The Ring Buffer Network Bus (RBNB) DataTurbine Streaming Data Middleware for Environmental Observing Systems

Sameer Tilak[†], Paul Hubbard[†], Matt Miller[‡], and Tony Fountain[†]
[†]San Diego Supercomputer Center and [‡] Erigo Technologies
{sameer, hubbard, fountain}@sdsc.edu and matt.miller@erigo.com

***Abstract*— The environmental science and engineering communities are actively engaged in planning and developing the next generation of large-scale sensor-based observing systems. These systems face two significant challenges: heterogeneity of instrumentation and complexity of data stream processing. Environmental observing systems incorporate instruments across the spectrum of complexity, from temperature sensors to Acoustic Doppler Current Profilers, to streaming video cameras. Managing these instruments and their data streams is a serious challenge. Critical infrastructure requirements common to all of these sensor-based observing systems are reliable data transport, the promotion of sensors and sensor streams to first-class objects, a framework for the integration of heterogeneous instruments, and a comprehensive suite of services for data management, routing, synchronization, monitoring, and visualization. In this paper we present the RBNB DataTurbine, an open-source streaming data middleware system, and discuss how the RBNB DataTurbine satisfies the critical cyberinfrastructure requirements core to these sensor-based observing systems. The discussion includes the results from real-world deployments.**

I. Introduction

The vision of large-scale sensor-based observing systems to address the National Research Councils Grand Challenges for environmental science relies on robust cyberinfrastructure [1]. Since the original NRC report in 2001, significant progress has been made in sensor based observing systems. The Arzberger report, CLEANER committee, and the national ecological observatory committee among others have all articulated the science, engineering, and educational issues [2]–[4].

To that end, the environmental science and engineering communities are actively engaged in planning and developing the next generation of large-scale sensor-based observing systems [5], [6]. These systems face many significant challenges including: **heterogeneity of instrumentation** and **complexity of data stream management**. Environmental observing systems incorporate instruments across the spectrum of complexity, from temperature sensors to Acoustic Doppler Current Profilers (ADCP) to streaming video cameras. Managing these instruments and their data streams is a serious challenge. An ideal solution to these challenges is a streaming data middleware providing modularity, flexibility, and control over complex data interactions.

RBNB DataTurbine started as a commercial streaming data product and has a track record of performance in several large-scale projects [7]–[9]. It satisfies a core set of critical infrastructure requirements that are common across a number of observing systems initiatives, including reliable data transport, the promotion of sensors and sensor streams to first-class objects, a framework for the integration of heterogeneous instruments, and a comprehensive suite of services for data management, routing, synchronization, monitoring, and geospatial data visualization. It has been tested in a variety of real-world streaming data applications [5], [7]–[9]. It facilitates the development of complex distributed streaming data applications, including real-time virtual observatories and telepresence collaboratories. Recently, the RBNB DataTurbine has been released open-source under the Apache 2.0 license [10]. In this paper, we describe several key features of open-source RBNB DataTurbine and discuss how these features address the core requirements. We also present the results from real-world deployments.

II. Background

A. Sensor-based Observing Systems Approach to Engineering and Environmental Science

The use of sensor-based observing systems to both gather data and perform experiments in engineering and environmental science is now a commonly used technique in the efforts to answer questions related to issues such as global warming, invasive species, infectious diseases, structural engineering, desertification, salination, pollution, and land use. NEON [5], NEES [7], GLEON [8], CREON [11], ORION [6], and EarthScope [12] are examples of large-scale observatories which are evolving to consist of thousands of sensors, tens of thousands of data streams, and tens of thousands of end users. Rapid advancements in technology and cyberinfrastructure have made it possible to create and use sensor networks with capabilities that were previously unattainable.

This paper is based on our experience in providing cyberinfrastructure to a number of observing systems domains and communities. All of them face similar challenges in deploying and managing a distributed collection of heterogeneous sensors and instruments. Each observatory though has their own unique requirements that mean the individual solutions to these challenges vary greatly from system to system. We briefly describe some representative domains here, and the interested reader is invited to pursue the references for more information.

The National Ecological Observatory Network (NEON) will be the first national ecological measurement and observation system designed both to answer regional- to continental-scale ecological questions and to have the interdisciplinary participation necessary to achieve credible ecological forecasting and prediction [5]. A key requirement for NEON is robust streaming data middleware to handle thousands of data streams from a large variety of sensing devices. While the NEON sites are distributed, the administration is done through a single central authority. The Network for Earthquake Engineering and Simulation (NEES) is a shared national network of fifteen experimental facilities, collaborative tools, and centralized data repository [7]. NEES also has a central repository where post-experiment data is uploaded into a data center. However, individual sites also collaborate on distributed experiments using RBNB DataTurbine to stream data between laboratories. The Global Lake Ecological Observatory Network (GLEON) is a grassroots network of limnologists, information technology experts, and engineers who have a common goal of building a scalable, persistent network of lake ecology observatories [8]. The Coral Reef Environmental Observatory Network (CREON) is a worldwide collaborative association of scientists and engineers working to design marine sensor networks [11]. The GLEON and CREON sites are geographically distributed and administered locally. While no two observatories are identical, there are strong commonalities. For classification purposes, we can divide into two categories based on how control and data are distributed and shared. The first category is federated data model, and the second is the centralized data model.

Federated model: Participating sites are geographically distributed and are administrated locally (under autonomous control), at the site-level (ref. Figure 1(a)). In particular, each site can design its own architecture, database schema and manage its own data using that schema. Thus, the system from sensor to data resource, and the interfaces to the rest of the participating sites (and the world) are the responsibility of the site.

Centralized model: Participating sites are geographically distributed but are administrated by a single administrative authority (ref. Figure 1(b)). All the sites gather their data and typically stream/upload it to a data center typically following a common database schema. It is therefore the responsibility of the data center to make the data available for the entire network of participants.

Regardless of their model, all these projects have similar cyberinfrastructure (CI) requirements with regard to data acquisition, instrument management, and state-of-health monitoring including reliable data capture and transport, persistent monitoring of numerous data channels, automated processing, event detection and analysis, integration across heterogeneous resources and systems, real-time tasking and remote operations and secure access to system resources. To that end, streaming data middleware provides the framework for application development and integration.

We now briefly describe a typical sensor network setup for environmental and ecological monitoring applications (ref. figure 2(b)). Typically, a digital or analog sensor is connected to a datalogger device, which acts as a simple computer to drive the sensor and store the results. The datalogger may be equipped with a wireless radio, allowing it to be periodically polled for data transfer. Some dataloggers are stand-alone, which means that a scientist must periodically visit the site to download data to a laptop or PDA. Davis, Campbell Scientific Inc. [13], National Instruments [14] are examples of datalogger vendors. In fact, these common dataloggers encompass the majority of systems deployed by our collaborating domain scientists (ref. Section II-A).

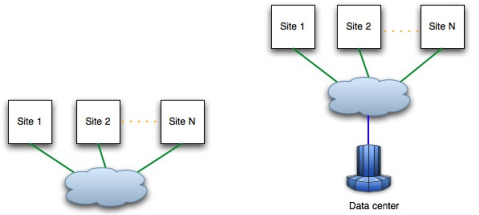
B. RBNB DataTurbine Background

Environmental observing systems want to have access to real-time data for a variety of reasons. Use cases include adaptive sampling rates, failure detection and correction, quality assurance and simple observation. In addition, real-time data access can be used to generate interest and buy-in from various stakeholders, especially the public.

It should be noted that our use of the 'real-time' description is more properly soft real-time, in that guarantees are not made with respect to latency or response. Hard real-time is not judged to be useful in these applications over routed networks.

Real-time streaming data is a natural model for many of the applications in observing systems, in particular event detection and pattern recognition. Many of these applications involve filters over data values, or more generally, functions over sliding temporal windows. By making data streams first-class objects, we have interfaces that support these operations in a natural way.

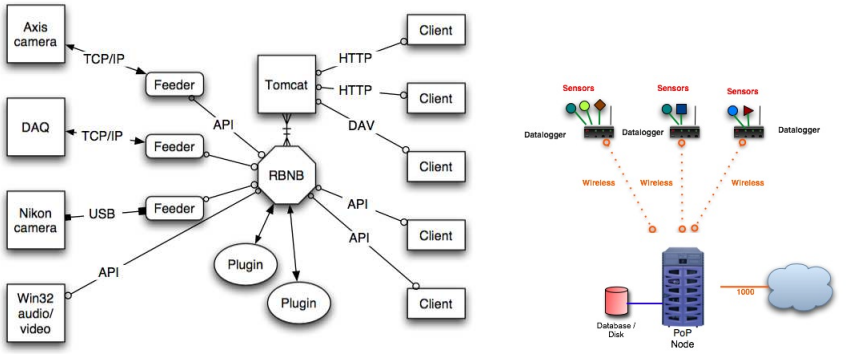
The management of real-time streaming data in large-scale collaborative applications also presents major processing, communication and administrative challenges. These applications must provide scalable and secure support for data acquisition, instrument and data stream management, and analysis and visualization. Most applications address these issues by building custom systems



(a) Observing systems: federated model.

(b) Observing systems: centralized model.

Fig. 1. Observing system models



(a) General architecture at collaborating site.

(b) Typical architecture of a site with field deployed sensor system.

Fig. 2. RBNB deployment and observing system

that are inevitably complex and difficult to support. Extensibility, scalability, and interoperability are often sacrificed under this approach. The goal of RBNB DataTurbine is to address these cyberinfrastructure challenges in a principled manner.

The RBNB DataTurbine middleware provides a modular, scalable, robust environment for meeting these challenges while providing security, configuration management, routing, and data archival services. The RBNB DataTurbine system acts as an intermediary between dissimilar data monitoring and analysis devices and applications.

As shown in Figure 2(a), we use a modular architecture, where a source or ”feeder” program is a Java application that acquires data from an external source (e.g. video camera, data acquisition (DAQ) system, microphone, etc) and feeds it into the RBNB server. Additional modules display and manipulate data fetched from the RBNB server. This allows flexible configuration where RBNB serves as a coupling between relatively simple and ”single purpose” suppliers of data and consumers of data, both of which are presented a logical grouping of physical data sources. RBNB supports the modular addition of new sources and sinks with a clear separation of design, coding, and testing (ref. Figure 2(a)).

From the perspective of distributed systems, the RBNB DataTurbine is a ”black box” from which applications and devices send data and receive data. RBNB DataTurbine handles all data management operations between data sources and sinks, including reliable transport, routing, scheduling, and security. RBNB accomplishes this through the innovative use of memory and file-based ring buffers combined with flexible network objects. Ring buffers are a programmer-configurable mixture of memory and disk, allowing system tuning to meet application-dependent data management requirements. Network bus elements perform data stream multiplexing and routing. These elements combine to support seamless real-time data archiving and distribution over existing local and wide area networks. Ring buffers also connect directly to client applications to provide TiVo-like services including data stream subscription, capture, rewind, and replay. This presents clients with a simple, uniform interface to real-time and historical (playback) data. RBNB’s Java implementation language lends it wide platform flexibility, as is further detailed in section III.

RBNB DataTurbine shares some common features with other existing data management systems; however RBNB DataTurbine is unique in its support for

science and engineering applications. Commercial programs such as MSMQ [15] and Websphere MQ [16], NaradaBrokering [17] and similar standards including Enterprise Service Bus [18], Java Message Service [19], and various publish-subscribe systems [20] provide support for guaranteed messaging, but fail on other science and engineering requirements. These enterprise messaging systems have subtly different requirements that are derived from their focus on business processing models. For example, science applications require persistence of delivered data, robust metadata annotation, and integration of heterogeneous instruments and data types, e.g., numeric, audio, and video data. The only other middleware system that approaches RBNB DataTurbine is the Antelope system from Boulder Real Time Systems (BRTT) [21]. Antelope is a successful streaming data product that supports science and engineering applications. It is also built on the concept of a ring buffer. Compared to RBNB DataTurbine, Antelope presents challenges in procurement and management. As a commercial product, it can be too expensive for many applications and communities, and the licensing restrictions make it difficult to manage. In our experience, RBNB DataTurbine provides all the core services of the BRTT Antelope, plus other attractive features: its open source, written in Java, and its easy to install and manage.

RBNB DataTurbine is highly portable and is available for many platforms, from 64-bit multi-core machines and desktop systems to handheld and embedded devices. As a concrete example, we have tested performance of RBNB DataTurbine on a 8 core Sun Fire T2000 ServerDual-core Linux servers, to Gumstix devices [22]. To give an idea, a typical Gumstix device has an Intel Xscale - 400 MHz processor with 64 MB RAM and 16 MB Flash and runs Linux 2.6 OS. On the other end, Sun Fire T2000 server has 8 cores (UltraSPARC T1 processor) , 16 GB memory, runs Solaris OS and is connected to a 9 TB storage (RAID). We have ported it to cell phones using J2ME.

III. RBNB Internals

The RBNB server is a standalone Java program in an executable jarfile. By default, it listens to port 3333 and can therefore be run by unpriveleged users. After it parses the command line, all interactions are via TCP connections. It is also bundled with an optional Tomcat [23] server with an RBNB extension providing HTTP and DAV access to the server. The server code is compatible with Java 1.1 or later, and runs without change on all known J2SE platforms.

RBNB DataTurbine uses a three-fold classification of programs: Sources, Sinks and Plugins. Sources generate data (writers), Sinks are readers, and Plugins are hybrids that act like Unix pipes, transforming data in the process. A Java program can have one or more of each type, generally however a program will have a single type for simplicity and reliability.

Key to the idea of RBNB is that all data is time-stamped. Timestamps can be explicit (provided by the source), or implicit (automatically provided by the client API or RBNB server). Time-varying metadata is handled as normal channels, for example the GPS location of a mobile system.

RBNB can also utilize disk to extend the span of a ring buffer. Clients specify the duration for which they they wish to retain data, both in disk and memory, and this is translated into a combination of memory and disk buffers. Clients and sources, however, simply use the unified addressing scheme detailed below, and the server manages the specifics.

Naming and addressing: Each data source can have multiple ”channels” of data, where channel data are referenced by name and time. Thus, the RBNB data structures are inherently a triad of data, name, and time. Each RBNB server has one or more data sources. Each data source has one or more frames of data. Data frames have one or more data channels, and each channel contains one or more blocks of data per frame. From an external point of view, an RBNB server resembles streams of data that are accessed by timestamp. Data streams are addressed via a three-part naming system of Server / Source / Channel e.g. Server name / DAQ NW Floor / Strain gage 16. Data is therefore addressed via a triplet of Name, Timestamp, and Duration.

Channel Map All RBNB data is organized in a data structure called ”channel maps”. A channel map consists of a collection of named channels, each with data and timestamp. RBNB clients manipulate channel maps as a means to make requests (sinks) and submit data (sources). A source client builds a channel map consisting of one or more named channels. For each channel it provides data of a specified type and quantity. It also specifies a timestamp for the channel map as a whole, or for the various pieces (channels and data) separately. After being so built, the channel map is sent from the source client to the RBNB server. This process can be repeated, adding new channels or new data to existing channels.

A sink client builds a channel map in order to request data. Here, the channel map consists of named channels and timestamps, which is sent to the RBNB server as a request. The response to this request is another channel map, this time with the data filled in for the various channels.

Sources: Source clients are ”active”, that is they initiate data transmission to the server. Each time a source sends some data to the server, it is called a ”frame”. A source can send a sequence of frames to the server. Each frame can consist of one or more named ”channels”. Each channel can consist of one or more data points per frame. Increasing the number of points per frame increases efficiency by using larger TCP packets, but adds to the total latency.

Sinks: Sink clients are ”active”, that is they initiate data retrieval from the server. Just as for a source, each time a sink gets frames of data from a server. Each frame consists of one or more named channels, with each channel consisting of one or more data points. A sink requests data by both channel name(s) and timestamp. The data returned to a sink can consist of multiple or partial source frames, depending upon the requested time slice. There are three modes by which a sink can get data from a server: Request, Subscribe, Monitor.

Requests are for a particular time interval, for which there is a single response for each such request. It is also possible to make a single request that is automatically repeated over a specified number of time intervals. Subscribe and Monitor modes are open-ended in that new data is automatically sent (from the server to the sink client) as it becomes available. Subscribe mode fetches all data, even if this means falling behind real-time. Monitor mode skips data in order to stay current.

Plugins: Plugin clients are ”passive”, that is they wait for data requests from the server, and send data to the server in response to those requests. PlugIns act like a kind of combined sink/source. The server passes to the Plugin any requests for Plugin channels. Upon receipt of a request, a Plugin acts as a source and sends its response to the server. A Plugin can optionally register the specific channels that it can provide. Registered channels do not have any data in them, they are a means of ”advertising” available channels. With registered channels, only requests for those specific channels will be forwarded by the server to that Plugin. Otherwise, any request (e.g. ”Plugin/anychan”) is forwarded to the Plugin. Thus, a Plugin can provide ”services” that can involve fetching and processing other RBNB data on the demand of third party applications. Plugins can process data from other Plugins, thus cascading sequences of processing steps.

IV. Use-case scenarios

Use-case scenarios are essential tools for understanding requirements and ensuring that proposed developments address critical domain needs. Through our collaborations with domain scientists and engineers across various science communities, we have developed five use-case scenarios to inform the software improvements and the prototype demonstrations. These scenarios capture core activities that are common across all environmental observing systems and rely on efficient cyberinfrastructure middleware services. Section V then explains how the DataTurbine addresses the requirements.

Data Acquisition: The primary objective of an observing system is to capture and annotate field measurements, including data and image streams. This requires the ability to collect data from deployed devices, annotate the data with appropriate metadata, and move the data reliably and efficiently from observation site to data collection center.

Discovery and Real-time Access to Data Streams from Remote Instruments: Many observing system functions require real-time or near-real-time operations on data streams (in addition to data archives). This is especially true for event detection and response, QA/QC, virtual observatories, and distributed experimentation. In these situations the ability to identify, address, and manipulate individual data streams is critical.

Data Visualization and Analysis: Understanding and utilizing real-time observing systems data requires complex visualization and analysis services. This raises requirements on the streaming data middleware to support efficient integration of custom and commercial tools.

Large-scale Distributed Collaborative Experiments: Large-scale observing systems must support collaborative experiments spanning multiple groups and physical laboratories. Remote real-time collaborations require streaming middleware support to ensure synchronized shared experiences with data, data products, and analysis and visualization tools. The challenges here include reliable data transmission in the face of unreliable networks, scalability to handle hundreds of collaborators, observers and outreach classes, the ability to handle enormous amounts of data with minimal latency.

V. Observing System Challenges and RBNB DataTurbine Features

Challenge 1 - Reliable data transfer: The challenge is to provide reliable data delivery in the face of variable network delays and frequent network partitions caused by the extreme physical environments where sensor networks are often deployed. The increasing usage of the often error-prone wireless medium as a networking solution further exacerbates the challenge of achieving reliable data delivery.

A simple solution is to have local nodes, running the RBNB server, continue to place data in a ring buffer that spans both memory and disk. After the network returns, data is available to be automatically streamed. For this reason, maximum reliability is achieved by placing a sensor network node running RBNB DataTurbine as close as possible to the field deployed sensor system. This minimizes the distance over which the interface is susceptible to failures resulting in data loss.

From recent experience where we successfully ran the RBNB server on a GumStix device, we know it is possible for these embedded devices running DataTurbine to either coexist with the sensor’s datalogger or, in some cases, even replace the datalogger. By running instances of DataTurbine in situ with the field-deployed sensors we are able to minimize potential loss of data.

To address the data delivery issues arising from transient network errors, the RBNB DataTurbine uses TCP for all transport, which provides a basic level of reliability that is well-understood. The RBNB server is able to extend the basic level of reliability of TCP through its buffer of received data frames until the circular queue

fills. The size of circular queue is user-defined allowing users an easy mechanism with which to balance the trade off between reliability and resource consumption. For data that is critical, larger queues (in either RAM or disk) that exceed the length of any anticipated network disruption can be used. Upon reconnect, clients simply request data (or commands) available since the last transmission.

It should be noted that the via use of ring buffers RBNB DataTurbine provides a level of reliability somewhere between plain TCP and mission-critical transactional systems. We have found its reliability to be a good compromise for streaming data. It strives for minimal delay and efficient use of resources. Since all data goes into circular queues, *delay tolerance* is inherent in the design. As explained earlier, buffering should be defined to exceed the maximum expected network disruption, up to the resource limits in the server.

Challenge 2 - Integration of heterogeneous instruments: The challenge here is to accommodate a broad spectrum of instruments that observing systems include. It is necessary to accommodate the diversity of instruments because of the external issues such as local requirements, budget constraints and scientific preferences affecting observing systems.

Because it is not possible to interface to every instrument, our design decision was to write drivers for the most common dataloggers and to publish standards and APIs so that interested users can easily add RBNB DataTurbine support for their equipment and experiments. More specifically, writing software to support a large number of individual sensors poses significant challenges in terms of scalability and obsolescence. Therefore, instead of interfacing directly to individual sensors, we decided to write device drivers for common off-the-shelf dataloggers from Campbell, National Instruments, and Davis. These dataloggers encompass the systems deployed by our collaborating domain scientists (ref. Section II-A). This is a scalable approach since writing a driver for a datalogger can help to integrate a vast number of sensors that can be attached to it (potentially thousands of sensors).

We have prototype versions of device drivers for National Instruments [14] and Campbell dataloggers [13]. These drivers were developed during our feasibility tests with RBNB DataTurbine. We are working on improvements to these drivers to ensure that they are production quality. We are also documenting our efforts to provide future guidance for writing driver for dataloggers from other vendors. Through our efforts, we have integrated numerous sensors with RBNB DataTurbine including, Apprise Templine, Vaisala Weather Transmitter WXT510, Vaisala Digital Barometric Pressure Sensor PTB210, Axis video camera and Axis cameras on pan-tilt-zoom (PTZ) platforms, Nikon 5700 Digital camera, Greenspan Dissolved Oxygen Sensor, Labview-based DAQs, and strain gages, string potentiometers, and linear variable displacement transducers (LVDTs).

Our design decision to interface RBNB DataTurbine to dataloggers was motivated by a desire to reach a broad range of sensors quickly and to mitigate the impact of rapidly evolving sensor interfaces. By focusing on datalogger interfaces we are able to support a large variety of sensors and users early in the design and development process, integrating sensors from a variety of vendors with minimal additional software engineering. This approach also facilitates the migration from one sensor product to another when a better product becomes available or an older product becomes obsolete.

Challenge 3 - Performance and Scalability: The challenge is that the next generation of observing systems will be more complex than any built to date. The version of DataTurbine currently deployed in observing systems is the 32-bit version which only supports 3.5GB of memory and a few tens of GB of disk. This translates into just one third of the data generated in a single experiment from 384 channels of strain gage data sampled at 1 Hz through 16 bit A/D [24]. Alternately, this is 8 minutes of video data from a high-resolution network video camera producing 30 frames per second at 250KB per frame. Many currently planned experiments will generate data well in excess of these limitations. As a concrete example, telepresence and photogrammetry experiments [25] are already sophisticated and large-scale users of the RBNB DataTurbine and they have designed experiments that are not possible with a 32-bit version of DataTurbine. Other observing system users have expressed interest in conducting experiments that require integration of high-definition video stream data, acoustic monitoring and use of current profilers, which will generate load that exceeds the 32-bit DataTurbine's capacity.

To address this challenge, we have ported RBNB DataTurbine to 64-bit version of JVM , which allows us to store considerably more data in ring buffers. We have also verified interoperability between 32/64-bit client and server instances. We are in the process of determining how well the 64-bit version will address the increased performance and scale requirements. Towards this end we are using a sensor testbed consisting of a dedicated Sun Fire T-2000 server as discussed previously. Our tests have shown for example that 64-bit RBNB is able to handle 30fps of video data with a peak throughput of 65MB/sec over a gigabit ethernet network.

Based on our direct experience in working with a number of observing systems domains and communities, we realized that researchers want data of an entire experiment available in the ring buffer so that they can rapidly move through both live and historical data in a transparent manner. This allows them to jump around in the experimental record, looking at features of interest, browsing the progress of an experiment, utilizing fast or slow replay etc.

The reader may now be wondering at the use of Data

Turbine instead of an SQL database: RBNB can be thought of as a simple non-relational database, where the keys are name and timestamp. There are several differences between the two approaches and we would like to point out that they complement each other. For example, while databases are geared towards long-term persistence of data, we see RBNB as occupying a unique niche for live data. SQL databases typically lack a publish/subscribe mechanism, so clients have to poll for new data. RBNB can process new data much more rapidly since the lack of ACID-type guarantees reduces transactional overhead. Finally, RBNB is designed with the assumption that buffers may wrap around (circular buffers), replacing the oldest data with the newest thereby providing transient space in contract to long-term persistent storage typically provided by databases.

Challenge 4- Routing and topology management: The challenge here arises from the fact that every observing system requires different network topologies. Additionally these topologies may need to be dynamic.

Based on our deployment experiences we can say that via intelligent selection of routing topologies, a sensor network designer can improve power efficiency, bandwidth utilization, or system response time. To that end, RBNB DataTurbine can be easily configured to support a broad variety of network topologies.

The simplest architecture of a system using RBNB DataTurbine has a single server, with one or more sources and one or more sinks. (See Figure 2(a)) In many scenarios, more elaborate routing topologies are useful. Note that the DataTurbine has host-based access control, which can be used when constructing a system to define who can connect and read or write. Figure 3(a) shows the basic architecture.

Using parent-child, arbitrarily deep routing trees can be constructed, with host-based access control at each vertex/server. This is often used to place a child or children behind a firewall or NAT, with the parent as public. This works because the child initiates the connection to the parent, and will periodically attempt to reconnect if an error occurs.

Other topologies are possible using the mirroring capabilities. RBNB DataTurbine can mirror between servers very easily. Mirroring is useful for load balancing, scalability and robustness. It is particularly useful in isolating client load from server load, so that inbound data is not delayed or lost.

Challenge 5 - Support for in-network processing and time synchronization: In-network processing provides opportunity for data fusion, event detection, and QA/QC across multiple sensor data streams [26]. To that end, RBNB provides capabilities for in-network data buffering and processing. For example, Figure 3(a) shows how data from multiple sources can be processed at multiple levels using tree topology. This provides an opportunity to perform application-specific in-network processing at various stages along the path between a data source and a data sink.

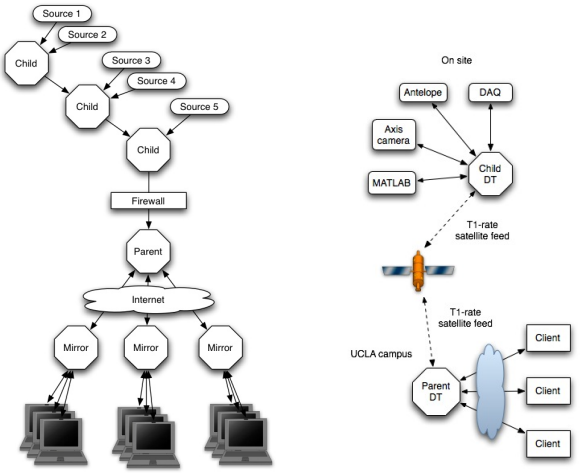
Time synchronization is especially critical in sensor networks for a variety of reasons including sensor data aggregation, power-efficient duty cycling. There has been a considerable research on this topic [27]mostly in the context of resource-constrained micro-sensor nodes.

As explained in Section III, key to the idea of RBNB is that all data is timestamped. Timestamps can be explicit (provided by the source), or implicit (automatically provided by the client API or RBNB server). Therefore, time synchronization is absolutely necessary. However, resource constraints were not an issue in any of our deployments, mainly because most of the use of line power or renewable energy source such as solar power. Therefore, we did not focus development on novel time synchronization protocols. In fact, in our deployments we used standard time synchronization protocols developed for resource-rich distributed systems. More specifically, we used NTP servers and locally-attached GPS clocks, depending on the experiment.

Challenge 6 - Support for Spatial Data and Visualization Services: Data from a sensor network is inherently spatial and temporal. The size and complexity of the planned observatories necessitates visual and map-based services for scientific analysis and system management. Online mapping interfaces provide a convenient user environment for managing sensors, visualizing their characteristics and mapping sensor data vis--vis a variety of spatial data layers.

To that end, a recent addition to the capabilities of the RBNB DataTurbine is Google Earth integration. Via 'plugins', a DataTurbine mechanism for closely coupled data transformation, streamed data is converted into KML format and displayed in Google Earth. For geospatial data such as Earth-observing systems, this provides a visual interface that domain experts find simpler to understand and navigate. It has also proven invaluable for understanding data trends over an area.

Challenge 7 - Coupling sensor data with modeling tools: Answering grand science questions depends critically on developing models for understanding complex processes demands integration of modeling and analysis tools that can use the real-time data streams from the RBNB DataTurbine. To that end, it is possible to meld the rich image and numeric toolkits of Matlab with the real-time data streams from the RBNB DataTurbine. For example, a Matlab program can perform the image processing on video stream that RBNB acquires from networked cameras. With Version 6 and later of Matlab, direct calls to Java are supported from the command line mode of Matlab. Thus, Matlab uses the native Java RBNB API directly. Several simple utility M-files are provided as examples along with the open-source version of the RBNB DataTurbine. We want to point out that integration with Matlab is just an example of how analysis tools can be integrated with RBNB DataTurbine. However, modularity and well documented APIs of RBNB facilitates integration of other tools.



(a) Generalized diagram of RBNB topologies.

(b) Parent-child routing as used for experiment.

Fig. 3. Observing system models

VI. Conclusion

Environmental science and engineering communities are now actively engaged in the early planning and development phases of the next generation of large-scale sensor-based observing systems. In all of these systems, streaming data has a central role. The RBNB DataTurbine, recently open-sourced, presents a compelling solution by addressing a core set of cyberinfrastructure requirements common across several environmental observing systems initiatives. Our deployments to date show that the RBNB DataTurbine middleware provides a modular, scalable, and robust solution.

VII. Acknowledgements

This work was partially supported by NSF grant OCI-0722067.

References

[1] "National research council (nrc) grand challenges in the environmental sciences," 2001.
[2] P. Arzberger, "Sensors for environmental observatories: Report of the nsf sponsored workshop," 2004.
[3] "Committee on the collaborative large-scale engineering analysis network for environmental research (cleaner), cleaner and nsfs environmental observatories," 2006.
[4] "Committee on the national ecological observatory network, neon: Addressing the nation's environmental challenges," 2003.
[5] "The national ecological observatory network," http://www.neoninc.org.
[6] "The ocean research interactive observatory network," http://www.orionprogram.org.
[7] "Nees: The network for earthquake engineering and simulation," http://www.nees.org.
[8] "The global lake ecological observatory network (gleon)," www.gleon.org.
[9] "Namma: The nasa african monsoon multidisciplinary analyses program," http://namma.msfc.nasa.gov/.
[10] "Apache license version 2.0," January 2004, http://www.opensource.org/licenses/apache2.0.php.
[11] "Creon: The coral reef environmental observatory network," http://www.coralreefeon.org/.
[12] "The earthscope project," http://earthscope.org.
[13] "Campbell scientific inc." http://www.campbellsci.com/index.cfm.
[14] "National instruments," www.ni.com/.
[15] "Microsoft message queuing (msmq) technology," http://msdn2.microsoft.com/en-us/library/ms701784.aspx.
[16] "Ibm websphere mq," http://www.devx.com/ibm/Article/31997.
[17] S. Pallickara and G. Fox, "Naradabrokering: A middleware framework and architecture for enabling durable peer-to-peer grids," in *Middleware*, 2003.
[18] D. A. Chappell, *Enterprise Service Bus*, 2004.
[19] "Jsr 914: Java message service (jms) api," http://jcp.org/en/jsr/details?id=914.
[20] Y. Liu and B. Plale, "Survey of publish-subscribe event systems," May 2003.
[21] "Brtt antelope real-time system," http://www.brtt.com/.
[22] "Gumstix," http://gumstix.com/.
[23] "Apache tomcat," http://tomcat.apache.org/.
[24] "A system for multi-axial subassemblage testing (mast): Design concepts and capabilities," 2002.
[25] E. A.S., B. Spencer, D. Kuchma, J. Ghaboussi, Y. Hashash, and G. Quan, "Multi-axial full-scale sub-structured testing and simulation (must-sim) facility at the university of illinois at urbana-champaign," in *Proceedings of the 13th World Conference on Earthquake Engineering*, 2004.
[26] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," 2002. [Online]. Available: citeseer.ist.psu.edu/madden02tag.html
[27] R. K. Jeremy, "Global synchronization in sensornets." [Online]. Available: citeseer.ist.psu.edu/764199.html