Real time data streaming in sensor networks: Integrating SAL with the RBNB Data Turbine

Real Time Data Streaming in Sensor Networks: Integrating SAL with the RBNB Data Turbine

Yong Jin Lee*, Jarrod Trevathan*, Ian Atkinson*, Wayne Read*, Nigel Bajema*, Adam Scarr*, Jochen Braun*, Andreas Knisch*, Andreas Seemann* and Ron Johnstone†
*eResearch Centre – James Cook University, Townsville, Australia
†School of Geography, Planning and Environmental Management – University of Queensland, Brisbane, Australia

***Abstract*—Developing a large sensor-based observation system faces two serious challenges: 1) incompatible sensor technologies from different manufacturers; and 2) complexity of the data streaming process. Sensor Abstraction Layer (SAL) is a middleware integration platform which enables a single interface to view and control heterogeneous sensors regardless of the technologies involved. Although SAL addresses the software compatibility issue of sensors from different manufacturers, it provides limited support for real-time visualisation of the sensed data. Real-time data streaming is extremely useful for scientific modelling and presenting study results for which the sensor network has been designed to investigate. This limitation of SAL can be improved through using existing purpose-built technologies such as the Ring Buffer Network Bus Data Turbine. The Data Turbine is an open-source data management system which provides services for data stream management, routing, monitoring and visualisation. This paper introduces SAL-T (Transmission) which integrates SAL with the Data Turbine. SAL-T is an extra software layer that facilitates the management of the sensed data from SAL into the Data Turbine. Performance tests have been conducted using SAL-T in a simulated data streaming environment indicative of a wireless sensor network. The tests showed that SAL-T dramatically reduced network traffic and improved transmission times.**

I. Introduction

Sensor-based environmental observation systems provide great benefits for scientists by collecting massive amounts of data on key environmental factors such as temperature, light, pressure, humidity, etc. They also have the ability to transfer the sensed data to the end-users via the network. This reduces the amount of time scientists need for collecting data and the automated process ensures a degree of data quality/integrity. It also allows for remote observation of how the environment changes over time [4], [8]. However, developing a large-scale system faces two major challenges: 1) complexity of the data streaming process; and 2) difficulties with integrating heterogeneous instruments/technologies.

A key role in sensor-based environmental observation systems is the ability to have access to real-time data [7], [5]. Real-time data is used for scientific modelling, presenting study results, and failure detection/correction. However, this is a difficult challenge when the system must support hundreds of sensors streaming copious amounts of data, and must also fulfil thousands of user requests. A further problem is that manufacturers typically provide their own proprietary software tools that only support their products. This is an extremely limiting factor which impedes the system's capabilities. The system can become extremely complicated when it is designed to support a large number of different sensor types. Therefore a middleware solution is required to alleviate this problem.

The *Sensor Abstraction Layer* (SAL) goes part of the way in providing a solution [2], [3], [10]. SAL is a *middleware integration platform* which enables a single interface to view and control heterogeneous sensors regardless of the technologies involved. New sensors from different vendors can be easily added by installing a driver analogous to the 'plug and play' paradigm for adding peripheral devices to a personal computer. SAL provides GUI (Graphical User Interface) for controlling the sensors and streaming data. However, SAL provides limited support for real-time data streaming and visualisation of the streamed data. Only a single SAL client is able to stream data from one sensor, other clients are not able to stream data from the sensor while it is in use.

This limitation can be improved with existing methods such as the *Ring Buffer Network Bus Data Turbine* [9]. The Data Turbine (DT) is an open-source middleware data stream management system, which provides modularity, flexibility and control over complex data interactions for near real-time data streaming and visualisation. It provides the ability to manage the server system resources, manage data from multiple sources, and manage multiple user requests over a network. The DT timestamps data and stores it into a ring buffer, which enables Ti-Vo [1]-like features for data retrieval and control. The DT also provides a Real Time Data Viewer (RDV) tool for visualization of stored data.

This paper presents *SAL-Transmission* (T). SAL-T is a transmission middleware which integrates SAL with the DT. This is an extra software layer that facilitates the management of the sensed data from SAL into the DT. The core object of the system is to improve the functionality of real-time data streams in SAL and data visualisation via the RDV. The performance analysis is presented to compare the SAL-T with SAL's current method (Remote Method Invocation (RMI)) for data streaming. Our results shows that SAL-T dramatically reduced network traffic and improved transmission times.

This paper is organised as follows: Section II provides a brief overview of SAL. Section III gives an introduction to the DT. Section IV proposes an architectural framework for SAL-T. Section V analyses the performance of SAL-T. Section VI

[1] www.mytivo.com.au

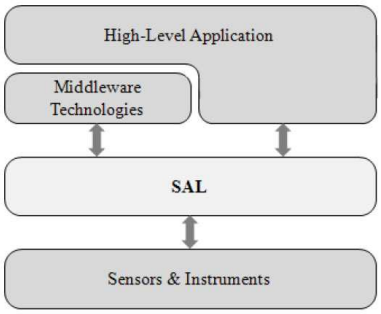

Fig. 1. The SAL software model. SAL sits between the lower middleware and the hardware for the sensors and instruments. It facilitates a transparent interface for communication between the high-level application programs and the sensor hardware.

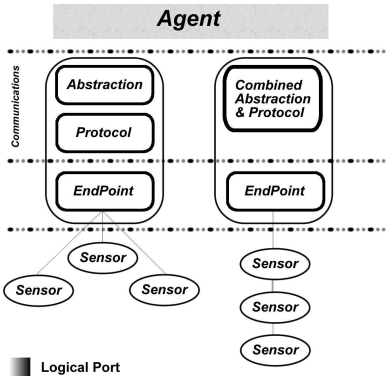

Fig. 2. SAL components and software layers

provides some concluding remarks and avenues for future work.

II. Sensor Abstraction Layer

Sensor networks (SN) are becoming increasingly essential as sensors are introduced into more areas of life. However, lack of standards makes it difficult to integrate heterogeneous sensors in a single SN. Therefore it is important to have middleware which is able to manage all types of sensors. This middleware is often implemented specifically for one SN, which makes it hardware dependent. Changes in the network, such as adding a new sensor, leads to manipulating the middleware code.

SAL is a middleware integration platform which provides a plug-in-based model where support for new types of sensors can be loaded to the running system via plug-in [2], [3], [10]. The system automatically detects and configures new sensors, if permitted by the hardware and operating system (OS). It provides a unified interface to all sensors by abstracting the sensor specific features. This simplifies access to a SN and the management/control of its sensors. SAL can be seen as a low-level software layer as it bridges a network of sensors with high-level applications or further middleware technologies (see Figure 1).

SAL consists of two components, the *SAL client* and the *SAL agent*. The SAL client represents an interface for SAL to either the user via a user interface or to other applications. It implements the SAL agent *Application Programming Interface* (API) in order to provide the SAL agent's functionality. The API is grouped into the following categories:

- **Sensor Management** -?Methods to manage the pool of sensors and including operations for enumerating, adding and removing sensors.
- **Sensor Control** – Methods to report on a sensor's capabilities and control the streaming of the data.
- **Platform Configuration** – Methods to adjust the platform, e.g., add support for a new sensor type.

Each category uses a different markup language. The Sensor Management methods use *SensorML* [1], which describes a sensor's configuration. The methods in the category Sensor Control use *CML*. The CML documents contain a list of commands which are supported by a sensor. The last category, Platform Configuration, uses *Platform Capabilities and Configuration Markup Language* (PML). PML documents contain information on the platform configuration in order to support a certain type of sensor technology.

The SAL agent implements the various features of SAL. It runs on a platform which is connected to the sensors and therefore is regarded as a *sensor gateway*. The connection between the platform and the sensors can be either direct using platform specific input/output (I/O) ports like USB or indirect by using wireless technology. The SAL agent manages the sensors which are directly connected and are found by the agent and the indirectly connected sensors it has been told of. The SAL agent consists of three layers: *Agent Layer*, *Communication Layer*, and *EndPoint Layer* (see Figure 2).

The *Agent Layer* is responsible for the communication with the SAL client. It receives messages, parses and forwards them to the underlying *Communication Layer* and sends the response back to the client.

The *Communication Layer* provides methods for managing and controlling sensors. The managing methods are used to configure and set up hardware while the controlling methods translate a generic command into a sensor native command, which then can be transmitted to the sensor. The generic commands are provided in the SAL API. For translating the generic commands two sub-layers are used. The *Abstraction Layer* is an adapter layer where the generic commands are implemented. From here the sensor-specific methods are called, which are implemented in the *Protocol Layer*.

The *EndPoint Layer* is tightly coupled to the I/O ports available on the sensor gateway. It is responsible for transmitting native sensor commands produced by the Protocol sub-layer to the sensor, and data from the sensor is transmitted to the SAL agent. This layer's software code layer is normally included in the OS. SAL ensures it is available and configured correctly.

The sub-layers *Abstraction* and *Protocol* and the *EndPoint Layer* form a *Logical Port*. Each sensor is connected to a Logical Port which allows a specific client to control multiple agents. The logical port allows SAL to scale vertically in that

an agent can control other agents in a hierarchical manner. The logical port treats an attached device as a data source, independent of whether it is a sensor or another agent. The ability to scale allows SNs of almost any topology or configuration.

III. The RBNB Data Turbine

This section describes the key features of the RBNB Data Turbine in terms of implementation.

The DT's key feature is that data is stored in a time-stamped *ring buffer*. It uses the system disk for data storage and the system memory for input and output access to the buffer. The buffer can store a specific amount of data which is overwritten by new data when it exceeds the storage capacity limit. The oldest data is overwritten first in a circular manner. The stored data can be retrieved/viewed using the RDV, or alternately can be consumed by a self-implemented process. The DT provides the ability to retrieve data by the time reference, which enables near real-time data streams.

The basic components of the system are:

- Channel map;
- Source; and
- Sink.

Each of these are discussed in turn below.

Channel map is a data structure, which contains one or more *channels*. A channel represents the data type and size, and contains one or more data blocks (i.e., a piece of data). A channel can be added by assigning it a unique name. It can then be accessed with its index which is a reference number returned when the channel is added. The index is used identify which channel is to be used for storing data.

The channel map also provides the ability to configure a *time-mode*, which refers to a method for time-stamping data. There are three main methods for time-stamping data:

1) *"NEXT"* stamps data starting from 1 and increases when source transfers data to the server.
2) *"TIMEOFDAY"* stamps data with the client system time when the data is stored in the channel.
3) *"SERVER"* stamps data with the server system time when server receives data.

Source initiates data transmission to server. It transfers the data package (i.e., channel map and all streamed data) to the server. This is referred as a *frame*. Source can send either a frame or a sequence of frames.

Source has the ability to control usage of the server system resources. It configures usage of the system resources by passing three arguments: *mode*, *catch* and *archive size*. They refer to how to use the system disk, and the capacity of RAM and the disk respectively. It has the ability to configure the server resource usage such as the size of disk and memory. It has four different types of mode:

- **NONE** – No server disk is used (used by default).
- **LOAD** – The existing archive that matches the application client name is used.
- **CREATE** – Always create a new archive: it deletes the existing archive which matches the application client name.

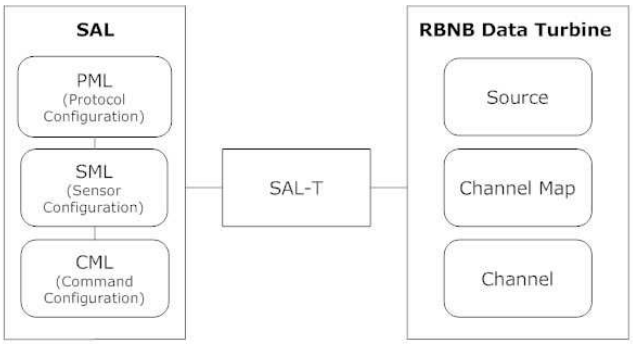

Fig. 4. The data interaction between SAL and the DT

- **APPEND** – Adds data to the existing archive if exist otherwise create a new archive.

Sink initiates data retrieval from the server. It makes a request by a channel map and time, which basically refers to where and which data to retrieve. A channel map has one or more data references where each data reference is the name of the source and channel. Time specifies a period of time in which the data is time-stamped. The server responses can be read by the *fetch* process. This returns a channel map with the requested channels and data points.

Sink has three types for data retrieval:

1) *SUBSCRIBE* is the simplest data fetch mode, which streams all the data from the time request sent to the newest data without gaps. This mode is the most efficient mode to stream data from the server.
2) *MONITOR* is used to get live information. This mode streams the current data up-to-date, which will skip some of the data to keep up if necessary.
3) *REQUEST* is used to get data at specific time. This mode takes three arguments: start time, duration and time reference. It streams data from the start time and runs for the given duration.

The RTV is a free open-source software tool for visualizing scientific and engineering data (see Figure 3). It provides support for a number of ways to visualize the numeric, video, image and audio data either from the local or remote DT server. This also can be extended to support other data types or other ways for visualisation.

IV. SAL-T

The section presents SAL-T which is a middleware transmission layer for integrating SAL with the DT.

SAL-T fulfils following tasks:

- **Data Interaction** – The ability to manage the data interaction between SAL and the DT.
- **Data Management** – The ability to manage sensed data from multiple sources.
- **Transmission** – The ability to perform transmission to the server.

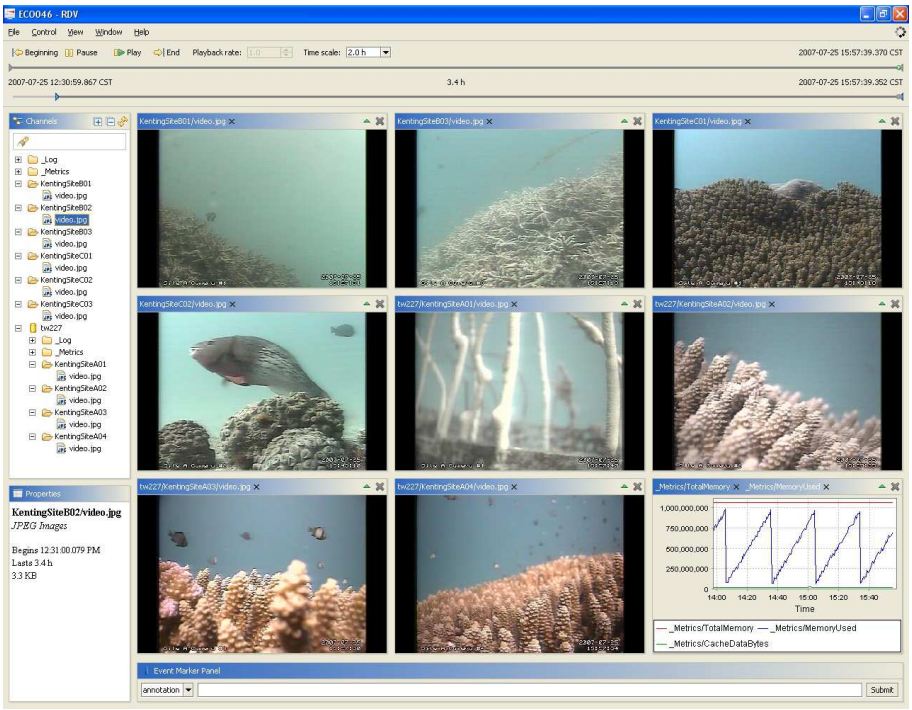

Fig. 3. The Real-Time Data Viewer showing sample data streamed from the DT

Figure 4 shows the data structure of SAL and the DT. SAL uses *PML, SML* and *CML* as a data structure for sensor description. These describe the protocol, sensor and command configuration respectively. The DT manages data by Source Name, Channel Name and Data Type.

SAL-T manages the data interaction between these two systems. The sensed data for a particular sensor is managed in the DT as:

- **Agent ID as Source Name** – An unique ID of the Agent to describe SAL.
- **PML Type and SML ID as Channel Name** – A protocol type and an unique ID of a particular sensor to identify the sensor.
- **CML Respond Data Type as Channel Data Type** – A data type of sensed data of a particular sensor.

It stores the sensed data in the server by the name of the SAL and the sensor.

One of the issues SAL has is that it provides a limited support for insuring the data quality and preventing data lose. SAL provides a direct transmission for the sensed data, which might provide the data quality by reducing the time consuming for transferring data to the end-user. However, this case is based on when there is a reliable network connection to support. The sensed data can be lost when the transmission process is either delayed or failed, which is most likely the case, especially in a wireless network.

SAL-T has two core channel maps for preventing data loss and to ensure the data quality. The system takes an argument, *Interval Time* for transmission process. It configures the channel maps with the mode, “TIMEOFDAY” and stores

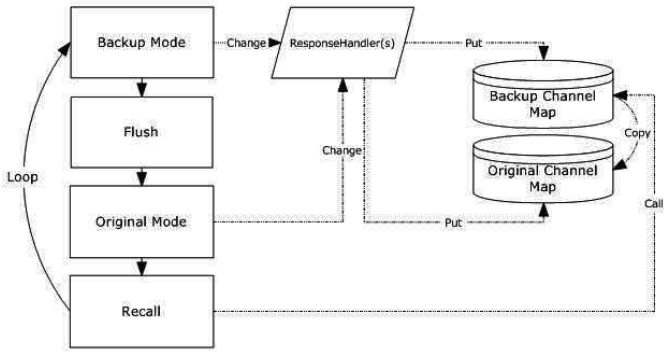

Fig. 5. The system flow in a data transmission process

the sensed data directly into the channel map. This method has an issue with the interaction between transmission and storing sensed data. A channel map is not accessible during the transmission process, in which the sensed data can be either completely lost or loses the data quality as the data is time-stamped when it is stored in the channel map.

ResponseHandler is an object in SAL, which collects the sensed data of a particular sensor. SAL-T has an extend object called *SALTResponseHandler*, which stores the sensed data into a channel map when it receives the data. ResponseHandlerFactory handles the mode for the sensed data. The mode refers to the configuration for which channel map is used as a data storage for sensed data.

Figure 5 shows the system flow in the data transmission

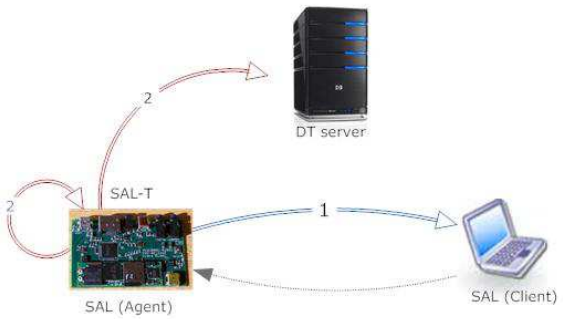

Fig. 6. The test environment – a simulated sensor-based observation system

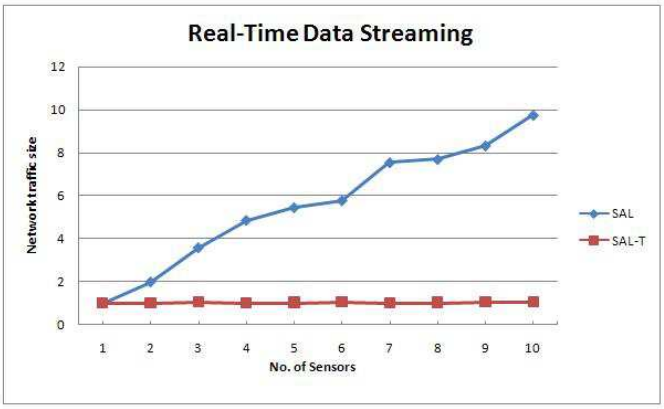

Fig. 7. The effect of a number of data streaming process on network traffic size (lower is better)

process. SAL-T initially stores the sensed data into the Original Channel Map (O-Map). When the transmission process occurs, it first changes the mode to the Backup Channel Map (B-Map), *Backup Mode*. Then, it transfers the data stored in the O-Map to the server, and flushes the memory (*flush*).

The B-Map is a temporary storage for the transmission process. After the transmission is completed, the system changes the mode back to the O-Map, (Original Mode). Once the process completes, the system consumes the data stored in the B-Map and stores it into the O-Map. It provides a solution for the interaction issue while maintaining the data quality.

Furthermore, this ensures against data loss from transmission failures. The transmission can fail due to the lose of network connection, a system crash in the server, network delay, etc. The data will be re-transferred with the new data in the next transmission.

SAL-T improves the limitations of SAL by integrating it with the DT. It provides a support for the data interaction between the two systems by manipulating the information provided by SAL into the DT format. Furthermore, it ensures the data quality with the features provided in the channel map, which time-stamps the sensed data when they are stored. This is also used as the temporary data storage to prevent data loss from the transmission failures.

V. Performance Analysis

This section presents a performance analysis of SAL-T. It compares the existing connection method used by SAL (RMI) with the modified version of SAL using the DT (SAL-T).

A. Test Setup

The test environment is setup with a *Gumstix* (computer on module), a PC, and a laptop. Figure 6 illustrates the test environment. The SAL-Agent and SAL-T are installed on the Gumstix (600 MHz ARM processor, 256 MB RAM and 256 MB Flash). The laptop (3.2 GHz RTM processor and 2 GB RAM) and the PC (4.66 GHz RTM processor and 2 GB RAM) are setup with the SAL-Client and the DT server respectively. The communication between the systems is established through 802.11g wireless network. The arrows refer to the flow of the data transmission process in SAL and SAL-T. In SAL, the agent directly transfers the sensed data to the client ((1) in Figure 6) whilst SAL-T collects the sensed data from the agent and transfers it to the server ((2) in Figure 6).

The test is designed to observe the differences between the systems in terms of performing the real-time data streaming process. In order to simulate a large environmental observation system, a test function is used to set-up a number of sensors and the data size for each sensor to read. The agent generates a series of simulated sensors and sensed data.

B. The Influences of Real-Time Data Streaming on Network Traffic

This test observed the effects of real-time data streaming processes on the network traffic size. That is, how many messages have to be sent between the streaming process and the SAL Client/DT-Server.

Ten simulated sensors are generated which produce 1 byte of data each. The test measured the traffic size with respect to the number of real-time streaming processes (i.e., number of sensors). The interval time for streaming data is set to 10 seconds. The traffic size is measured every 1 min for total of 5 mins and the average value is computed.

Figure 7 presents the test results. We observed the variance of the network traffic size when the number of real-time data streaming processes increases. The Y-axis describes the traffic size with respect to the initial point, which is the traffic size of one process. The X-axis describes the number of data streaming processes. The results show that the network traffic size in SAL dramatically increased with each additional data streaming process. In comparison, SAL-T remained nearly constant for each additional sensor.

This is due to the transmission process in SAL. Each sensor in SAL undertakes the data transmission process, in which it transfers the sensed data when the data is collected. This indicates that streaming n sensors generates m transmission processes ($n = m$). SAL-T effectively manages multiple real-time data streaming processes through the use of the channel map. Sensed data is stored in each respective channel via the

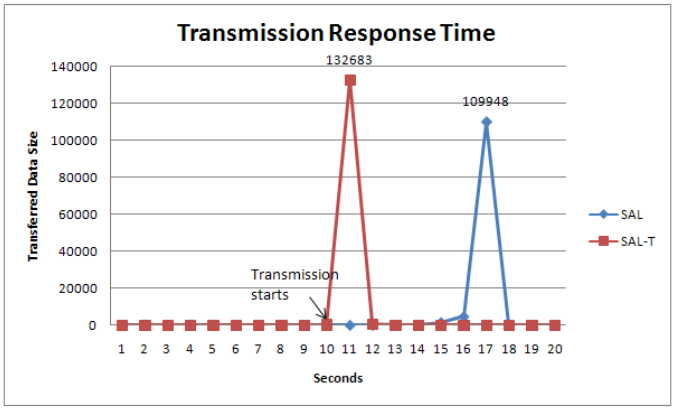

Fig. 8. The elapsed time for the data transmission process

channel map, and is transferred to the ring buffer as a batch process at predefined intervals.

C. Transmission Reliability

This test observed the reliability of the data transmission. That is, how each respective method deals with transmission speed, data loss, and timeliness.

A simulated sensor was generated which produces 100,000 bytes of data. The test measured the total transmission time from the point at which the data was collected and transmitted, to when it was received by the SAL Client/DT-Server. The data is requested at 10 second intervals after setting up the test environment, and analysed the transmission time in seconds.

Figure 8 illustrates the results from the test. The X-axis and Y-axis gives the time (in seconds) and the transferred data size respectively. SAL took 7 seconds (17 secs) whereas SAL-T only took 1 second (11 secs) for the transmission process. This result shows that SAL-T provides faster transmission and more reliable transmission. If a message is delayed in SAL and new data is generated, then the previous value is destroyed. SAL-T on the other hand uses time-stamping and the ring buffer to ensure delayed messages are not lost or overwritten by newer messages.

VI. Conclusions

While SAL effectively handles the complexity of heterogeneous sensors in large-scale, sensor-based environmental observation systems, it only provides limited support for real-time data streaming and data visualisation. The data streaming process is inefficient and is restricted to only supporting a single user. It also does not provide mechanisms for insuring data quality and preventing data loss.

This paper presented SAL-T which is designed to address these limitations. SAL-T is a middleware transmission which integrates SAL with the DT. It provides temporary data storage (i.e., a channel map) for preventing data loss from transmission failures, where it keeps the data for the next transmission process. Furthermore, the DT time-stamps data when it is stored in the channel map. This ensures data quality in the event of a network delay or server crash etc.

SAL-T reduces the network traffic size by managing mulitple data streaming processes simultaneously. Multiple transactions in SAL proportionally increases the amount of network traffic. Whereas SAL-T merges these into one transaction process. Our test shows that this reduces the network traffic size significantly and it is also shown that the DT transmission protocol provides more reliability than RMI (SAL). This reduces the network delay for the data transmission and ensures delayed messages aren't lost.

Future work involves storing the DT on the Gumstix to better manage system resources, and to improve the robustness of the system against crashes. Furthermore, we intend on exploring the possibility of modifying SAL-T to give priority in terms of system resources to particular sensors.

Acknowledgment

This work was supported in part by the Queensland Government National and International Research Alliances Program.

References

[1] Aloisio, G., Conte, D., Elefante, C., Marra, G. P., Mastrantonio, G., and Quarta, G., "Globus Monitoring and Discovery Service and SensorML for Grid Sensor Networks," In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 201–206, 2006.

[2] Gigan, G., Atkinson, I., "Sensor Abstraction Layer: a unique software interface to effectively manage sensor networks," In *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 479–484, 2007.

[3] Gigan, G. and Atkinson, I., "Towards a uniform software interface to heterogeneous hardware," In *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 429–434, 2008.

[4] Hubbard, P., Fountain, T. and Tilak, S., "Using RBNB Data Turbine for Observational Cyberinfrastructure", *American Geophysical Union*, Fall Meeting 2007.

[5] Prowell, I. and Elgamal, A., "Internet Accessibility of Data and Video in Real Time Demonstrated for a Highway Bridge Testbed", *Journal Computer in Civil Engrg.* Volume 24, Issue 1, pp. 11-24, 2010.

[6] Rajasekar, A. , M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S. Chen, R. Olschanowsky, "Storage Resource Broker - Managing Distributed Data in a Grid," *Computer Society of India Journal*, Special Issue on SAN, Vol. 33, No. 4, pp. 42-54, 2003.

[7] Rajasekar, A., Lu, S., Moore, R, Vernon, F., Orcutt, J. and Lindquist, K., "Accessing sensor data using meta data: a virtual object ring buffer framework," In *Proceedings of the 2nd international workshop on Data management for sensor networks*, pp. 35-42, 2005.

[8] Strandellz, E., Tilaky, S., Chouz, H., Wangz, Y., Lin z, F., Arzbergery, P, Fountain, T., Fan, T., Janx, R. and Shaox, K., "Data Management at Kentings Underwater Ecological Observatory", In *3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, pp 715-720, 2007.

[9] Tilak, S., Hubbard, P., Miller, M. and Fountain T., "The Ring Buffer Network Bus (RBNB) Data Turbine Streaming Data Middleware for Environmental Observing Systems", In *Proceedings of the third IEEE International Conference on eScience and Grid Computing*, 2007.

[10] Trevathan, J., Atkinson, I. M., Read, W., Johnstone, R., Bajema, N. and McGeachin. J., "Establishing Low Cost Aquatic Monitoring Networks for Developing Countries", In *Proceedings of the International Conference on Wireless Communications and Information Technology in Developing Countries*, pp. 37–48, 2010.