Watcher User's Guide

Geoff Lawler <geoff.lawler@cobham.com>
Michael Elkins <michael.elkins@cobham.com>

January 26, 2010

Contents

1 Introduction

The *watcher*, or *watcher system* allows users to visualize emulated Mobile As-Hoc Networks.

2 General Watcher Information

2.1 Watcher Architecture, Communications, and Component Interactions

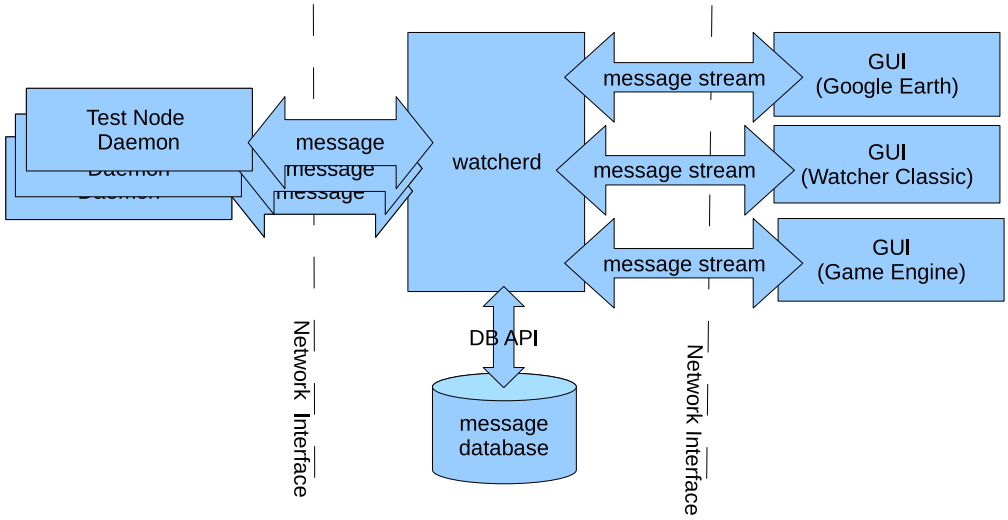

Figure 1: The basic architecture of the watcher system.

Figure 1 shows the three main components in the watcher system: test node daemons (or "feeders"), a watcher daemon, and a number of graphical user interfaces (or "GUIs"). The basic operation is this: the test nodes feed state information to a running instance of the watcher daemon, *watcherd*. The watcher daemon saves all the state information locally to a database. A GUI connects to a running watcher daemon and requests a stream of state information, then displays it using some (most likely) graphical mechanism. The daemon acts as a message cache and demulitplexer, streaming multiple message streams from multiple test nodes into a single stream of messages.

The interface between the watcher daemon and the test node is a simple message interface. The test node daemon connects to a daemon, sends one or more messages, then disconnects. Some messages contain abstract state information about that test node such as current set of neighbors, current location, etc. Other messages contain lower-level graphical display commands like "change color" or "draw an edge between these two nodes". For a list of all supported message, see the *watcher::event::Message* class in the watcher developer documentation.

The interface between the GUI(s) and the watcher daemon is also message based, but also includes the concept of a message stream. The stream is composed of messages and has an interface that manipulates the stream content, time, direction, and rate. A message stream can start at any time t for which there is data. The stream rate can be set as a multiplier of real time. For instance a GUI can request that the

stream go 10x the speed at which the messages were received. (This only will work of course if there is a cache of messages that can be sent at 10x - once the message cache is exhausted, the stream is sent at the rate the messages are received, 1x real time.) The concept of manipulating a message stream by rate and time allows GUIs to act as test bed "TiVO"s, playing back an arbitrary stream of test node data in reverse, forward, jumping around the stream as the user requests.

Depending on which components are running and how they are started, the watcher system can be said to be running in different modes: a "live" mode, a "playback" mode, and a "record" mode. In all modes, a *watcherd* instance must be running. These modes can be moved into and out of in real time depending on the needs of the user. In record mode, the test nodes and a watcher daemon are run. The test nodes send state data to the daemon, the the daemon writes those messages to a local database. In live mode, the test nodes are feeding data to the daemon (which writes the information to a database) and some number of GUIs connect to the daemon and request a message stream. Note that the GUIs can request messages from the start of the test run - they are not constrained to "live" messages; live mode simply means that there are, at that moment, new messages arriving at the daemon and being recorded. The last mode is playback mode and does not involve test node daemons. In this mode, the watcher daemon is started with an existing database full of messages and it just waits for connections from a GUI. When a GUI connects, the watcher daemon streams it the requested messages, this "playing back" a "recorded" test run.

2.2 Obtaining and Building Watcher

2.2.1 Obtaining

The watcher is GPL'd software although it is not currently publically available. This will change in the near future. Future versions of this document will be updated after the watcher is publically released giving the download location. In the meantime please contact the authors of this document to obtain instructions on getting the watcher.

The watcher system is written in C++.

2.2.2 Repository Structure

The watcher system uses the GNU auto-tools build system for most components. (GUIs are an exception as the GUI libraries used, generally have their own build systems, qmake for QT, etc). The source code for all watcher components is in ./src. The basic layout of the source tree is given below. Note that only the "important" directories are listed to minimize clutter.

./src Holds all watcher system source code.

- **./src/clients** Anything that is a client of the watcher daemon, including GUIs and test node daemons, is in here.
- **./src/doc** Holds the LaTeX code for generating Doxygen based watcher documentation including the Watcher User's Guide.
- **./src/idsCommunications** Holds the code for building the (optional) hierarchy communications library which the watcher uses to talk to hierarchy software running on the test nodes. See section 3.2.2 for details.
- **./src/libwatcher** Holds the source code for two libraries used in the watcher system: libwatcher, which is a generic communications mechanism used by watcher components to talk to one another,

and libwatchermessage, which contains the code for all watcher messages and a mechanism to parse them.

./src/logger The code the watcher system uses for logging. This is a small wrapper around calls to log4cxx, a standard logging library from the good people at Apache.

./src/util Code for a utility library used by some watcher components.

./src/watcherd The code for the watcher daemon, *watcherd*.

./doc Watcher documentation, including basic architecture documents and the location where the Doxygen generated developer's guide is when it is built.

./doc/html The html based developer's guide. (Must be built - `make -C ./src doc` to do so.)

./doc/man If configured, this is the location of the man pages built by `make -C ./src doc`

./doc/latex If configured, this is the location of the latex-based developer's guide built by `make -C ./src doc`

./tars Some source code (tar'd up) for third party libraries used by the watcher system including **log4cxx** (and its required libraries **apr and apr-util), libconfig, libidmef, and libqwt.**

2.2.3 Dependencies

The watcher system has the following dependencies:

Third Party Watcher uses a number of third party libraries.

sqlite 3 The watcher daemon uses sqllite for all database operations.

libconfig 1.3.1 The system uses this for creating, reading, and writing to configuration ("cfg") files.

log4cxx 0.10.0 (This depends on lib apr 1.3.3 and lib apr util 1.3.4). This is used for all logging in the watcher. It is a clone of Java's log4j logging package.

boost 1.38 The watcher makes extensive use of Boost. This C++ library used more many things: serialization, network code, regular expressions, etc.

Legacy Watcher The "legacy watcher" is the Qt/OpenGL based GUI. If not building the legacy watcher, these libraries are not needed.

Qt 4.3 Qt is used by the legacy watcher to supply the GUI widgets (windows, frames, menus, buttons, etc) and to wrap the OpenGL which displays the MANET. If not building the legacy watcher, Qt is not needed.

qwt-5.1.1 A small Qt-based library used to draw the 2d scrolling graphs in the legacy watcher. If not building the legacy watcher, Qwt is not needed.

OGRE Watcher The OGRE watcher is an OGRE-based GUI. If not building the OGRE watcher GUI, these libraries are not needed.

OGRE The ogreWatcher uses the Object-Oriented Graphics Rendering Engine (OGRE) for graphics.

OIS The ogreWatcher uses the Object Oriented Input System (OIS) for mouse and keyboard input.

CEGUI The ogreWatcher uses Crazy Eddie's Graphical User Interface (CEGUI) for GUI widgets.

Watcher3D The Delta-3D based watcher GUI. If not building the watcher3d, these libraries are not needed.

delta 3d Delta 3d is used for graphics rendering in watcher3d.

Internal These are the libraries that watcher uses internally.

libwatcherutil Holds a few things that most components need, like parsing command line args to get to the cfg file.

liblogger A small wrapper around log4cxx to abstract which logging library we use.

libwatcher The core of the watcher messaging system, the watcher APIs between test node daemons, a watcher daemon, and the GUI(s)s.

2.2.4 Building

First make sure that the dependencies above are built and installed on the build machine. For most dependencies you can use the package management system to install the dependencies. For example, on Fedora 11, the command `yum install boost-devel boost qt qt-devel log4cxx log4cxx-devel libconfig libconfig-devel ...` will install the required dependencies. Some libraries may not be widely available (like libidmef), in these cases check in the `tars` directory. The source code may be there.

Once the required dependencies are installed, build the core watcher components (libwatcher, watcherd, and test node binaries). These are auto-tools based. It should be as easy as the canonical. It should be as easy as the canonical `./configure && make && sudo make install`. (If ./configure does not exist, run `./autogen.sh` to generate it.

./configure does have a few watcher-specific arguments:

–**enable-testnodeonly** Only configure and build test node components. Do not build the watcher daemon.

–**enable-ogreWatcher** Enable compilation of ogreWatcher.

–**enable-watcher3d** Enable compilation of watcher3d GUI.

–**enable-hierarchy_client** Enable compilation of the watcher hierarchy client (both libidsCommunications and watcherHierachyClient).

Both the `ogreWatcher` and `watcher3d` are built as part of the normal build if the appropriate arguments, `--enable-ogreWatcher` and `--enable-watcher3d` respectively are passed to ./configure at configure time.

The legacy watcher is built using Qt's make system, `qmake`, to generate normal Makefiles. So it is not part of a normal build. To build it, change directories to ./src/clients/legacyWatcher and type `qmake && make && sudo make install`.

2.3 Log Property Files

Most components in the watcher system use log properties files, usually named `log.properties`. These files control the amount, location, and even format of logging in the watcher system. They are based on

standard Apache log4j logging properties files. There are sample log properties files in ./etc. As part of the build process `log.properties` will be created in most directories where they are needed. A full explanation of all logging options in a `log.properties` file is beyond the scope of this document. See `http://logging.apache.org/log4j` for more information.

The watcher system supports the following log levels: `trace`, `debug`, `info`, `warn`, `error`, and `fatal`. All functions in the watcher system log a trace message when they enter and exit the function, so a full execution trace is possible.

Figure 2 gives a sample log.properties file. This file sets the default log level to `debug` and the level to `trace` for the ClientConnection class. The log output goes to both the console (stdout) and a local file. The file is overwritten every time the component restarts.

```
# Global logging level.  Create two loggers:  stdout and flog log4j.rootLogger=debug,
flog, stdout

# trace the Client Connection for debugging.
log4j.logger.ClientConnection=trace

# stdout is set to be a ConsoleAppender, i.e.  append to the console.
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%-5p] (%F:%L) - %m%n

# flog is set to be a non-appending file at ./watcherd.log
# don't append to the log file.
log4j.appender.flog=org.apache.log4j.FileAppender
log4j.appender.flog.layout=org.apache.log4j.PatternLayout
log4j.appender.flog.file=watcherd.log
log4j.appender.flog.append=false
log4j.appender.flog.layout.ConversionPattern=%-4r [%t] %-5p %c (%-12F:%-4L) - %m%n
```

Figure 2: Sample `log.properties` file

2.4 Configuration (cfg) Files

Most watcher components store their configurable parameters in configuration files. While the content of the configuration files varies, the format is basically the same across the components as they all use the same library, *libconfig*. system users. The library used is libconfig. See http://www.hyperrealm.com/libconfig for complete documentation about the format syntax of the configuration files. The files are human readable and are designed to be easily edited by watcher system users. Most components only have a few parameters that can be set, address of the server, etc. For these simple cases the syntax is a basic `name = ''value''` pair. See figure 3 for a sample configuration for the watcher daemon.

Most components in the system take the location of the configuration file on the command line given with a `-c` or `-f` option. If you're unsure which to use, all components support `--help`, which will show a usage statement. By convention watcher configuration files follow the form *program-name.cfg*. So the configuration

for the ogreWatcher binary would be called `ogreWatcher.cfg`. This is not enforced, it's just a convention - "cfg" files can be called anything.

The config files are created two ways. The build process will copy sample "cfg" files from ./etc into the local directory. Also, most watcher components will create a default configuration file if they find an empty or non-existent one when they start up. So if you don't have a configuration file when you need one, you may just be able to run the component, then stop it and a default configuration file will be created in the current working directory. Note though that sometime the defaults are not what you want, so you'll have to modify the defaults by hand.

```
logPropertiesFile = "watcherd.log.properties";
server = "glory";
port = "8095";
serverThreadNum = 8;
databaseName = "event.db";
databasePath = "event.db";
```

Figure 3: Sample `cfg` file

For the specific settings in a component's configuration file, see the section in this document on that component.

3 Test Node Components

3.1 Scripting Interface

For each watcher message there is a command line binary to send that message. These binaries can be used directly in shell scripts or invoked via a system call from most scripting languages to send an instance of that message to a running watcher daemon instance. Each binary allows the user to specify the content of the message and the daemon instance to send the message to.

In many cases, the node that the message "comes from" can be set as well. This allows a user on any machine that can connect to a watcher daemon, the ability to modify nodes, edges, labels, etc of any test node. This is useful for debugging or real time modification of an aspect of the test bed. For instance a single machine could monitor traffic rates between nodes and update the edges between those nodes with the current traffic rate.

The available commands are:

- sendColorMessage (page 11)
- sendEdgeMessage (page 12)
- sendGPSMessage (page 13)
- sendConnectivityMessage (page 14)
- sendDataPointMessage (page 15)

- sendLabelMessage (page 16)
- showClock (page 17)

The following pages give details for each command.

3.1.1 sendColorMessage

sendColorMessage is a test node command line program that sends a ColorMessage message to the watcher daemon, specifing that a node should change it's color.

Usage: `sendColorMessage -s server -c color [optional args]`

Required Arguments:

- `-c, --color=color`, The color of the node. Can be ROYGBIV or RGBA format, string or hex value. Supports transparency.
- `-s, --server=address`, The addres of the node running watcherd

Optional Arguments:

- `-n, --node=address`, The node to change color, if empty the local node's address is used
- `-f, --flash=milliseconds`, Flash between the new color and the orginal color every milliseconds seconds, 0 for no flash.
- `-x, --expiration=seconds`, How long in seconds to change the color. 0==forever
- `-p, --logProps`, log.properties file, which controls logging for this program
- `-h, --help`, Show help message

Examples:

- This tells the GUI(s) that are listening to the daemon running node *glory* the node at 192.168.1.101 should be drawn in blue:

  ```
  sendColorMessage -s glory -c blue -n 192.168.1.101
  ```

- This tells the GUI(s) that are listening to the daemon running node *glory* the node at 192.168.1.101 should be drawn in a transparent blue. Transparent format is R.G.B.A, where A is alpha transparency:

  ```
  sendColorMessage -s glory -c 0.0.255.64 -n 192.168.1.101
  ```

- This tells the GUI(s) that are listening to the daemon running node *glory* the node at 192.168.1.107 should be drawn in green for 5 seconds:

  ```
  sendColorMessage -s glory -c green -n 192.168.1.107 --expiration 5000
  ```

- This tells the GUI(s) that are listening to the daemon running node *glory* the node at 192.168.1.104 should flash for 10 seconds:

  ```
  sendColorMessage --server glory --color green --node 192.168.1.104 --flash --expiration
  10000
  ```

3.1.2 sendEdgeMessage

sendEdgeMessage is a test node command line program that sends a GPSMessage message to a watcher daemon, specifing a node's current GPS coordinates.

Usage: `sendEdgeMessage -s server -t tail [optional args]`

Required Arguments:

- `-s, --server=address`, The address or name of the node running watcherd to which the message is sent.
- `-t, --tail=address` The node to attach the tail of the edge to. If no head is given, the local node is used.

Optional Arguments:

- `-h, --head=address` The node to attach the head of the edge to.
- `-c, --color=color` The color of the edge. Can be ROYGBIV or RGBA format, string or hex value. Supports transparent colors.
- `-w, --width=width` The width of the edge in some arbitrary, unknown unit.
- `-y, --layer=layer` Which layer the edge is on in the GUI.
- `-d, --bidirectional=bool` Is this edge bidirectional or unidirectional. Use 'true' for true, anything else for false.
- `-l, --label=label` The text to put in the middle label (This program only supports creating a middle label, although the message supports labels on node1 and node2 as well. May add that later)
- `-f, --labelfg=color` The foreground color of the middle label. Can be ROYGBIV or RGBA format, string or hex value. Supports transparent colors.
- `-b, --labelbg=color` The background color of the middle label. Can be ROYGBIV or RGBA format, string or hex value. Supports transparent colors.
- `-z, --fontSize=size` The font size of the middle label
- `-x, --expiration=seconds` How long in milliseconds to diplay the edge
- `-p, --logProps` log.properties file, which controls logging for this program

Examples:

- Draw an edge between node 101 and node 102 on the "QoS" layer and make the color a translucent red.

  ```
  sendEdgeMessage -s glory -h 192.168.1.101 -t 192.168.1.102 -l QoS -c 255.0.0.64
  ```

3.1.3 sendGPSMessage

sendGPSMessage is a test node command line program that sends a GPSMessage message to a watcher daemon, specifing a node's current GPS coordinates.

Usage: `sendGPSMessage -s server -x value -y value -z value [optional args]`

Required Arguments:

- `-s, --server=address`, The address or name of the node running watcherd to which the message is sent.
- `-x, --latitude=value`, The latitude of the node.
- `-y, --longitude=value`, The longitude of the node.
- `-z, --altitude=value`, The altitude of the node.

Optional Arguments:

- `-n, --fromNode=address|name`, The node that the coordinates refer to. If not given, assume the local node.
- `-l, --logProps`, log.properties file, which controls logging for this program
- `-h, --help`, Show help message

Examples:

- This tells the GUI(s) attached to the watcherd on glory that node 192.168.1.101 is now at 79.23123123, 43.123123123, 20

  ```
  sendGPSMessage -s glory -n 192.168.1.101 -x 79.23123123 -y 43.123123123 -z 20
  ```

- This tells the GUI(s) attached to the watcherd on glory that the local node is at 0.0123123 0.123123123 123

  ```
  sendGPSMessage --server glory -n 192.168.1.101 -x 0.0123123 -y 0.123123123 -z 123
  ```

3.1.4 sendConnectivityMessage

sendConnectivityMessage is a test node command line program that sends a watcher::event::ConnectivityMessage message to the watcher daemon, specifying the current list of neighbors that the node has. The GUI(s) that are listening to that daemon, then draw the neighbors in a way that is relevent for that particular GUI.

Usage: `showColor -s server [optional args] nbr1 nbr2 nbr3 ...  nbrN`

Required Arguments:

- `-s, --server=address`, The addres of the node running watcherd.
- `nbr1 nbr2 nbr3 ...  nbrN` - the list if neighbors by ipaddress.

Optional args:

- `-l, --layer=layer`, the layer that these neighbors should show up on when displayed in the GUI(s).
- `-p, --logProps=log.propertiesFile`, the log properties file to use.
- `-f, --fromNode=fromNodeAddr`, the node that has these neighbors, if not given the local node is assumed.

Examples:

- This tells the GUI(s) that are listening to the daemon running on 'glory' the local test node has neighbors 192.168.1.101 and 192.168.1.102

  ```
  sendConnectivityMessage -s glory 192.168.1.101 192.168.1.102
  ```

- This tells the GUI(s) that are listening to the daemon running on 'glory' the local test node 192.168.1.101 has neighbor nodes 192.168.1.110-192.168.1.115 and they should be displayed on the "children" layer. (Note that 192.168.1.11{0..5} is a bashism which expands to the sequenctial list of nodes 192.168.1.110-192.168.1.115.)

  ```
  sendConnectivityMessage -s glory -l children -f 192.168.1.101 192.168.1.11{0..5}
  ```

3.1.5 sendDataPointMessage

sendDataPointMessage is a test node command line program that sends a DataPointMessage message to the watcher daemon, specifing a set of timestamped data point(s) for the node. The data point is labeled with a string saying what the data points represent. The GUI(s) then display this information is some way. In the case of the legacy watcher [5.1], 2D scrolling graphs are displayed showing past data points. The legacy watcher assumes that data points are given once a second.

Usage: `sendDataPointMessage -s server -g name [optional args] -d dp1 -d dp2 ...  -d dpN`

Required Arguments:

- `-s address|name`, The address or name of the node running watcherd.
- `-g name`, the "graphname" of the data - what the data is measuring.
- `-d datapoint`, a single data point measuring something.

Optional args:

- `-n, --node=address`, the node the data is from
- `-h, --help`, Show help message

Examples:

- Update the watcher about the current CPU usage on the machine 192.168.1.105.
  `sendDataPointMessage -s glory -g "CPUUsage" -n 192.168.1.105 -d .45432`
- Update the watcher about the current number of user's logged in to the local machine.
  `sendDataPointMessage -s glory -g "LoggedInUsers" -d 23`
- Update the watcher about the local node's current level of self satisifation.
  `sendDataPointMessage -s glory -g "BoyAmIGreatLevel" -d 23`

3.1.6 sendLabelMessage

sendLabelMessage is a test node command line program that sends a watcher::event::LabelMessage message to the watcher daemon, specifing that a label should be attached to the specified node (or float if given coords).

If address is specified, the label will attach to the node with that address. If cooridinates are specified, the label will float at those coordinates. The node address takes precedence. If neither option is specified, the label will attach to the node from which the message saw sent.

Usage: `sendLabelMessage -s server -l label [optional args]`

Required Arguments:

- `-l, --label=text`, The text of the label
- `-s, --server=address`, The address—name of the node running watcherd, the server.

Optional args:

- `-n, --node=address`, The node to change color, if empty the local node's address is used
- `-x, --latitude=coord` The latitude to float the node at.
- `-y, --longitude=coord` The longitude to float the node at.
- `-z, --altitiude=coord` The altitude to float the node at.
- `-t, --fontSize=size` The font size of the label
- `-f, --foreground=color` The foreground color of the label. Can be ROYGBIV or RGBA format, string or hex value.
- `-b, --background=color` The background color of the label. Can be ROYGBIV or RGBA format, string or hex value.
- `-e, --expiration=seconds` How long in millisecond to diplay the label
- `-r, --remove` Remove the label if it is attached
- `-L, --layer=layer` Which layer the label is part of. Default is "Physcial".
- `-x, --expiration=seconds`, How long in seconds to change the color. 0==forever
- `-p, --logProps`, log.properties file, which controls logging for this program
- `-h, --help`, Show help message

Examples:

- sendLabelMessage -s glory -n 192.168.1.102 -l "Correlation Layer" -e 1500 -f red -b green -L Correlation
- sendLabelMessage -s glory -n 192.168.1.102 -l "Physical Layer" -e 1500 -L Physical
- sendLabelMessage -s glory -n 192.168.1.104 -l "Attack Detected" -f yellow -b blue -L Physical

3.1.7 showClock

showClock is a command line program that "draws" a clock by arranging a set of nodes and edges into the shape of an analog clock. The "clock" is updated once a second to move "the hands" of the clock around. This program is mostly used to test the TiVO-like functionality built into the watcher system. But it can also be used to simply test if the watcher system is working properly. It does not need to be run on a test node - it is generally run on the same machine as the watcher daemon is running, but of course does not have to be. An example can be seen in Figure 4 on page 21.

Usage: `showClock -s server [optional args]`

Required Arguments:

- `-s, --server=address|name`, The address or name of the node running watcherd

Optional Arguments:

- `-r, --radius`, The radius of the clock face in some unknown unit
- `-S, --hideSecondRing` Don't send message to draw the outer, second hand ring.
- `-H, --hideHourRing` Don't send message to draw the inner, hour hand ring.
- `-p, --logProps=file`, log.properties file, which controls logging for this program
- `-e, --expireHands` When drawing the hands, set them to expire after a short time.
- `-h, --help`, Show help message

Examples:

- This shows a clock with a radius or 10 units.

  ```
  showClock -s glory -r 10
  ```

- This shows a clock with a radius or 10 units, but no outside minute ring and the edges which make up the hands are refreshed every second.

  ```
  showClock --server glory --radius 10 --hideHourRing --expireHands
  ```

3.2 Test Node Daemons

These daemons are run on the test nodes and feed information about the test nodes to the watcher daemon. This information is then streamed to GUI(s) attached to the daemon, which display it.

3.2.1 GPS Feeder

The GPS Feeder, *gpsFeeder.py*, updates the watcher daemon on the current location of a single node on the test bed. It is usually run on the test node itself, thus every test node updates its own location. The GPS is read from a locally running *gpsd* process. (See http://gpsd.berlios.de for details of *gpsd*.) The watcher GPS Feeder can be found at ./src/clients/gpsFeeder. It consists of a python script that uses the python interface exported by *gpsd*. Once a second, it retrieves the curent GPS location of the local node and makes a system call via the shell to sendGPSMessage 3.1.2. (Since it calls *sendGPSMessage*, *sendGPSMessage* must be in the PATH of the shell in which gpsFeeder is launched.

Usage: `gpsFeeder -s server`

Required Arguments:

- `-s,--server=address|name`, The address or name of the node running watcherd.

Example:

- Run the gpsFeeder to give current location information to the watcher daemon once a second and save the output as a log to `/var/log/gpsFeeder.log`.

  ```
  gpsFeeder.py -s glory 2>&1 /var/log/gpsFeeder.log
  ```

Note: the *emane* system uses the gpsd from gpsd.berlios.de and this is the daemon that the gps feeder supports. There is an older *gpsDaemon* that supports a *MANE* environment, but that is not supported directly by this watcher system release. Use the unsupported *gpsDaemon* and the auxillary program *watcherHierarchyClient* (see page 18) for *MANE* support in this release of the watcher system.

There are plans to create a global GPS Feeder that would feed all location information to the watcher daemon directly in order to cut down on network traffic, which would be very useful once the watcher is run on a system with hundreds or thousands of nodes. But these are still just plans.

3.2.2 watcherHierarchyClient

The *watcherHierarchyClient* daemon is used to support a test bed running the the dynamic hierarchy software. The watcher was orginally written with the hieracht daemons as a transport layer and as part of a dynamic hierarchy MANET platform. Thus the *watcherHierarchyClient* daemon is used for backwards compatibility. The basic idea of the `watcherHierarchyClient` is that it connects to a hierarchy instance, subscribes to all the messages that a watcher GUI may care about and when it receives such a message it translates it into something that the (new) watcher system can understand.

watcherHierarchyClient is the glue between hierachy land and watcher land. *watcherHierarchyClient*, when started, connects to a running hierarchy daemon and subscribes to all watcher related messages. When it receives a watcher related messages, it converts the message into something the watcher system can understand and sends it to the watcher daemon that it is connected to. *watcherHierarchyClient* is meant to offer backward compatibility to all "old style" watcher clients. It acts as a go-between between old hierarchy

messages and the new watcher messages.

Usage: `watcherHierarchyClient -s watcher_daemon_name_or_address -u hierachy_daemon_node_address`

Arguments:

- `-s`, The address or name of the node running the watcher daemon, watcherd.
- `-u`, The address of the node running the hierachy daemon.

4 The Watcher Daemon

The watcher daemon is responsible for collecting events from the test nodes and sending event streams to the Watcher GUIs. Events from the test nodes are stored in an SQLite databased (named "event.db" by default). The Watcher Daemon determines whether a connection is a test node or GUI by the type of the first event received.

When recording events from the test nodes into the database, events are appended to the existing database, or a new database is created if it does not exist. The Watcher Daemon may also be invoked in read-only database mode using the command line option `-r` or `--read-only`, in which case events are not stored in the database. Read-only mode is useful particularly when replaying events from a database from some time in the past. In this case, it may not make sense to append any current event stream from the test nodes when a large time gap exists between past and present runs.

The watcher daemon source code can be found in `./src/watcherd`. The binary produced after building is named `watcherd`.

4.1 Live Mode

When a GUI connects to the Watcher Daemon, it will by default subscribe to the live event stream coming from the test nodes. In this case, the Watcher Daemon is simply retransmitting received events to all listening GUIs rather than fetching events from the database. The events are also stored in the database for later replay.

If a GUI pauses, rewinds, or slows playback, it will switch to Playback mode.

4.2 Playback Mode

In Playback Mode, the Watcher Daemon fetches events from the database and sends them to the GUI. Each GUI connection has an independent notion of the current playback time offset, direction and speed. Stopping playback in one GUI will not cause playback to stop in another GUI.

The Watcher Daemon will automatically pause Playback when the last event from the database has been sent to the GUI. Thus, if a GUI were playing at a time offset near the end of the database, and faster than real time, the GUI will be paused when the last event is sent, even if additional events arrive from test nodes.

4.3 Configuration

The Watcher Daemon has several optional configuration options. By default, watcherd will read its configuration from a file named `watcherd.cfg`. An alternate configuration file can be specified using the `-c` command line option.

- `server`, the DNS name or IP address of the network interface to listen for connections
- `port`, the TCP port number to listen on for connections (default: 8095)
- `serverThreadNum`, the number of threads to spawn for handling connections
- `databasePath`, the full pathname to the test node event database (default: `event.db`)

- `dataNetwork`, if it exists, this network address will be mapped onto any incoming feeder messages that do not specify a source address. For example, if the `dataNetwork` is set to 192.168.10.0 and the incoming feeder message is from ip address 192.168.1.121, the incoming feeder message is set to 192.168.10.121. This is needed as the watcher daemon will use the ip address of incoming feeder messages to set the `from address` on the messages. If the watcherd is running on a network that is different than the test nodes (which is frequently the case when using a control network), this will map the control network addresses to the data network address space, so all feeder messages appear to come from the same network. If the dataNetwork was not specified, then this would generate 2 different addresses for a single node, confusing any attached GUI.

4.4 Command Line Options

- `-c` or `--config`, specify an alternate configuration file (default: `watcherd.cfg`)
- `-r` or `--read-only`, mark the event database Read Only so that no new events from test nodes will be written
- `-h` or `--help`, display a list of all supported command line options

5 Watcher GUIs

The watcher daemon exports a message streaming interface that supports any number of clients. This means that multiple GUIs can connect to a running watcher daemon and request a stream of data. One GUI can be viewing a single node at the start of the run while another can be viewing the whole field in live mode. This gives us the ability to write different GUIs to support different ends. The standard GUI is usually just called "the watcher". Or due the fact that there are now multiple GUIs and they have the potential to become the go-to GUI, the orginal GUI is sometimes called the "legacy watcher" - its the GUI that's been around since the start. There are currently three other GUIs, one of which is not yet hooked into the watcher API (so it cannot send or receive watcher messsages). Details about these GUIs are found below.

5.1 Legacy Watcher

The legacy watcher is the standard (for now) watcher system GUI. It is written in OpenGL, wrapped in a creamy Qt shell.

legacyWatcher source code can be found in `./src/clients/legacyWatcher`. The binary produced after building is named `watcher`.

5.1.1 Command Line Options

- `-h` or `--help`, show a usage message and exit.
- `-f` of –configFile, gives the location of the configuration file. If not given a default one will be created, used, and saved on program exit.

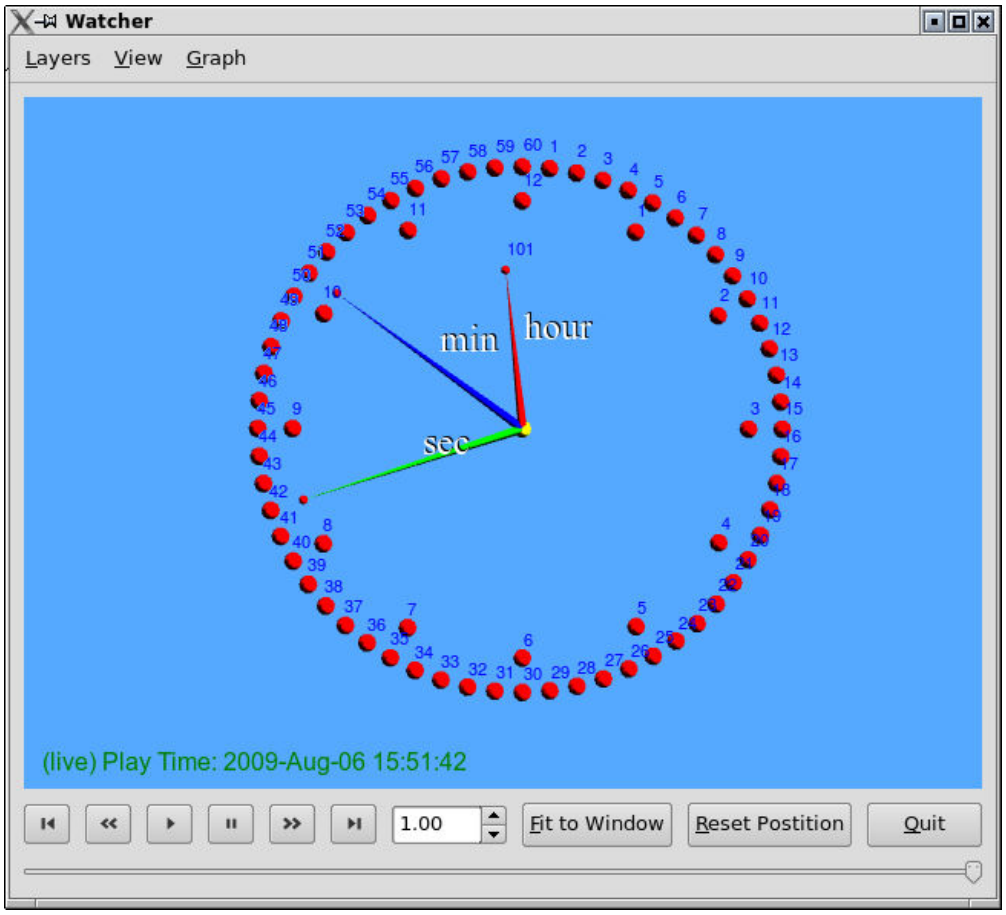

Figure 4: The legacy watcher showing a running instance of showClock.

5.1.2 Configuration

- `server`, name or ipaddress of the server to connect to.
- `service`, name of service (usaully "watcherd") or port number on which the server is listening.
- `layers`, a listing of layers from the Layers menu. All layers the watcher knows about will show up here in a `layername = bool` pair. If the bool is `true`, the layer will be active (and shown), if `false` the layer will not be shown. Every `layername` will become an entry in the `layers` menu in the GUI.
- `nodes3d = bool`, if `boolval` is `true`, the GUI will use 3d shapes to diplay the MANET elements, otherwise 2d shapes will be used.
- `monochrome = bool`, if `bool` is true all GUI elements in the main window will be black and white. Otherwise colors will be used.

- `displayBackgroundImage = bool`, if `bool` is true and a background image file is given, the background will be displayed. Otherwise it will not.
- `showVerboseStatusString = false`, if `true` show extra debugging information in the status string in the main window.
- `showWallTime = true`, if `true` show the current system time in the status string in the main window.
- `showPlaybackTime = true`, if `true` show the current playback time (when the events happened) in the status string in the main window.
- `showPlaybackRange = true`, if `true` show total time range (first event, last event) of playback in the status string in the main window.
- `statusFontName = ''name''`, the font name of the status string in the main window. e.g. "Helvetica".
- `scaleText = float`, how much to scale the text in the main window.
- `scaleLine = float`, how much to scale lines in the main window.
- `layerPadding = float`, how much padding (in pixels) to place between layers.
- `gpsScale = float`, a constant to multiply GPS coordinates against.
- `antennaRadius = float`, how big the "antenna radius" is in meters (for display only - does not effect connectivity).
- `ghostLayerTransparency = float`, how transparent to make the "ghost layer", when layers are spread more than a few pixels apart.
- `statusFontPointSize = float`, the font size of the status string.
- `backgroundImage:imageFile=filename`, the location of the background image to use, or "none" (with quotes) if not using a background image.
- `backgroundImage:coordinates = [ x, width, y, height]`, the initial location of the background image. This can be adjusted manually by using the right mouse button, while holding the shift key.
- `viewPoint : angle = [x, y, z]`, the initial angle of the main window view point with respect to the MANET.
- `viewPoint : scale = [x, y, z]`, the initial scale of the main window view point with respect to the MANET.
- `viewPoint : shift = [x, y, z]`, the initial shift of the main window view point with respect to the MANET.
- `backgroundColor : r=float`, the *red* component of the background color expressed as number between 0 and 1.
- `backgroundColor : g=float`, the *green* component of the background color expressed as number between 0 and 1.

- `backgroundColor :` `b=float`, the *blue* component of the background color expressed as number between 0 and 1.
- `backgroundColor :` `a=float`, the *alpha* component of the background color expressed as number between 0 and 1.

5.1.3 User Interface

The user interface for the watcher consists of a main window which displays the MANET, playback controls, a few buttons, and three pull down menus.

The main window is the heart of the visualization of the MANET. It is here where all the nodes, connections, labels, and layers are displayed. The user can use interact with the main window via the mouse or keyboard shortcuts. Mouse usage and keyboard shortcuts are given below.

5.1.4 Layer Menu

Figure 5: An instance of the layer menu in the watcher GUI. Layers appear here dynamically.

5.1.5 View Menu

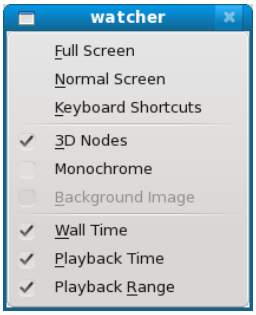

Figure 6: The view menu in the watcher GUI.

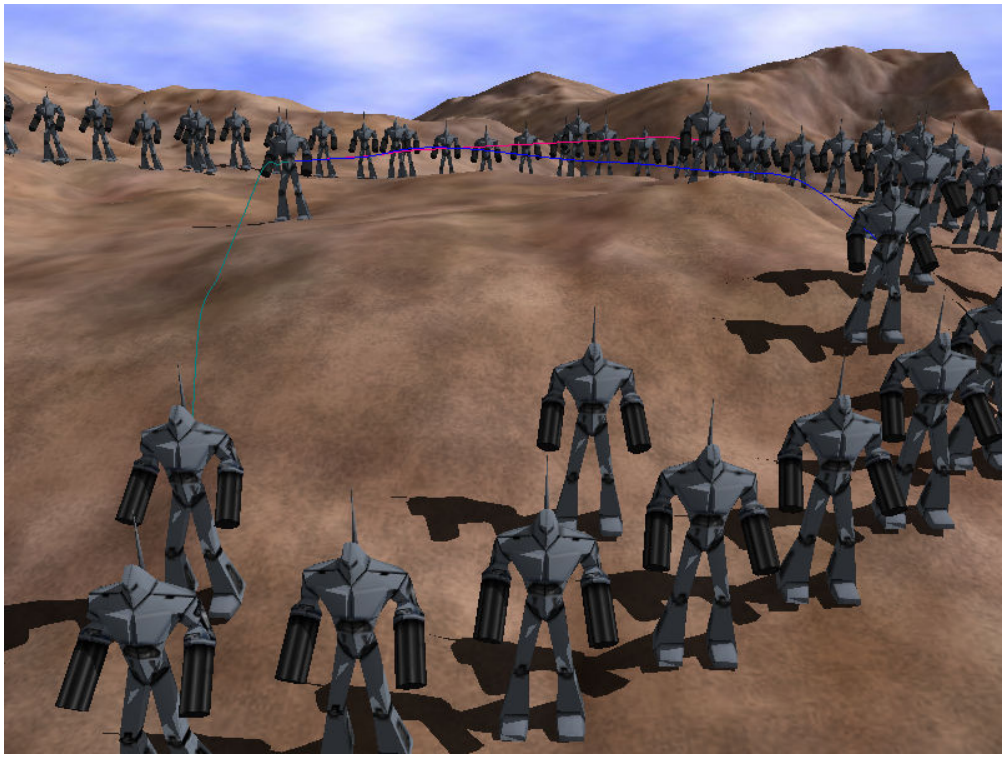

Figure 7: The ogreWatcher showing a running instance of showClock.

5.1.6 Graph Menu

5.2 ogreWatcher

Grr! Arg!

orgeWatcher source code can be found in `./src/clients/orgeWatcher`.

Built with OGRE.

5.2.1 Configuration

- `-h` or `--help`, show a usage message and exit.
- `-c configfile`, gives the location of the configuration file. If not given a default one will be created, used, and saved on program exit.

5.2.2 Command Line Options

- `server`, name or ipaddress of the server to connect to.
- `service`, name of service (usaully "watcherd") or port number on which the server is listening.

5.3 Watcher3d

Built with Delta-3d.

5.4 messageStream2Text

messageStream2Text is not your typical GUI in that it has no graphical interface. It's in the GUI section though as it uses the GUI watcher API to talk to a running watcher daemon. It was written (and is used) as a tool to debug the message stream interface. It simply connects to a daemon instance, requests a message stream, and dumps all the recieved messages to the console on stdout. Think of it as a tcpdump for watcher message streams. See figure 8 for a sample of the output.

messageStream2Text source code can be found in `./src/clients/messageStream2Text`.

5.4.1 Command Line Options

- `-h` or `--help`, show a usage message and exit.
- `-c configfile`, gives the location of the configuration file. If not given a default one will be created, used, and saved on program exit.

5.4.2 Configuration

- `server`, name or ipaddress of the server to connect to.
- `service`, name of service (usaully "watcherd") or port number on which the server is listening.

5.5 Earth Watcher

The Earth Watcher is a daemon application for Linux systems that allows visualization of Watcher events in the Google Earth GUI.

earthWatcher source code can be found in `./src/clients/earthWatcher`. The binary produced after building is named `earthWatcher`.

5.5.1 Command Line Options

- `-h` or `--help`, show a usage message and exit.
- `-c` of –configFile, gives the location of the configuration file. If not given a default one will be created, used, and saved on program exit.
- `-a` or `--latoff`, translate GPS coordinates relative to a given latitude
- `-A` or `--altoff`, translate GPS coordinates relative to the given altitude
- `-c` or `--config`, specify a configuration file (default: earthWatcher.cfg)
- `-d` or `--speed`, specify the event playback rate
- `-h` or `--help`, display this help message

```
Message # 158: from: 192.168.3.60 version: 1 type: "3 (gps)" time: 1250210220433 x:
9.9999999999999467 y: 21.5 z: 0 format: 0 layer: Clock
Message # 159: from: 192.168.1.100 version: 1 type: "3 (gps)" time: 1250210221433
x: 10 y: 10 z: 0 format: 0 layer: Clock
Message # 160: from: 192.168.1.101 version: 1 type: "3 (gps)" time: 1250210221434
x: 3.1529667948633628 y: 8.5446181642756738 z: 0 format: 0 layer: Clock
Message # 161: from: 192.168.1.121 version: 1 type: "5 (edge)" time: 1250210221434
node1: 192.168.1.100 node2: 192.168.1.101 edgeColor: red expiration: 0 width: 2
dir: unidirectional layer: Clock add: true node1Label: NULL middleLabel: from:
0.0.0.0 version: 1 type: "4 (label)" time: 1250210221434 label: hour (floating) font
size: 0 layer: Clock fg: (white) bg: (black) exp: 0 add: true lat: 0 lng: 0 alt:
0 node2Label: NULL
Message # 162: from: 192.168.1.102 version: 1 type: "3 (gps)" time: 1250210221434
x: 3.3086939364114212 y: 2.5685517452260545 z: 1 format: 0 layer: Clock
Message # 163: from: 192.168.1.121 version: 1 type: "5 (edge)" time: 1250210221434
node1: 192.168.1.100 node2: 192.168.1.102 edgeColor: blue expiration: 0 width: 2
dir: unidirectional layer: Clock add: true node1Label: NULL middleLabel: from:
0.0.0.0 version: 1 type: "4 (label)" time: 1250210221434 label: min (floating) font
size: 0 layer: Clock fg: (white) bg: (black) exp: 0 add: true lat: 0 lng: 0 alt:
0 node2Label: NULL
Message # 164: from: 192.168.1.103 version: 1 type: "3 (gps)" time: 1250210221434
x: 11.045284632676534 y: 19.945218953682733 z: 2 format: 0 layer: Clock
```

Figure 8: Sample *messageStream2Text* output.

- `-i` or `--icon-scale`, adjust the size of node icons
- `-I` or `--icon-path`, specify the node icon to use
- `-o` or `--output`, specifies the output KML file
- `-O` or `--lonoff`, translate GPS coordinates relative to a given longitude
- `-r` or `--refresh`, write the the output every SECS seconds
- `-s` or `--server`, connect to the watcher server on the given host
- `-S` or `--seek`, start event playback at the specified timestamp. May be specified relative to the first and last timestamps in the Watcher database by prefixing the offset with "+" or "-". Example: +5000 means 5 seconds after the first event in the database.
- `-t` or `--steps`, number of points to use when drawing a spline (default: 2)

5.5.2 Configuration

- `server`, name or ipaddress of the server to connect to.
- `service`, name of service (usaully "watcherd") or port number on which the server is listening.

- `layers`, a listing of layers from the Layers menu. All layers the watcher knows about will show up here in a `layername = bool` pair. If the bool is `true`, the layer will be active (and shown), if `false` the layer will not be shown. Every `layername` will become a folder in the `Places` menu in the Google Earth GUI.
- `layerPadding = float`, how much padding (in feet) to place between layers.
- `latOff = float`, translate GPS coordinates relative to a given latitude.
- `lonOff = float`, translate GPS coordinates relative to a given longitude.
- `outputFile = string`, specifies the output KML file.
- `iconPath = string`, specify an alternate icon to use for nodes.
- `splineSteps`, the number of points in a spline when drawing edges.

5.5.3 Reloading Configuration File

EarthWatcher checks for changes in the configuration file while running. The purpose of this feature is to allow for defined layers to be toggled on or off during runtime. No options other than the layers are checked for changes during runtime.

5.5.4 Translating GPS coordinates

If the GPS coordinates stored in the Watcher database during a test run are not accurate, the EarthWatcher has options to translate the coordinates relative to some location on the Earth. For example, the origin is located in the Atlantic Ocean off the coast of West Africa. Using the `latOff` and `lonOff` options can place the nodes in a more geographically interesting location.

5.5.5 Using EarthWatcher with Google Earth GUI

EarthWatcher writes a KML file, by defauled named `watcher.kml`. Follow these steps to view the generated KML file in Google Earth:

- Launch the Google Earth GUI.
- Select the *Add* menu.
- Select the *Network Link* menu item.
- Click the *Browse* button next to the text input box labaled *Link*.
- Browse to the folder containing the KML file created by EarthWatcher.
- Click on the *Refresh tab*.
- In the *Time-Based Refresh area*, click the drop down box next to *When* and select *Periodically*.
- Set the time to 0 hours, 0 minutes and 1 second.
- Click OK.

In the *Places* pane in Google Earth, there will be a *Untitled Network Link* under *Temporary Places* in the tree view that contains all the Watcher related elements. Double-clicking on *Untitled Network Link* will cause Google Earth to center and zoom in on the Watcher objects.

5.5.6 Using Google Earth when disconnected from the Internet

Google Earth is normally used on an Internet-connected workstation. When browsing an area of interest, the satellite images are broken into tiles and sent to the Google Earth client on demand. In order to avoid sending the same tiles repeatedly, the Google Earth client caches the tiles locally on the hard drive. Once the tiles are cached, it is possible to use the Google Earth client to browse areas that are contained in the cache while disconnected from the Internet.

On a Linux-based operating system, the image cache is located in the `$HOME/.googleearth/Cache` directory. The image cache consists of two files named `dbCache.dat` and `dbCache.dat.index`. In addition to the satellite image cache, there are also subdirectories containing cached placemark icons and 3-D models. It is possible to package up the `Cache` directory and move it to another machine for use.

Google Earth allows a maximum size of 2 gigabytes for the satellite image cache. When attempting to cache multiple different areas of interest, this limitation may require creation of separate cache files for each area rather than a single cache file. The cache files can be swapped and Google Earth restarted in order to switch the area of interest for browsing. The easiest way to handle this is to have separate `Cache` directories for each location of interest. For example, `Cache.NewYork` or `Cache.GrandCanyon`. The appropriate directory can be renamed `Cache` or a symbolic-link can be used to select which image cache to use.

In the sitation where the disconnected workstation used for browsing Google Earth can not initially be connected to the Internet, it is necessary to perform one additional initial step, otherwise Google Earth will fail to work. The first time Google Earth is run some registration information is exhanged with Google's servers. When running in disconnected mode, this information must exist, or it can't be used for offline browsing. On a Linux-based machine, the generated configuration files reside in the `$HOME/.config/Google` directory. These files can be copied from any Internet connected machine and will allow offline browsing on another machine.

In the simplest case where a single image cache is generated on an Internet-connected machine and moved to another offline machine for browsing, the following command is sufficient for creating a package that can be moved to the offline machine:

```
# tar jcvf ge-cache.tar.bz2 .config/Google .googleearth
```

This file can be unpacked in the home directory of the Google Earth user on the offline machine.