CANONICAL

Technical White Paper

Centralised logging with rsyslog

By Peter Matulis – September 2009

www.canonical.com

Overview

The management of multiple systems requires the setup of tools to control the servers behaviour in real time and post analysis. Moreover, regulations and best practices often require the IT department to maintain an accurate log of all events happening in their systems in order to allow for later analysis. Performing such analysis on each system is time consuming and is relatively insecure because if a server is compromised, the attacker, having gained root access, will be able to cover its traces by removing the portions of the logs that he wants.

This white paper analyses the tools available in Ubuntu to perform the task of centralised logging, recommends the use of rsyslog and provides the steps needed to configure a set of servers to send their events to a central logging facility.

Table of Contents

Introduction

Every computer system, regardless of its operating system, has a mechanism that records activities taking place within it. Such 'logs' do not normally concern the average end-user but play a crucial role in the life of a system administrator or support analyst when problems arise because errors that occur are included in those activities and are the first stop in attempts at troubleshooting.

In addition, since logging is essentially a record of what is happening, with systems processing company-sensitive or even secret data the subject assumes a whole new dimension. In large organisations, where the number of computer systems can range in the thousands, there is the task of managing such logging data. Geographically diverse branch offices bring another element to the mix. Finally, logs play a vital role when a system has been compromised by an external (or internal) hostile agent.

This white paper also tries to address how a company technically manages the potentially huge volume of logs its computer systems generate.

Other questions deserving of serious consideration but which are not covered by this technical paper are:

- Authorisation (i.e. **who** should have access to such logs?)
- Legally, how far back in the past must a company retain its logs (particularly when managing client data)?

The software that is covered in this document is *rsyslog*. Possible alternatives are the stock Linux/Unix *syslog* system or *syslog-ng*. This paper describes the reasons for the choice of rsyslog in the section *Logging Software* and provide technical caveats and background information in the section *Technical Considerations and Historical Background*.

This paper is not an introduction to the field of system logging. See *Appendix A, "The Ins and Outs of System Logging Using Syslog"* for the basics.

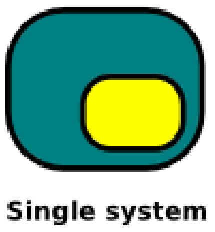
***Note**: at the time of publication, Ubuntu 9.10 (karmic koala) is in alpha and uses rsyslog as its default tool for logging, replacing sysklogd that was the previous default . The analysis performed for this white paper is what triggered this change.*

Logging models

This section surveys several typical architectural models of computer system logging.

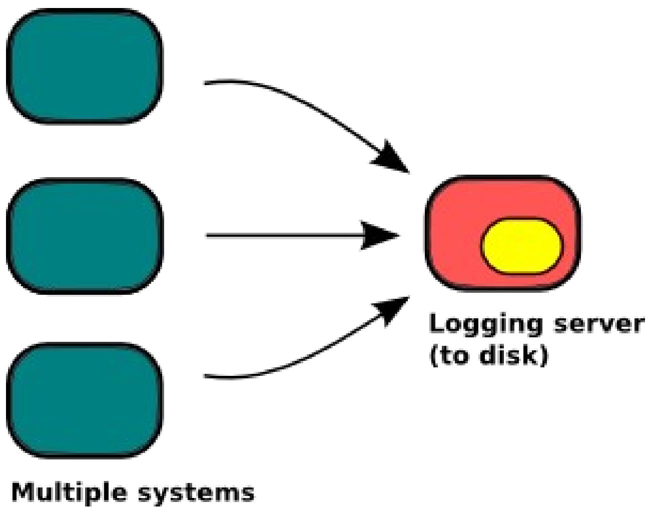
1. Single system (to disk)

Individual computer systems, by default, perform logging. Messages typically get written to the local hard drive but Network Attached Storage (NAS) or Storage Area Network (SAN) are also valid storage options for this model.



2. Multiple systems (to disk)

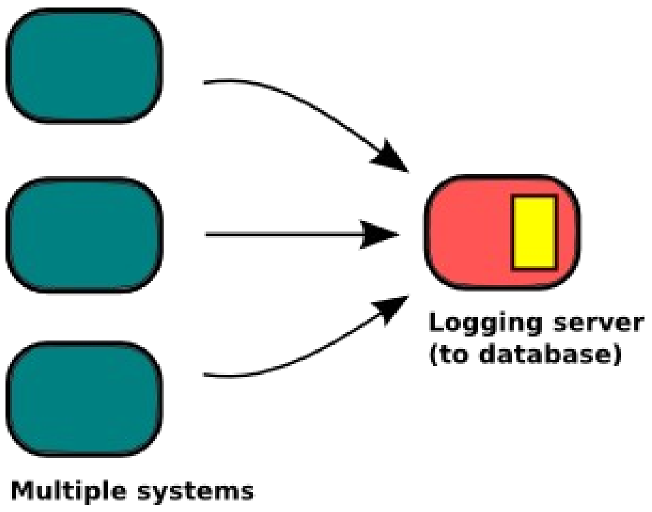
Known as *central logging*, many systems forward their logs over the network to a central logging server. Analogous to the single-system model, on the server-side, messages get written to the local hard drive or to some other available storage.



3. Multiple systems (to database)

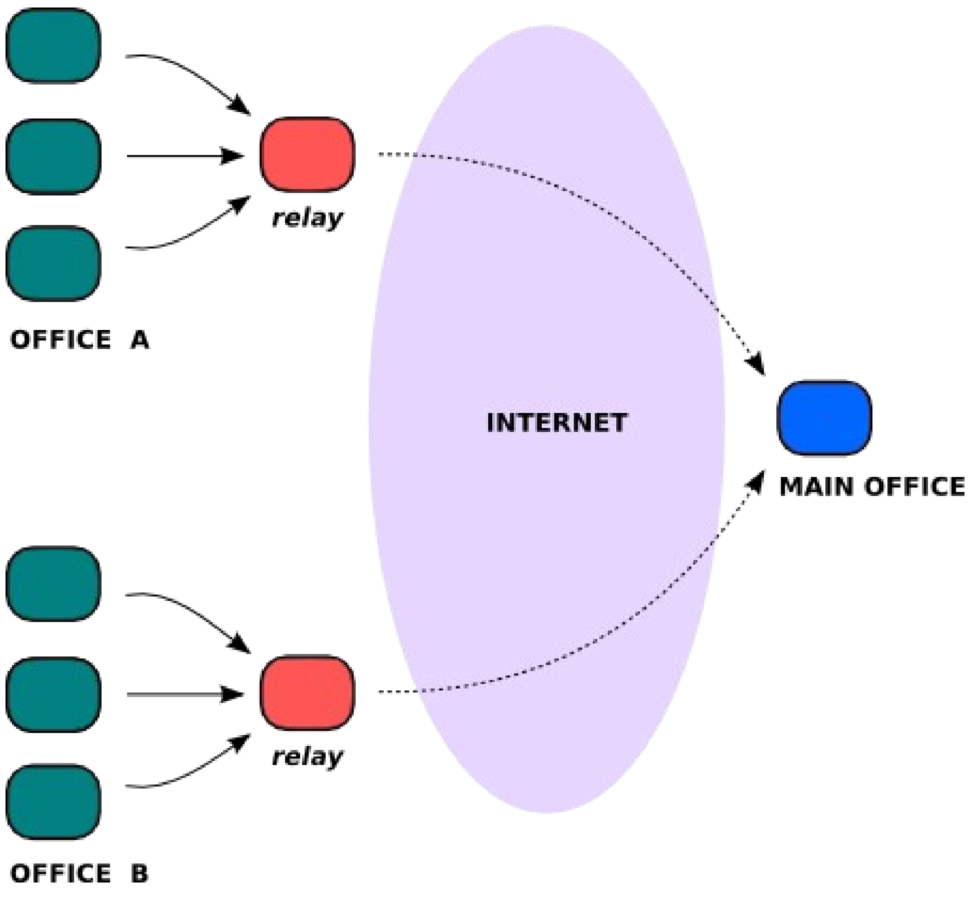
A common option is to have the remote messages stored directly into a database on the server with, possibly, a web-based interface acting as a viewing/query tool.

The database need not reside on the logging server (as shown in the diagram); it can be placed onto a separate system.



4. Branch offices (remote storage)

We continue the logical progression where multiple branch offices are each implementing the #2 or #3 model. Their central logging servers now relay their logs to a second-level central logging architecture (typically residing at the company head office or data centre). The fact that sensitive information is being transported over a non-trusted network (here the internet) is a vital facet that needs to be addressed by your company's security team.



Technical considerations to central logging

Network logging reliability

Traditional Unix syslog uses the UDP protocol. This is unsuitable for central/network logging due to the protocol's lossy/unreliable nature. Alternative software such as syslog-ng and rsyslog include support for the TCP protocol. This is a great improvement but there remains nonetheless a reliability issue even with TCP. Thousands of messages can be lost if the network connection with the logging server breaks as there is no mechanism in TCP that notifies the sender immediately (its send buffer continues to fill up). The rsyslog project is currently developing a truly reliable logging protocol: RELP (Reliable Event Logging Protocol). The usage of RELP however is outside the scope of this white paper however.

Database logging

When logging to a database you need to make sure your database can actually handle the rate of messages (messages per second, mps) that is being directed at it. Rsyslog offers ways to handle this and will be mentioned in the **Advanced Rsyslog Features** section but the database will often present an I/O bottleneck. Multiple servers may need to be used in such an environment.

TLS connections

The TCP listener can currently only listen on a single port. You therefore cannot at this time, use a dual-mode setup (encrypted and unencrypted channels).

Logging software

The *rsyslog* tool was chosen over the more popular *syslog-ng* for the following reasons:

1. **Licensing and software features**
   Syslog-ng is dual-licensed. A commercial product has been forked from the open-source (GPL) project and the more advanced features are found only in the commercial offering. Affected features of import so far are i) native TLS/SSL support (i.e. not using stunnel) and ii) on-disk spooling of messages. It's unknown how these forks will diverge in the future.

2. **Truly reliable message delivery (RELP)**
   Rsyslog is confronting the unreliability of TCP in a logging environment through the development of the RELP protocol whereas syslog-ng is not.

3. **Compliance with IETF regarding reliable TCP transport (RFC 3195)**
   Rsyslog is compliant with the standards regarding reliable TCP transport whereas syslog-ng is not.

4. **Native support for traffic encryption (TLS/SSL)**
   Rsyslog supports TLS natively whereas the GPL fork of syslog-ng does not.

5. **SNMP support**
   Rsyslog supports SNMP traps whereas syslog-ng does not.

6. **BSD-style hostname and program name blocks**
   Rsyslog supports powerful BSD-style hostname and program name blocks for easy multi-host implementations whereas syslog-ng does not.

7. **On-disk message spooling**
   Rsyslog has on-disk file spooling features that are lacking in GPL syslog-ng:
   - on-demand (as needed) spooling
   - independent spool files per log action
   - spool files on multiple disks
   - Process spooled messages during configured timeframes

8. **Include config files**
   Rsyslog has configuration include file support that syslog-ng lacks. This allows one to organize and split one's configuration into multiple files.

9. **Native support for email alerts**
   Rsyslog natively supports the ability to send email alerts based on log message content. Syslog-ng needs to pipe data to an external process.

Getting started with rsyslog

This section covers:

- Installation
- Configuration structure
- Rules/actions
- Timestamps
- Templates
- Properties-based filters
- Expression-based filters
- Queue processing

Installation

Installing rsyslog on Ubuntu is easy:

```
$ sudo aptitude install rsyslog
```

To get extra documentation:

```
$ sudo aptitude install rsyslog-doc
```

Rsyslog is only compatible with sysklogd/klogd when it is run in the specified compatibility mode (option '-c0'). By default, rsyslog runs in a mode of '-c3' (not considered compatibility mode) and displays a warning if this is not the case. See the */etc/default/rsyslog* file.

All configuration, with the exception of the above defaults file, is placed in the */etc/rsyslog.conf* file or in files found under the */etc/rsyslog.d* directory. In this document, configuration examples are coloured like this.

Rsyslog should now be running. It has converted the existing syslog.conf file into its own rsyslog.conf file. A diff of the two files has been included in Appendix B.

In addition, there is no separate kernel log daemon (klogd) running. Kernel logging is incorporated into rsyslog via its **imklog** input plug-in (enabled by default).

Local logging functionality is provided by the **imuxsock** plug-in (also enabled by default).

```
$ModLoad imuxsock
$ModLoad imklog
```

Naturally, any configuration changes require a restart:

```
$ sudo /etc/init.d/rsyslog restart
```

Or if the version of Ubuntu you are running is 8.10 (intrepid) or more recent:

```
$ sudo service rsyslog restart
```

Configuration structure

Configuration files are structured in the following manner:

- Modules
- Global directives
- Filter rules

All modules and global directives need to be specified one per line and must start with a dollar-sign ($). They affect all rules.

Rules/actions

Rules consist traditionally of '**selector action**' (where selector consists of '**facility.priority**'). This method has been retained from regular sysklog because they are effective but also for backward compatibility with sysklog configuration files. However, rsyslog provides other unique and powerful methods of building rules as we'll see.

The facility and priority are defined in RFC 3164. Here is a summary:

Facilities		
Numerical Code	Keyword	Facility
0	kern	Kernel
1	user	Regular user processess
2	mail	Mail system
3	daemon	System daemons
4	auth	Security (authentication and authorisation) related commands
5	syslog	Syslog internal messages
6	lpr	Line printers system
7	news	NNTP subsystem
8	uucp	UUCP subsystem
10	authpriv	Private authorisation messages
16-23	local0-7	Site specific use

Priorities		
Numerical Code	Keyword	Facility
0	emerg	Emergency: system is unusable
1	alert	Alert: action must be taken immediately
2	crit	Critical: critical conditions
3	err	Error: error conditions
4	warning	Warning: warning conditions
5	notice	Notice: normal but significant conditions
6	info	Informational: informational messages
7	debug	Debug: debug level messages

Selectors can contain the special keywords ' * ' and 'none', meaning all or nothing, respectively. A selector can include multiple facilities separated with commas. Multiple selectors can also be combined with semicolons.

You may also precede the priority with an '=' and/or an '!' to specify, respectively, the inclusion of a single priority or the exclusion of the priority and any higher (numerically). Using both ('!=') will exclude the single priority. These features are taken from the BSD sources.

Examples:

1. A rule that sends messages for one facility and of any priority to file:

```
daemon.*                  /var/log/daemon.log
```

2. A rule combination that sends messages for two facilities and of any priority to one file and then everything else to another:

```
auth,authpriv.*           /var/log/auth.log
*.*;auth,authpriv.none            /var/log/syslog
```

3. A selector that picks out messages for one facility and of a single priority:

```
mail.=info
```

4. A compound selector that picks out messages for all facilities but only of priority info, notice and warning:

```
*.info;*.!err
```

5. A rule that sends messages for one facility and of all priorities except that of warning:

```
kern.*;kern.!=warn        /var/log/kernel/nowarnings
```

The terms "rule" and an "action" are often used synonymously. A rule *defines* an action.

Queues and queue parameters are covered later in this paper but for now we need to remember that:

- An *action queue* is created each time an action is specified.
- Action queue *parameters* are reset after an action queue has been created (allowing the creation of a new action queue and its corresponding parameters).

Output file syncing

Due to performance degradation, rsyslog no longer retains sysklog's default of file syncing [1] if not specified otherwise (by placing a dash in front of the output file name). Even if rsyslog finds sync selector lines it will ignore them. In order to enable file syncing you must explicitly do so at the top of your configuration file:

```
$MainMsgFileEnableSync on
```

Timestamps

Rsyslog comes with high-precision timestamp support (disabled by default). This feature is controlled by the following parameter:

```
$MainMsgFileDefaultTemplate RSYSLOG_TraditionalFileFormat
```

Comment it out in order to gain this feature (turn on high-precision timestamps).

Here is a comparison of traditional and high-precision timestamps, respectively:

```
Nov 17 12:01:44 client_hostname ubuntu: test

2008-11-17T12:02:47.372490-05:00 client_hostname ubuntu: test
```

Templates

Templates in rsyslog are user-definable formats that are applied to message queues. A template is associated with every type of output and can also be used for dynamic file name generation. In particular, for database logging, a template is used that converts a message into a proper SQL statement. Since templates are integrated into rsyslog at a basic level, hardcoded templates are available upon installation (i.e. those that are compatible with the stock sysklog).

A template is defined in this way:

```
$template template_name,"text %property% text %property%\n"
,<options>
```

Where options can be only sql-related at this time. They are applied automatically as needed and do not concern us here.

We can modify properties in interesting ways where each of the above properties can be extended:

```
%property:fromChar:toChar:options%
```

1 - *File syncing* writes every message immediately upon reception and waits until that write has been deemed successful before continuing.

Where 'fromChar' and 'toChar' are character addresses. These enable us to begin and end a property's value at certain places (ex: 1:2 are the first two characters in the value of the specified property). Property options are listed in Appendix D.

We apply this template to messages by associating it with the default template for file action (we can do the same for forwarding/network action):

```
$ActionFileDefaultTemplate template_name
```

Default forwarding templates used with UDP or TCP are defined with the following parameter:

```
$ActionForwardDefaultTemplate
```

Examples:

1. A template named "simple_template" that adds the word 'text' before and after the regular syslog message. Following the example is output after the string "test" was sent with the 'logger' command:

   ```
   $template simple_template,"text_before %msg% text_after\n"
   ```

   ```
   text_before  test text_after
   ```

2. A template named "uppercase_template" that does the same but puts the property value in uppercase:

   ```
   $template uppercase_template,"text_before %msg:::uppercase%
   text_after\n"
   ```

   ```
   text_before  TEST text_after
   ```

Property-based filters

This type of filter is unique to rsyslog. Property-based filters provide the capability to filter on message properties like hostname, syslogtag and msg (full list of properties provided in Appendix C). Each property can be evaluated against a string (the value) using a specified "compare-operation":

```
:property, [!]compare-operation, "value"
```

Naturally, the '!' is to negate the comparison.

To assist in debugging, rsyslog provides debug information for property-based filters during their evaluation. Invoke rsyslogd in the foreground while specifying the "-d" option to enable this.

Comparison operators

contains	value is contained in property contents
isequal	value is identical to property contents
startswith	value is found at the beginning of property contents
regex	value is matched against property contents using a POSIX regular expression

Examples:

1. Send messages beginning with a certain string to their own file:

```
:property, startswith, "pam_unix"
/var/log/properties/startswith-pam_unix
```

2. Send messages matching a certain regular expression (e.g.. "/dev/sda2") to their own file:

```
:property, regex, ".*sd.*"
/var/log/properties/regex-sd
```

Queue processing

All incoming messages are placed in the main message queue where they are then filtered by configured actions (what to do with certain messages) and assigned to the action's queue and processed accordingly. This is all applied serially. The consequence of this is that every action's processing is only as fast as the sum of all the actions. When even one action is regularly slow this can become a serious problem. This is true even to the point of actions ceasing to be processed. This can occur, for example, when an action writes to a remote database and the database becomes overloaded or simply unavailable. The answer here is to *de-couple* the slow action queues from the main queue, effectively creating parallel processing. This is simply accomplished with rsyslog.

In the configuration files, the main queue is denoted by *MainMsg* and a de-coupled action queue is denoted by *Action*. In this document, queue parameters generically contain the place name <object> to refer to the queue type. So replace that with either of the two queue types when using them.

Examples:

1. Specify (and enable) a disk queue for the main message queue:

   ```
   $MainMsgQueueFileName some_filename1
   ```

2. Specify (and enable) a disk queue for an action queue:

   ```
   $ActionQueueFileName some_filename2
   ```

Central logging scenarios

This section looks at how to implement logging models #2, #3, and #4 encountered earlier.

Multiple systems (to disk)

Each client system should have rsyslog installed. The server software does not need to be rsyslog as long as it is compatible with the chosen transport protocol.

You first need to decide what protocol you want to use: UDP, TCP.

On the server, assuming you are running rsyslog, you do this by enabling the appropriate input module, as well as specifying the port to be used:

UDP:

```
$ModLoad imudp
$UDPServerRun 514
```

TCP:

```
$ModLoad imtcp
$InputTCPServerRun 514
```

On the client, you specify either the UDP or TCP protocol with the '@' or '@@' characters respectively.

Examples:

- Here we are directing the client to forward all its logs via UDP to the logging server whose IP address is 192.168.0.1:

  ```
  *.*         @192.168.0.1:514
  ```

- The same but over TCP:

  ```
  *.*         @@192.168.0.1:514
  ```

Multiple systems (to database)

Rsyslog supports the following databases:

- MySQL
- PostgreSQL
- Oracle
- SQLite
- Microsoft SQL
- SyBase

- Firebird/Interbase
- Ingres
- mSQL

MySQL and PostgreSQL are supported natively (plug-ins provided) while the rest are supported via *libdbi*, a database abstraction layer. Below we provide guidance for MySQL.

Each client should be set up for central logging. There are no other requirements for database logging on the client side.

First, install the module on the logging server:

```
$ sudo aptitude install rsyslog-mysql
```

Next, the server should load the output module **ommysql** and be configured to connect to the database. Its configuration should be similar to the following:

```
$ModLoad ommysql
*.* :ommysql:localhost,Syslog,rsyslog,abc123
```

This configuration is set up automatically when you install the above package. It is implemented as an include file found: */etc/rsyslog.d/mysql.conf .*

In addition, the MySQL database creation process is also automated. The newly created database is called 'Syslog', containing two tables: 'SystemEvents' and 'SystemEventsProperties'. The installation process will prompt you for a password (user 'rsyslog' is used by the software to access the database). A password will be generated if you do not provide one and the credentials end up in the configuration above (i.e. I provided the password 'abc123' during installation). The only privilege required by the database user is INSERT.

Branch offices (remote storage)

This extension to the central logging model involves the use of a non-trusted network such as the Internet. Securing the connection over which the syslog data is transported may be required. In a branch office environment it is probable that a VPN is already in place. If so, this option should be used. In the absence of a company VPN, however, you may choose to use the TLS/SSL protection that rsyslog natively provides.

We will provide the basic steps required to set this up. See Appendix A, "The GNU Transport Layer Security Library" for more on TLS.

On the system where you will be creating keys and signing certificates you will need to install the necessary tools and create directories to manage the various files:

```
$ sudo aptitude install gnutls-bin
$ mkdir -p ~/tls/{ca,server,client}
$ chmod go-rwx ~/tls/{ca,server,client}
```

Notes:

- You need to create a separate certificate for each machine (client and server).
- When generating a certificate (-c option) use the proper DNS name of the machine in question (dnsName dialogue) as this is the name used in the certificate. Here, we assume the names of the server and client are, respectively, *server.example.com* and *client.example.com*.
- Protect all private keys.
- For security reasons, try to keep the machine acting as CA not permanently connected to a network.

For simplicity, create all keys, requests and certificates on the CA:

On the Certificate Authority

1. Manage the CA:

   ```
   $ cd ~/tls/ca
   ```

2. Create the private CA key (**ca-key.pem**):

   ```
   $ certtool -p --outfile ca-key.pem
   ```

3. Self-sign the public CA certificate (**ca.pem**):

   ```
   $ certtool -s --load-privkey ca-key.pem --outfile ca.pem
   ```

4. Manage the server:

   ```
   $ cd ~/tls/server
   ```

5. Create the private server key (**server-key.pem**):

```
$ certtool -p --outfile server-key.pem
```

6. Generate a signing request (**request.pem**):

```
$ certtool -q --load-privkey server-key.pem \
      --outfile request.pem
```

7. Sign the request with the CA private key to obtain the server's certificate (**server-cert.pem**):

```
$ certtool -c --load-request request.pem \
      --outfile server-cert.pem \
      --load-ca-certificate ../ca/ca.pem \
      --load-ca-privkey ../ca/ca-key.pem
```

8. Manage a client:

```
$ cd ~/tls/client
```

9. Create the private client key (**client-key.pem**):

```
$ certtool -p --outfile client-key.pem
```

10. Generate a signing request (**request.pem**):

```
$ certtool -q --load-privkey client-key.pem \
      --outfile request.pem
```

11. Sign the request with the CA private key to obtain the client's certificate (**client-cert.pem**):

```
$ certtool -c --load-request request.pem \
      --outfile client-cert.pem \
      --load-ca-certificate ../ca/ca.pem \
      --load-ca-privkey ../ca/ca-key.pem
```

12. Securely transfer the necessary files to the server (ca.pem, server-cert.pem, server-key.pem) and each client (ca.pem, client-cert.pem, client-key.pem).

On the logging server

Configuration:

```
$ModLoad imtcp

$DefaultNetstreamDriver gtls

$DefaultNetstreamDriverCAFile ca.pem
$DefaultNetstreamDriverCertFile server-cert.pem
$DefaultNetstreamDriverKeyFile server-key.pem

$ActionSendStreamDriverAuthMode x509/name
$ActionSendStreamDriverPermittedPeer client.example.com
$ActionSendStreamDriverMode 1

$InputTCPServerRun 10514
```

On a logging client

Configuration:

```
$DefaultNetstreamDriver gtls

$DefaultNetstreamDriverCAFile ca.pem
$DefaultNetstreamDriverCertFile client-cert.pem
$DefaultNetstreamDriverKeyFile client-key.pem

$ActionSendStreamDriverAuthMode x509/name
$ActionSendStreamDriverPermittedPeer server.example.com
$ActionSendStreamDriverMode 1

*.* @@192.168.0.1:10514
```

Advanced Rsyslog features applicable to central logging

Rsyslog has a number of interesting and powerful advanced features. Here are two such features as applicable to central logging:

- BSD-style blocks
- Logging queues
- Discard watermarks

BSD-style blocks

We can create blocks of rules with each one separated by the previous by a program or hostname label. The block will only process messages corresponding to the program and/or hostname given.

Use '!program' or '-program' to include or exclude programs and '+hostname' or '-hostname' to do the same for hostnames. These features are also taken from the BSD sources and help in a central logging environment.

Examples:

- Send all messages generated by the named process to file:

```
!named
*.*                        /var/log/named.log
```

- In a central logging context, set up a rule on the server for a particular host:

```
+mail.example.com
mail.*                     /var/log/mail/hosts
```

Logging queues

A logging queue is simply a buffer for log messages. There are several types provided by rsyslog.

Disk Queues

A *Disk Queue* is a queue that resides solely on a hard drive. Reliability is increased while performance suffers. You can impose a limit to the queue size in this way

```
$<object>QueueSize 15000         # 15k maximum elements
```

In-Memory Queues

An *In-Memory Queue* is a queue that resides solely in volatile memory. Reliability is sacrificed while performance is increased. It is worth mentioning however that this method is not recommended when any form of reliability is required. There are two subtypes to this kind of queue:

- FixedArray
- LinkedList

FixedArray uses a pre-allocated array that holds pointers to queue elements whereas *LinkedList* allocates resources dynamically. There is some processing overhead with the second type but can save you memory. The former is the default (with a limited size of 10k messages/elements). You can impose a limit to the queue size as for disk queues.

```
$<object>QueueType LinkedList  # dynamically grow this in-memory
queue
```

Hybrid Disk-Assisted In-Memory Queues

This is a combination of an In-Memory Queue and a Disk Queue. In that order. So data is written to disk (and read back) on an as-needed basis. You set up such a queue by defining **both** directives for each respective queue:

```
$<object>QueueType LinkedList    # or FixedArray
$<object>QueueFileName filename
```

There is no actual queue size for a DA (disk-assisted) queue. It is limited by hard disk space only. In order to limit the space available use

```
$<object>QueueMaxDiskSpace 100G # a maximum disk space of 100 GB
```

Queueing and de-queueing

It is possible to enable disk queues only during specific time intervals. A queue could be made active during peak traffic hours (say 8 am to 8 pm) and de-activated for the rest of the day. This would essentially allow the queue to dump its contents during the night.

```
$<object>QueueDequeueTimeBegin 20  # activate queue at 8 pm
$<object>QueueDequeueTimeEnd 8      # de-activate queue at 8 am
```

Logging queue examples

Here are some examples of using queues in various situations. Add the following lines to your configuration to enable queueing features.

Local disk logging

Create a default (FixedArray) queue for a standalone system:

```
$WorkDirectory /var/log/queue     # destination queue directory
$MainMsgQueueFileName filename    # set file name for this
action; enables disk mode
```

Remote disk logging

When logging to a remote server there may be times when the database is no longer able to cope with the traffic volume. We set up a LinkedList In-Memory Queue; specify to save the queue's memory-resident data if rsyslog ever shuts down; and connect to server 192.168.0.1 over the TCP protocol on port 514:

```
$WorkDirectory /var/log/queue     # destination queue directory
$ActionQueueType LinkedList       # de-couple this action queue
$ActionQueueFileName filename     # set a file for this action;
enables disk mode
$ActionResumeRetryCount -1        # infinite retries on failure
$ActionQueueSaveOnShutdown on     # save in-memory data if
rsyslog shuts down
*.*     @@192.168.0.1:514         # connect to remote server
```

Remote database logging

We use the same setup as above but swap the last line with the following one. We will access a MySQL server at 192.168.0.1 containing database 'logs' with user 'rsyslog' and a password of 'abc123':

```
*.*     :ommysql:192.168.0.1,logs,rsyslog,abc123;
```

Discard watermarks

When logging centrally, there may be times of sudden bursts of traffic. When a queue reaches a threshold of a number of queued elements, less important messages can be discarded to help alleviate the problem. The threshold in this context is called a 'discard watermark'. The objective is to save queue space for more important messages. The algorithm discards both incoming messages and those currently queued.

The discard watermark should be set sufficiently high to not discard messages unnecessarily but low enough to allow for large message bursts.

```
$<object>QueueDiscardMark some_threshold     # number of elements
$<object>QueueDiscardSeverity some_severity # numerical severity
```

This directive accepts both the usual textual severity keyword as well as a numerical code as defined in RFC 3164.

To turn message discarding off simply make the discard watermark higher than the queue size. An alternative is to specify a discard severity of 8. This is the default setting (to prevent unintentional message loss).

Examples:

- Set up a main queue to discard messages less severe than Error (i.e. warning, info, notice, and debug) when the queue exceeds 8000 messages:

  ```
  $MainMsgQueueDiscardMark 8000
  $MainMsgQueueDiscardSeverity 4
  ```

- Same as above:

  ```
  $MainMsgQueueDiscardMark 8000
  $MainMsgQueueDiscardSeverity warning
  ```

Appendix A: References and useful Links

- Rsyslog home page
  http://www.rsyslog.com
- Rsyslog mailing list (rsyslog-users)
  http://lists.adiscon.net/mailman/listinfo/rsyslog
- Rsyslog public forums
  http://kb.monitorware.com/rsyslog-f40.html
- The Ins and Outs of System Logging Using Syslog
  http://www.sans.org/rr/whitepapers/logging/1168.php
- Comparison between rsyslog and syslog-ng
  http://www.rsyslog.com/doc-rsyslog_ng_comparison.html
- RFC 3164 (The BSD Syslog Protocol)
  http://www.ietf.org/rfc/rfc3164.txt
- RFC 3195 (Reliable Delivery for Syslog)
  http://www.ietf.org/rfc/rfc3195.txt
- The GNU Transport Layer Security Library
  http://www.gnu.org/software/gnutls/manual/html_node/index.html
- List of log analysers
  http://www.syslog.org/wiki/Main/LogAnalyzers
- Rsyslog main developer blog
  http://blog.gerhards.net/
- SANS Information System Audit Logging Requirements (2006)
  http://www.sans.org/resources/policies/info_sys_audit.doc
- NIST Information System Audit Logging Requirements (2006)
  http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf
- Distributed syslog architectures with syslog-ng Premium Edition (2008)
  http://www.balabit.com/dl/white_papers/syslog-ng-v2.1-whitepaper-distributed-syslog-architectures-en.pdf

Appendix B: rsyslog.conf / syslog.conf diff

```
$ diff rsyslog.conf syslog.conf

- XXX this should be diff -u, only RMS still uses context diffs
1c1
< #  /etc/rsyslog.conf  Configuration file for rsyslog v3.
---
> #  /etc/syslog.conf   Configuration file for syslogd.
3,32c3,4
< #                     For more information see
< #                     /usr/share/doc/rsyslog-
doc/html/rsyslog_conf.html
<
<
< #################
< #### MODULES ####
< #################
<
< $ModLoad imuxsock # provides support for local system logging
< $ModLoad imklog   # provides kernel logging support
(previously done by rklogd)
< #$ModLoad immark  # provides --MARK-- message capability
<
< # provides UDP syslog reception
< #$ModLoad imudp
< #$UDPServerRun 514
<
< # provides TCP syslog reception
< #$ModLoad imtcp
< #$InputTCPServerRun 514
<
<
< ###########################
< #### GLOBAL DIRECTIVES ####
< ###########################
<
< #
< # Use default timestamp format.
< # To enable high precision timestamps, comment out the
following line.
< #
< $ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
---
> #                     For more information see syslog.conf(5)
> #                     manpage.
35c7
< # Set the default permissions for all log files.
---
> # First some standard logfiles.  Log by facility.
37,39d8
< $FileOwner root
< $FileGroup adm
< $FileCreateMode 0640
41,53d9
< #
< # Include all config files in /etc/rsyslog.d/
< #
< $IncludeConfig /etc/rsyslog.d/*.conf
<
<
< ###############
< #### RULES ####
< ###############
<
< #
< # First some standard log files.  Log by facility.
< #
68c24
< mail.warn                         -/var/log/mail.warn
---
> mail.warning                      -/var/log/mail.warn
71,72c27
< #
< # Logging for INN news system.
---
> # Logging for INN news system
79c34
< # Some "catch-all" log files.
---
> # Some `catch-all' logfiles.
84c39
< *.=info;*.=notice;*.=warn;\
---
> *.=info;*.=notice;*.=warning;\
101c56
< #     *.=notice;*.=warn       /dev/tty8
---
> #     *.=notice;*.=warning    /dev/tty8
114c69,70
<       *.=notice;*.=warn       |/dev/xconsole
---
>       *.=notice;*.=warning    |/dev/xconsole
>
```

Appendix C: Message properties

Property	Meaning
msg	entire message
rawmsg	entire message exactly as it was received from the socket
hostname	hostname of original sender
source	alias for hostname property
fromhost	hostname of immediate sender (may be different from original sender)
fromhost-ip	IP address of 'fromhost'
syslogtag	message Tag (see appendix A; "The BSD Syslog Protocol")
programname	name of reporting program
pri	priority (undecoded)
pri-text	priority (textual form)
iut	MonitorWare InfoUnitType - used when talking to a MonitorWare backend
syslogfacility	facility (numerical form)
syslogfacility-text	facility (textual form)
syslogseverity	severity (numerical form)
syslogseverity-text	severity (textual form)
syslogpriority	alias for syslogseverity property (not pri)
syslogpriority-text	alias for syslogseverity-text property
timegenerated	high resolution timestamp of received message
timereported	message timestamp
timestamp	alias for timestamp property
protocol-version	contents of the PROTCOL-VERSION field from IETF draft draft-ietf-syslog-protcol
structured-date	contents of the STRUCTURED-DATA field from IETF draft draft-ietf-syslog-protocol
app-name	contents of the APP-NAME field from IETF draft draft-ietf-syslog-protocol

procid	contents of the PROCID field from IETF draft draft-ietf-syslog-protocol
msgid	contents of the MSGID field from IETF draft draft-ietf-syslog-protocol
inputname	identifier of input module that generated the message (if available)

Appendix D: Property options

Option	Meaning
uppercase	convert property to uppercase
lowercase	convert property to lowercase
drop-last-lf	remove last linefeed
date-mysql	format as mysql date
date-rfc3164	format as RFC 3164 date
date-rfc3339	format as RFC 3339 date
date-subseconds	subseconds of a timestamp (always 0 for low precision timestamps)
escape-cc	replace control characters (ASCII value 127 and values less then 32) with an escape sequence. The sequnce is "#<charval>" where charval is the 3-digit decimal value of the control character. For example, a tabulator would be replaced by "#009".
space-cc	replace control characters by spaces
drop-cc	drop control characters - the resulting string will neither contain control characters, escape sequences nor any other replacement character like space.
sp-if-no-1st-sp	returns either a single space character or no character at all. Field content is never returned. A space is returned if (and only if) the first character of the field's content is NOT a space. This option is a hack to solve a problem rooted in RFC 3164 which specifies no delimiter between the syslog tag sequence and the actual message text. Almost all implementation in fact delimit the two by a space. As of RFC 3164, this space is part of the message text itself.
secpath-drop	Drops slashes inside the field (e.g. "a/b" becomes "ab"). Useful for secure pathname generation (with dynafiles).
secpath-replace	Replace slashes inside the field by an underscore. (e.g. "a/b" becomes "a_b"). Useful for secure pathname generation (with dynafiles).

Note: options escape-cc, space-cc, or drop-cc require that $EscapeControlCharactersOnReceive is set to off.