QRAAT

John Muir Institute of the Environment

May 20, 2013

Abstract

This could eventually be our doc for QRAAT. For now, this just outlines configuration stuff for the prototype system. Build this doc with pdflatex (sudo apt-get install pdflatex).

Contents

1 Overview

1.1 Hardware

Along with the antenna, preamps, RMG receiver, batteries, and solar panels, a node in the QRAAT network consists of a router to the network, a field computer, and a network-addressible powerswitch. The last two are pictured together with the receiver in Figure 1.

Field computer. To run the software defined radio remotely, we require a headless system that can run on limited power and be able to withstand high temperatures. The ideal system would have no moving parts—fanless, solid-state memory instead of a harddrive—and draw only a few amps. However, in our tests with various solutions, we've concluded that a dual-core CPU is required at a minimum.[1] The parallelisation that can be achieved with multiple cores drastically improves the performance of the signal processing software. There are a number of products available that strike a comprimise between power consumption and processing power. The fit-PC3 suites our needs well.[2] The fit-PC3 has a dual-core, 32-bit AMD processor, 2 GBs of ram, and a 8GB solid state drive. Although it is designed to run any general-purpose desktop OS, the fit-PC3 can easily be configured to run as a headless system. It even comes with a mini-rs232 console interface, though we don't make use of this in QRAAT at the moment. With our software running full-bore, the fit-PC3 draws 46 W.

Figure 1: Network powerswitch, field computer, and RMG receiver.

Powerswitch. The powerswitch, field computer, and network router are connected with an ethernet switch. The field computer and RMG receiver are powered by a network-addressible powerswitch. The computer's power is switched over the network by the server, and the the computer is set up to switch the receiver. This makes it possible to turn the equipment on and off remotely and, with certain products, monitor power levels. We've deemed this feature necessary for two reasons: one, an intermittent issue related to the RMG receiver's USRP[3] can be fixed by cycling the power. Remote switching makes fixing this problem much more convenient; two, turning off equipment, in particular the RMG receiver, when it's not needed is the best way to save battery power. (Note that it is possible to deploy QRAAT without such switches.)

Again, there are various solutions for network power switching. Most of these input mains power, i.e. 120 V AC, and require us to mofidy them to work with our solar panels (12 V DC). The PingBrother EPIW104P[4] passive POE ethernet switch not only allows us to input the proper power, but provides an array of features uesful for our deployment of QRAAT. The EPIW104P can switch a 12 V DC relay instead of POE, and the ethernet ports can be used as a switch. In addition, its http-based interface can be used to monitor the input voltage and power consumption. Our nominal hardware specification with the PingBrother powerswitch and fit-PC3 is as follows:

Network The network router and field computer are plugged into the powerswitch's ethernet switch. The powerswitch and computer are assigned static IP addresses in the subnet of the router. Static addresses are necessary because the QRAAT server must know how to reach the site; this information is not distributed when the node comes into the network. (See software overview section for details.)

[1] One solution we tried was the net6501-70 from Soekris Engineering, which ships with a single 1.6 Ghz Atom core. The backend software was installed on top of the OpenWRT operating system, a stripped-down version of GNU/Linux meant for embedded systems. The Soekris board failed to process the radio signal at line-rate with the system running full-bore.

[2] For details, visit `fit-pc.com`.

[3] For details, visit `ettus.com`.

[4] For details, visit `pingbrother.com`.

Power The field computer's power is attached to the powerswitch's first relay on NC ("normally closed"). If power goes down and is then made available the field computer will be powered automatically. The RMG receiver is attached to the second relay on NO ("normally opened") so that it won't get power until it is needed by the computer.

Receiver The receiver has a two ports: one, a straight-through serial port for communication with the receiver's PIC interface, and two, a standard USB 2.0 port for data transfer from the USRP. The fit-PC3 doesn't have a straight-through serial interface, but an off-the-shelf a serial → USB converter is sufficient. The PIC uses a 9600,8n1 baud-rate/parity-bit scheme.

1.2 Software

We begin this section with an eagle eye view of the QRAAT dataflow. Each transmitter emits a periodic pulse on a specific radio frequency, e.g. 164.599 Mhz. The pulse is received and amplifed on four channels, then converted to a digital signal by the RMG receiver. The software-defined radio filters the signal and inputs it into a pulse detector. When the pulse detector is triggered, the signal region surrounding the pulse on all four channels is dumped to file. This kind of record is created for each pulse and each transmitter that the SDR is conifgured to listen to. These files are produced and stored in memory on site; when the server requests them, they are tranferred over the network and deleted from the field computer.

The backend software package is comprised of two elements: one, the software defined radio module, and two, a system for managing QRAAT's resources and collecting data. We now describe these in turn. This section refers to the software repository published on Github at `github.com/QRAAT`.

1.2.1 RMG module

The SDR pulse detector is implemented with a mixture of C++ and Python on top of GNU Radio.[5] In GNU Radio terms, the module is made up of a radio signal processing block with a suite of high-level applications that run it. The transmitters that the RMG module listen for are specified by a file called `tx.csv`. (See Appendix A for file format.) When the software starts, a signal filter and peak detector is instantiated for each frequency passed by `tx.csv`, up to 32. The pulse data files are stored in a directory structure that reflects the time and date it was created. For example:

```
/tmp/ramdisk/det_files/2013/03/22/15/20/03439743.det
```

The file name gives the second it was created with millisecond precision, i.e. `SSUUUUUU.det`.

1.2.2 Dataflow

Each QRAAT node has a state associated with it: *down*—computer should shutdown, *up*—computer should be on, but the RMG receiver should be off and not listening for transmitters, or *active*—the RMG receiver should be on and pulse detector should be running. The state is specified by a file on the server called `sitelist.csv`. (See Appendix A for file format.) Along with the state, the site list specifies the configuration information of each site. The parameters are:

Name — of the site,

CompIP — hostname or IP address of the remote computer,

PowerIP — hostname or IP address of the network addressible power switch,

CompOutlet — outlet number of the computer,

RxOutlet — outlet number of the RMG receiver,

[5] `gnuradio.org`.

PowerType — power source type, e.g. Netbooter, PingBrother, or WebPowerSwitch, and finally

State $\in$ {*Up*, *Down*, *Active*}.

When a site's configuration information needs to be updated or the user wishes to change its state, the parameters (*PowerIP, RxOutlet, PowerType, State*) are transmitted to the site computer as a configuration file called `site.csv` (Appendix A).

Node. Although the node's *goal* state is specified by the server, the server cannot directly coerce its state; only the node can power on its RMG receiver and start the RMG module or power itself down. We've designed it this way so that the nodes can function independently of the server in case of network failure. In addition to controlling its own state, the node implements monitoring and error recovery by checking its state periodically. If the pulse detector crashes for any reason, the process is controlled by a monitor script that cycles the recevier's power and attempts to restart the module. If the process crashes a certain number of time before the system successfully reaches *active*, the node gives up. Each time the site's state is checked or transitioned, the *real* state, which we'll call *SiteStatus*, is reported to a file called `status.log`.

Server. There are three essential tasks that the server implements: one, pole the sites for status information, two, update the sites' configuration and goal state, and three, download and store pulse data. When a site is poled, the server collects network and site status. If the site is reachable, then the server reads the last line of `status.log`. Site status is reported as (*NetworkStatus, SiteStatus*). After a site is updated (the `site.csv` file is transmitted to the field computer), the server instructs it to check its state.

Finally, when the server fetches the pulse data files, it downloads all available files excluding those that were recently produced (within the last minute). This is to avoid attempting to download a file that is currrently being written by the RMG module. Note that this design critically depends on real time clock synchronization, which we achieve through the `ntpd`[6] daemon.

1.3 Usage

In the QRAAT protoype system, dataflow and site management are implemented with shell scripts. The user-level programs are `rmg` and `rmg-node`. When invoked, they immediately set up environment variables specifying data, configuration files, etc.

rmg.[7] The server control program. `rmg` takes as arguments a task and a list of sites. Before performing the task, the status of the site is checked. Possible tasks are the following:

status — Just report site's current status as (*NetworkStatus, SiteStatus*).

update — Passive site update. Copy new `site.csv`.

up/down/start/stop — New goal state. Update `sitelist.csv` and copy new `site.csv`. Tell the site to check its new state (`rmg-node check`, see below.)

on/off — Switch the field computer's power supply.

updatetx –file *FILE* — Update site's `tx.csv`.

cyclerx — Cycle the site's receiver power. This is needed sometimes.

fetch — Download `.det` files.

rmg-node. Control a site's state locally. This program is used for monitoring and is invoked by `rmg` when the user changes the site's goal state. It can also be safely called directly. As its argument, it accepts either a new state or `check`:

active — Check that RMG receiver is on and the SDR is running.

[6] Network time protocol (NTP).

[7] The name "rmg" is admittedly a bit of a misnomer, since it has nothing to do with "Rapid Multichannel Goniometers". It just stuck, I suppose.

up — Check that the RMG receiver is off and the SDR is stopped.

down –delay *M* — Set a timer to shutdown. If such a timer exists (i.e., state is already down), the timeer has elapsed and no M isn't specified, than power off the computer. If a delay is specified, reset timer to M minutes.

check — Look up goal state in `site.csv` and transition as described.

2 Dataflow specifications

QRAAT is composed of nodes all capable of running software defined radio to perform initial signal processing. One of these is desginated as the QRAAT server, where pulse data are collected for higher-order processing. We've designed QRAAT with modularity in mind; it's possible to run the SDR on the same machine as the server. This dataflow is implemented as a set of shell and Python programs. Except for the top-level `rmg` and `rmg-node` scripts, these are used by both the nodes and server. In this section, we describe these programs in detail. The complete specification of the nodes and server are also given.

2.1 Environment variables (`rmg_env`)

Dataflow is implemented as a set of shell scripts and Python programs. Locations of data, metadata, and miscellaneous variables are defined in a bash script called `rmg_env` which is included in any top level program. (See Appendix C for suggested values.)

`RMG_SITE_METADATA_DIR` — Location of metadata on a remote machine. Files included in this directory are: `site.csv`, `tx.csv`, and `status.log`. Any debug information is also stored here. We currently log the standard output and standard error of the last run of the pulse detector. This value is used by the server and must be the same at each site.

`RMG_SITE_DET_DIR` — Location of data on a remote machine. This is typically a subdirectory of a temporary filesystem created with `/etc/fstab`. This value is used by the sever and must be the same at each site.

`RETRY_MAX` — The maximum number of attempts to start the pulse detector before a fatal error state is reached. This value is needed by the site.

`RMG_SERVER_METADATA_DIR` — Location of metadata on the server. Files included in this directory are `sitelist.csv` and transmitter spec files.

`RMG_SERVER_DET_DIR` — Location of raw pulse data files fetched from the nodes.

`RMG_SERVER_EST_DIR` — Location of callibrated data files.

`RMG_SERVER_SSH_KEYFILE` — Location of private RSA encryption key.

2.2 Server

Along with managing the field computers and collecting data, the server is responsible for callibration and triangulation. The `rmg` script allows us to power the computer and RMG module on and off, cycle the RMG module power, start and stop the software defined radio, update the field computer's transmitter file, and collect `.det`s. The server has a file called `sitelist.csv` that stores various parameters as well as the state of nodes. Because the server doesn't direclty control the state of the nodes, this is referred to as the goal state. The following parameters are specified per site:

Name — of the site,

CompIP — hostname or IP address of the remote computer,

PowerIP — hostname or IP address of the network addressible power source,

CompOutlet — outlet number of the computer,

RxOutlet — outlet number of the RMG receiver module,

PowerType — power source type, e.g. Netbooter, PingBrother, or WebPowerSwitch, and finally

State $\in \{$*Up*, *Down*, *Active*$\}$.

Possible states are:

Up — Site computer is on, the RMG receiver is off, and the pulse detector is not running.

Down — The RMG receiver is off and either the computer is off or there is a timer set on site to shutdown. The server does not directly run the shutdown the on the node.

Active — Site computer is on, the RMG receiver is on, and the pulse detector is running.

When a site's goal state is to be updated, the parameters (*PowerIP, RxOutlet, PowerType, State*) are transmitted to the site computer in a file called `site.csv`. The site computer then immediately verifies that it's actual state is *State*.

Status is reported as the pair (*NetworkStatus, SiteStatus*). *NetworkStatus* is either *ErrorPower*, meaning the site's powerswitch couldn't be reached over the network, or *ErrorComputer*, meaning the powerswitch could be reached, but not the computer. If the computer is reachable, then *NetworkStatus* = *Ok*. *SiteStatus* is the actual state reported by the site computer. If it can't be reached, then *Unknown* is reported.

To determine reachibility of the computer, the powerswitch is queried for the power status of the computer. If it's off, then it's assumed to be reachable if it were turned on; hence, (*Ok*, *Down*) is reported. If it's on and the computer is unreachable, then (*ErrorComp*, *Unknown*) is reported. Note that this status will be reported even when the field computer shuts itself down cleanly; though the system is halted, its input is still powered and must be switched.

2.3 Nodes

Field computers run the pulse detection software on GNU Radio. As such, QRAAT must be run on GNU/Linux. Section 3 details the installation and configuration of our software on Ubuntu Server 12.04 LTS. In addition to GNU Radio, each site must run an ssh daemon, git, and ntpdate. Each node has a user called *rmg* who runs the SDR.

RSA encryption is used over ssh for secure data transfer between nodes and server. The public key is stored on the node in `/home/rmg/.ssh/authorized_keys` and the private key is stored on the server in the file specified by `RMG_SERVER_SSH_KEYFILE`. (See Appendix B.)

The software defined radio emits a file per pulse per transmiter and stores it locally in memory. To accomplish this, a temporary file system, called a ramdisk, is created in `/etc/fstab`. The location must be the parent directory of `RMG_SITE_DET_DIR`.

Though the *goal* state is specified by the server, nodes control and report their actual state The site computer periodically checks the goal state given in `site.csv` and verifies the computer is in that state, looks up the last reported status, and reports change in the real state to `status.log`. *SiteStatus* is reported as one of the following:

Up — Site computer is on, the RMG receiver is off, and the pulse detector is not running.

Down(N) — Site computer is on, the RMG receiver is off, the pulse detector is on, and a timer is set to power down the system. If N seconds have elapsed since the site last checked its status, then *Shutdown* is reported and computer powers itself off.

Shutdown — If this is the last reported status and the goal state is *Down*, then the shutdown timer is reset.

Active — Site computer is on, the RMG receiver is on, and the pulse detector is running

ErrorRetry(i) — The pulse detector crashed and is restarting. The software may crash for various reasons, sometimes resulting from upstream code such as GNU Radio or the driver for the USRP. Before retrying, the RMG receiver's power is cycled. i gives the retry count.

ErrorFatal — $i =$ `RETRY_MAX`, do not attempt to recover. The computer remains on, but the RMG receiver is off.

2.4 Programs

2.4.1 `rmg_csv`

```
usage: rmg_csv <site> <parameter> [<value>]
          -c,--column <parameter> [<value>]
          -r,--row <site> [{parameter list}]
          -l,--last-row   [{parameter list}]
 < path/to/original.csv [> path/to/new.csv]
```

This program is used to look up and modify values in configuration files. It's functionality has evolved to meet our needs. The input is a CSV-formatted file on standard input. If a the name of a site and a paramter are given as arguments, then `rmg_csv` emits that field. If the field is empty, it emits *nil*. If a value is also passed, then the CSV file is dumped to standard output with the field replaced. The '`-c`' option is similar, but it's applied to each site (row). The '`-r`' option emits a row as a CSV file. If a parameter list is given, then the fields for the row are outputted on separate lines. '`-l`' has a similar functionaility, but it applies to the last row. This is uesful for polling site status from `status.log`.

2.4.2 `rmg_fetch`

```
usage: find <dir> -type d | rmg_fetch
```

`rmg_fetch` is used to compile a list of directories of pulse data files to fetch from a remote computer. `scp` is used to copy whole directories from the nodes to the server. Recall that data files are stored in a directory structure based on the date and time they were created. In order to prevent copying a file currently being written by the pulse detector, we exclude directories created within the last minute. Thus, if the current time is 15:20 on 22 March 2013, we can't simply the entire directory `/tmp/ramdisk/det_files/2013`; we have to copy all the sibling directories of `/tmp/ramdisk/det_files/2013/03/22/15/20` individually. `rmg_fetch` combines non-"hot" directories in order to reduce the number of calls to `scp`. It reads a directory structure on standard input and outputs a list of subdirectories that can be copied at once.

2.4.3 `rmg_powerswitch`

```
usage: rmg_powerswitch <pwr_type> <pwr_ip> <outlet> {ON, OFF, CYCLE, QUERY}
                       [--invert={true,false}]
```

We use this script to control network addressible powerswitches. `pwr_type` $\in$ {*pingbrother*, *webpowerswitch*, *netbooter*, *nil*} is the type (brand) of powerswitch, `pwr_ip` is its address, and `outlet` is the identifier (number) of the outlet to be switched. `QUERY` returns the status of the outlet specified, either 1 or 0. If the '`--invert`' option is passed, then *ON* is sent to the relay instead of *OFF* and vise-versa. This feature is useful for the PingBrother system when a device is attached to NO instead of NC.

2.4.4 rmg_runretry

```
usage: rmg_runretry <pwr_type> <pwr_ip> <rx_outlet>
```

`rmg_runretry` is a wrapper script for the pulse detector and as such is only run on nodes. It starts the pulse detector and waits for it to fail. Once it fails, it cycles the power of the RMG receiver with `rmg_powerswitch` and retries up to `RETRY_MAX` times. After each failure, the status is written to `status.log` with the retry count. After `RETRY_MAX` failures, *ErrorFatal* is reproted and the program closes. Because this script is normally called from a higher level program, `RETRY_MAX` is expected to be declared before runnning.

2.4.5 rmg-node

```
usage: rmg-node state/check [-options]
```

(Detailed usage in overview section.) `rmg-node` is used to change the state of a QRAAT node locally. Either the state is specified or *check* is given and the program verifies the real state is consistent with the goal state specified in `RMG_SITE_METADATA_DIR/site.csv`. First, the script loads `rmg_csv` and verifies that configuration information is available. If no `status.log` exists, one is created. Next, current status information is collected, including the RMG receiver power state, the process I.D. of the pulse detector (if it's running), and the timestamp of the last-known status. We set up each node with the cron task `rmg-node check`.

2.4.6 rmg

```
usage: rmg task [-options] {site list}/all
```

(Detailed usage in overview section.) `rmg` is used to monitor and manipulate the state of QRAAT. Before *task* is performed, the status of the site is checked, i.e. (*NetworkStatus, SiteStatus*) is reported. If the task is only to report the status, a warning is emitted if the site's goal state is not consistent with the server-specified goal state. When an update or state-transition is to be performed, the server transmits the relevant `site.csv` to the field computer and runs `rmg-node check`.

`rmg` is also capable of some other miscellaneous tasks that don't cause state checks. These are `cyclerx`, which cycles the RMG receiver's power directly from the server, and `updatetx`, which transmits a new transmitter file to the site. Note that `updatetx` doesn't cause a running pulse detector to reload its tranmitter configuration.

3 Configuration instructions

3.1 Field computers

The field computers were configured by building and installing the software on one system, creating an image from its harddrive, and copying this image to the other sites. To create the image, we first installed a stock copy of Ubuntu Server 12.04 on a 7.5 GB partition, leaving 512 MB for swap. We configured no automatic updates, since these computers spend the majority of their time not connected to the internet. After booting the system and updating, the first thing to do is install the following packages via apt-get:

1. `git-core` - clone our software repository from github.
2. `openssh-server` - each field site needs to run an ssh daemon.
3. `minicom` - serial interface to RMG receiver module.
4. `ntpdate` - remote computers will be clock synced to the server via `ntpd`.

3.1.1 Building and installing the software

Clone the online repository to get things going:

```
$ git clone github.com/QRAAT/QRAAT.git
```

The first thing to build is GNU Radio along with the UHD driver. For conveniance, we provide a copy of the gnuradio build script written by Marcus Leech. Type

```
$ QRAAT/build-gnuradio -v all
```

to start downloading and building. This may take a few hours, but you'll only need to do this once. When the build finishes, it will output a couple post-install tasks. Make sure you do these. Next, building and installing the QRAAT software is essentially the same procedure, though it will take less than two minutes:

```
$ QRAAT/build-rmg -v install
```

To handle updates in this early development stage, we've set up each node to clone the software repository from the QRAAT server. The code base is stored locally in `/home/rmg/QRAAT`. To update, ssh to the site, switch to this directory, and type 'git pull'.

Before testing your build, we need to setup communication with the RMG recevier's PIC interface. First we add a rule to udev[8] to set permissions for `/dev/ttyUSB0`, the serial interface for the RMG receiver. Create a new file:

```
$ sudo vi /etc/udev/rules.d/101-serial-usb.rules
```

and type the following line:

```
KERNEL=="ttyUSB0", MODE="0666"
```

To verify that this worked, we'll try to communicate with the PIC via minicom. Open up the minicom configuration screen. Under *Serial port setup*, set *Serial Device* to `/dev/ttyUSB0`. Set *Bps/Par/Bits* to `9600 8N1`. Lastly, set *Hardware/Software Flow Control* to `No`. Restart minicom. See if you're able to communicate with it by typing "`?`".

3.1.2 Post-configuration

The SDR pulse detetcor can potentially produce a lot of files, and since they aren't needed permanently down stream, we store them locally in memory. We create a temporary file system in RAM in order to accomplish this. Add the following line to the end of `/etc/fstab` (be careful!):

```
tmpfs   /tmp/ramdisk  tmpfs  nodev,nosuid,mode=1777,size=1024M   0   0
```

1 GB is reasonable since the field computers have 2 GBs of ram.

If linux shuts down uncleanly, e.g. the site loses power, the GRUB bootloader will wait for user input before rebooting the operating system. This is bad for headless systems, so we need to configure GRUB to timeout in this situation. Add the following to /etc/default/grub:

```
GRUB_RECORDFAIL_TIMEOUT=2
```

Then type `sudo update-grub`.

Up until now, we've been using the field computer with a monitor, keyboard, and ethernet connection to the internet. The next thing we have to do is configure the network interface so that we can access it over ssh without a head. In `/etc/network/interfaces`, comment out the default ethernet interface and add a static IP address:

[8] `udev` is a common domain on GNU/linux systems used to handle new devices when they're plugged in. For instance, when you installed GNU Radio, rules were installed for the USRP—a hardware component of the RMG receiver—and the UHD driver.

```
# auto eth0
# iface eth0 inet dhcp
auto eth0
iface eth0 inet static
  address 10.20.1.55
  netmask 255.0.0.0
```

To connect to the computer directly through an ethernet cable, just setup a static IP address on the host system in the same subnet. See the section on networking for details.

The next thing to do is change the system's hostname. This needs to be changed in two places: `/etc/hostname` and `/etc/hosts`.

Lastly, since we will be managing the power of this system remotely, we want to be able to shutdown and reboot the computer without typing in a sudo password. Type

```
$ sudo visudo
```

and add the following lines to the end of the sudoers file:

```
rmg ALL=NOPASSWD: /sbin/halt
rmg ALL=NOPASSWD: /sbin/reboot
rmg ALL=NOPASSWD: /sbin/poweroff
```

3.1.3 Creating an image

The preceeding configuration is time-consuming, though not difficult. For configuring many sites, it's best to create an image from a fully configured system and copy this to other sites. The simplest way to do this is with an Ubuntu live-USB. Plug a monitor, keyboard, and mouse into the FitPC and boot a live copy of Ubuntu on a thumbdrive. Grab a terminal and make sure the harddrive is unmounted. ("`mount | grep sda`"). Verify that /dev/sda is indeed the harddrive.) Plug in an external harddrive or thumbdrive that can fit the 8 GB file we're about to create. Change the directory to the external drive and run:

```
$ dd if=/dev/sda of=qraat.img bs=4K
```

To copy the image, boot another computer in the same way. Plug in the external drive, change to its directory, and type

```
$ dd if=qraat.img of=/dev/sda bs=4K
```

Both of these commands should take about 30 minutes. Once you've finished copying the image, you'll want to edit the hostname and network interfaces as described in the preceeding section.

3.2 Networking

These instructions apply to our deployment at Quail Ridge. The ethernet interface needs to be configured to use the Qurinet router as its gateway. In /etc/network/interfaces, you'll find an interface that is commented out. Uncomment this and comment out the old one.

```
# Default interface in Quirinet
auto eth0
iface eth0 inet static
  address 10.20.1.55
  netmask 255.255.0.0
  gateway 10.20.1.1
```

The second part is to make sure the git repository is pointed to the right place. In /home/rmg/QRAAT/.git/config, you'll find a line that reads

```
url = christopher@10.253.1.55:work/QRAAT # change me!
```

Change this to the correct user and directory and you're set!

Appendices

A Configuration files

QRAAT metadata is stored in CSV-style files. Elements are seperated by commas and spaces are not allowed. The first row specifies the column parameters. Values may be ommited.

tx.csv. The transmitter file maps a name to frequency and also passes parameters to the pulse detector. These are described in detail in the backend software manual.

```
use,name,freq,type,pulse_width,rise_trigger,fall_trigger,filter_alpha
yes,robin1,164.42,Pulse,20,3.0,2.0,0.01
yes,robin1_heart,164.65,Continuous,,,,
no,noisy,164.5,Pulse,20,1.010,1.005,0.01
yes,test_tx,165.99,Pulse,25,3.00,1.15,.001
yes,mouse1,164.3,Pulse,25,3.0,2.0,0.01
```

sitelist.csv. The site list stores information about the nodes that is needed by the server.

```
name,comp_ip,power_ip,comp_outlet,rx_outlet,powertype,state
site0,10.0.0.55,10.0.0.56,1,2,webpowerswitch,down
site1,localhost,,,,,up
site2,10.2.1.55,10.2.1.56,1,2,netbooter,up
site10,10.10.1.55,10.10.1.56,2,3,pingbrother,up
site13,10.13.1.55,10.13.1.56,1,2,pingbrother,active
site20,10.20.1.55,10.20.1.56,1,2,pingbrother,down
site21,10.20.1.55,10.21.1.56,1,2,pingbrother,active
```

site.csv. Each field computer has a file that specifies its own configuration. This is necessary so that it can switch the receiver's power, as well as check its own goal state.

```
power_ip,rx_outlet,powertype,state
10.13.1.56,2,pingbrother,active
```

status.log. Each field computer keeps a log as a time stamp in seconds since the epoch and its status when checked.

```
timestamp,site_status
```

...

```
1366307941,up
1366312980,active
1366312996,up
1366313467,down(300)
1366313445,shutdown
1366323435,down(300)
1366323974,up
```

B Creating an encryption key

We use RSA encryption for communication between server and nodes for secure transmission as well as efficient SSH scripting, since RSA keys don't require a password. Here's how we made a key.

On the server:

```
$ ssh-keygen -t rsa
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/christopher/.ssh/id_rsa): [type "rmg_rsa"]
Enter passphrase (empty for no passphrase): [press enter]
Enter same passphrase again: [press enter]
... some output ...
$ mv rmg_rsa  /.ssh/rmg_rsa
$ scp rmg_rsa.pub rmg@siteN:rmg_rsa.pub
$ ssh rmg@siteN
```

On remote site N:

```
$ mkdir -p .ssh && mv rmg_rsa.pub .ssh
$ cat .ssh/rmg_rsa.pub >> .ssh/authorized_keys
```

Test setup (on RMG Server):

```
$ ssh -i .ssh/rmg_rsa rmg@siteN
$ rmg fetch siteN
```

C Suggested environment variable values

QRAAT's variables are defined in `rmg_env` and are included at the start of both `rmg` and `rmg-node`. The default values for these variables are given as follows:

```
RMG_SITE_METADATA_DIR="/home/rmg"
RMG_SITE_DET_DIR="/tmp/ramdisk/det_files"
RETRY_MAX=10
RMG_SERVER_SSH_KEYFILE="$HOME/.ssh/rmg_rsa"
RMG_SERVER_DET_DIR="$HOME/det_files"
RMG_SERVER_EST_DIR="$HOME/est_files"
RMG_SERVER_METADATA_DIR="$HOME"
```