ARL-TR-8863 ● DEC 2019

Visualization and Processing for Embedded Research Systems (ViPERS) Web-based Interface for Internet of Things (IoT): Implementation Guide

by Kevin E Claytor, Alex George, Zachary Drummond, and Abby Snellman

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

ARL-TR-8863 ● DEC 2019

Visualization and Processing for Embedded Research Systems (ViPERS) Web-based Interface for Internet of Things (IoT): Implementation Guide

Kevin E Claytor
Sensors and Electron Devices Directorate, CCDC Army Research Laboratory

Alex George and Zachary Drummond
General Technical Services

Abby Snellman
Army Educational Outreach Program (AEOP) College Qualified Leaders (CQL)

REPORT DOCUMENTATION PAGE						*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**						
1. REPORT DATE *(DD-MM-YYYY)* December 2019	**2. REPORT TYPE** Technical Report					**3. DATES COVERED (From - To)** May 2017–October 2019
4. TITLE AND SUBTITLE ViPERS Web-based Interface for Internet of Things (IoT): Implementation Guide						**5a. CONTRACT NUMBER**
						5b. GRANT NUMBER
						5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kevin E Claytor, Alex George, Zachary Drummond, and Abby Snellman						**5d. PROJECT NUMBER**
						5e. TASK NUMBER
						5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) CCDC Army Research Laboratory ATTN: FCDD-RLS-SP Aberdeen Proving Ground, MD 21005-5066						**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-8863
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)						**10. SPONSOR/MONITOR'S ACRONYM(S)**
						11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES ORCID ID(s): Kevin E. Claytor, https://orcid.org/0000-0002-6072-5033						
14. ABSTRACT Visualization and Processing for Embedded Research Systems (ViPERS) provides a way to run data processing and visualization code on an edge device, such as a Raspberry Pi or Snickerdoodle single-board computer, and present the results to a personal computer or mobile device. This enables the rapid transition of cutting-edge research results from the researcher (who can develop using the ViPERS platform) to the Warfighter (who can view the results on the ViPERS interface). This is done securely so only authorized users can view the data. Additional tools are provided so that the networking can be configured, allowing or disallowing local or wide area networking and cloud-connectivity while still maintaining a functional processing node. By providing the researcher a process manager and access to real-time log files, ViPERS enables a feedback loop for rapid prototyping of new algorithms and processing techniques, but these can simultaneously be presented as actionable information to the Warfighter in a relevant environment.						
15. SUBJECT TERMS Internet of Things (IoT), embedded processing, transition, live data visualization, device management, network configuration						
16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**		**19a. NAME OF RESPONSIBLE PERSON** Kevin E Claytor
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	44		**19b. TELEPHONE NUMBER (Include area code)** (301) 394-2162

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction/Motivation

With the proliferation of Internet of Things (IoT) devices, there is now a spread of sensing and computing devices from traditional data acquisition units (DAQs) and PCs to tiny embedded IoT sensors. These IoT devices are frequently no more than a small microprocessor and WiFi or Bluetooth chip, able to sense their environment and transmit that data to a remote server. Conversely, DAQs and PCs can not only interface with the sensor(s), but provide a powerful computing and analysis platform. Between these devices is a range of systems that are ideal for research, and transition to Department of Defense (DOD) users. Here, the application space may not be defined enough to have a headless IoT device. However, the size, weight, and power (SWaP) constraints may not permit a full DAQ or PC. These devices, such as the Raspberry Pi,[1] Snickerdoodle,[2] Beaglebone,[3] or Intel NUC,[4] are powerful enough that they can perform processing and data storage on the device, but have sufficiently low power draws that they are practical for extended unattended operation. Thus, the benefits of IoT sensing can be realized in a military environment, which may include degraded communications, requiring the device to operate independently.

These single-board computers (SBCs), however, have a major drawback restricting their use as a research tool and transition to military operators: the lack of an interface. As they do not have a display, one must either attach an external display or tunnel into the device using a command-line tool such as secure shell (SSH). An attached display may negatively impact SWaP, for while these may be quite, quite small, they may cost a significant fraction of an SBC and can have a negative impact on battery life. On the other hand, command-line tools such as SSH are quite powerful, but require an experienced user.

This software Visualization and Processing for Embedded Research Systems (ViPERS) attempts to address these limitations by providing an easy-to-extend website-based interface to the device. This interface can be viewed from any device capable of rendering web content, such as a PC, smartphone, tablet, or even an integrated display. In addition to simply providing a user interface, ViPERS also provides a process manager that can aid development and a networking controller that can assist in configuring the device for local area network (LAN) or wide area network (WAN) connections.

To achieve this, ViPERS consists of three parts:

1) A webserver to interface with and configure the SBC and to configure and view data.

2) A process controller (Dataserver) that can be used to run data processing programs and streaming visualizations.

3) A network manager (NetManager) to configure the network state of the device.

Since it runs as a webserver, the researcher or customer can bring their own device, such as a mobile phone or tablet to interact with the SBC. This removes the need for a power-hungry integrated monitor or a bulky laptop. Secondly, Dataserver allows the researcher to upload and quickly prototype their processing algorithms, while viewing log files and streaming output data to a visualization. Finally, NetManager allows the customer to configure the device to their specific network, a necessity since DOD installations frequently have specific network requirements.

This report has three main aims:

1) Introduce the need for ViPERS and survey the available open-source libraries that can be combined to create our solution and select the specific ones that we used.

2) Describe the ViPERS architecture and how it is used in an example application.

3) Describe how to customize and extend the core ViPERS elements for other computing platforms.

2. Survey

2.1 Language

Web applications can be written in practically any programming language. For the case of ViPERS, we chose to use Python[5] for two main reasons:

1) It is flexible with large and growing community support.[6]

2) Many scientific programs are written in Python, allowing the scientific programmer to apply their knowledge to tailor ViPERS to their need.

2.2 Webservers

Within the Python ecosystem, there are two popular web frameworks: Django[7] and Flask.[8]

Django is a “batteries included” framework that includes user authentication, database management and migration, support for subapplications, and administrative management, among many other tools. However, it also has a steep learning curve and a fairly rigid structuring and layout.

On the other hand, Flask is a "microframework" that provides only the essential elements and relies on an ecosystem of add-ons to fill out functionality. This more decentralized approach gives it flexibility in how one approaches creating a web application and also gives it an easier learning curve, only requiring one to learn a single new component at a time. While both are valid options for a project such as this, the goal of minimizing learning time and maximizing flexibility encouraged the use of Flask.

Although Flask provides a development webserver, it is not recommended for production environments. Consequently, we rely on Gunicorn[9] for the webserver gateway interface (WSGI) and NGINX[10] for the webserver.

2.3 Plotting

There are several popular Python plotting frameworks including the following:

- Matplotlib[11]
- Qt[12]
- Plotly[13]
- Dash[14]
- Bokeh[15]

For this application, there were two driving requirements:

- Plots need to be generated locally (there may be no Internet connection).
- Streaming/dynamic plots are needed to facilitate user interaction.

It is exceedingly difficult to get Matplotlib to generate dynamic, streaming plots. Similarly, Qt is much more for interaction on desktop devices versus web/mobile devices. Finally, Plotly requires an internet connection and data transfer/authentication with plotly's servers.

Because of these limitations, Bokeh appeared to be the best choice. It facilitates very customizable, modern plots and graphs that can either be static or a dynamic application served with the `bokeh serve` command. Additionally, Bokeh has very good Jupyter/IPython integration, allowing one to prototype plots and applications in an interactive environment and then transfer them to ViPERS when ready.

After Bokeh was selected for this project, Plotly released Dash, an open-source application similar to Plotly, but not dependent on their servers. This may be a good candidate if one is already using Plotly in other applications or otherwise familiar with its application programming interface (API).

2.4 Databases

Even with the ubiquity of WiFi hotspots and cellular data, such as 3G and 4G, cloud connectivity is never guaranteed and there may be long stretches of time when the IoT device is left without connection to a cloud-based database. In fact, depending on the application, the data during times of network outage are the most important data. As a result, it is essential that an IoT device store data locally in addition to pushing data to the cloud. By maintaining a local database of events, when Internet connectivity is restored, data may be replicated back to the cloud. Thankfully, there is no shortage of database options from which to choose. A few are shown in the Table 1.

Table 1 Survey of databases and their advantages and disadvantages for the ViPERS architecture

Database	Type	Advantages	Disadvantages
Flat file	NoSQL	Easy to write into.	Difficult to query into. Complexity grows with additional data.
MySQL[16]	SQL	Simple, reasonable performance.	Memory consumption. Relational model may not be suited for time-series data.
PostgreSQL[17]	SQL	Good performance, stable, good query speed.	Memory consumption. Relational model may not be suited for time-series data.
SQLite[18]	SQL	Lower memory consumption, single file database.	Relational model may not be suited for time-series data.
Berkley DB[19]	NoSQL	Good performance for inserts, long stable history.	Need to build a time-series engine around the key/value store.
Redis[20]	NoSQL	High-performance in-memory key/value store.	Need to build a time-series engine around the key/value store.
OpenTSDB[21]	NoSQL	Simple. Open source.	Immature software, still under substantial development. Limited support for math on queries. Rollups and retention policies need to be performed manually.
TimescaleDB[22]	SQL	Stability of PostgreSQL with improved insert performance for time-series data.	Difficult to install, especially on older versions of operating systems. Rollups and retention policies need to be performed manually. Large database size.
InfluxDB[23]	NoSQL	Good insert speeds for low-cardinality data. Single binary distribution. Open-source community edition. Rollups and retention policies are first class. Performs data compression for small database size.	Has some stability issues, inconsistent type-conversion.

Note: SQL = Structured Query Language

Most of these databases will run on an embedded Linux system such as a Snickerdoodle or Raspberry Pi. In the end, we chose InfluxDB to include with the ViPERS installer because 1) it was a single binary and easy to include, 2) the NoSQL structure allows one to quickly start storing data without having to worry about creating a schema, and 3) the included Chronograf visualization software makes exploring and querying the database easy for a nontechnical user.

It is interesting that the capabilities of TimescaleDB are growing fast. For high-cardinality data, it appears that TimescaleDB may outperform the insert performance of InfluxDB.[24] Additionally, it is able to leverage the stability and the query planner of PostgreSQL giving high-reliability into one's data and fast queries. Finally, a third-party visualization software called Grafana[25] can interface almost all of the databases described earlier giving a good visualization solution regardless of the database back end.

2.5 Networking

In addition to the different use cases for IoT devices, every installation has a different approach to network security. To accommodate as many networking configurations as possible, a research IoT device cannot simply rely on WPA2 secured WiFi—the standard for homes and small businesses. For this reason, we built in a network manager into ViPERS.

To configure Linux networking, one can rely on new tools such as `ip` and `network-manager`. However, for older Linux distributions, these may not be available. In this case, we use `ifconfig` to manage the interfaces (mainly through `ifdown` and `ifup`). Instead of painstakingly managing the `wpa_supplicant.conf`, however, we use the `wpa_cli`[26] tool.

Additionally, ViPERS can configure the system to host its own network. This allows data collection and real-time visualization of data in environments where network infrastructure is not available, such as field sites. For these, `hostapd`[27] and `isc-dhcp-server`[28] allow the SBC to serve as an access point and hand out Dynamic Host Configuration Protocol (DHCP) addresses.

These tools allow us to reach a broad range of Linux systems while avoiding some of the low-level frustrations of working with networking interfaces. Additionally, we are able to cover the range of network configurations—from Ethernet to Wi-Fi, host or client, dynamic or static IP—ViPERS is able to configure it.

3. Architecture

ViPERS comprises three components and an optional database layer graphically displayed in Fig. 1:

- The webserver is the front-facing user interaction and send command signals to the other components. Additionally, it allows for the embedding and visualization of the data.
- Dataserver is a process manager, which is used to run the data processing code. It also exposes an API that allows the webserver to start and stop modules.

- NetManager is an interface to lower-level Linux networking commands. This allows one to tailor both the interfaces and connect to wireless networks.

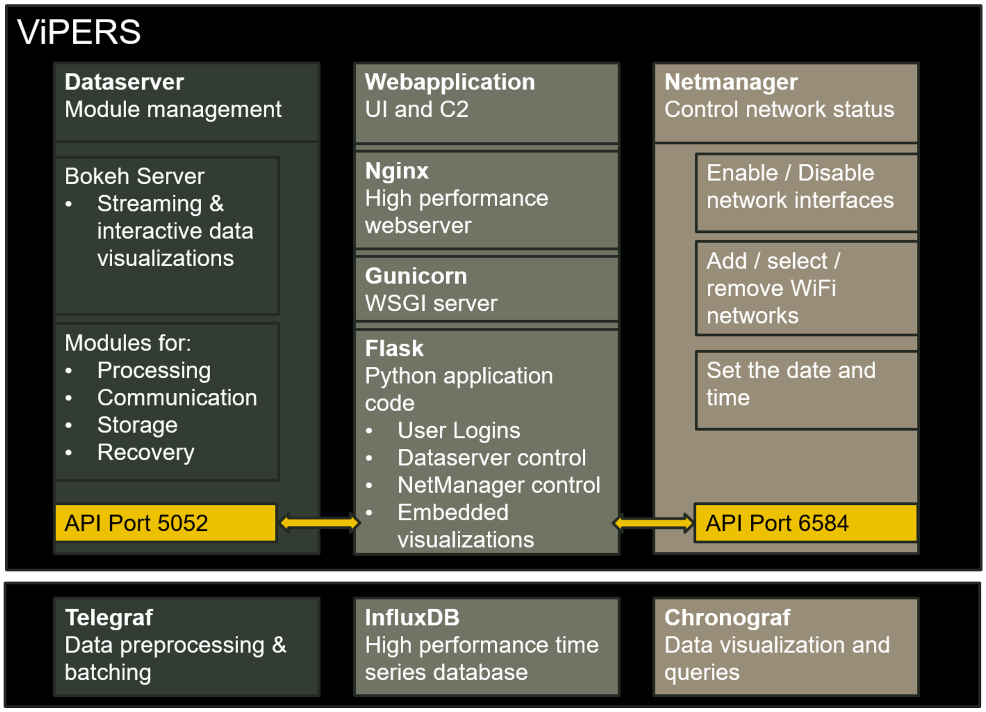

Fig. 1 Schematic of the ViPERS architecture. The webserver is the main user interaction component, which communicates to Dataserver and NetManager through a Transmission Control Protocol (TCP) API. The database is lightly integrated and can be replaced with a database of one's choosing.

When installed using the included installer, symbolic links to all three components and the Python virtual environment is generated in `/usr/local/vipers`. Additionally, all log files are sent to `/var/run/vipers/` as `vwebserver.log`, `vdataserver.log`, and `vnetmanager.log`, respectively.

These three components and their APIs are described in depth in the next three sections.

3.1 ViPERS Webserver

The ViPERS webserver is how the user interacts with both Dataserver and NetManager. Additionally, it allows one to view the status of the system and ViPERS components. There is an integrated help system—wherever a question mark appears in a title, clicking it will take the user to the corresponding help pages.

3.1.1 Viewing the Web Application

Depending on the network configuration, the address of the webserver is variable. Table 2 describes how to determine the address of the webserver.

Table 2 Mapping between network mode and IP address to use to connect to ViPERS

Networking mode	Address
Ethernet DHCP host	10.10.0.1
Wi-Fi DHCP host	11.11.0.1
Static IP address	<assigned static ip>
DHCP IP address	<assigned dhcp ip>
All	<hostname>.local
All	<hostname>.lan

Once connected, the user can view the login screen, authenticate, and see the main index, as shown in Fig. 2. The main index page is always available by clicking on the title text present on any page.

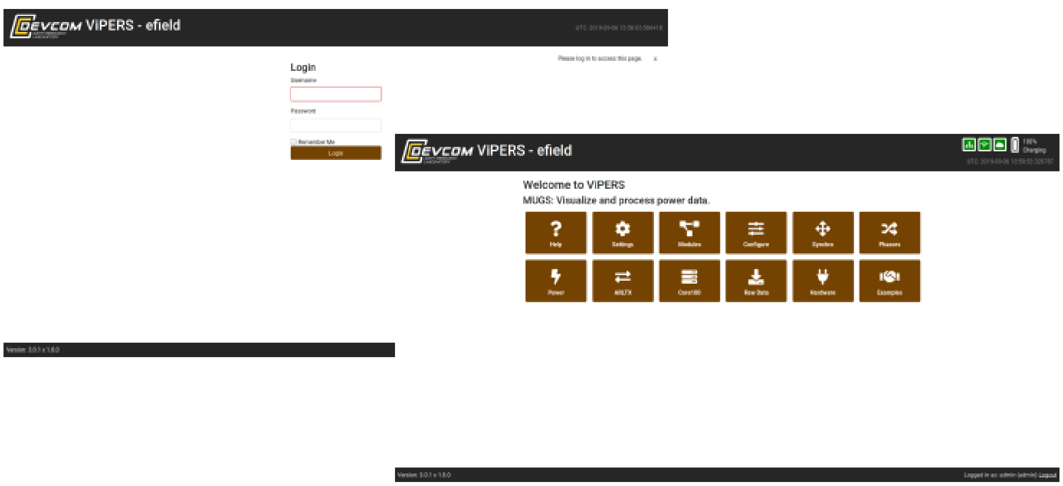

Fig. 2 All pages are protected by a login screen. Once authenticated, the index page allows quick access to main ViPERS components.

3.1.2 Running the Webserver

If the main install script (`../install.sh`) or the webserver install script (`../install/setup_vwebserver.sh`) was run then the webserver should be installed as a startup script. One can view the status of the webserver by running the following:

```
sudo service vwebserver status
```

The service can be restarted with the following:

```
sudo service vwebserver restart
```

The webserver can be run manually with the following:

```
cd vipers
workon vipers
gunicorn webapp:app --bind 0.0.0.0:8000 --log-level DEBUG
```

Finally, for development or debugging, one can run the Flask development server. The development server actively watches for code changes and reloads when the source changes, allowing one to rapidly create and test new pages. Additionally, when an error is encountered, the development server displays an interactive stack trace, which allows the user to enter a Python command line at various points through the error. This allows one to inspect variables and the state of the program at the point of the crash and in the stack frames above the crash. This does require entering a personal identification number, which is displayed on the command-line when the development server runs.

To run the development server, one can use the following convenient script:

```
cd vipers
workon vipers
./run_flask.sh
```

Do not run the development server in production. Use the production Gunicorn server that is provided as part of the startup script. Additionally, the default port for the debug server is port 5000, as opposed to 8000 (for the Gunicorn server). This must be specified when connecting (for example, in the browser's address bar).

3.2 ViPERS Dataserver

Dataserver is a process manager that presents both a Python API and a JavaScript Object Notation (JSON)-encoded socket API that can be used to start, stop, and/or restart the processes it manages.

In addition, it also runs the `bokeh serve` command and passes it a list of plots. The Bokeh session is secured with a secret key, which Dataserver can share with other applications to allow them to access the Bokeh sessions.[29] This prevents unauthenticated users from viewing the Bokeh applications.

The Python module also implements a programmatic API, which one can use to control Dataserver. Additionally, other languages may use command port to send JSON-encoded command messages.

3.2.1 Implementation Details

The Dataserver directory structure is as follows:

```
dataserver/
├── controller.py
├── dataserver.py
├── settings.conf
├── saved/
├── plots/
│   ├── plots.py
│   └── ...
└── modules/
    ├── conf/
    ├── data/
    ├── logs/
    ├── modules.py
    └── ...
```

The purpose of these programs and folders are listed in Table 3.

Table 3 Description of the core Dataserver functions and folders

File/folder	Description
controller.py	Provides an interactive control session to Dataserver.
dataserver.py	The main executable.
settings.conf	The configuration file that specifies which modules and plots are enabled.
saved/	A folder for saved configuration files.
plots/	Folder for Bokeh plot scripts.
modules/	Folder for modules (scripts/executable files).
modules/conf/	Convenience folder for modules to store configuration files.
modules/data/	Convenience folder or modules to store data files.
modules/logs/	Standard output and error from modules is piped to files in this folder.

3.2.2 Process Management

Modules are started as subprocesses using the `psutil.Popen` command.[30] Standard output and standard error are piped to a log file: `dataserver/modules/logs/<module filename>.log`.

Modules are killed using SIGTERM, and it is recommended that the module handles SIGTERM and shuts down gracefully after receiving the signal. Additionally, all child processes of the module are terminated in a similar manner.

On shutdown of Dataserver, all internal modules are killed, even if they are no longer in the configuration file. This ensures that all running child processes are terminated.

3.2.3 Bokeh Management

In addition to the module processes, there is a special process that corresponds to Dataserver running the `bokeh serve` command. This is equivalent to running the following:

```
export BOKEH_SECRET_KEY=`$(bokeh secret)`
export BOKEH_SIGN_SESSIONS=true
bokeh serve --allow-websocket-origin=* --session-ids=external-sig
ned <list of plot files>
```

The flag `--allow-websocket-origin=*` allows any host to connect to the server, which is needed as we do not know *a priori* where the user is connecting from. Given the networking options we support, the user could be connecting over wired or wireless LAN or DHCP directly or through a LAN.

This allows any device on the local network to connect by going to the server URL, for instance, `http:192.168.1.4:5006/synchro` if the device has an IP address of 192.168.1.4. This is clearly suboptimal, as it allows anyone to view our stream of data. To protect against these attacks, we can set Bokeh to only allow through sessions with a secret key.

By setting the flag `--session-ids=external-signed` Bokeh only allows sessions that present a key derived from the environment variable `BOKEH_SECRET_KEY`. This keeps the data secure in transit and ensure that only those who have `BOKEH_SECRET_KEY` can connect.

Unfortunately, the `BOKEH_SECRET_KEY` is only exported to Dataserver's process and child processes. As a result, the webserver cannot access it; however, Dataserver retains memory of the secure secret and can provide that to other applications (e.g., the web application) via its API. This allows other applications such as the webserver to sign Bokeh sessions and embed plots. This is done through the `dataserver.secret()` function.

Since Dataserver only accepts connections from the localhost, only other applications on the IoT device can request the secret (such as the Flask web application) and use it to authenticate against the Bokeh server to embed plots.

3.2.4 Configuration File

Dataserver uses a configuration file `settings.conf`. This is a TOML[31]-encoded configuration file with entries of the following form:

```
plots = []
[[modules]]
  filename = "mycode.py"
  command = "python" # Can be: python, java, octave, execute
  options = ""    # Command-line arguments to pass
  enabled = true   # Is the module enabled or not?
```

It is recommended to use the Python API to read and write this file.

3.2.5 Python API

It is recommended to use the Python API where available. This abstracts out the implementation and provides convenience functions for Dataserver functionality.

To use this, simply import Dataserver and call one of the functions shown Table 4.

Table 4 Functions in dataserver.py that define the public API

Function	Description
(modules, plots) = read_config()	Return the current module and plot dictionaries.[a]
write_config(modules=None, plots=None)	Write modified module and plot dictionaries to a file.
start(*args)	Start modules specified by args.[b,c]
stop(*args)	Stop modules specified by args.[b]
reload(*args)	Reload (stop then start) modules specified by args.[b,c]
reload()	Reload (stop then start) all modules.[c]
replot()	Restart the Bokeh server.
secret()	Return the Bokeh secret key (needed to authenticate against the Bokeh session).
show()	Show the running modules and provide their status (zombie/running/sleeping) and central processing unit usage.
quit()	Cause the Dataserver to exit

[a]The module dictionary matches the modules section of the config file. The plots dictionary uses the module filenames as keys and a Boolean for the value corresponding to whether or not they are running:

```
modules = {
  "module.py": {
    "filename": "module.py", # The filename
    "command": "python",   # One of; "python", "exec", or "octave"
    "options": "",      # Command-line arguments
    "enabled": True,     # Runs the module if enabled
  }
}
plots = {
 'mod1.py': True,  # Pass `mod1.py` to Bokeh if enabled
 'mod2.m': False,
}
```

[b]`*args` is an unpacked list of module filenames, for example, `start('mod1.py', 'mod2.m')`.
[c]If the module is not enabled in `settings.conf`, it will not be started.

3.2.6 JSON API

If using a programming language other than Python, one can still issue commands to Dataserver using its JSON-based socket API.

Create a TCP connection to Port 5052 then transmit a JSON message of the form: The JSON data should conform to the following:

```
{<COMMAND>: <OPTIONS>}
 COMMAND is a string and is one of
  - "QUIT"  (no OPTIONS)
  - "SHOW"  (no OPTIONS)
  - "START"  (OPTIONS is required)
  - "STOP"  (OPTIONS is required)
  - "RELOAD" (OPTIONS is optional)
  - "REPLOT" (no OPTIONS)
  - "SECRET" (no OPTIONS)
 OPTIONS is a list of module names, eg;
  ['mod1.py', 'mod2.m', 'mod3']
```

The server will then reply with JSON in the following form:

```
[RESULT, MESSAGE]
 RESULT is either "ACK" for success, or "NAK" for failure.
 MESSAGE is a list of messages. This can be an error, or the resu
lt of the command.
```

3.3 ViPERS NetManager

NetManager presents a JSON-encoded socket API that can be used to change the networking state of the host system. It is designed to run as root and enable or disable specific network interfaces, as well as add, remove, and/or connect to wireless access points, and finally can also set the date and time.

NetManager modifies the network interfaces by rewriting the files in `/etc/network/interfaces.d/`, in particular, `eth0` for Ethernet and `wlan0` for wireless. Additionally, it uses `wpa_cli`[26] for scanning and managing wireless access points. For a useful list of `wpa_cli` commands, see McLeod.[34]

In addition to helping connect to existing Ethernet or wireless networks, NetManager can also configure the interfaces to host their own network. In this case, the device will take on a specific IP address:

- `10.10.0.1` when hosting an Ethernet network
- `11.11.0.1` when hosting a wireless network (called "ARTEMIS")

This is done through `hostapd`[27] and `isc-dhcp-server`.[28]

Since these two networks are separated, one can have the device host *both* a wired and wireless network.

The most likely point of friction will occur with the interface *name* and the *wireless driver*. There are a few places where these occur and are pointed out in the following.

3.3.1 Implementation Details

NetManager consists of two classes and a set of config definitions (Table 5).

Table 5 Classes and configuration functions used by NetManager

Item	Type	Description
DateWrapper	Class	Used for setting system date and time.
WpaCliWrapper	Class	Used for issuing commands to wpa_cli.
InterfaceWrapper	Class	Sets the interface files and restarts the interfaces.
NetManager	Class	Wraps the above three classes and presents the API.
_get_configs_test()	Config	Contains configuration dictionary for test cases.
_get_configs_1404()	Config	Contains configuration dictionary for Snickerdoodle Ubuntu 14.04.
_get_configs_1604()	Config	Contains configuration dictionary for Ubuntu 16.04.

3.3.1.1 Class Specifics

When run, the `NetManager` class is created. This first determines the platform release (e.g., Ubuntu 16.06) and selects the appropriate config. Next, it creates a `WpaCliWrapper` that finds `wpa_cli` and determines if it needs root privileges to run.

The `NetManager` then listens for valid JSON commands (see Section 3.3.4) on port `6584` forever. If a valid message is received, it attempts to execute it.

When a valid command is received, the appropriate config text is retrieved and filled in, and the `update_interface()` function is run. This function 1) disables the interface (e.g., `ifdown eth0` or `ifdown wlan0`), which ensures that the interface is brought down according to the current config mode it is in; 2) overwrites the interface file with the new config; and 3) enables the interface with the new config (e.g., `ifup eth0`).

3.3.1.2 Configs

The config functions return a nested dictionary of contents for the `/etc/network/interfaces.d/` files.

The dictionary structure follows that of the API: `CONFIG[<INTERFACE>][<MODE>]`, for example; `CONFIG["WLAN0"]["DHCP-HOST"]` would return the contents that will be inserted into `/etc/network/interfaces.d/wlan0`.

Some config strings require additional information, specifically the `STATIC` modes require a static IP and netmask.

In the following, we annotate some of the configs specifically.

3.3.1.3 Disable

These are empty config files used to disable the interface:

```
# Wired DISABLED (NetManager)
```

3.3.1.4 Static

These set the static IP address and netmask and attempt to get network time. Note: If we are unable to get network time we include the `|| true` so that the interface continues to be brought up.

```
# Wired STATIC CLIENT (NetManager)

auto eth0
allow-hotplug eth0
iface eth0 inet static
address %s
netmask %s
post-up ntpdate ntp.ubuntu.com time.nist.gov || true
```

3.3.1.5 DHCP-client

This example is from the 16.04 config, where one can see that the driver and wpa-conf are also specified. The user may need to replace the wpa-driver argument with their own driver.

```
# Wireless DHCP CLIENT (NetManager)

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
  wpa-driver nl80211
  wpa-conf /etc/wpa_supplicant.conf
post-up ntpdate ntp.ubuntu.com time.nist.gov || true
```

The 14.04 version uses pre-up and post-up scripts, which may exist for the user's driver as well:

```
pre-up /usr/share/wl18xx/sta_start.sh || true
# ...
post-down /usr/share/wl18xx/sta_stop.sh || true
```

3.3.1.6 DHCP-HOST Ethernet

In this case, we have to restart the `isc-dhcp-server` *after* we assign the static IP address. When `isc-dhcp-server` sees that the device has this "magic" static IP address, it successfully starts and begins serving DHCP addresses (see dhcpd.conf in Section 3.3.1.8).

```
# Wired DHCP SERVER (NetManager)

auto eth0
allow-hotplug eth0
iface eth0 inet static
address 10.10.0.1
netmask 255.255.255.192
post-up service isc-dhcp-server restart
post-down service isc-dhcp-server restart
```

Similarly after we bring down the interface, we restart the `isc-dhcp-server`. The device no longer has the "magic" IP address and the `isc-dhcp-server` fails to start and does not serve IP addresses.

3.3.1.7 Wireless

This has a few additional steps. In addition to the `isc-dhcp-server`, we have to remove any lingering elements of old, obtained IP addresses from other networks. This is done with the `ip addr flush dev wlan0` in the `pre-up` line. Additionally, `hostapd` has to be started before the interface is online, so this also appears on the `pre-up` line.

```
# Wireless DHCP SERVER (NetManager)

auto wlan0
iface wlan0 inet static
pre-up ip addr flush dev wlan0 && service hostapd restart
address 11.11.0.1
netmask 255.255.255.0
post-up service isc-dhcp-server restart
post-down service hostapd stop && service isc-dhcp-server restart
&& ip addr flush dev wlan0
```

Note: The `pre-up` line appears *before* the static IP address is assigned.

Finally, in addition to restarting `isc-dhcp-server` on the `post-down` line, we also stop `hostapd`, and remove the static IP address with the `ip addr flush` command.

3.3.1.8 Additional Files

There are some additional files in `networking/install/` that one may want to customize.

`dhcpd.conf` contains the "magic" IP addresses for when the device enters into DHCP-HOST mode, as well as the range of addresses that will be handed out. Some of the key values are reproduced:

```
subnet 10.10.0.0 netmask 255.255.255.0 {
 range 10.10.0.2 10.10.0.17; # Address range of clients
 ...
 option domain-name "ARTEMISE";
 ...
 option routers 10.10.0.1;
 ...
}
```

`hostapd.conf` contains details for hosting the wireless access point and needs the interface and driver set properly:

```
interface=wlan0
driver=nl80211
```

`isc-dhcp-server` includes the interfaces on which one will run the DHCP mode:

```
INTERFACES="wlan0 eth0"
```

3.3.2 Platform Dependence

Because Netmanager modifies the network interface files, the user may have to customize it to their specific system. NetManager comes preconfigured for two basic systems: 1) Ubuntu 16.04 and 2) snickerdoodle Ubuntu 14.04.

Both versions are fairly generic, and should likely work on most systems. However, both versions have some scripts or driver specifications specific to the wireless driver found on the Snickerdoodle board (if these scripts do not exist, they are not run). The user's system may have similar driver-specific scripts. See Sections 3.1.1.5, 3.1.1.6, and 3.1.1.8 for how to specify the driver and interface name.

3.3.3 Testing and Command-Line Arguments

Running NetManager with the `-t` argument places it testing mode. Instead of modifying the `/etc/network/interfaces.d/` files, it will create a new directory `test/` and modify `eth0` and `wlan0` in that directory. Similarly, instead of actually issuing commands to `wpa_cli`, it will instead print the command to the screen. This is useful for automated unit tests.

Additionally, one can run NetManager with a higher level of debugging using the `-d` argument.

Finally, if the wireless interface is not `wlan0`, the user can specify this with the `-i` argument. A full example would be the following:

```
python netmanager.py -d -t -i wlp3s0
```

3.3.4 JSON API

Netmanager runs a TCP server on port 6584 and accepts a JSON-encoded string for commands. The most recent API can be found in the NetManager class in `netmanager.py`. It is reproduced here for convenience.

```
NetManager - Control and manage network interfaces

Control network interfaces (eth0, wlan0) in access point, DHCP, o
r static modes.

Public API:
This runs on port: 6584 and accepts a JSON string for data (descr
ibed below).

DATE - Set the system datetime
{"DATE": <UNIX TIMESTAMP>}
{"DATE": 1557518882.967937}
To get the unix timestamp: time.time()

ETH0 - Set the ethernet adapter
{"ETH0": [<COMMAND>, <MODE>, <OPTION>]}
{"ETH0": ["DHCP-HOST"]}
{"ETH0": ["DHCP-CLIENT"]}
{"ETH0": ["STATIC", "10.10.0.1", "255.255.255.192"]}
{"ETH0": ["STATIC", "192.168.1.144", "255.255.255.255"]}
{"ETH0": ["DISABLE", ]}

WLAN0 - Set the wireless adapter
{"WLAN0": [<COMMAND>, <MODE>, <OPTION>]}
{"WLAN0": ["DHCP-HOST"]}
{"WLAN0": ["DHCP-CLIENT"]}
{"WLAN0": ["STATIC", "10.10.0.1", "255.255.255.192"]}
{"WLAN0": ["STATIC", "192.168.1.144", "255.255.255.255"]}
{"WLAN0": ["DISABLE", ]}

WPA_CLI - Send commands to wpa_cli for configuring wireless netwo
rks
NOTE: This uses a list of command, option pairs. The commands are
executed
  in the order in which they appear in the list.
```

```
NOTE: <idx> is an integer specifying the network index in list_ne
tworks
{"WPA_CLI": [(<COMMAND>, <OPTION>, <OPTION>, ...), ]}
{"WPA_CLI": [("scan", ), ]}
{"WPA_CLI": [("scan_results", ), ]}
{"WPA_CLI": [("list_networks", ), ]}
{"WPA_CLI": [("enable_network", <idx>), ]}
{"WPA_CLI": [("select_network", <idx>), ]}
{"WPA_CLI": [("disable_network", <idx>), ]}
{"WPA_CLI": [("remove_network", <idx>), ]}
{"WPA_CLI": [("add_network", ), ]}
{"WPA_CLI": [("set_network", <idx>, key, value), ]}
{"WPA_CLI": [("reassociate", ), ]}

NOTE: Multiple commands may be sent at once:
{"WPA_CLI": [("scan", ), ("scan_results", ), ]}
{"WPA_CLI": [
  ("set_network", 5, "ssid", "blueberry_pie"),
  ("set_network", 5, "psk", "kiss_the_cook"),
  ("set_network", 5, "scan_ssid", True),
]}

NOTE: One can also send multiple types of commands at once:
{
  "WLAN0": ["DHCP"],
  "WPA_CLI": [
    ("set_network", 5, "ssid", "blueberry_pie"),
    ("set_network", 5, "psk", "kiss_the_cook"),
    ("set_network", 5, "scan_ssid", True),
  ]
}
```

3.4 ViPERS Python Virtual Environment

Only NetManager is written targeting Python 2.7 with only the standard libraries, both the ViPERS webserver and Dataserver require Python 3.6+ and third-party packages. As a result, the ViPERS installer checks for the presence of a Python 3.6 interpreter and if none is found installs one. A virtual environment is then created using Virtualenvwrapper.[29] One can activate this environment with the command `workon vipers`, which will make Python 3.6 the default Python and enable all of the third-party packages. To stop working on the environment, issue the command `deactivate`. Note that packages installed in this environment will not be available to the system Python 2.7 or the system Python 3. Additionally, a symbolic link is created in `/usr/local/vipers/env`, which allows the startup scripts for the webserver and Dataserver to access the virtual environment.

Additionally, the software licenses for the core Python packages installed in this environment are shown in Section 7. These packages may be dependent on additional packages, whose license can be obtained from the parent package.

4. Case Study: ViPERS-ARTEMIS

The Electric- and Magnetic-Field sensing team at the US Army Combat Capabilities Development Command (CCDC) Army Research Laboratory (ARL) has developed a hardware platform called Autonomous Real-Time Electric/Magnetic Integrated Sensor (ARTEMIS). The latest "mobile" version is a Snickerdoodle-based IoT electric-power sensing and processing platform. ViPERS-ARTEMIS is a customization layer on ViPERS that supports this by interfacing a number of power-processing specific modules and plots. These custom modules are summarized in Table 6 and discussed in detail in the following sections. Additionally, ViPERS-ARTEMIS contains software to integrate into the existing power processing software; distributed Live Animated Multi-Phasor (dLAMP), published separately as a MATLAB analysis tool.

Table 6 Custom ViPERS-ARTEMIS modules and their descriptions

Module	Description
`core100.py`	Set hardware (filter frequencies, log raw data to internal secure digital [SD] card, etc.) properties.
`synchro.py`	Select data from a range of sources and format it to a standard dLAMP packet.
`relative.py`	Combine synchrophasors into meaningful relative phasor and power measurements.
`influx.py`	Store synchrophasors or relative and power phasors to InfluxDB.
`extras.py`	Store GPS, inertial measurement unit (IMU), and battery data to InfluxDB.
`proxy.py`	Use to connect between a data server (`synchro.py` or `relative.py`) and a dLAMP-PC server.

Note: `core100.py` should successfully exit and show "zombie" status in Dataserver, while all the other modules are designed to run forever and should show a "running" or "sleeping" status.

4.1 Core100 Module

Purpose: Configure ARTEMIS hardware, field-programmable gate array software, and Core100 software.

Command-line arguments: `--debug`

Configuration file: `conf/phasors.conf` (scheme: toml)

This program waits for Core100 to start then sets initial filtering parameters, data source, and raw data logging. Table 7 details the fields this module uses.

Table 7 Fields used by `core100.py` in the `phasors.conf` configuration file

Configuration field	Description
hardware/initial_arltx	Initial ARLTX command to send. Useful for setting filtering parameters.
hardware/log_raw_sd1	Enable raw data writing to internal SD card 1.
hardware/log_raw_sd2	Enable raw data writing to internal SD card 2.
hardware/bank	Select hardware data source, BNC, BNC with attenuation, or LEMO.

4.2 Synchro Module

Purpose: Connect to data source and standardize output to dLAMP packet.

Command-line arguments: `--debug --channels --noise --frequency --rate`

Configuration file: `conf/phasors.conf` (scheme: toml)

This program subscribes to a data source (either the hardware data source [dbus] or a number of simulated data sources) and then republishes a standardized dLAMP packet on port 5683. It will attempt to automatically reconnect to the dbus if the connection is dropped. Table 8 details the fields this module uses, Table 9 describes the values used for those fields, and Table 10 shows the command-line arguments.

Table 8 Fields used by `synchro.py` in the `phasors.conf` configuration file

Configuration field	Description
hardware/source	Source for data. See Table 9 for possible values.
hardware/broadcast	Broadcast data to 0.0.0.0 (all clients) instead of localhost.
hardware/n_channels	Number of channels.
hardware/n_harmonics	Number of harmonics per channel (currently, only 3 is supported).

Table 9 Values and their interpretation for the source field

Data source	Description
dbus	Read phasors published on the dbus by Core100.
inl	Read phasors published by the INL application.
simulator	Create a simulated three-phase and cable sensor signal. The first bank mimics data from a three-phase source (VA, IA, IB, IC) and the second bank mimics data from a cable sensor (H1, H2, H3, DDot).
simulator-3phase	Full three phase connection. The first bank simulates all voltages (VA, VB, VC, VN) and the second bank all currents (IA, IB, IC, IN). Useful for simulating DELTA or WYE circuits.
none	No data are read or published.

Table 10 Command-line arguments and their interpretation for the module

Command-line argument	Description
channels	Number of simulated phasor channels to produce.
noise	Magnitude noise of simulated phasors (in percent).
frequency	Standard deviation of frequency noise of simulated phasors (in Hz).
rate	Rate at which simulated phasors are produced.

4.3 Relative Module

Purpose: Convert raw synchrophasors into physically meaningful relative and power phasors.

Command-line arguments: `--debug`

Configuration file: `conf/phasors.conf` (scheme: toml)

This subscribes to the synchrophasors published on localhost:5683, transforms them to relative phasors and publishes those, their reference synchrophasor, and the corresponding power phasor, frequency, and total harmonic distortion value on port 5684. Table 11 details the fields this module uses.

Table 11 Fields used by `relative.py` in the `phasors.conf` configuration file

Configuration field	Description
hardware/broadcast	Broadcast data to 0.0.0.0 (all clients) instead of localhost.
hardware/bank	Used for scaling synchrophasors to physical units.
channels	Used for scaling synchrophasors to physical units.
circuits	Used for constructing referenced relative phasors and corresponding power phasors.

The circuits are scaled according to the following equation:

$$scaled = \frac{(measured * impedancefactor * multiplier)}{(sensitivity * attenuation)}, \qquad (1)$$

where the factors are pulled from the configuration file channels fields as described.

Channel scaling to physical units is set in the [[channels]] array (Table 12) and phase references are set in the [[circuits]] array (Table 13). The received data and the transmitted data should be in the form of level 10 or 11 dLAMP phasor packets.

Table 12 Fields of the [[channels]] list of dictionaries in the `phasors.conf` configuration file

Channel field	Description
name	The name of the channel. This is arbitrary, although channel names can contain neither a dot (".") or colon (":") and should be unique to avoid data loss in InfluxDB.
type	Can be one of; "current", "voltage", "hfield", "efield", "undefined"
impedance	"50" or "1M", which sets the impedance factor to 1 or 2, respectively.
sensitivity	The numeric component of the sensor sensitivity
units	The units of the sensor sensitivity.
multiplier	Arbitrary numeric multiplier. Useful for voltage/current transformers.
hw_channel	Integer identifier to the corresponding hardware channel (zero indexed).

Table 13 Fields of the [[circuits]] list of dictionaries in the `phasors.conf` configuration file.

Circuit field	Description
name	The name of the circuit. This is arbitrary, although channel names can contain neither a dot (".") or colon (":") and should be unique to avoid data loss in InfluxDB.
wiring	One of "3WYE", "3DELTA", "SINGLE", "SPLIT" (currently unused).
signal	An array of signal channels - this maps to the hw_channel used above.
reference	An array of reference channels - this maps to the hw_channel used above. This must be the same length as the signal array.
phase	An additional phase angle to subtract from the signal channel. This must be the same length as the signal array.

Once the channels are scaled, the signal and reference channels are specified. The phase of the reference channel(s) is then subtracted from the signal channel(s). This phase referencing transforms a synchrophasor into a relative phasor. Additionally, the power phasor is produced by multiplying the relative phasors magnitude by the magnitude of the reference channel. The combination of signal and reference channels is called a circuit, and 0-N circuits may be created.

4.4 Influx Module

Purpose: Listen to dLAMP phasor packets (either from `synchro.py` or `relative.py`) and store the results in InfluxDB.

Command-line arguments: `--debug --address --port`

Configuration file: `conf/influx.conf` (scheme: toml)

By default this module listens on `localhost:5684` for dLAMP Level 10 or 11 packets, parses the metadata from the ssn_name and channel names and inserts it into InfluxDB. The address and port can be specified with command-line arguments, allowing one to use this program to insert data from another dLAMP phasor source. For example, one can run `synchro.py` or `relative.py` on an ARTEMIS unit and use this program on a PC on the local area network to insert data into a local database, or a database in the cloud. Table 14 details the fields this module uses.

Table 14 Fields used by `influx.py` in the `influx.conf` configuration file.

Configuration field	Description
local/host	Hostname for local InfluxDB database (default: “localhost”).
local/port	Port for local InfluxDB database (default: 8086).
local/database	InfluxDB database to publish measurements to.
local/username	Username for authentication against the local InfluxDB database.
local/password	Password for authentication against the local InfluxDB database.
influx/use_line	Use influxdb-python’s default line protocol interpreter (more robust, but more computation) rather than the default string format method.
influx/interval	Interval for pushing data to Influx. Set this to zero to push data on every sample received.
influx/resample	Average over this period (in seconds). Note that averaging is done by real/imaginary components, so this can yield incorrect results for rapidly varying frequencies.

4.5 GPS / IMU / Battery Modules

Purpose: Store GPS, IMU, and battery information to InfluxDB.

Command-line arguments: `--debug, --interval`

Configuration file: `conf/influx.conf` (scheme: toml)

These three modules, gps.py, imu.py, and battery.py, log GPS, IMU (roll, pitch, yaw, and quaternion, and 9-degree-of-freedom), and battery data, respectively. The interval for each can be specified independently. Table 15 details the fields this module uses.

Table 15 Fields used by `extras.py` in the `influx.conf` configuration file

Configuration field	Description
local/host	Hostname for local InfluxDB database (default: “localhost”).
local/port	Port for local InfluxDB database (default: 8086).
local/database	InfluxDB database to publish measurements to.
local/username	Username for authentication against the local InfluxDB database.
local/password	Password for authentication against the local InfluxDB database.

4.6 Replication Module

Purpose: Duplicate measurements from the local InfluxDB to a cloud instance of InfluxDB.

Command-line arguments: `--debug`

Configuration file: `conf/influx.conf` (scheme: toml)

This module replicates data stored in the local database to data stored in a remote database. It queries the last `size` points from each `measurement` and then attempts to POST them to a cloud instance of InfluxDB. If successful, it then marks that region in a separate “replication” measurement that tracks what points have been replicated. On future passes, any successfully replicated regions from the “replication” measurement are excluded from the local data query. This allows both newer and older points to be replicated on the second pass. After the second pass, any overlapping regions of replication are combined into one active region. Eventually, this scheme allows all historical data to be replicated while simultaneously ensuring that the most recent data are also replicated. Table 16 details the fields this module uses.

Table 16 Fields used by `replication.py` in the `influx.conf` configuration file

Configuration field	Description
local/host	Hostname for local InfluxDB database (default: “localhost”).
local/port	Port for local InfluxDB database (default: 8086).
local/database	InfluxDB database to publish measurements to.
local/username	Username for authentication against the local InfluxDB database.
local/password	Password for authentication against the local InfluxDB database.
cloud/	Same parameters as local, but for the cloud InfluxDB instance.
replication/ interval	Interval for replicating points.
replication/size	How many points to query when replicating data.
replication/measurements	List of measurement names to replicate.

4.7 Proxy Module

Purpose: Connect output from `synchro.py` or `relative.py` to dLAMP-PC.

Command-line arguments: `--source --destination --port`

Configuration file: `<NONE>`

Since `synchro.py` and `relative.py` transmit data by hosting a TCP server, and dLAMP-PC connects to data sources by hosting a TCP server and sending connection information, the two cannot natively connect to one another. This program solves that by creating two clients and passing data between them. The first client connects to `source` port 5684 (relative phasors) by default and transmits data to the second client, specified by the IP address `destination` and `port`, which are specified by the command-line arguments.

4.8 Raw Data Module/API

Purpose: Allows users to remotely request and data from raw data files stored on the ARTEMIS sensors.

Command-line arguments: `<NONE>`

Configuration file: `<NONE>`

This API provides users with the ability to remotely request raw data from any time period, file, or storage location on the ARTEMIS sensor unit. After a request is made, this API will read data blocks from either one file or multiple files and produce a double array containing all of the raw data collected during the requested time period. If collecting data from multiple files, this API ensures that overlapping data are not concatenated. This API also allows users to stream and process raw data packets from a user datagram protocol or TCP stream.

From the ViPERS web application, users are able to download full files from both the internal SD cards and an external hard drive (HDD) if one is attached, as well as analyze both live and historical data from a raw data analyzer tool. This tool both provides a plot of the raw data, as well as some basic calculations for average frequency, root-mean-squared (RMS) amplitude, and total harmonic distortion (THD) to help the user further analyze the raw data.

4.9 Backup to HDD Module

Purpose: Allows users to remotely toggle data backups to a hard disk whenever one is attached to ARTEMIS sensor.

Command-line arguments: `<NONE>`

Configuration file: `<NONE>`

This module, once started, either via command line or the ViPERS web application, first detects whether an external HDD is attached to the ARTEMIS sensor unit. If one is attached, it will begin backing up files from local SD cards to an HDD. This module waits until there is 4-GB worth of data stored on the SD cards and then concatenates these files into one large file. This module only concatenates files together if they occur one after the other. This feature allows users to determine if there are any time periods where no raw data were collected by the sensor. This module also writes header files for each file copied to the HDD so that they can be processed using the dLAMP software at a later time.

This module should be stopped only from either the ViPERS web application or by issuing the kill command with the –SIGTERM argument in the command line. This will ensure that the module exits cleanly, and finishes copying the current file it is working on to the HDD. If the HDD is just ejected from the device, this program will wait for a new HDD to be attached in order to continue the backup. However, some files on the original HDD may be corrupted as they have not been properly flushed (data written and files closed) before ejecting the disk.

4.10 dLAMP Integration

Purpose: Provide a Pythonic API for encoding and decoding standardized dLAMP binary data.[32]

Command-line arguments: `<NONE>`

Configuration file: `<NONE>`

The dLAMP Python module was designed to encode data in a standardized binary format (the dLAMP data standard). It provides a Pythonic API for formatting arbitrary data as binary, hiding the low-level operations of converting Python data to the dLAMP binary format and back. Higher-level modules such as the `synchro.py`, `relative.py`, and `influx.py` use the dLAMP module as a backend for communication. Due to the flexibility of the dLAMP data packets, the dLAMP module can be used for communication between different embedded systems or other PCs. The end user is presented with a file-like object[33] that provides well-known methods such as `read()` and `write()`, as well as classes that encode provided data into specific dLAMP formats. This file-like object can write either to a binary file or a TCP socket making communication easy. Other Python features such as context manager functionality are provided to make usage as simple as possible.

5. Troubleshooting

Occasionally either the web application, the plots, networking, or the InfluxDB database will fail to perform. This section discusses some common issues with each.

5.1 Common Webapp Issues

The log file is located at `/var/run/vipers/vwebserver.log`. The application may be restarted with `sudo service vwebserver restart`. On Ubuntu 16.04 and later the log may also be viewed with; `sudo journalctl -u vwebserver`.

Unless the system date and time are set to within a few years of the current date and time, the application will fail to start. Simply update using network or GPS time and reconnect. Due to this issue, the ViPERS installer includes the `fake-hwclock` package. Once network or GPS time is obtained, this package writes that to disk so that after a system restart, the time is reset to the last known time. This allows the web application to start and from there one can adjust to the correct time.

5.2 Common Dataserver Issues

The log file is located at `/var/run/vipers/vdataserver.log`. The application may be restarted with `sudo service vdataserver restart`. On Ubuntu 16.04 and later the log may also be viewed with; `sudo journalctl -u vdataserver`.

Dataserver is fairly robust and no issues have been reported yet.

The main log file contains both the log for Dataserver and the `bokeh serve` subprocess. Other processes have their log files piped to unique files in `vipers/dataserver/modules/logs`.

When the "zombie" state is shown, this can either indicate a successful exit or a crash. Unfortunately, there is no way of differentiating between the two.

5.3 Common NetManager Issues

The log file is located at `/var/run/vipers/vnetmanager.log`. The application may be restarted with `sudo service vnetmanager restart`. On Ubuntu 16.04 and later the log may also be viewed with; `sudo journalctl -u vnetmanager`.

NetManager was designed to run under the system Python, which is still Python 2.7 for most distributions. When run under Python 3, some of the exception classes have changed names and will fail.

When setting the date, one may receive a “NAK” message despite the action being carried out.

Depending on how long the system takes to execute “ifup”, one may need to extend the timeouts.

One can also manually execute `wpa_cli` commands. A summary of wpa_cli commands can be found online.[34]

5.4 Common InfluxDB Issues

The log file is located at `/var/log/influxdb/influxdb.log`. The application may be restarted with `sudo service influxdb restart`. On Ubuntu 16.04 and later the log may also be viewed with; `sudo journalctl -u influxdb`. One can replace `influxdb` in the above with `telegraf`, `chronograf`, or `kapacitor` to view the corresponding log or restart the corresponding service.

The InfluxDB documentation[35] is quite extensive should be a primary source of information for understanding and debugging the database.

When writing data, the first write to a field determines the data type (string, integer, Boolean, or float). Future writes will be ignored if they do not match this type. This can be an issue if the data is floating point, but the first measurement is a zero represented as an integer—all subsequent writes will be ignored! Therefore, it is helpful to explicitly type cast before formatting the insert string.

InfluxDB can consume a tremendous amount of memory on startup. This is when it scan previous write-ahead-log (wal) files and attempts to write their contents into the database. This is especially pronounced when using the `inmem` index. Switching to the `tsi1` index stores the cardinality index on the disk and improves memory usage.

If the system was shut down while writing, one of these files can become corrupted. On startup, InfluxDB will repair the first instance of this corruption, but if another corrupted file is encountered it will crash. As a result, either the corrupted files must be removed, or InfluxDB must be continuously restarted until all files are repaired.

If Chronograf is not showing any data under the “Explore” tab, first ensure that a connection has been created under the “Configuration” tab. If no source is still showing, then try another browser or clearing the browser cache.

6. Software Licenses

Tables 33 and 34 list the core software and their licenses used in this project.

Table 17 Core software, their license, and their license file

Software	License	URL
Python	BSD-Style	https://docs.python.org/3/license.html
Flask	BSD	https://github.com/mbr/flask-bootstrap/blob/master/LICENSE
nginx	BSD 2-clause	http://nginx.org/LICENSE
Bootstrap	MIT	https://github.com/twbs/bootstrap/blob/v4.1.3/LICENSE
Telegraf	MIT	https://github.com/influxdata/telegraf/blob/master/LICENSE
InfluxDB	MIT	https://github.com/influxdata/influxdb/blob/master/LICENSE
Chronograf	MIT	https://github.com/influxdata/chronograf/blob/master/LICENSE
Kapacitor	MIT	https://github.com/influxdata/kapacitor/blob/master/LICENSE
wpa_cli	GPL 2, BSD	https://linux.die.net/man/8/wpa_cli
hostapd	BSD	https://w1.fi/cgit/hostap/plain/hostapd/README
isc-dhcp-server	MPL 2.0	https://www.isc.org/licenses/

Table 18 Python packages, their license, and their license file.

Package	License	URL
Bokeh	New BSD	https://github.com/bokeh/bokeh/blob/master/LICENSE.txt
Flask-Bootstrap4	BSD	https://github.com/mbr/flask-bootstrap/blob/master/LICENSE
Flask-Login	MIT	https://github.com/maxcountryman/flask-login/blob/master/LICENSE
Flask-Migrate	MIT	https://github.com/miguelgrinberg/Flask-Migrate/blob/master/LICENSE
Flask-SQLAlchemy	BSD	https://github.com/mitsuhiko/flask-sqlalchemy/blob/master/LICENSE
Flask-WTF	BSD	https://github.com/lepture/flask-wtf/blob/master/LICENSE
gps3	BSD	https://pypi.org/project/gps/
gunicorn	MIT	https://pypi.org/project/gunicorn/
influxdb	MIT	https://github.com/influxdata/influxdb-python/blob/master/LICENSE
markdown2	MIT	https://github.com/trentm/python-markdown2/blob/master/LICENSE.txt
numpy	BSD	http://www.numpy.org/license.html
pandas	BSD	https://github.com/pandas-dev/pandas/blob/master/LICENSE
psutil	BSD	https://github.com/giampaolo/psutil/blob/master/LICENSE
pydbus	LBGLv2.1+	https://github.com/LEW21/pydbus/blob/master/LICENSE
pytest	MIT	http://doc.pytest.org/en/latest/license.html
virtualenvwrapper	MIT	https://pypi.org/project/virtualenvwrapper/
Werkzeug	BSD	https://pypi.org/project/Werkzeug/
WTForms	BSD	https://github.com/wtforms/wtforms/blob/master/LICENSE.rst

7. Conclusion

The CCDC ARL is always looking to make transitions of cutting-edge technology available to the Warfighter. As more of this technology requires powerful, embedded computers, both researchers and Warfighters need a flexible platform on which they can collaborate. ViPERS solves this need by providing its three key services. Dataserver provides a platform on which the researcher can develop software and visualizations. The webserver provides a portable interface that can be used with desktops, laptops, tablets, or mobile devices and provides both the researcher and Warfighter a way to live, real-time, streaming data. Tying everything together is NetManager, which allows the networking interface of the IoT device to be configured, adapting it to either a research, prototype, or field environment. We expect that this software will rapidly accelerate technology development, demonstration, and transition.

8. References

1. Foundation RP. Teach, Learn, and Make with Raspberry Pi Raspberry Pi. [accessed 2019 Sep 12]. https://www.raspberrypi.org/.
2. krtkl. Snickerdoodle | krtkl. [accessed 2019 Sep 12]. https://krtkl.com/snickerdoodle/.
3. BeagleBoard.org. BeagleBoard.Org - bone. [accessed 2019 Sep 12]. https://beagleboard.org/bone.
4. Intel. Intel NUC. [accessed 2019 Sep 12]. https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html.
5. The Python Software Foundation. Welcome to Python.Org. [accessed 2019 Sep 12]. https://www.python.org/.
6. Cass S. The top programming languages 2019 - IEEE Spectrum. [accessed 2019 Sep 12]. https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019.
7. Foundation DS. Django. [accessed 2019 Sep 10]. https://www.djangoproject.com/.
8. Mönnich A, Ronacher A, Lord D, Unterwaditzer M. Flask. Flask. [accessed 2019 Sep 12]. https://palletsprojects.com/p/flask/.
9. Chesneau B. Gunicorn - Python WSGI HTTP server for UNIX. [accessed 2019 Sep 12]. https://gunicorn.org/.
10. NGINX Inc. NGINX | High performance load balancer, web server, & reverse proxy. [accessed 2019 Sep 12]. https://www.nginx.com/.
11. NumFOCUS. Matplotlib: Python plotting Matplotlib 3.1.1 documentation. [accessed 2019 Sep 12]. https://matplotlib.org/.
12. The QT Company. Qt for Python | The official Python bindings for Qt. [accessed 2019 Sep 12]. https://www.qt.io/qt-for-python.
13. Plotly. Modern analytic apps for the enterprise - Plotly. [accessed 2019 Sep 12]. https://plot.ly/.
14. Plotly. Dash user guide and documentation - Dash by Plotly. [accessed 2019 Oct 21]. https://dash.plot.ly/.

15. Bird S, Canvan L, Hulsey C, Paprocki M, Rudiger P, Van de Ven B. Welcome to Bokeh. Bokeh 1.3.4 documentation. [accessed 2019 Sep 12]. https://bokeh.pydata.org/en/latest/.

16. Oracle. MySQL. [accessed 2019 Sep 12]. https://www.mysql.com/.

17. PostgreSQL. PostgreSQL The world's most advanced open source database. [accessed 2019 Sep 10]. https://www.postgresql.org/.

18. SQLite Consortium. SQLite home page. [accessed 2019 Sep 12]. https://www.sqlite.org/index.html.

19. Oracle. Oracle Berkeley DB downloads. [accessed 2019 Sep 12]. https://www.oracle.com/database/technologies/related/berkeleydb-downloads.html.

20. redislabs. Redis. [accessed 2019 Sep 12]. https://redis.io/.

21. The OpenTSDB Authors. OpenTSDB - a distributed, scalable monitoring system. [accessed 2019 Sep 10]. http://opentsdb.net/.

22. Timescale Inc. Timescale. [accessed 2019 Sep 10]. https://www.timescale.com/

23. InfluxData. InfluxData The leading platform for monitoring & analytics. [accessed 2019 Sep 10]. https://www.influxdata.com/products/.

24. Freedman M. TimescaleDB vs. InfluxDB: Purpose built differently for time-series data. Timescale Blog. 2019 [accessed 2019 Sep 10]. https://blog.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/.

25. Grafana Labs. Grafana. [accessed 2019 Sep 12]. https://grafana.com/.

26. Malinen J. Wpa_cli WPA client - Linux man page. [accessed 2019 Sep 10]. https://linux.die.net/man/8/wpa_cli.

27. Malinen J. Hostapd IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. [accessed 2019 Sep 10]. https://w1.fi/hostapd/.

28. Ubuntu Wiki. Isc-dhcp-server. [accessed 2019 Sep 10]. https://help.ubuntu.com/community/isc-dhcp-server.

29. Bird S, Canvan L, Hulsey C, Paprocki M, Rudiger P, Van de Ven B. Running a Bokeh Server Bokeh 1.3.4 documentation. [accessed 2019 Sep 12]. https://bokeh.pydata.org/en/latest/docs/user_guide/server.html.

29. Hellmann D. Virtualenvwrapper 4.8.5.Dev5 virtualenvwrapper 4.8.5.Dev5 documentation. [accessed 2019 Sep 12]. https://virtualenvwrapper.readthedocs.io/en/latest/.

30. Rodola G. Psutil documentation psutil 5.6.4 documentation. [accessed 2019 Sep 12]. https://psutil.readthedocs.io/en/latest/.

31. Preston-Werner T. Toml-lang/toml. 2019 [accessed 2019 Sep 12]. https://github.com/toml-lang/toml.

32. Parks B, Hull D. US Army Research Laboratory Live Animated Multi Phasor (ARL-LAMP) analysis software. Adelphi Laboratory Center (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-TR-7436.

33. The Python Software Foundation. Glossary — Python 3.8.0 Documentation. [accessed 2019 Oct 2]. https://docs.python.org/3/glossary.html#term-file-object.

34. McLeod R. WPA CLI commands. Gist. [accessed 2019 Sep 10]. https://gist.github.com/penguinpowernz/1d36a38af4fac4553562410e0bd8d6cf.

35. influxdata. InfluxDB 1.7 documentation | InfluxData Documentation. [accessed 2019 Sep 12]. https://docs.influxdata.com/influxdb/v1.7/.

List of Symbols, Abbreviations, and Acronyms

API	application programming interface
ARL	Army Research Laboratory
ARTEMIS	Autonomous Real-Time Electric/Magnetic Integrated Sensor
CCDC	US Army Combat Capabilities Development Command
DAQ	data acquisition
DHCP	Dynamic Host Configuration Protocol
dLAMP	distributed Live Animated Multi-Phasor
DOD	Department of Defense
GPS	global positioning system
HDD	hard drive
IMU	inertial measurement unit
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	local area network
PC	personal computer
SBC	single-board computer
SD	secure digital
SQL	Structured Query Language
SSH	secure shell
SWaP	size, weight, and power
TCP	Transmission Control Protocol
TOML	Tom's Obvious Markup Language
UDP	User Datagram Protocol
ViPERS	Visualization and Processing for Embedded Research Systems

WAN	wide area network
WSGI	webserver gateway interface

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	CCDC ARL FCDD RLD CL TECH LIB
2 (PDF)	CCDC ARL FCDD RLS SP K E CLAYTOR FCDD RLS S A GEORGE