Introduction

This document provides installation and configuration instructions for the Arednsig web application. The Arednsig web app provides a convient way to view charts of signal statistics for any calendar period for which data points are stored in the database. The app uses a round-robin database (RRD) that can be configured for any depth, limited only by file system storage capacity. The signal data can be accessed by any client with a web browser that is on the mesh network. The instructions given in this document apply to most Linux distributions. The author has successfully run this app on a Raspberry Pi 3 running the Raspbian operating system.

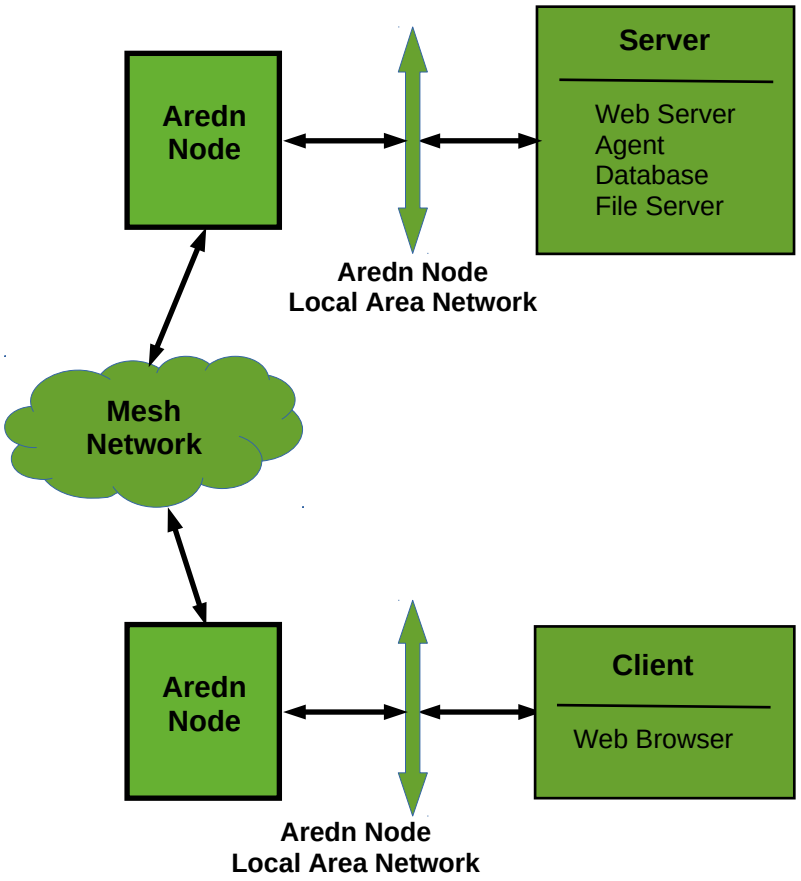

Figure 1. Overall conceptual view of client and server on AREDN mesh network.

AREDN Mesh System Overview

Figure 1 provides a conceptual overview of how the AREDN mesh provides network connectivity between client and server systems. Each node has two network interfaces: a wireless interface to other AREDN nodes, and a local area network interface to clients and servers. A node routes requests from clients on its LAN, via other nodes, to servers connected to the LAN's of other nodes. Refering to figure 1, a possible series of events goes as follows

1. The client sends, over the local node's LAN, a request for services available on a server elsewhere on the mesh network.
2. The local node routes the request, via other nodes, to the remote node where the server is located.
3. The remote node routes the request to the appropriate server on its own LAN.
4. The server sends, over the remote node's LAN, the reply to the requesting client.
5. The remote node routes the reply back to the local node.
6. The local node routes, over it's LAN, the reply back to the client.

Typically, the node's LAN is implemented by a Virtual Local Area Network (VLAN) capable switch, such as a Netgear GS105Ev2 switch. The node uses IEEE 802.Q VLAN identifiers to route traffic to devices on its own LAN, as well as routing external requests to an external network connect to a "WAN" port on the VLAN switch. An external request can be a request to any service on a host *not* on the LAN or somewhere else on the mesh network.

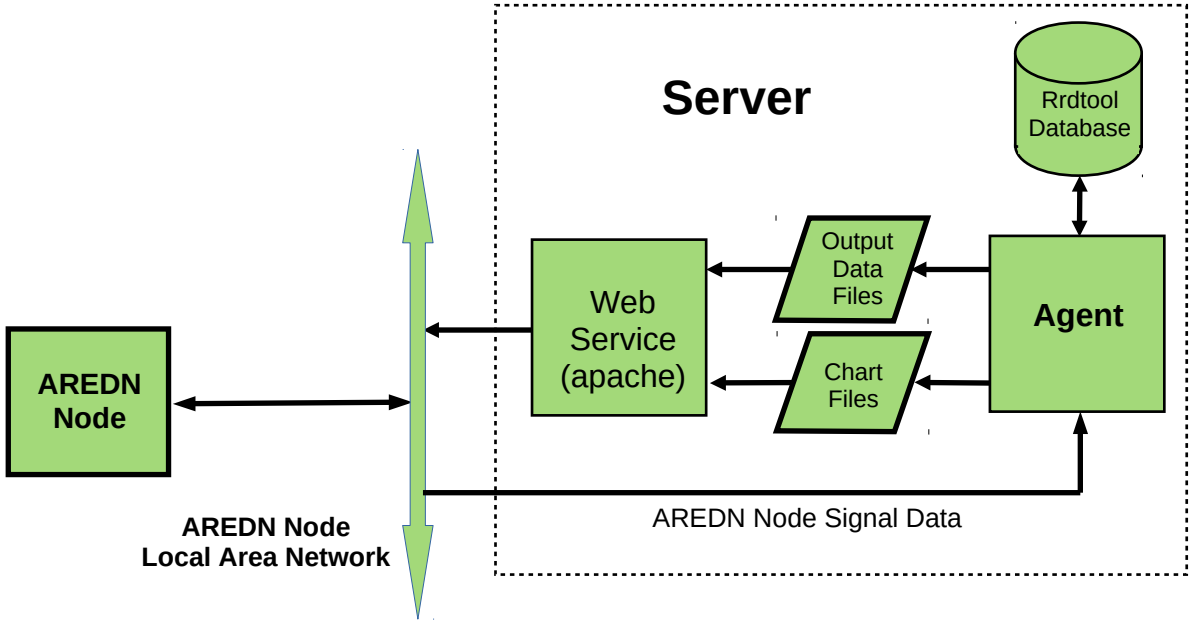

Figure 2. Block diagram of server software components showing data i/o, agent process, web service, and database elements.

Server Software

Referring to figure 4, the server software consists essentially of two components: a web server component and an agent component. The web server component includes a PHP script along with HTML documents containing embedded Javascript. The PHP script processes POST requests sent from the requesting client. The agent, a Python script, requests data from the local node, manages data conversion and re-formatting, updates a database, and generates stock charts.
Events flow in the following manner.

The agent periodically sends a request to the local node for signal data. The agent then converts specific data items to other formats where required. Selected data items get written to a round-robin database for permanent storage. Besides these functions, the agent manages generation of a stock set of charts for display in HTML documents. After formatting the node signal data, the agent writes the data to the *Output Data File* for use by HTML documents. When a client browser requests the Arednsig HTML document, Javascript embedded in the document reads the output data file and displays this data in an HTML document. The agent, as mentioned earlier, generates graphic charts for display in HTML documents. The graphic charts are stored as image files which Javascript in the HTML documents can load and display in the Arednsig web page.

When the users requests custom charts, the browser launches a PHP script on the server. In this case, the PHP script processes client requests for charts covering custom, user supplied dates. First the PHP script validates the user supplied date range against the date range of the data points stored in the RRD. The date range request by the user must be covered by the range stored in the database. Next the script fomats and runs an **rrdtool graph** command with the parameters to create charts covering the user supplied date range. Finally the created charts get downloaded to the requesting client.

Installing the Server Software

The arednsig application software should be installed on a Linux host which meets the following requirements:

- The server software should be installed on a recent Linux distribution such as Debian, Ubuntu, or Raspbian. (The author has successfully developed and installed the software on a Raspberry Pi running the Raspbian operating system.)
- Apache2 should be installed and configured to allow serving HTML documents from the user's public_html folder.
- PHP should be installed and configured to allow running user PHP scripts from the user's public_html folder.
- Rrdtool should be installed - type **sudo apt-get install rrdtool**
- Python 2.7 usually comes pre-installed in virtually all Linux distributions. Type "python" at a command line prompt to verify Python has been installed.
- See the Appendix for detailed configuration notes.

Software Inventory
The following software items in the install zip file need to be installed on the server

HTML folder:
- index.html
- arednsig.php
- arednsig.html
- static/chalk.jpg

bin folder:

- createArednsigRrd.py
- arednsigAgent.py
- ardstart
- ardstop

Getting the Arednsig Software

From the github repository:

1. On a computer connected to the Internet download the zip file from

   https://github.com/fractalxaos/ham/archive/master.zip

2. From the downloaded file “ham-master.zip”, extract the “arednsig” folder to the desktop.

From the Willamette Valley AREDN mesh network:

1. On a computer connected mesh network download the zip file from

   http://ka7jlo-raspi1.local.mesh/file-manager/files/KA7JLO/Apps/arednsig.zip

2. Save the zip file to the desktop or downloads folder.
3. From the downloaded file “arednsig.zip”, extract the “arednsig” folder to the desktop.

Installation

Note that the following installation procedure assumes that the document root for the arednsig HTML documents will be the user’s public_html folder. Typically the full path name to this folder will be something like “/home/{user}/public_html”.

1. If it doesn’t already exist, use **mkdir** to create a folder **public_html** in the user’s home folder. The home folder is of the form **/home/{user}**, where **{user}** is the user name of the account hosting the web documents, for example, **/home/pi**.

2. In the **public_html** folder, use **mkdir** to create a folder **arednsig** to contain the ardensig HTML and PHP files.

3. In the **arednsig** folder created in step 2, use **mkdir** to create a folder **dynamic** to contain dynamic content used by arednsig html documents.

4. From the **html** folder in the dowloaded **arednsig.zip** file, move all files and folders to the folder created in step 2.

5. The output data files (see figure 2) get overwritten frequently; similarly the chart files get overwritten frequently. On SD card systems, such as a Raspberry Pi, it is inadvisable to do frequent writes to any file system mounted on the SD card. To use the ram based temporary file system (tmpfs) to store these files, continue

with step 6. Otherwise, to write dynamic content to the disk drive, continue with step 10.

6. Assure that the server **/tmp** folder gets mounted to the temporary file system. This can be done by adding the following line to the **/etc/fstab** file

   tmpfs /tmp tmpfs nodev,nosuid,size=50M 0 0

7. The web service cannot freely access files and folders outside of the public_html document root folder. Therefore, the dynamic folder must be bound to the temporary file system by running the following commands

   mkdir /tmp/arednsig
   chmod g+w /tmp/arednsig
   sudo mount --bind /tmp/arednsig /home/{USER}/public_html/arednsig/dynamic
   sudo chown :www-data /tmp/arednsig

8. The above four commands may be placed in a startup shell script and run at boot up time by launching the startup script with the **su** command from **/etc/rc.local.** For example, place the above commands in a script **/home/pi/bin/startup.sh** and place the following line in the **/etc/rc.local**.

   (su - pi -c "bin/startup.sh")&

   Whenever the host boots up, the **startup.sh** script will run the commands in step 7. Note that, although requiring superuser permissions, the above provides a far more secure environment than merely enabling write permission for everyone.

9. Alternatively you can enable apache2 access to files in the the tmpfs **/tmp** folder by backing up and then modifying the file **/lib/systemd/system/apache2.service** Replace the line “#PrivateTmp=true” with the line “PrivateTmp=false”. Note that it is a good idea to use comments to indicate where and why a change has been made to a configuation file. For example,

   ```
   # changed 2020-01-15 by JLO [my initials] to enable apache to follow
   # symlinks to the /tmp folder in tmpfs
   #PrivateTmp=true
   PrivateTmp=false
   ```

   Reload system deamons by running

   sudo systemctl daemon-reload

   Restart apache2 service by running

   sudo systemctl restart apache2

10. If it does not already exist, use **mkdir** to create a folder named **bin** in the user home folder. For example, the full path name should look like **/home/pi/bin**.

11. From the **bin** folder in the dowloaded arednsig folder, move all files to the folder created in step 10. In most Linux installations the user's bash profile will automatically add the user's bin folder to the command search path. If such is the case, then the agent can be started up by simply typing **arednsigAgent.py** followed by ENTER. For example, typing **arednsigAgent.py -v** will launch the agent in verbose debug mode.

12. In the user home folder, create a folder named **database**. In the **bin** folder run the python script **createArednsigRrd.py**. Running this script creates an empty round robin database file where the agent will periodically store signal data from the AREDN local node.. This script should be run once and then kept in a secure place. Running it accidentally at some future date will result in *total loss of all previously stored data*.

13. In the user's home folder create a folder named **log**. This is where the agent will keep its error logs.

14. For convenience two scripts have been provided to make it easy to turn the agent on and off. The **ardstart** script starts up the agent and causes all diagnostic output and error messages to be written to a log file in the log folder. The **ardstop** stops the agent from running. Start the arednsig agent by running **ardstart**. Alternatively the **ardstart** command can be placed in the **startup.sh** script mentioned in step 8 to automatically start the agent when the host boots up.

This completes installation of the server software.

References and Resources

The following resources describe how to configure a Raspberry Pi to be a server

- Using an SSH key pair: https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
- Reducing wear on the SD card: https://www.zdnet.com/article/raspberry-pi-extending-the-life-of-the-sd-card/
- Installing a web server: https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md
- Linux system administration and devops https://www.guru99.com/unix-linux-tutorial.html

The following tutorials are useful for more in depth understanding of the server software:

- Javascript http://www.w3schools.com/js/default.asp
- PHP http://www.w3schools.com/php/default.asp
- HTML http://www.w3schools.com/html/default.asp
- Rrdtool http://oss.oetiker.ch/rrdtool/
- Python http://greenteapress.com/thinkpython/thinkpython.html

Appendix

The following steps describe how to build a web server on a Raspberry Pi. The steps below configure the Raspberry Pi to serve web pages from the user's document root folder. When user html folders are enabled, Apache looks for the files in the user's **public_html** folder. For example, a browser request **http://raspi.local/~pi/myDocument.html** would cause the host **raspi.local** to look for **myDocument.html** in the folder **/home/pi/public_html**. With this setup documents can also be served from the host's **/var/www** folder. For example, **http://raspi.local/aDocument.html** would cause Apache to look for **aDocument.html** in the folder **/var/www/html**.

1. If upgrading from a previous Raspbian version, first backup the user home folder by running

   zip -r pi_bak.zip /home/pi/*

   Copy the **pi_bak.zip** file to a thumb drive or some other storage media.

2. Copy the new Raspbian OS disk image to the SD card. Follow raspbian instructions for copying the disk image to the SD card.

3. On the router turn off any port forwarding to the raspberry pi. This is a temporary security precaution until ssh keys are installed.

4. Attach a monitor, keyboard, and mouse. Run the command

   sudo raspi-config

   and change the following items

   hostname {your host name}
   password {your password}
   ssh ON
   region US
   timezone {your timezone}
   keyboard US

5. If not already connected, connect the raspberry pi to your LAN.

6. Use ssh to shell to user pi with the password set in step 4

   ssh pi@{your host name}.local

7. Update the package database and install vim

   sudo apt-get update
   sudo apt-get install vim

vim is an editor that makes it easy to make changes in configuration files. Alternatively you can use **nano** or some other terminal based editor of your choice.

8. On the client computer, create an ssh key pair. Create in the user home folder a folder named **.ssh**. In the **.ssh** folder create a file named **authorized_keys**, and copy the public key to it.

9. If they will not be used, disable WiFi and Bluetooth. Backup and then modify **/boot/config.txt** by adding the following two lines to the end of the file:

```
dtoverlay=pi3-disable-wifi
dtoverlay=pi3-disable-bt
```

and run the once off command:

```
sudo systemctl disable hciuart
```

10. Backup and then modify **/etc/ssh/sshd_config** as follows

```
#PermitRootLogin prohibit-password
PermitRootLogin no
⋮
#PasswordAuthentication yes
PasswordAuthentication no
⋮
#X11Forwarding yes
X11Forwarding no
```

The above enhances security when using secure shell (ssh) to access the raspberry pi.

11. Setup the temporary file system (tmpfs) by backing up and then modifying **/etc/fstab**. Add the following lines to the bottom of the file.

```
# uncomment if needed for web appstial logs in ram to reduce
# stress on the SD card due to frequent writes.
tmpfs /tmp tmpfs nodev,nosuid,size=20M 0 0
tmpfs /var/tmp tmpfs defaults,noatime,nosuid,size=20m 0 0
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,size=20m 0 0
tmpfs /var/spool/mqueue tmpfs
defaults,noatime,nosuid,mode=0700,gid=12,size=20m 0 0
```

This highly recommended step configures the raspberry pi to store all log files and temporary files in RAM. Unless configured to use an external hard drive, the raspberry pi mounts the root file system to the SD card. Log files and temporary files are frequently written to the file system, resulting in wear on the SD card. Storing these files in RAM saves the SD card from such wear. Note that all log

files and temporary files are lost upon reboot or power down.

12. Reboot the raspberry pi.

13. Optionally run all software updates

    sudo apt-get upgrade
    sudo reboot

14. Backup **/etc/rc.local** and then add the user start up script. For example add the following line to **/etc/rc.local**

    (su - pi -c "bin/startup.sh")&

15. Install rrdtool

    sudo apt-get install rrdtool

    rrdtool maintains round-robin databases and generates charts for display in web pages. **rrdtool** can run on the same host as **Maria (mySql)**. However they are not interoperable with each other.

16. Install LAMP. This is the standard Linux web server “stack”.

    Apache2
    ======
    sudo apt-get install apache2 -y
    sudo a2enmod rewrite
    systemctl restart apache2

    PHP
    ===
    sudo apt-get install php libapache2-mod-php -y
    systemctl restart apache2

    MySQL
    =====
    sudo apt-get install mariadb-server mariadb-client php-mysql -y
    systemctl restart apache2

17. Backup and then modify **/etc/apache2/mods-available/userdir.conf**. For example,

    # changed 2020-01-15 by JLO to allow user .htaccess file
    #AllowOverride FileInfo AuthConfig Limit Indexes
    AllowOverride All

    This enables apache to use the **.htaccess** file in the user’s document root folder.

18. Enable user directories in apache2 by running

    a2enmod userdir

19. Backup and then modify **/etc/apache2/mods-available/php7.3.conf** to allow PHP scripts to run in user directories by commenting the lines at bottom of file. For example,

```
# changed 2020-01-15 by JLO to enable user php scripts
#<IfModule mod_userdir.c>
#    <Directory /home/*/public_html>
#        php_admin_flag engine Off
#    </Directory>
#</IfModule>
```

20. Backup and then modify **/etc/apache2/sites-available/000-default.conf**. For example,

```
# changed 2020-01-15 by JLO to make user
#  pi the html document root
#DocumentRoot /var/www/html
DocumentRoot /home/pi/public_html
```

    This makes the pi user public_html folder the document root for the raspberry pi. For example a client request **http://raspi.local/myDocument.html** would cause Apache to look for the document in the folder **/home/pi/public_html**.

21. Backup and then modify **/etc/apache2/envvars** to create apache2 logs in tmpfs. Add the following lines at the top of the file, for example,

```
# added 2020-01-15 by JLO to allow apache to use tmpfs
if [ ! -d /var/log/apache2 ]; then
mkdir /var/log/apache2
fi
if [ ! -d /var/log/mysql ]; then
mkdir /var/log/mysql
fi
```

    Without these lines Apache will fail to start when the system reboots. When the system reboots the folders **/var/log/apache2** and **/var/log/mysql** will not exist if logs are using the temporary file system in RAM. Apache will fail to start if these folders are not present. Adding the above lines causes Apache to create these folders when the system boots up.

22. Enable Apache to access files in the the tmpfs **/tmp** folder by backing up and then modifying **/lib/systemd/system/apache2.service**.

```
# changed {date} by {name} to enable apache to follow
# symlinks to the /tmp folder in tmpfs
```

#PrivateTmp=true
PrivateTmp=false

This change enable Apache to follow symbolic links in the user document root (public_html) folder to folders and files in the temporary file system. Making this change eliminates the need to bind the /tmp folder to the dynamic folder, as described in step 7 in the software installation section above.

Reload system deamons

sudo systemctl daemon-reload

Restart apache2 service

sudo systemctl restart apache2

23. Reboot the raspberry pi.

24. Copy the backup file **pi_bak.zip** from the thumb drive media to the **/home/pi** folder. Restore desired files and folders from backup archive by running

    unzip pi_bak.zip

    Use **mv** to move folders and files to their appropriate locations.

25. Test all the above modifications.