Benthic Boundary Layer Project

Kilo Nalu Real Time Array

Requirements Document and Data Management Plan

Sep 13, 2011

Dr. Margaret McManus
Department of Oceanography
University of Hawaii at Manoa

TABLE OF CONTENTS

SUMMARY 3

REQUIREMENTS 3

SOFTWARE REQUIREMENTS 3

NEAR REAL-TIME STREAMING OF SCIENTIFIC DATA 3

NEAR REAL-TIME STREAMING OF ENGINEERING DATA 3

RAW DATA ARCHIVAL 3

NEAR REAL-TIME PROGRAMMATIC ACCESS 4

SNAPSHOT-BASED WEB ACCESS 4

SIMPLE, ASCII-BASED DATA ACCESS 4

TIME-BASED ARCHIVAL QUERIES BY INSTRUMENT 4

INSTRUMENT CONTROL 4

SYSTEM EXTENSIBILITY 4

UPTIME EXPECTATIONS 4

HARDWARE REQUIREMENTS 5

SHORESIDE STATION DATA SYSTEMS 5

CAMPUS DATA SYSTEMS 5

DATA SYSTEMS CONNECTIVITY 5

DATA MANAGEMENT PLAN 6

SUMMARY 6

RING BUFFER NETWORK BUS SOFTWARE 7

NAMING CONVENTIONS 7

SERVER NAME CONVENTIONS 8

SOURCE NAME CONVENTIONS 8

CHANNEL NAME CONVENTIONS 9

FILE NAMING CONVENTIONS 9

TCP-BASED STREAMS OF SCIENTIFIC DATA 10

TCP-BASED STREAMS OF ENGINEERING DATA 10

ONLINE AND OFFLINE FILE-BASED ARCHIVES 11

DATA TURBINE PLUGINS FOR PROGRAMMATIC DATA ACCESS 11

DATA TURBINE PLUGINS FOR SCHEDULED PROCESSING 11

DATA TURBINE WEB-BASED ADMINISTRATION 12

MANAGING DATA AVAILABILITY VIA THRESHOLD ALARMS 12

LINUX-BASED SERVER SYSTEMS 12

WIRELESS VPN LINK 13

APPENDIX A - ARCHITECTURE DIAGRAM 14

1. Summary

The BBL Project intends to stream data from various oceanographic instruments connected to the Kilo Nalu Nearshore Reef Observatory's underwater cabled array over an ethernet network. This cabled array connects to the shoreside station via fiber optic cables, which provide ethernet access to the instruments from computer systems located in the shoreside station. This document enumerates some of the hardware and software requirements needed to enable near real-time, continuous streaming of the data from the instrumented array to a variety of client applications.

Likewise, this document outlines a plan for implementing the near real-time system using Object Ring Buffer technology to access and control the various data streams. Many of the data streams will come from the scientific instruments in the water, while others will include engineering data that inform the Kilo Nalu team about the status of the system and internal environmental conditions within the array components.

2. Requirements

2.1. Software Requirements

2.1.1. Near real-time streaming of scientific data

The software system must provide near real-time data from the scientific instruments to the desktop systems of researchers monitoring the array.

Near real-time in this context will mean that the raw data stream will be available within seconds from collection in unprocessed form. Processed versions of the data streams should be available within a fraction of a minute from collection, depending on the extent of the processing. Reliability and integrity of the data stream is prioritized over speed of the data stream.

There is no current requirement to provide absolute real-time data (such as video over RTP) because the instruments largely collect time series measurements rather than a continuous feed. The software system should be extensible enough to enable *real time* data acquisition in future plans for the Kilo Nalu array.

2.1.2. Near real-time streaming of engineering data

The software system must provide near real-time engineering data from each of the array's nodes to the desktop systems of researchers monitoring the array.

Near real-time in this context will mean that the raw engineering data stream will be available within seconds from collection in unprocessed form. Processed versions of the engineering data streams should be available within a fraction of a minute from collection, depending on the extent of the processing. Reliability and integrity of the data stream is prioritized over speed of the data stream.

There is no current requirement to provide absolute real-time engineering data, although feedback to the engineering team must occur rapidly in order to allow them to respond to environmental problems within the nodes of the array.

2.1.3. Raw data archival

Each raw scientific data stream must be permanently archived to both online disk and offline media in order to reduce the potential for data loss. *Raw data* in this context will mean the original bytes of data coming from each instrument in its native format. Engineering data must also be archived in order to analyze trends for troubleshooting problems in the array's hardware. Offline archival will occur using an

automated system that relies very little on human intervention (i.e. using a tape auto changer or external hard disk rather than manual rotation of media).

2.1.4. Near real-time programmatic access

Each of the scientific data streams must be accessible programmatically via a uniform and open Application Programming Interface (API) in order to allow researchers to process the data in near real-time. The API should accommodate common data analysis applications used in oceanography such as Matlab. Likewise, the API should accommodate lower-level programming languages in order to process and display the data in near real time.

2.1.5. Snapshot-based Web access

Likewise, the API must accommodate lower-level programming languages in order to enable middleware applications to use the data for presentation over the Web. The system must allow for on-the-fly transformation of the data into graphs and images that convey the current and historical observations on the Kilo Nalu array.

2.1.6. Simple, ASCII-based data access

Based on the raw archival of data streams from the scientific instruments and engineering instruments, the software system must also be able to provide the data streams as simple, ASCII text based data in near real time. This will allow for easy import into multiple analysis applications, and will also enable quick inspection of the data in a human-readable format.

2.1.7. Time-based archival queries by instrument

The software system must allow searching of the time series data from each instrument. The online archive should accommodate at least one year worth of scientific data streams in order to identify seasonal events. A goal is to have multiple years of archival data available at least in summary form rather than the entire raw data streams.

2.1.8. Instrument control

The API must accommodate communication with the scientific instruments in order to at least connect to, start and stop the array. A goal is to also provide a uniform programming interface that also allows re-configuration of the instruments in order to capture anticipated events. For instance, instrument sample rates could be changed on-the-fly during high-intensity swell events.

2.1.9. System extensibility

The software system must be able to accommodate increasing numbers of instruments being deployed on the array without shutting the system down, and must be able to accommodate multiple types of instruments in an extendable manner.

2.1.10. Uptime expectations

The reliability of the software system must exceed the reliability of the hardware array. For instance, if the the Kilo Nalu instruments are unavailable, the online archived streams must still be available. A goal is to provide 99% uptime, with infrequent server maintenance downtime.

2.2. Hardware Requirements

2.2.1. Shoreside station data systems

The data collection server at the Shoreside station must be configured as the definitive archive of the data streams. It must provide both online and offline archival of the data streams, and must be able to replicate data to other servers in the BBL project. The system must have enough physical memory to accommodate at least 1 month of streaming data, and must have enough disk space to accommodate 1 year of streaming data. The disk sub-system must be fault tolerant and must be able to endure 1 full disk failure without losing the entire sub-system. It must also accommodate hot-swapping of disks during maintenance or upgrades to the filesystem. The server system must provide a tiered backup facility of daily, weekly, and monthly data archives to offline media (such as tape or external hard disk). Offline media must be physically removed from the Shoreside station on a monthly basis to prevent data loss in case of fire, etc.

2.2.2. Campus data systems

The data distribution server on the UH campus must have an equal capacity to that of the Shoreside station's server in terms of memory and disk space so that data may be replicated to the campus server in real time. This server must also provide access to the data as Web-based snapshots, at approximately a 15 minute time interval. This server must accommodate concurrent access to the data archive for multiple researchers and publicly accessible web pages.

2.2.3. Data systems connectivity

Connectivity between the Shoreside server and the Campus server must be reliable and must accommodate broadband data rates, at approximately 50 Megabits per second. This will allow the servers to maintain replicated data stores during the operation of the Kilo Nalu array, and will allow querying of the near real time streams by client applications.

3. Data Management Plan

3.1. Summary

The BBL project has evaluated software systems that meet the majority of the requirements outlined above, and has chosen to deploy an open source, Java-based, real time streaming engine called RBNB Data Turbine. The Data Turbine software was initially developed by Creare, Inc.[1] in Hanover, NH under contracts from NASA, the U.S. Department of Energy, the U.S. Air Force, and the National Science Foundation. Due to demand in the scientific community, Creare has released the 3.0 version of the software under an Apache open source license. The software has been adopted by major research initiatives including the NSF's Network for Earthquake Engineering Simulation (NEES)[2] and the National Ecological Observatory Network (NEON)[3]. NEES has in turn developed open source client applications that enable easy real-time access to data streams being served by the Data Turbine software.

The diagram in Figure 1 below shows the overview of the planned software architecture that uses the Data Turbine as the primary real-time data stream handler for both incoming and outgoing streams. Instruments deployed on the Kilo Nalu Cable Array will communicate with the Data Turbine via customized Java driver software that understands the stream formats of each instrument. The primary Data Turbine installation will mirror the data streams to the secondary Data Turbine installation at the UH Manoa campus via a wireless VPN connection. Various data handling clients will be written to convert data into readable ASCII streams, archive data to disk-based files, and to access data via Matlab, the Kilo Nalu web page, and other software. Files stored on the Shore Station RAID array will be archived to external hard disk, and will also be archived on the Campus system via a nightly rsync process that mirrors the data directories over the VPN connection.

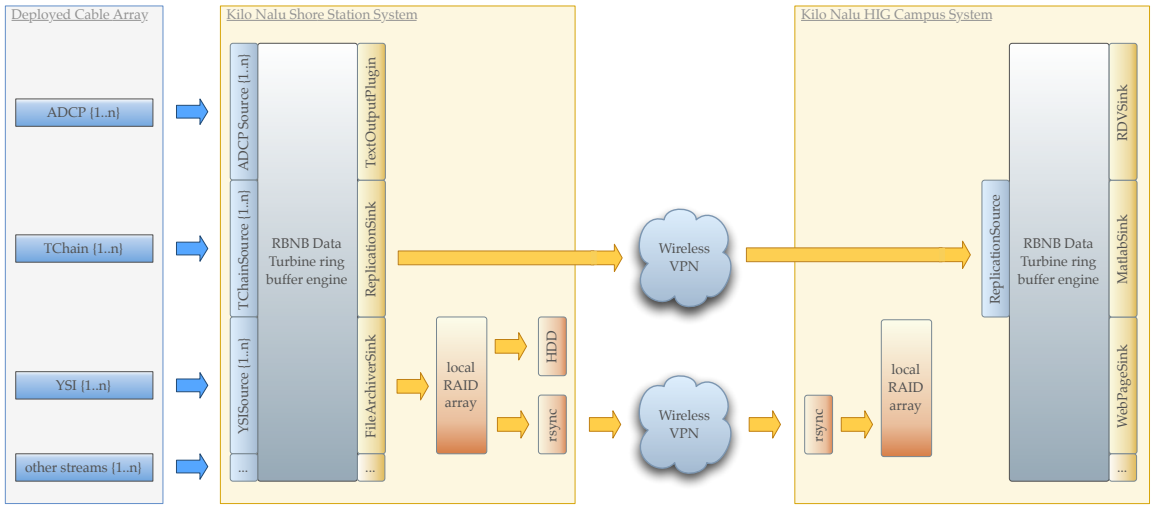


Figure 1. Overview of the Kilo Nalu real-time data software and backup architecture. Data are mirrored to the campus system using the Data Turbine software, and are accessed by desktop and web-based applications.

[1] see http://www.creare.com/rbnb

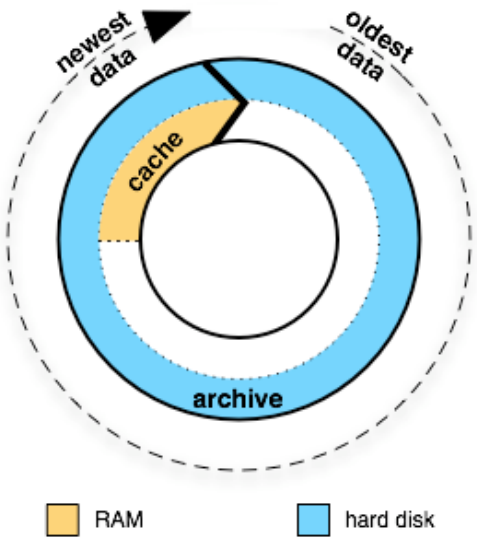
[2] see http://nees.org

[3] see http://neoninc.org

3.2. Ring Buffer Network Bus software

Note: *The following description of the ring buffer software summarizes the NEES IT website description of the Data Turbine, which provides a concise overview of the system architecture.*[4]

The basis of the RBNB Data Turbine software is a fixed-size data structure called a ring buffer. The ring buffer is a structure that continuously accepts new, incoming data by overwriting the oldest data in the ring buffer. The Data Turbine uses a new ring buffer for each data source that connects and streams data to the turbine. The amount of data that can be stored in the ring buffer is configurable, and depends on the amount of disk and memory available on the server system, the size of each observation being stored, and the rate at which data are saved.



Schematic diagram of the Data Turbine ring buffer courtesy of the NEES IT "Data Turbine Getting Started Guide".

The Data Turbine software archives data to hard disk, and also maintains a smaller cache copy of the most recent data for increased real-time performance. The amount of data that is available via the memory cache is limited by the amount of RAM installed in the server system. The amount of memory cache requested by all of the data sources must not exceed the available system RAM, else the Data Turbine will generate out-of-memory errors and will not serve data.

As mentioned above, applications that stream data to the Data Turbine are called *sources*, and applications that fetch data from the Data Turbine are called *sinks*. Source applications communicate with the Data Turbine by sending *channel map* objects to the server. Each channel map consists of a collection of time-stamped channels of data. Sink application request data from the Data Turbine by passing *channel map* objects as well, and specify which channels of data should be returned, along with the desired time frame. Sink applications can *subscribe* to a real-time, continuous stream from the Data Turbine, *monitor* a data stream by allowing the Data Turbine to 'skip ahead' and send the most recent data frame to avoid the stream from aborting, and *request* a specific duration of data from a source or sources.

The BBL Project will adopt the Data Turbine as the backbone of the Kilo Nalu real-time system, and will develop source applications that stream data from the various scientific and system monitoring instruments to the Data Turbine software installed in the shore station lab. Likewise, the BBL project will develop sink applications that fetch data from the Data Turbine for permanent archival, and will adopt open source client software that has been designed to present data from the Data Turbine in near real-time to researcher desktops and to the web. The built-in replication features of the Data Turbine will be utilized to mirror the Data Turbine archive from the Shore Station lab to the HIG building at the UH Manoa campus.

3.3. Naming conventions

In order to keep track of the various instruments, data streams, data servers, and archived data files, the BBL Project will adopt a set of naming conventions used to identify the various components. Researchers

[4] see http://it.nees.org/library/telepresence/getting-started-with-data-turbine-37.php

will then be able to connect to a specific server and request data from specific data channels collected by a specific data source.

3.3.1. Server Name Conventions

There will initially be two data servers installed in the Kilo Nalu real-time system, indicated in the table below. The definitive data archive will be stored on the Shore Station server, and will be mirrored to the HIG Campus server for distribution to the web and desktop client software. Data sink applications will use the Data Turbine names below to request data from the server.

Server	Location	Data Turbine Name
kilonalu.ancl.hawaii.edu	Kilo Nalu Shore Station Lab	KNShoreStationDataTurbine
bbl.ancl.hawaii.edu	HIG Building, UH Manoa Campus	KNHIGCampusDataTurbine

3.3.2. Source Name Conventions

To readily identify a specific data source, the BBL Project will adopt the following naming convention that consists of the following components:

Site ID							Instrument Depth (m)			Instrument Type				Water Depth (m)			Replicate		
ArrayID		Node		Subnode															
K	N	0	1	0	1	_	0	1	0	A	D	C	P	0	1	0	R	0	0

- Site ID - a unique identifier that is comprised of an Array ID (KN), the array Node where the instrument is located (01, 02, 03, etc...), and the Subnode where the instrument is located (01, 02, 03, etc...). In the case where a Subnode is not appropriate, an XX will be used in the Subnode identifier. The Site ID portion of the identifier will be followed by an underscore to delimit the rest of the deployment identifier.
- Instrument Depth - The depth of the deployed instrument in the water column, expressed in meters of water below Mean Sea Level (MSL).
- Instrument Type - An identifier of up to 4 letters that acts as a code for the type of instrument deployed. In the cases where less than 4 letters are used, an X is used to fill the remaining letters. The codes may be one of:
    1. ADCP (RDI Acoustic Doppler Current Profiler 1200KHz Workhorse)
    2. ADVX (Nortek Acoustic Doppler Vector/Vectrino current meter)
    3. TCHN (Precision Measurement Engineering custom temperature chain)
    4. YSIX (YSI Inc. Water Quality Sonde Model 6920)
    5. LIST (Sequoia Instruments LISST 100 Particle Size Analyzer)
    6. APNA (SubChem Systems, Inc. UH APNA prototype)
- Water Depth - The depth of the water column where the Node and Subnode are located, expressed in meters of water below Mean Sea Level (MSL).
- Replicate - the replicate number of the instrument in cases where more than one instrument of the same type is deployed at the same Site ID, Water Depth, and Instrument Depth (R00, R01, R02, etc...).

The example table above shows a data source that is an ADCP instrument on the Kilo Nalu (KN) array at Node 01, Subnode 01, with single instrument depth of 10 meters below MSL, deployed in 10 meters of water, labeled as Replicate 00.

3.3.3. Channel Name Conventions

It's important to be able to uniquely identify both the data sources (scientific instruments and engineering instruments) and each of the channels of data presented by those data sources. The Data Turbine allows for a hierarchical approach to naming data sources in order to easily query the time series data across multiple data sources. It simply models a typical file hierarchy such as '`/Server/DataSource01/Channel01`' and enables each data stream to be accessed individually or across streams using a wildcard syntax.

Each data source that streams data to the Data Turbine software will do so by populating a channel map with one or more channels of data which represent the raw data as they are streamed from the instrument. Once in the turbine, each of these channels will be decoded into new channels that include each of the measurement variables and the accompanying metadata fields found in the data stream. The table below shows the channel naming conventions that will be used for raw data streams.

Instrument	Channel Name	Description
ADCP	BinaryPD0EnsembleData	Raw, binary ensemble data in PD0 format
ADVX	TBD	TBD
TCHN	ASCIIDelimitedTChainData	Raw ASCII temperature & pressure data in space delimited format with newline endings
YSIX	TBD	TBD
LIST	TBD	TBD
APNA	TBD	TBD

3.3.4. File Naming Conventions

Site ID							Instrument Depth (m)			Instrument Type				Water Depth (m)			Replicate				Begin Timestamp														Doc #			Rev		Ext		
Array ID		Node		Sub Node																																					
K	N	0	1	0	1	_	0	1	0	A	D	C	P	0	1	0	R	0	0	_	2	0	0	7	0	9	0	1	1	3	4	5	2	2	.	1	0	.	1	.	t	x	t

The file-based naming convention will include the source convention, followed by an underscore, and the following four fields:

- Begin Timestamp - The year, month, and day, hours, minute, and second of the first observation found in the data file. This will allow for easy time-based identification of the file at a glance. A period delimiter will follow the Begin Date field.
- Document ID - This is a two-digit numeric identifier that is unique to the deployment. No other file with the same combination of Site ID through Begin Date fields will have the same Document ID, which allows us to unambiguously identify a deployment file and its level of processing. The Document ID numbers will use the following convention to indicate levels of processing, and can continue at increments of 10 (10, 20, 30, 40, 50 ...):
  - 10 - indicates raw data are archived in the file with no processing from the data stream
  - 20 - indicates the first level of processing, such as minimal QA/QC procedures
  - 30 - indicates the second level of processing, such as converting to ASCII encoding

  A period delimiter will follow the Document ID field.

- Revision - A one digit number indicating the which revision the file has undergone. In some cases, a file with the same deployment fields and Document ID will be processed incorrectly, and will need updating. This field indicates how many times it has been revised so that files of previous revisions are not deleted or removed. A period delimiter will follow the Revision field.
- Extension - A three letter (or number) extension indicating the type of the file. Typical file-endings will be txt (ASCII encoded text files), dat (binary data files), csv (comma-separated values text files), etc.

The table above shows an example file name with the previous deployment fields in the Source example above, with a begin date of 01Sep2007, a document ID of 10, which is at the first revision, and is an ASCII-encoded text file.

3.4. TCP-based streams of scientific data

Instruments that are deployed on the array are connected to serial-to-IP converters such as the Digi Portserver TS MEI. To enable communication between the instruments and the Data Turbine software, the serial converters will be configured as TCP servers, allowing client software to establish a TCP connection and read raw TCP packets from the converter. The NEES program has already put in a great deal of effort in establishing a Java-based interface for communicating with data sources, and has made this code available through an open source license. Driver software that is written to communicate with each instrument in the BBL Project will implement this established programming interface, and will provide the following methods:

- connect() - connect to the data source via TCP
- disconnect() - disconnect to the data source via TCP
- start() - start reading the data stream over the established TCP connection
- stop() - stop reading the data stream over the established TCP connection
- isRunning() - report if there is a connected and streaming process
- setArgs() - handle command line arguments passed to the source class on startup

Once the TCP connection is established and started, each source driver will interpret the data stream appropriately, and is responsible for connecting to the Data Turbine, creating a channel map object to be sent to the turbine, populating the channel map with the streamed data, and flushing the channel map to the turbine as a time-stamped frame. The size of the frame is arbitrary, but should coincide with the logical measurement unit of the instrument. For instance, for ADCP data, the ADCPSource class will send a data frame to the turbine for each ensemble streamed from the instrument. Client software that connects to the Data Turbine and reads the incoming per-ensemble ADCP frames will be responsible for decoding the ensemble data and creating new data channels after processing the ensemble, such as the TextOutputPlugin and the MatlabSink.

3.5. TCP-based streams of engineering data

The Kilo Nalu array will also be streaming system-level engineering data from sensors within the node and subnode housings to which the scientific instruments connect. Measurements such as internal temperature, humidity, and power levels will be streamed to the Data Turbine software in the same fashion as the scientific instruments. A Java-based source driver will be written for each of the internal node sensor types, and will stream the engineering data to the Data Turbine, to be monitored in real-time by applications such as the NEES RDV application (Real-time Data Viewer).[5]

[5] see http://it.nees.org/software/rdv

3.6. Online and offline file-based archives

As seen in Figure 1 above, the Kilo Nalu system will provide a continuous file-based archive for data that are streaming into the Data Turbine. Since the Data Turbine uses ring buffers of fixed size, the oldest data in the ring buffer needs to be archived to disk prior to being overwritten by the newest data. The BBL Project will create a Java-based client class called a FileArchiverSink that will continuously read data channels from the Data Turbine and create time series snapshot files of both the original raw data streams, and an ASCII-text version of the data which is processed to provide a column-based view of the data. These text files will adhere to the NEES DAQ[6] format (simple tab-delimited time series data) in order to take advantage of real-time streaming software such as the RDV viewing application.

Once files are archived to the designated directory on the Shore Station server, they will be backed-up to an external hard disk (HDD) on a daily, weekly, and monthly schedule. Likewise, to protect the integrity of the files, an offsite backup of the data directory will also occur via a nightly rsync process. The rsync command will be used to mirror the directory from the Shore Station server to the HIG Campus server in case of physical problems at the the Shore Station (such as fire, etc.).

3.7. Data Turbine plugins for programmatic data access

The Data Turbine software includes functionality to replicate data from one turbine to another. In the Kilo Nalu system, the replication feature will be used to mirror data from the Shore Station server to the HIG Campus server, where researchers will be able to query data from Data Turbine engine. In order to query, read, and view data in real time, software will be used that allows programmatic access to the data streams. The Data Turbine software ships with a set of Matlab functions that provide a uniform interface for fetching data from the system. Existing Matlab software written by BBL researchers will be modified to take advantage of this toolbox, and will read data directly from the Data Turbine in real time, rather than from single data files.

The three modes of acquiring data from the Data Turbine include:

- Subscribe - All new data is to be streamed without gaps
- Monitor - "Current" up-to-date live information is to be sent best-effort.
- Request - A particular time-slice of data is requested

The Data Turbine software provides the programming methods to perform each of these modes of data acquisition, and they are described in the Data Turbine Developer Guide[7]. Although this guide is slightly outdated (Version 2.1), it provides an excellent programming tutorial for fetching data from the system.

The BBL Project will create plugin software that uses the RBNB programming interface to process data. The first will be a FileArchiverSink program, which will archive data streams to disk on a continuous basis. Secondly, a TextOutputPlugin program will be written to convert binary data streams from the instruments into ASCII-based[8] data channels (put back into the turbine). These channels can then be subscribed to via client applications such as RDV, Matlab, or other custom software.

3.8. Data Turbine plugins for scheduled processing

As described above, the Data Turbine API will be used to access data streams in the turbine software. In order to process and display data on the Kilo Nalu web page, existing Matlab software will be modified

---

[6] see DAQ Data Format Final Specification, page 6

[7] see http://outlet.creare.com/rbnb/WP/V2.0/Documentation/Sapi/DeveloperGuideV2.1.html

to periodically connect to the Data Turbine, fetch a time-slice of data, process the data to produce readable graphics, and post the graphics to the web server. This process will allow for periodic snapshots of up-to-date summaries of data streaming from both the scientific instruments and the engineering sensors.

3.9. Data Turbine Web-based administration

The Kilo Nalu array consists of multiple instruments connected to multiple nodes, all accessible via ethernet communication. Communicating with any single instrument is fairly straightforward, but communicating with all of the instruments in a coordinated manner takes more effort. Performing calls such as `start()` and `stop()` will initially be performed using command-line programs. However, we plan to also create a web-based administration page that provides for coordinated communication with all instruments on the array. We will create a servlet-based program to handle requests from a web page so that BBL staff can bring the array up or down cleanly.

3.10. Managing data availability via threshold alarms

Once the Kilo Nalu system is up and running with data streaming through the Data Turbine, we plan to write plugin software that will monitor the data signals. For instance, when conditions such as humidity reach a specific threshold within a node casing, the plugin software will send an email message to appropriate staff to inform them of the condition. Likewise, the Data Turbine software collects system metrics such as disk and memory consumption, and staff will be notified when usage hits particular thresholds.

3.11. Linux-based server systems

Two Linux-based servers will be installed, one in the Hawaii Institute of Geophysics (HIG) building, and one in the Shore Station at Point Panic. The machine specifications are as follows:

Item	HIG Campus server	Shore Station server
Model	Dell PowerEdge 2950	Dell PowerEdge 2650
Processors	2 x 2.33Ghz Dual-Core Intel Xeon 5148LV, 4MB cache	2 x 2.0 Ghz Dual-Core Intel Xeon, 512KB cache
Memory	4GB @ 667Mhz (2 x 2GB)	3.5GB @ 667Mhz (2 x 1GB, 1 x 512MB)
Storage	6 x 73GB SAS disks, 10K rpm	5 x 36GB U160 SCSI disks, 10K rpm
OS	Redhat Enterprise Linux 5	Redhat Enterprise Linux 5

The Shore Station server will act as the primary data collection and archive machine, and will run the Data Turbine software, and will provide SSH access via the wireless VPN uplink for administration. The HIG Campus server will act as the data mirror, will run the Data Turbine software, and will be accessible via the campus network as `bbl.ancl.hawaii.edu` over SSH.

3.12. Wireless VPN link

Connectivity between the Shoreside data systems and the UH Campus systems will be accomplished via a dedicated Virtual Private Network (VPN). As depicted in Figure 1, wireless antennae have will be installed on the UH Biomed building to provide line-of-sight communication with the Shoreside station at Point Panic. The 802.11a and 802.11g standards will be used to provide a 54 Mbps uplink to the station.

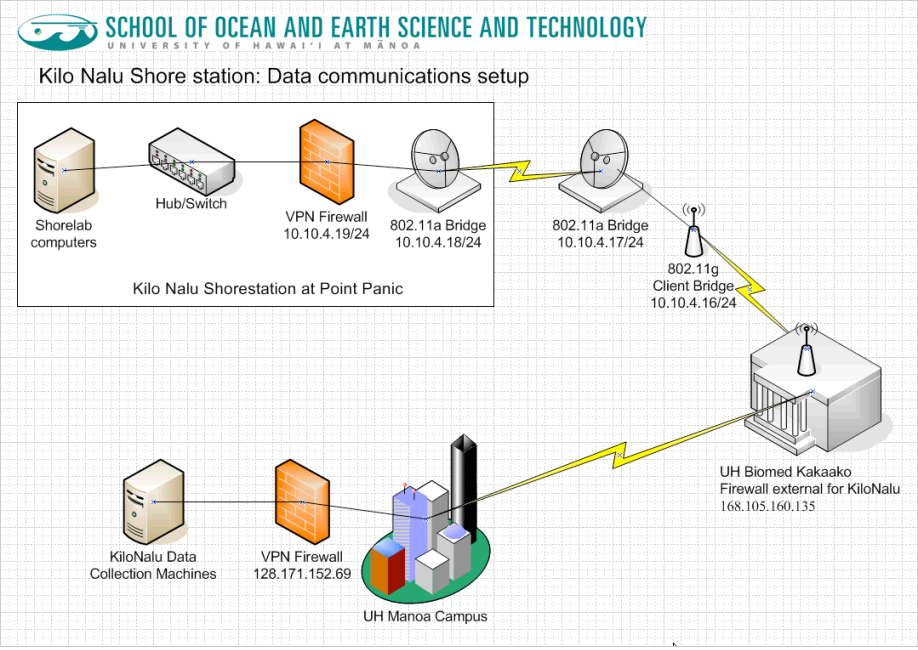


Figure 3. Data communications diagram created by Brian Chee

4. Appendix A - Architecture diagram

Kilo Nalu Real-time System Architecture Overview Diagram