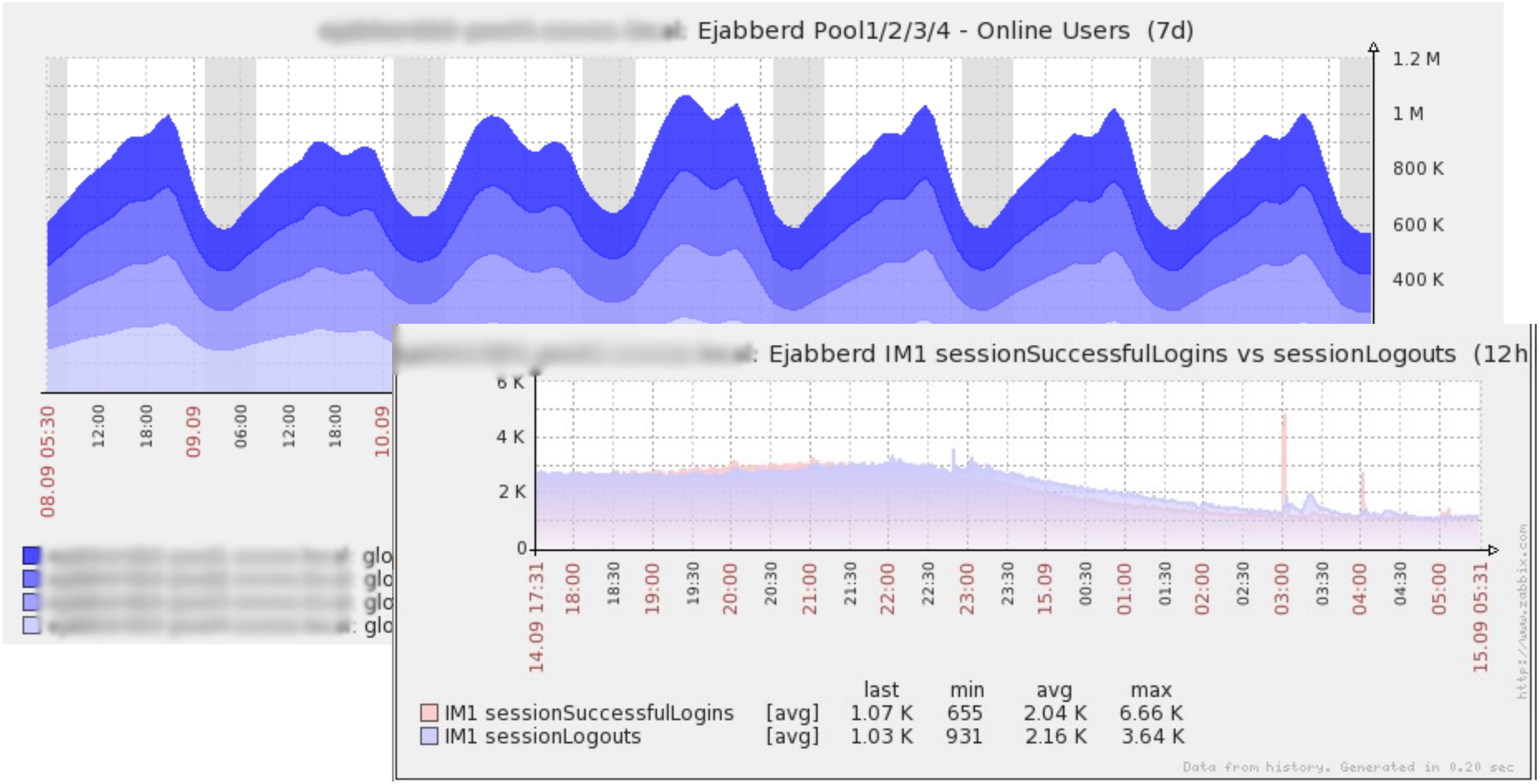
- per-node statistics
  - no global state required
  - no central point of synchronization
- based on ETS tables and update_counter operations
  - atomicity
  - parallel access
- available as ejabberd module: mod_snmp
- counters can be enabled on module-level basis

ejabberd: SNMPification

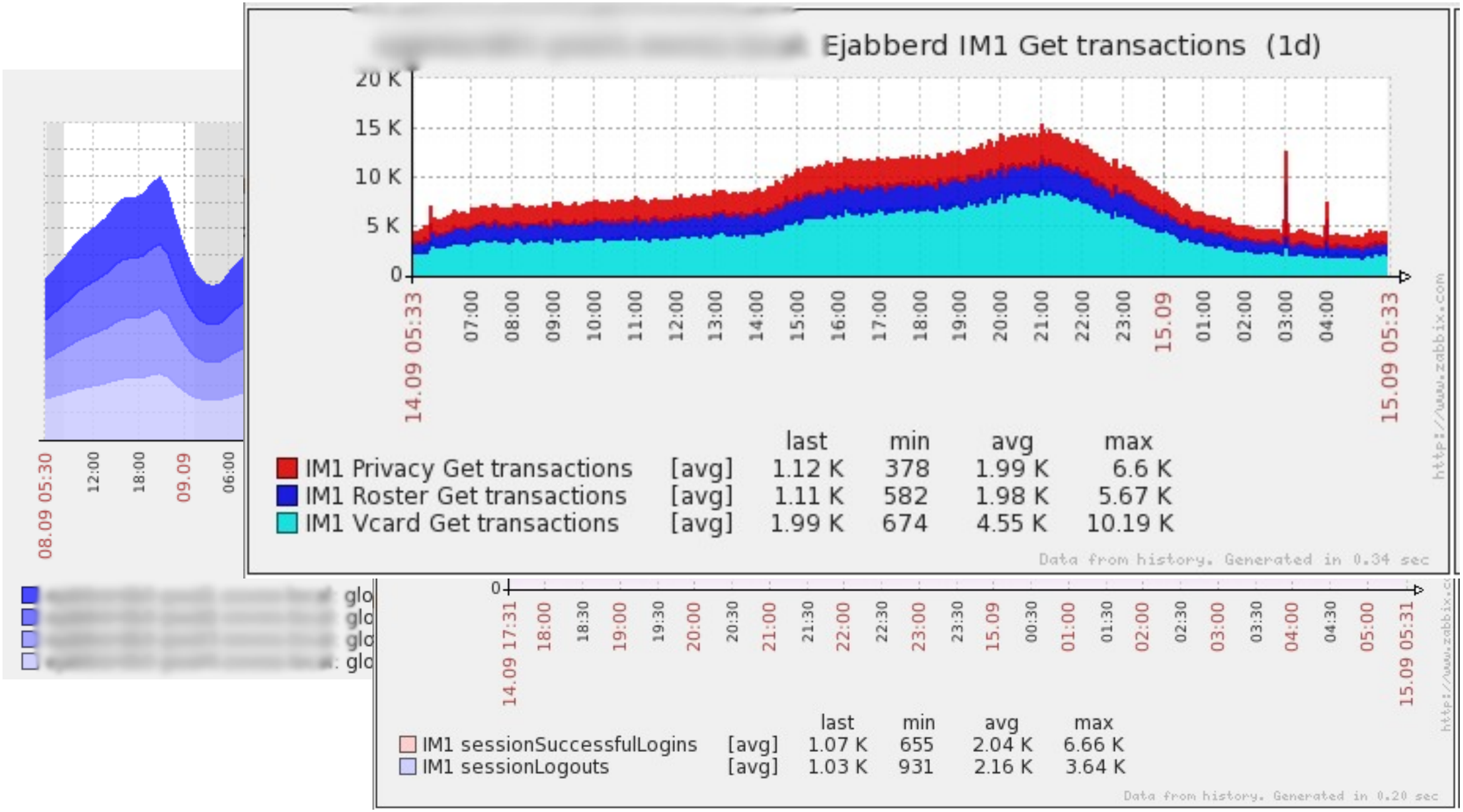
ejabberd: SNMPification



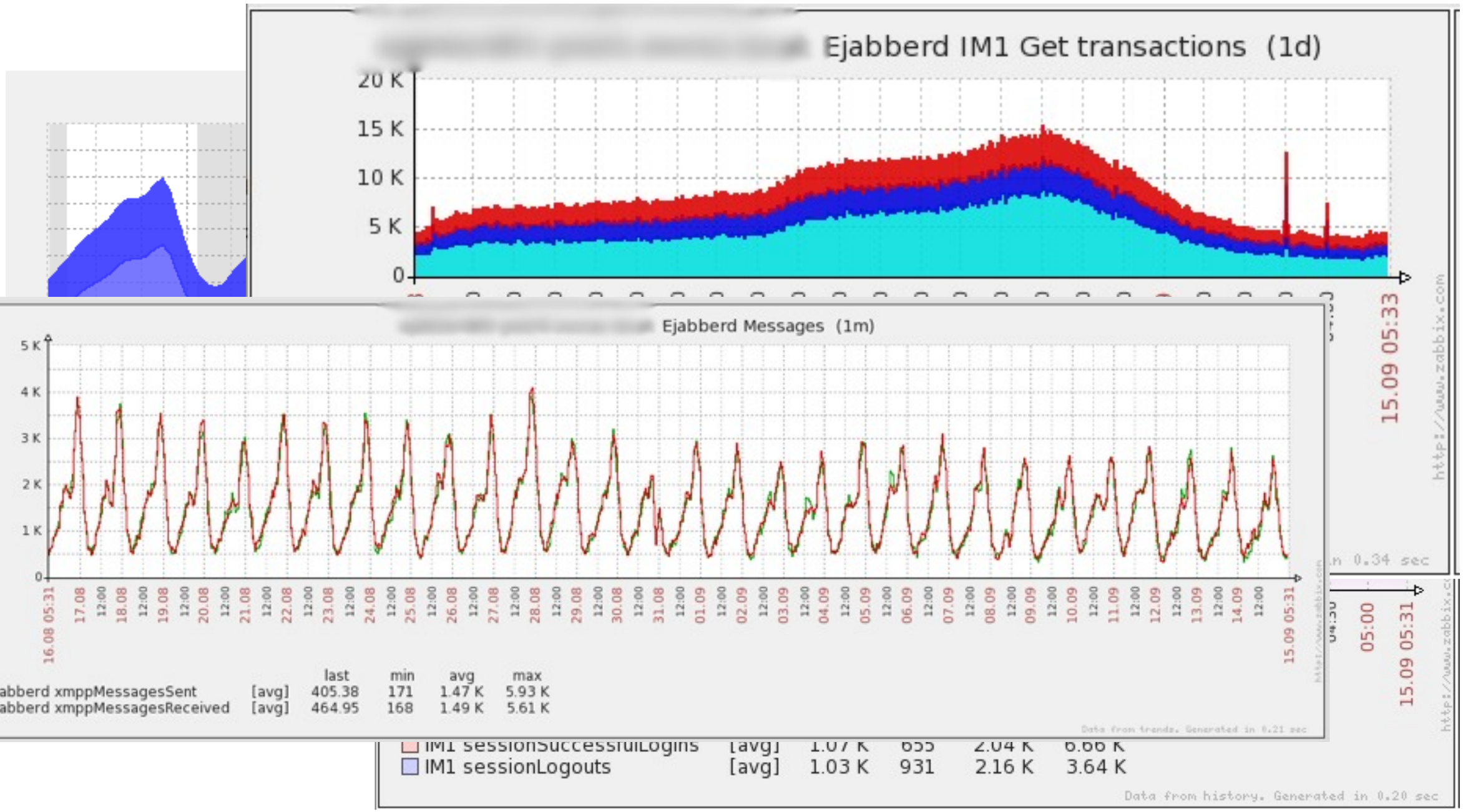
ejabberd: SNMPification



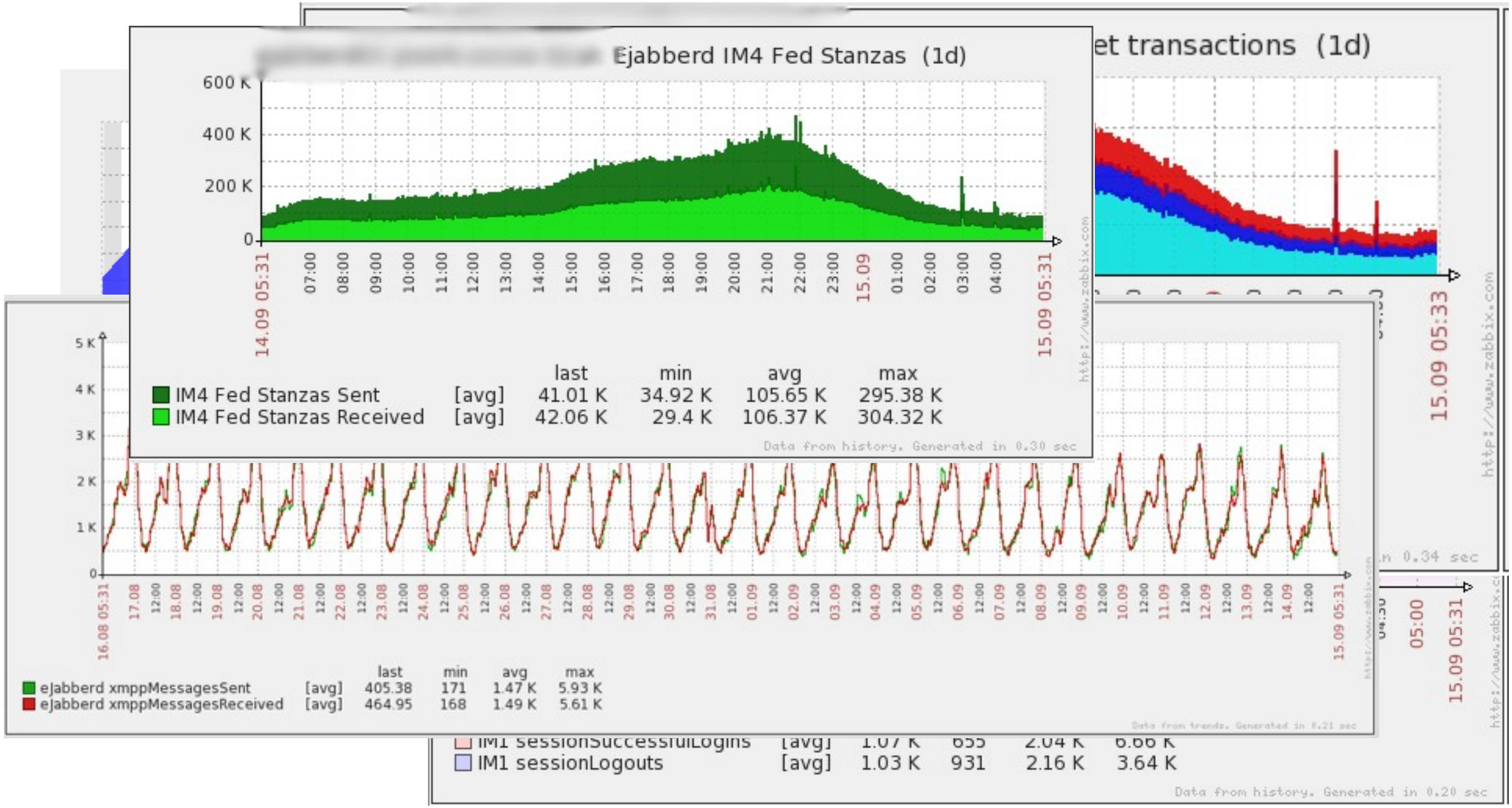
ejabberd: SNMPification



ejabberd: SNMPification



ejabberd: SNMPification



ejabberd: future

- waiting for 2.1.10 and 3.0 releases
- listening to feedback
- reimplementing more modules
- adding new features
- optimizing the code
  - sometimes loosing the generality
  - first goal: 500k CCU per box

ejabberd: OMP

- Open Messaging Platform
- the fastest way to get ejabberd up and running in 15 minutes time on your server
- pre-compiled packages
- including ESL ejabberd fork
- perfect for testing ejabberd and its capabilities

ejabberd: summary

- www.github.com/esl/ejabberd
- OTP + rebar + binaries + SNMP power
  - 40% less memory than 2.1.8 vanilla branch
- enjoy!