Ease the rsyslog admin's life...

Rainer Gerhards

Never touch a running system

- Of course not, but sometimes you need to
- And if you need to, DON'T stick to outdated versions!
- Many distros still ship v5, or even older
  - Missing features (e.g. wildcards in imfile, json)
  - Hard to get right config language
  - Bad performance
  - Long-solved bugs
  - Very limited support by the rsyslog community

Make your life much easier: Upgrade to current (v8.8 now)

- Of course, only if you need to touch the system
- Config will continue to work
- Adiscon has made packages available for support customers, but everyone is free to use them
  - RHEL/CentOS
  - Debian
  - Ubuntu

rsyslog – Mozilla Firefox

sentatio... | rsyslog Ver... | Vorträge – ... | Inbox (2) - r... | Recibidos (... | Deskpro Tech ... | Deskpro Tech ... | Adiscon Gm... | Veranstaltu... | rsyslog

www.rsyslog.com

guug ffg 2015

RSYSLOG

THE ROCKET-FAST SYSTEM FOR LOG PROCESSING

Search

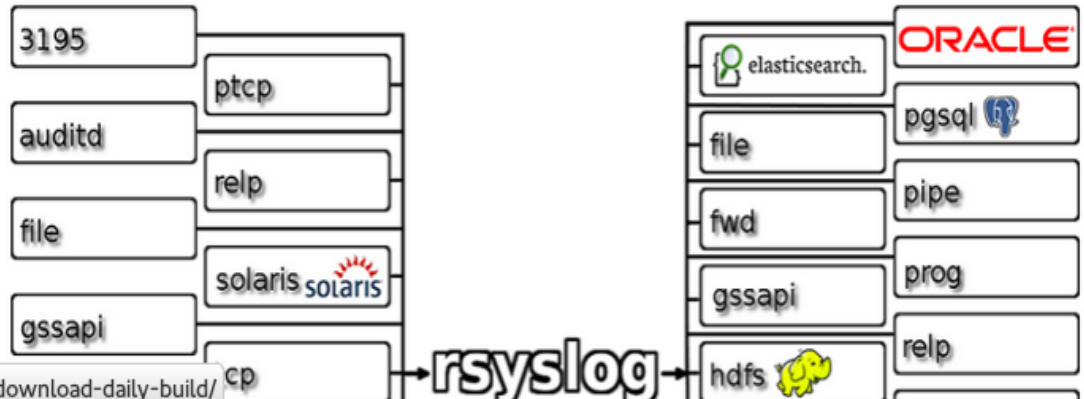
HOME | PROJECT | HELP | TOOLS | PROFESSIONAL SER | WIN S AGENT

HOME

RSYSLOG is the **r**ocket-fast **sys**tem for **log** processing.

It offers high-performance, great security features and a modular design. While it started as a regular syslogd, rsyslog has evolved into a kind of swiss army knife of logging, being able to accept inputs from a wide variety of sources, transform them, and output to the results to diverse destinations.

RSYSLOG can deliver over one million messages per second to local destinations when limited processing is applied (based on v7, December 2013). Even with remote destinations and more elaborate processing the performance is usually considered "stunning".



slog.com/downloads/download-daily-build/

Current Version

8.8.0 [doc] [download]

next: 8.9.0, 2015-04-07

daily build

[packages and older versions]

Top News

rsyslog daily builds and

January 26, 2015

The past days, we have worked on making rsyslog daily builds and tarballs a reality. We hope this wi ...

rsyslog's new release

December 2, 2014

While we are at it: rsyslog version numbering

- Traditionally [major].[minor].[increment]
- We now do 6-weekly releases and increment the minor version
- Automatic testing has much improved, so all numbered releases are stable
- Devel version available via git master branch and as daily tarball/package
- Leads to much earlier availability of new features

Getting Help

- Community support
  - latest stable and devel (daily build)
  - Mailing list (suggested) or web forum
  - Report confirmed issues and feature requests to github issue tracker
- Professional Support
  - via Adiscon, rsyslog's main sponsor (90%+)
  - Any version supported (but we still suggest going current)
  - Guaranteed support with NDA no problem
  - Includes development & consulting hours

The new configuration language

- Originally (~2007) we thought we would just need few additional statements
- As it turned out, they became more and more
- What made things even worse is that they have strict order requirements, not always intuitive
- Very hard to work with, very easy to get wrong
- So we had to settle for something better

This is what drove **me** crazy:

```
# The "regular" logging...
mail.*  /var/log/mail.log

$RuleSet remote10514
*.*     /var/log/remote10514

$RuleSet remote10516
mail.*  /var/log/mail10515
&       ~
*.*     /var/log/remote10515


$InputTCPServerBindRuleset remote10514
$InputTCPServerRun 10514

$InputTCPServerBindRuleset remote10515
$InputTCPServerRun 10515
```

- took **me** a while to figure correct order
- change anything, nothing will work :-)
- You need to be **very** brave if you add things like ruleset queues...

This is new style:

```
# The "regular" logging...
mail.*  /var/log/mail.log

ruleset(name="remote10514") {
  action(type="omfile" file="/var/log/remote10514")
}

ruleset(name="remote10515") {
  If prifilt("mail.*") then
      action(type="omfile" file="/var/log/mail10515")
  else
     action(type="omfile" file="/var/log/remote10515")
}

input(type="imtcp" port="10514" ruleset="remote10514")
input(type="imtcp" port="10515" ruleset="remote10515")
```

Or how about this one? [from rsyslog mailing list]

```
$ActionQueueType LinkedList
$ActionQueueSize 100000
$ActionQueueDiscardMark 95000
$ActionQueueDiscardSeverity 0
$ActionQueueTimeoutEnqueue 0
$ActionQueueDequeueSlowdown 1000
$ActionQueueWorkerThreads 2
$ActionQueueDequeueBatchSize 128
$ActionResumeRetryCount -1

local4.*  /var/log/ldap/ldap.log
local4.*  @@somehost
```

- It adds a queue…
- … in front of the file write action …
- … but not when forwarding to the remote host!

In new style this would have been obvious

```
local4.* {
   action(type="omfile" file="/var/log/ldap/ldap.log"
          queue.type="LinkedList" queue.size="100000"
          queue.discardMark="95000" queue.discardSeverity="0"
          queue.timeoutEnqueue="0" queue.dequeueSlowdown="0"
          queue.workerThreads="2" queue.dequeueBatchSize="128"
          action.resumeRetryCount="-1")
  action(type="omfwd" protocol="tcp" target="somehost")
}
```

You can mix old and new style

```
local4.* {
   /var/log/ldap/ldap.log
   action(type="omfwd" protocol="tcp" target="somehost"
          queue.type="LinkedList" queue.size="100000"
          queue.discardMark="95000" queue.discardSeverity="0"
          queue.timeoutEnqueue="0" queue.dequeueSlowdown="0"
          queue.workerThreads="2" queue.dequeueBatchSize="128"
          action.resumeRetryCount="-1")
}
```

Suggestion

- Old-style config is fine for simple things
  - Simple is “mail.info /var/log/mail.log”
  - Anything that requires parameters is NOT simple
  - Rulesets (as on last side) is questionable
- Use new-Style for everything more complicated
  - Note that some statements do not yet have new style equivalents → $IncludeConfig
  - Old configs still work, no need to migrate just to upgrade

Writing plugins

- Traditionally, plugins
  - Are written in C
  - Macros hide interface plumbing
  - Fairly easy to write for the C-literate
  - Still perceived as “complicated”
- V8 goal
  - Enable everyone to write plugins (sysadmins!)
  - Support any language (Python, Perl, …)
  - Ability to execute security-sensitive plugin outside of rsyslog security context

Writing plugins

- Traditionally, plugins
  - Are written in C
  - Macros hide interface plumbing
  - Fairly easy to write for the C-literate
  - Still perceived as “complicated”
- V8 goal
  - Enable everyone to write plugins (sysadmins!)
  - Support any language (Python, Perl, …)
  - Ability to execute security-sensitive plugin outside of rsyslog security context

Types of Plugins

- Output (actions)
  - Deliver message to some destination system, e.g. file, ElasticSearch, MongoDB, Solr, …
  - **Any language supported in v8.2.0+**
- Message Modification Plugins (Modules)
  - Permit on-the-fly modification of message content (e.g. anonymization, credit card removal)
  - **Any language supported in v8.3.0+**
- Input
  - Accept input messages
  - *Currently on hold – syslog(3) is fine…*

Ultra-quick tutorial for output plugins...

- Choose any language you like
- Implement the pseudocode below
  - Messages arrive via stdin, one message per line
  - Read from stdin until EOF
  - Process each message read as you like
  - Terminate when EOF is reached
- **That's it!**

```
While not EOF(stdin) do {
    Read msg from stdin
    Process msg
}
```

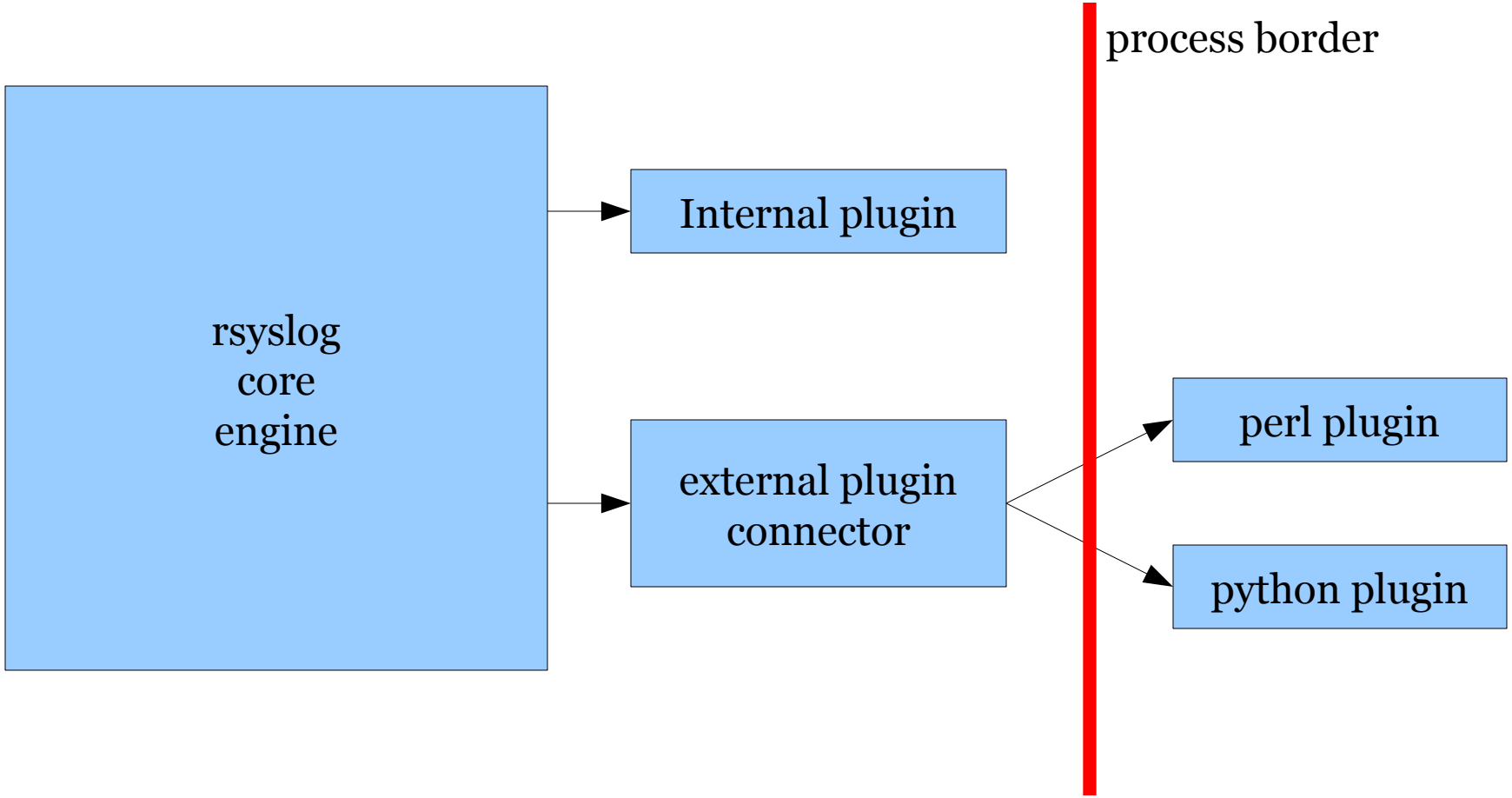
Make rsyslog call plugin

- Regular filtering applies (as with any action)
- You can specify message format via a template
- Use omprog for the call

```
module(load="omprog") # needed only once in config!

if $rawmsg contains "sometrigger" then
   action(type="omprog"
          binary="/path/to/your/plugin")
```

A bit more detail: Interface Overview



Interface Details: communication

- uses pipes
- stdin
  - one message per line
  - format can be customized via rsyslog templates
  - multi-line messags via JSON
- stdout/stderr
  - Must NOT be written in initial version
  - Message modification module returns changes via stdout [later also error state via stderr]
- Template specifies input format (with JSON recommended for more complex cases)

Interface Details: Threading

- Do NOT care about threading
- Write app according to single-thread paradigm
- rsyslog will spawn multiple instances of your plugin if there is need to do so
  - Happens based on config in busy cases
  - Works well in most cases (e.g. http connects)
  - Can be disabled if necessary
  - **If your program can run in multiple ter-minal sessions concurrently, it can also be run as multiple rsyslog action instances.**

Startup & Termination

- rsyslog will startup the plugin automatically
- Plugin needs to read stdin until EOF
- Do NOT terminate before EOF is reached
- On EOF, cleanup and terminate
- If the plugin dies, rsyslog restarts a new instance

Skeletons

- The rsyslog project provides sample plugin skeletons
- Available in ./plugins/external/skeletons
- These contain
  - some plumbing
  - often a kind of abstraction layer to make writing plugins even easier
  - often performance-enhancement features
- Can simply be copied to create your own plugins, don't care about the (minimal) plumbing!

External Plugins...

- Let's have a look at actual code...

Call to Action

- If you need to send logs to a destination that is not yet supported, you can quickly write an external plugin – in any language you know!
- Writing rsyslog plugins is easy
  - If there is already a skeleton for your language, copy it and add your app-specific code
  - If not … no problem, the interface is dumb easy

If you can write a script that reads stdin and does something useful with it, you can also write a rsyslog plugin!

impstats statistics module

- Provides insight into running instance
  - Queue sizes
  - File cache behavior
  - Messages processed
  - … and much more
- Reports
  - Periodically
  - To either regular syslog stream or local file
- Extremely useful for tuning and troubleshooting (even health monitoring…)

Activating impstats

```
module(load="impstats"
       interval="600"
       severity="7"
       log.syslog="off"
       /* need to turn log stream logging off! */
       log.file="/path/to/local/stats.log")
```

A sample pstats file…

imudp(*:514): submitted=2327203
imptcp(*/5514/IPv4): submitted=0

main Q[DA]: size=0 enqueued=0 full=0 discarded.full=0 discarded.nf=0 maxqsize=0
main Q: size=67 enqueued=283 full=0 discarded.full=0 discarded.nf=0 maxqsize=67

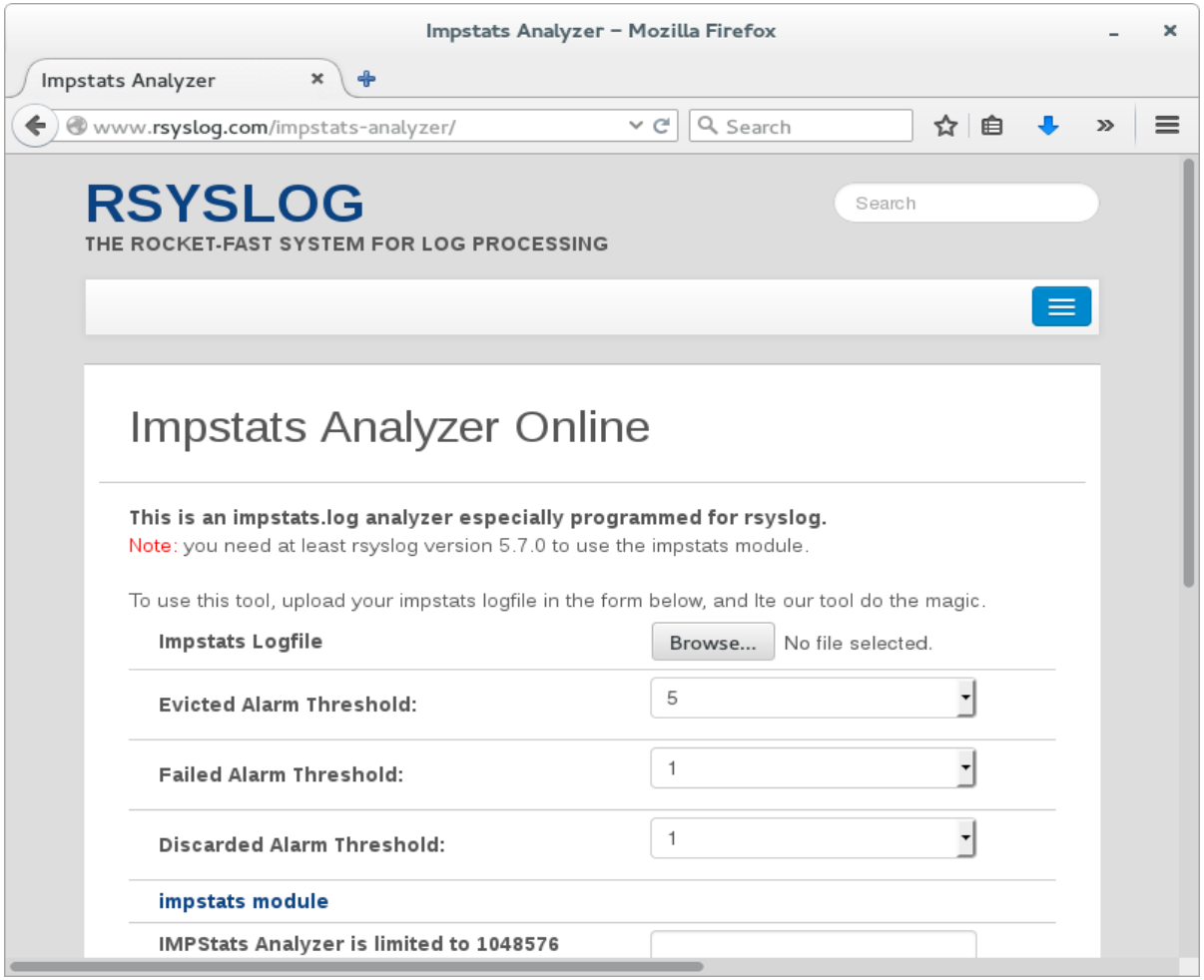
imuxsock: submitted=7 ratelimit.discarded=0 ratelimit.numratelimiters=3

remote[DA]: size=1040 enqueued=1041 full=0 discarded.full=0 discarded.nf=0 maxqsize=1040
remote: size=7190 enqueued=2922166 full=0 discarded.full=0 discarded.nf=0 maxqsize=8176

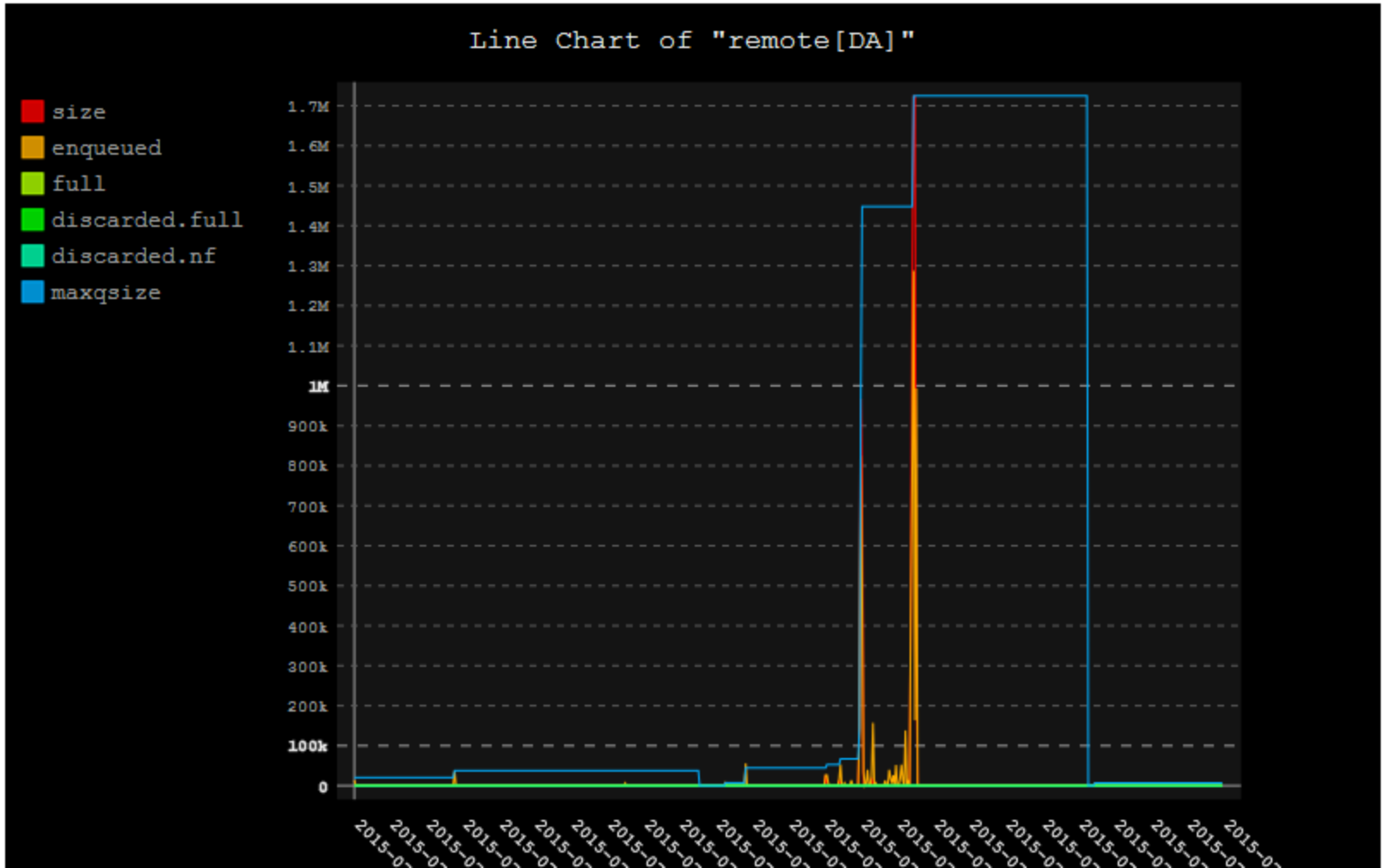
dynafile cache file_Watchdog: requests=2913980 level0=2913979 missed=1 evicted=0 maxused=1

Web pstats analyzer

- Great to gather some quick insight
- Available from the www.rsyslog.com homepage (under tools)



Web pstats analyzer: sample graph



The rsyslog doc project

- The doc just sucks… but a bit less now…
- In 2014 spawned a new project to create better one: https://github.com/rsyslog/rsyslog-doc
- Initiated by James Boylan (a sysadmin)
- Please help
  - Complain ;-)
  - open issues
  - Write some doc…
- We are especially interested to learn what is hard for beginners!

Log normalization

- PoC (liblognorm) available for some time now, already has good results
- Now working on the “final version”
  - Retain and improve realtime-capability
  - Better rule base format
  - Easier to use
- Will contain “simple log structure analyzer” (slsa)
  - Statistical structure mining
  - Goal: turn undetected messages into rules fast
  - Also promising for anonymization

Call for log samples

- In order to move forward with the project, I am in deep need for actual sample logs
- Please contribute
  - Public would be great
  - Under NDA also possible and appreciated!
- There is a bonus for you
  - I help with the creation of rulebases for your **current** environment (using liblognorm 1.x)
  - In the future, you get a tool that semi-automatically creates new rules for you (plus an even better normalization algo!)

Questions?

rgerhards@adiscon.com

Please contribute Logs!!!