eMews Installation and Usage Guide

eMews version 0.32

Contents

1 Introduction

This guide serves as a quick introduction to installing eMews and using it for running experiments, including creating new services. Much of this document is a work in progress and incomplete, but should provide enough information to get started, hopefully.

eMews was introduced originally in a paper presented at IRI 18 [1]. Version 0.32 of eMews, the version used in the paper, is soon to be replaced with version 0.4, which incorporates many changes over 0.32. Version 0.32 is released mainly for those who wish to replicate the experiments performed in the IRI 18 paper [1], or for those who want to get their feet wet with eMews (data generation, general curiosity, etc).

We will assume that eMews is being used with the CORE network emulator, and that the reader is familiar with it. While this is not a requirement per-se, the original experiments were performed with CORE, and eMews works with CORE out of the box.

2 Installation

eMews is written in Python, and thus is technically cross-platform. However, the framework targets Linux machines, and some of the lower level functionality may not work in other operating systems, such as Windows. If planning to run eMews on a non-Linux operating system, and everything works, please let us know!

2.1 Prerequisites

Python 2.7 is required to run eMews. In addition, the Python interpreter needs to be CPython, which is usually the default interpreter when installing Python. CPython uses what's called the *global interpreter lock (GIL)*, preventing CPU context switching of threads outside of I/O blocking or a suspended state such as sleeping. eMews services are designed with the GIL in mind, providing concurrency by utilizing I/O blocking and sleep. While a different interpreter can be used, the lack of a GIL may cause unnecessary context switching, adding to the overhead of the target machine.

2.1.1 Python Packages

The following packages are required for eMews to run:

- ruamel.yaml - Provides YAML support for eMews configuration files.

The following packages are required if running the included eMews client-side autonomous services (HTTP/HTTPS or SSH):

- numpy - Provides the distributions required for the user behavioral models. Also is required if using any of the sampler helper classes.

- mechanize - Required if using the HTTP/HTTPS (webcrawler) service.
- pxexpect - Required if using the SSH (AutoSSH) service.

It is recommended to NOT install the packages locally (ie, to your $HOME), as eMews by default runs under root when launched by CORE. Install the packages as root instead.

2.2 Installing eMews

Once the environment is setup, all that remains is simply copying the eMews Python modules to a desired path on the target machine. Make sure that where ever eMews is copied to, that its directory structure remains intact, including the 'emews' root directory.

2.3 CORE Specific Setup

CORE uses its own services, and eMews contains CORE services to start the eMews daemon and the single service client under different configurations. The single service client is a standalone process which connects to the daemon for the purpose of eMews service launch requests.

2.3.1 Modifying CORE Service Classes

In version 0.32, CORE service classes have both the $PYTHONPATH and eMews paths hard-coded. As such, the correct path will need to be added to these classes.

The modules in question can be found under /coreservices. AutoSSH contains multiple CORE services, one for each experiment performed in the IRI 18 paper [1]. For each eMews service you wish to run, it's corresponding CORE service class needs to be modified accordingly (emewsdaemon.py is required):

- In emewsdaemon.py, scroll to line 36 and change the $PYTHONPATH default path to the path of your eMews installation. Then change the path in the next line, after 'python', to the same path as your $PYTHONPATH. Be sure to keep everything from '/emews/' onward, only replacing what precedes it with your own path.
- In each of the remaining CORE service classes that you wish to run, perform the same modifications as for emewsdaemon.py. The lines in question are 33 and 34. In line 34, make sure to keep everything from '/emews/' onward, only replacing what precedes it with your own path.

2.3.2 CORE Utility Service Modification

To properly run the experiments from the IRI 18 paper [1], the CORE HTTP server service needed to be configured to support HTTPS.

To apply this modification, under /coremods/emulation_server/core/services/, copy the module 'utility.py' to your CORE installation, under /core/services/, replacing the utility.py module that is already there.

2.3.3 CORE 5.0 Service Modification

CORE 5.0 contains a bug which prevents custom services from being installed in the normal way. While CORE recognizes these services, unfortunately multiple runs of a CORE scenario produce duplicate service instances. To work around this, we treat our EMEWS CORE services as built-in to CORE. Note that CORE 5.1 has fixed this issue.

Under /coremods/emulation_server/core/services/, __init__.py will need to be modified to include your path to EMEWS. In the last line, modify the path to reflect your path. As with CORE services (Section 2.3.1), make sure to keep everything from '/emews/' onward, only replacing what precedes it with your own path.

Once the module has been modified, copy it to your CORE installation, under /core/services/, replacing the __init__.py file which is already there.

3 Usage

Once EMEWS is installed, the relevant services should show up when creating CORE topologies in the CORE GUI. Any EMEWS CORE services you add through the CORE GUI will automatically launch their respective EMEWS services (through the daemon) when you start the experimental run.

Note, the EMEWS daemon CORE service is required for any node that will run other EMEWS services. So if any EMEWS services are selected, then the daemon must also be selected.

3.1 eMews System Configuration

The main system configuration is located under the root EMEWS directory, named 'system.yml'. When running under CORE, this single file is shared among all the daemon instances for all nodes.

Configuration options are documented in the file. The options given were used for the last experimental run in the IRI 18 paper [1]. Distributed logging is enabled, with a node named 'n1' (CORE by default names nodes starting at 'n1') assigned to be the designated log server. The designated log server logs all entries from all other nodes to a file in the root of that node's container.

3.2 Distributed Logging

EMEWS distributed logging enables real-time experiment monitoring. One node is configured to start the logserver service, called the *designated node*, and all other nodes configured to send messages to this designated node. Details on how to set this up are located in the EMEWS system.yml.

By default, the distributed log is configured to be a file. An easy way to access this file during an experimental run is to ssh into the physical host running CORE (or just open a terminal if on the host locally), and cd to '/tmp/pycore.XXXXX', where 'XXXXX' is some number. The 'pycore.XXXXX' directory contains all the filesystem info for each node. For the node that is the designated node, find its directory (named 'nXXXX.conf', where 'XXXX' is the node number), and cd to it. Within this directory is the eMews distributed log file, named 'emews_log.txt'.

3.3 Standalone Service Launcher

For tasks such as testing a new eMews service, the standalone service launcher bypasses the daemon and directly launches a service. This module can be found under /standalone.

When directly launching services, distributed logging will be unavailable, so be sure to configure system.yml accordingly for local log output (output to the console or to a file).

3.4 Creating an eMews Service

Creating a custom eMews service only requires subclassing emews.services.baseservice.BaseService. This gives your service direct access to its configuration, logging output, and base service functionality.

Some important methods of the service API are the following:

- run_service(): The entry point for the service to start. Must be implemented.
- config(): Returns the service configuration.
- dependencies(): Returns the dependencies of this service, or None if none are defined. Dependencies are objects which the service requires, specified in the service's configuration file.
- sleep(time): Given an amount of time, in seconds, the service will sleep for that duration.
- interrupted(): Returns true if the service has been interrupted (requested to stop) by the eMews daemon.

3.4.1 Service Configuration

Service configuration files contain three main sections:

- config: Contains the main configuration options for the service.
- dependencies: Contains the classes the service requires. Each dependency may have its own configuration options, which are also defined here.

- decorators: Any decorators which the service requires are defined here. 'LoopedService' is a common (and in 0.32, only) decorator, which restarts the service after it has finished. For example, after AutoSSH has finished an SSH session, LoopedService restarts it after some time has elapsed, for a new SSH session.

3.4.2 Service Perfomance Concerns

As eMews services run within a shared environment, services should be rather lightweight. Also be aware that even though each eMews service runs in a separate thread, a computationally heavy service will not yield to any other service until it either finishes, blocks, or sleeps. This means that if a service requires say a minute of computational time before yielding, and if multiple nodes are running the service, then other services may be suspended long enough for significant timing issues to occur.

Try to design your service such that any time spent executing code is less than the time spent I/O blocking for received data over the network, or sleeping [1]. While this isn't a perfect solution, it helps to spread CPU usage among all the running services, while limiting the additional overhead spent context switching.

References

[1] B. Ricks, P. Tague, and B. Thuraisingham. Large-scale realistic network data generation on a budget. In *19th International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018.