Real-Time Remote Access Laboratory with Distributed and Modular Design

Ananda Maiti, *Student Member, IEEE*, Alexander A. Kist, *Senior Member, IEEE* and Andrew D. Maxwell, *Member, IEEE*

***Abstract*— Remote Access Laboratories (RAL) are online environments for operating instruments and collecting measurement data over the Internet. Such systems are often deployed by Universities to support undergraduate student and are generally follow the client-server paradigm. This paper discusses a RAL system that enables peer-to-peer (P2P) experiment design and sharing. For this, a modular design is required that allows participating nodes to create rigs and host those individually at distributed locations. The proposed architecture is generic and can be used with any distributed P2P network control systems over the internet. In this paper, a distributed remote control framework is presented with regards to a P2P RAL system. The experiments in the RAL require three subsystems handling the user interface, instruction interpretation and instruction execution which can be organized and operated in different manners depending upon the experiment. The key component for creating and controlling experiments are microcontrollers that can be easily obtained, configured and setup for use over the internet. The most popular microcontrollers are examined for suitability to the distributed control architecture. The basic layout of a message-based network protocol suitable for programming the devices and communication between peers for remote instrumentation and control is discussed and queuing and flow control mechanisms are compared and tested for the proposed framework.**

***Index Terms*— Remote laboratories, networking, message-oriented middleware, microcontrollers, web instrumentation**

I. Introduction

REMOTE Access Laboratories (RAL) allow access and operation of instruments and devices over the network for education [1-3]. As such, they are similar to network controlled systems (NCS) [4] with a high volume of interaction between instruments and users (students). Users change parameters of experimental setups and then collect corresponding data. Different technologies [5] and frameworks such as LabVIEW, Java [6], Asynchronous JavaScript and XML (AJAX) [7], LXI [8] etc. have been used to connect users with the RAL systems. All of these follow a client-server centric model [2]. Some of the most widely used systems are iLabs [9], Weblab-Deusto [7], VISIR [44] and Labshare [10] all of which provide remote access to laboratory experiments for engineering education. The aim of these systems is to accommodate higher number of students with limited resources and remove the time and space related constraints. Such systems are widely used for teaching at universities including automation [11], electronics [9] and control systems [12].

The client-server architecture and different technologies to support RAL have been investigated in detail [7, 1]. Figure 1 depicts the operations of a client-server model of RAL where the hardware and Remote Laboratory Management System (RLMS) are hosted by the universities also responsible for authentication and scheduling of users access. Experiment hardware and related software take input from user and run the experiment residing on the server side. The users log in through the internet and give instruction to run these experiments. The nature of such architectures features the notion of a service provider that adds additional experiments at server side. This architecture provides little operational autonomy in regards to the physical location and design of rigs. This limits the pedagogies that can be employed in the remote laboratory space: students can learn the outcomes, e.g. use experiments to collect results; but not how to produce them, e.g. design experiments [2, 13].

A RAL system may be expanded by increasing the flexibility in design of new experiments by enabling user to create experiment setups and allowing users to run experiments created by others and evaluate them. However, in order to implement a P2P RAL system, certain characteristics have to be met:

- The system must be able to establish point-to-point connections between users and providers.
- A suitable hardware platform must be used to create rigs that are robust, network capable as well as easy to use. Once designed, the rigs have to be programmed to communicate with the system and accept commands and send results which then have to be mapped to a particular user interface.

These core technical aspects of P2P RAL provide the context of this paper. It presents a modular peer-to-peer

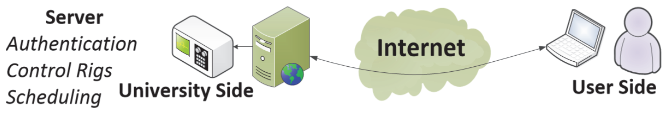

Fig 1. The traditional RAL structure.

Manuscript received April 15, 2014; revised August 19, 2014; accepted October 9, 2014.

This research is supported by the Digital Futures CRN Program of the Australian Government.

A. Maiti, A. A. Kist and A. D. Maxwell are with the School of Mechanical and Electrical Engineering, Faculty of Health, Engineering and Sciences, University of Southern Queensland, Toowoomba, Australia. (e-mail: anandamaiti@live.com; kist@ieee.org; andrew.maxwell@usq.edu.au)

architecture for distributed RAL instrumentation and control where any user or any experiment can be joined or removed at the users' discretion. A peer-to-peer (P2P) system also enables a high level of collaborations between users. This design of the RAL system is centered on the use of micro-controllers as the key motion control and decision making component in an experiment setup. Different configurations of the experiment setups to achieve this modular distributed architecture are presented and feasibility of existing electronics devices to realize this framework is also discussed. The proposed architecture can be adapted for any distributed network controlled and monitoring system e.g. home automation.

A P2P RAL system will require a significant change from the regular approach of client-server RAL architectures: experiments are geographically distributed and students are able to build their own experiments. Traditional RALs often offer a static experiment environment with a fixed set of experiments. The technologies available for collaboration between students in many RAL systems are typically limited. There are few examples of RAL experiments where the user plays a major role in deciding the operation of the RAL rigs. For e.g., in [12] using remote laboratories that shares equipment for research applications is described. It allows user defined programs for controller in an automatic control laboratory. This however, still does not allow the students to create the experiment setup. In previous studies, several students were involved in a P2P approach of RAL [13, 30, 43] which demonstrated the feasibility of the users being able to setup certain experiments and connect them to the network.

Current RAL architectures and related solutions are discussed in Section II. The proposed distributed approach is introduced in Section III and different communication related analyses of microcontrollers to realize such a system are discussed in Section IV and V. The components of the modular design are discussed in Section VI and an implementation and its performance results are reported in the Section VII.

II. Related Work

This section discusses existing architectures for distributed instrumentation and control.

A. Distributed RAL Systems

It has been long realized that due to resources being scattered throughout a geographic area, a multitier distributed architecture has to be used to connect resources to allow remote laboratory services [14-18]. Initial attempts were to create an efficient brokerage between several physical labs across a wide geographic region. These systems give the users a variety of experiments across multiple laboratories and universities manage their local resources optimally [14].

Sharing Laboratories has been suggested in [17] and a smooth interface between the physical laboratories is said to be crucial to determine the extent of sharing. Labshare and LiLa are collaborative projects of consortiums of Laboratories that work in this direction. A flexible architecture that connects and deploys hardware from different physical laboratories into an experiment has been proposed in [16]. The main obstacle identified were the service oriented architecture (e.g. SOAP) which is difficult to manage across heterogeneous networks and socket based communication is suggested as an alternative. With the advent of HTML5, new capabilities of JavaScript and WebSockets the problem of inter-hardware communication has been eased [45]. These technologies can work in bidirectional full duplex mode and in real-time.

In most cases distributed technology and resulting benefits are aimed at the service model i.e. the universities, RAL developers and administrations. The overall architecture of the system remains the same client/server where the user can only view and perform a set of instructions and then acquire results. Laboratory as a Service (LaaS) has been proposed that views laboratories as independent component modules [19]. These are based on Web Services which are slow [31] and more importantly, these rigs are not flexible enough and no universal approach is provided for students to build them. These are an organized approach for sharing existing remote laboratories among '*institutions*'. From a user's point of view, the system architecture remains in the service oriented model.

Web Instrumentation is the practice of controlling the actions of an instrument through a network environment. This methodology is popular in RAL systems [20]. Web instruments use a set of web services associated with the components of the instrument to operate them by calling the respective web service. This method is slow as it initiates HTTP like connections procedure every time a web service is called and also too complex, involving acute understanding of object-oriented programming, creation of objects and attaching and mapping of methods. This makes it unsuitable to be implemented by individuals, particularly students and school teachers.

The concept of devising a common hardware platform that is able to integrate multiple experimental rigs potentially increasing collaboration between '*institutions*' and lower design costs have been explored in [40]. This approach uses FPGAs based on the IEEE1451.0 standard to attain a modular architecture for RAL. With respect to the current context, drawbacks of this approach include the complexity and the use of a separate micro-computer to intermediate between the user and the FPGA. The proposed approach implements the control unit of the experimental rig as a 'ready-to-go' component that can be directly plugged to the internet.—Personalized environments can improve the learning experience of the users [41-42]. In [41] the monolithic user interfaces such as the Java Applets are replaced by a set of even smaller applications - the Web Widgets. This method allows the users to rearrange the UI as they want. However, this approach still does not allow the users to handle the actual rigs or configure the instruments which are required in the context of this research project.

More recently, desktop sharing technologies have been used to share laboratory experiments between users of different laboratories. A Relay Gateway Server (RGS) Architecture has been proposed in [21], where it is used for connectivity between students, instructors, and experiments. The architecture consists of a publicly accessible RGS which acts as an intermediary and pass information between the users and the laboratory setups. In this system, the users conveniently access remote labs in web-browsers using Java and Flash platforms. Whereas the systems discusses in this section allow experiment access via a common portal and to be shared between institutions, these service oriented approaches are not flexible enough to allow for individual experiments sites without extensive infrastructure requirements.

B. Remote Instrumentation and Grids

Grid computing is the collection of resources at several locations that work towards a common goal. Unlike distributed systems these are loosely coupled i.e. they share no knowledge about other separate resources in the grid. They mainly address the requirements related to computational power and data storage for computer based applications. Recently instruments have been incorporated as a resource in such grids to enable grid instrumentation. A grid based RAL architecture has been proposed by [15]. It incorporated a three tier setup - an internal serial *remote lab bus* connecting Web-based control units and all other physical components, a *bus protection unit* to authorize access to control units and a *protection unit* to check the validity of the commands executed to protect the instruments.

Grid based network resource allocation optimized for quality of services parameters for remote instrumentation has been implemented in [22]. The GRIDCC [22] project has used simple, straightforward procedures for adoption of the grid technologies to run instruments remotely. Instruments are represented in the architecture as an abstract format called Instrument Elements (IE). IE details are stored in a centralized information system. It uses the web services methods to communicate between the sites. The instrument element design has also been used in [23-24] to describe a standards-compliant model for the representation of instruments in a grid and for booking of instruments in advance (time-booking) or immediately (queue). However, grids are complex to build and maintain. Grids are also more static in structure and topology throughout their operational period. Moreover the proposed distributed RAL system needs to operate between independent and dynamic users directly. This is difficult to realize with a grid system.

C. Remote Control for Reconfigurable rigs

A smart devices based approach to empower clients side has been presented in [45] where the aim is to make the remote 'smart device' ubiquitous and autonomous. It outlines of the requirements and characteristics of using such devices in RAL. The minimal requirements of this 'smart device paradigm' [45] are also incorporated here, namely *State Measurement* in form of *READ* instructions and *State Control* in form of *WRITE* Instructions described later. However, the described approach in [45] does not decentralize the remote laboratory completely. Also it is not stated as how the users will set up their experiments; connect to the system or the organization of smart devices or their impact over such communication over the internet as studied in this paper.

Programmable Logic Controllers (PLC) have been integrated with the SCADA (Supervisory Control and Data Acquisition) system, usually used for automation of manufacturing to create a highly reconfigurable RAL architecture [25-26, 13]. SCADA is usually designed for monitoring and control of industrial equipment and hence *not suitable for peer-to-peer remote control*. It requires expensive components and complex setup mechanisms that are unfit for experimental setups for the target users. However, the basic concept of SCADA for decentralized control system such as data acquisition, communication and presentation are applicable here as well. A multi-tiered RAL architecture consisting of remote users using web browsers, a central web server and regional experiment servers with control units is discussed in [18]. But these do not support creating rigs at the user end.

Radio Frequency (RF) based components and communication techniques for monitoring and control system using Micro-Controller Units (MCUs) has been proposed in [27]. This system focuses on ensuring a low traffic between nodes to increase efficiency. This system is however an automation system, built on components based on close proximity using RF which is different from the peer-to-peer remote control through the Internet. Another example of reconfigurable rigs is presented in [28] where household robots fitted with microcontrollers and sensors are adapted to be used for RAL. A WEB Micro-server has been developed by RExLab [29] targeted for Mobile learning. Its functionalities can be expanded to monitor and control other devices. This however requires other devices to be controlled and lacks the support for being a controlled experiment rig by itself.

D. Industrial Control Protocols and Equipment

There are existing standardized instrument control protocols like LAN extension for Instrumentation (LXI) and Common Industrial Protocols that contains Control Area Network Bus (CAN), Highway Addressable Remote Transducer Protocol (HART), Ethernet/IP. However these are not usable for a distributed remote laboratory with individual users as:

- Either these technologies are not based on the TCP/IP protocol (such as CAN and HART) which is needed to connect through the internet or the MCUs are not compliant with them (such as LXI and Ethernet/IP).
- They are platform and hardware specific and require specialized compliant hardware for operation. Hence they are mostly used by industries and limited in educational uses. For example, Agilent devices are compliant with GPIB and LXI are widely used in RAL, but it is costly to interface it with the internet.
- They are constructed as client-server application and not optimized for internet based peer-to-peer operations. The topologies supported by these protocols are not ideal for P2P communications through the web and thus not suitable for the modular architecture of a distributed RAL.

These are not suitable for operating ‘ad-hoc’ rigs with both motion control and decision making elements, created by individuals with MCUs over the internet. As the distributed

RAL operates on the internet, many of the features that are reliable in in localized implementation, such as the periodic clock synchronizations, are not effective. This paper focuses only on tools and methodologies for experiment design in a distributed RAL architecture with its different possible configurations and investigate the usability of the suitable devices (MCUs) using a message based protocol for communication to implement these peer-to-peer arrangements. MCUs are proposed to be used as the fundamental building block in experimental rigs as well as core control components of the real-time system where the remote instruments must respond to all users' input within a given deadline.

III. The Distributed Approach

The distributed laboratory approach aims to expand the one-to-many approach where one central laboratory serves many users to a many-to-many approach with many users using multiple equipment set up by different providers. In a distributed RAL, experiments are to be created and hosted by individuals [41-42]. Users are all scattered in the network and anyone can connect to anyone. In this model of RAL, the experiment module is no longer a part of the RLMS as in a client-server model. This results in two types of modules –

- The experiment modules - This contains the hardware and the software related to it,
- The user modules - This remains the same as before i.e. just using the interface of the experiment, and

The RLMS in P2P setup is a set of tools to assist with searching, authentication and relaying messages that may be implemented as a distributed system such as the overlay networks [32]. It has very limited functionalities.

Designing an experiment will include assembling an equipment setup, program and run experiments locally and share the experiment with others by putting it on the internet [13]. A distributed architecture has the following characteristics:

1. *Modularity:* A modular design consists of individual modules or entities, such that each of them can operate independently as well as work together towards a larger goal. It allows users to combine separate experiments to create the workbench without the need of integrating it to a larger structure. In a modular design, new and improved experiments setups that are built subsequently may replace older ones. It is not necessary or expected that any of the experiments will be hosted for a long period of time. This will enhance collaboration as several makers can work on individual experiments at the same time, and then combining them together for a bigger project.
2. *Scalability:* The experiments repository may extend without bounds with users adding their creations to the system. New experiments could be directly added and made usable to others students. This gives the creators full liberty on design and operational paradigms. Any experiment can be added or removed from the system without having to change the rest of the system.

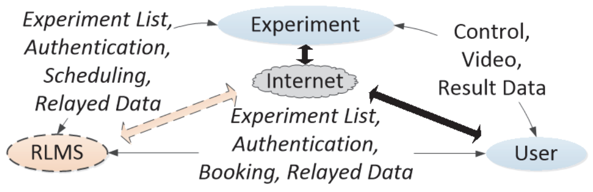
Experiment List, Authentication, Scheduling, Relayed Data
Experiment
Control, Video, Result Data
Internet
Experiment List, Authentication, Booking, Relayed Data
RLMS
User

Fig 2. The modular nature of the distributed RAL

To ensure these, the components used to create an experiment should be portable i.e. should be operable on different hardware platforms and operating systems. Figure 2 shows the structure of the modular design and the data that flows between them. The experiment nodes and user node are the two end points in the architecture mediated by the RLMS. The User node goes through time scheduling (e.g. time slotted or queuing) with the RLMS. The experiment node then authorizes the access at the appropriate time allowing the user node to start issuing the instruction commands. These commands and the corresponding result data are exchanged through an underlying overlay network [32] as a part of the RLMS on the internet. To run an experiment remotely, three components are required -

i. *User Interface* (UI) - This is what the learners see on their computers, at the users' side. This interface is used for taking users' input commands and displaying experiment outputs.
ii. *Instruction Execution Layer* (IEL) - This is for running the read/write operations at the experiment site. The IEL is a static algorithm to process and execute instructions and will have a unique but similar implementation on all platforms.
iii. *Instruction Interpretation Layer* (IIL) – It is a complex and dynamic programing environment required to create and host the program logic of the operation of the rig. The programming environment including the compiler is same for all MCUs but the program logic is dependent on the experiments. The IIL may be attached with the UI on the user side or the IEL on the MCU.

IV. Distributed Scenarios

The distributed system can be implemented in different ways depending upon the nature of the experiments.

A. Scenario 1

In the simplest form an instrument is controlled through the web by sending repeated signals. The IIL and the UI are at the users' site. Each signal is a message containing an instruction. This instruction is then executed and a reply is sent. The advantage of this approach is that all MCUs need only an IEL making it possible to support more types of MCUs by the system. The programmer needs to create a simple interface and upload it. The problem with this is that, each instruction necessary for operation must be sent over the network. If a message is lost, it will create additional overhead for processing and causes time delays. Many experiments can have a section of instructions that are static set of code for the rig which is repeated over time. Users may be required to input a few parameters occasionally. In such cases, sending individual instruction is not efficient.

B. Scenario 2

Since the previous method is not good for experiment that are less user interactive compared to the amount of movement and change at the experiment end, the architecture may be inverted by running the IIL at the experiment side and sending only necessary signals to the experiment. This way the number of messages is reduced. This in turn reduces the message drop rate and overall delay. For this, two interfaces are required to be designed - UI for the user inputs and the machine interface with the IIL and the IEL to process instructions.

One drawback of this approach is that it requires the provider to make a considerably more complex program at the experiment end by keeping track of what inputs are required and maintain synchronicity with the UI. As the IIL is now at the experiment end, it becomes difficult to support multiple platforms of MCUs. In case of more interactive experiments the performance may not improve compared to the increased complexity of double layered parsing of instructions on a slower environment. Also, the program in question may become too large to fit in the memory of an MCU.

C. Scenario 3

A third scenario involves installing a cloud or relay service that stores a programming logic. The relay service has to accept the users' inputs and translate them to instructions and send them to the appropriate target experiment where it can be executed immediately. This method may be used if:

i. The operation of the rig requires large storage memory e.g. if it needs to store images etc.

ii. If an activity is composite i.e. two or more experiment setups are located at different places and needs to operate in a co-operative manner. Different locations do not have to be geographically apart but more than one MCU being part of the same rig due to hardware orientations. Also it is suitable with only 'slow' experiments where events occur in sequence but the deadlines for such events are flexible.

iii. The program logic for operation requires higher computational power than what is available at the user node (for e.g. using a mobile device) or the experiment node (which has an MCU) such that it would be quicker for an intermediate node to process information and issue commands on behalf of the user.

This scenario requires the same type of and complexity in programming the rig from the provider's perspective, but it can provide additional features and any hardware related deficiency could be overcome. The relay programming interface can be same for all of MCUs. The server can be built as a website and acts as a proxy node between the client and provider ends. The message routing technologies has to be efficient to make sure that overhead latency in relaying is minimal. From functional perspectives this is similar to the Scenario 1, where the instruments are run by successive continuous stream of instructions.

All three can be implemented and operated as two or multi-modules experiment setup with the provider building the UI and IIL. As the Scenario 1 requires only one program to be built by the provider for the user (as integrated UI and IIL), it is most easy to implement followed by Scenario 2 and 3. All of these scenarios can be implemented in with high scalability of the P2P system, although the Scenario 1 is most simple to implement and thus most scalable followed by Scenario 2 - 3.

V. THE RAL CONTROL UNIT - MICROCONTROLLER UNITS

Micro-Controller Units such as Arduino [33], Raspberry Pi (RP), BeagleBone Black (BBB), Lego Mindstorms EV3 etc. are suitable to control the experiments remotely. These MCUs have control ports that can be used to set and reset properties of a rig component like motors and servos. They can also collect various kinds of data from the surroundings through sensors. They also have network capabilities to connect to the internet with TCP/IP based protocols. They are small, compact, cheap, readily available and the ideal CU for P2P RAL activities as well. Microcontroller units can act as PCs and provide equal functionalities, but have low computational power and networking capabilities. Table 1 shows a comparison of MCUs with regards to their requirements as a Control Unit of a distributed RAL system [34].

A. Achievable Throughput

Some experiments may require higher bandwidths for proper operation along with transmission of videos which consumes high bandwidth. All of the MCUs are equipped with Ethernet connections but the maximum bandwidth supported by each of them varies depending upon the computational capacity. To establish real performance parameters an experiment was performed that involved transferring files over the network and the real throughput was measured. Several files of different sizes from 200KB-95MB were transferred from a PC running Linux to the MCUs. The transport had to be adapted for the MCUs. For the BBB, Raspberry Pi and EV3

TABLE I. COMPARING MCUS

Properties	***Arduino(UNO, Due, Mega)***	***Raspberry Pi***	***BeagleBone Black***	**EV3 Mindstorms**
Native Programming	Yes	No	Yes	Yes
Adaptive Programming	No	Yes	Yes	No
Pins	Analog/Digital	Digital Only	Analog/Digital	Custom (I2C)
Network speed	Good	Good	Good	Good (Wi-Fi Only)
Processing capability	Arm 7 (16- 90 MHz – Fair)	ARM 11 (700 MHz - Good)	ARM Cortex-A8 (1 GHz - Very Good)	ARM 9 (300 MHz -OK)
Visual capacity	No	Yes	Yes	No
Control Capacity	Medium - High	Medium	High	Medium (Custom parts)
Community Support	Very Good	Good	Fair	Very Good

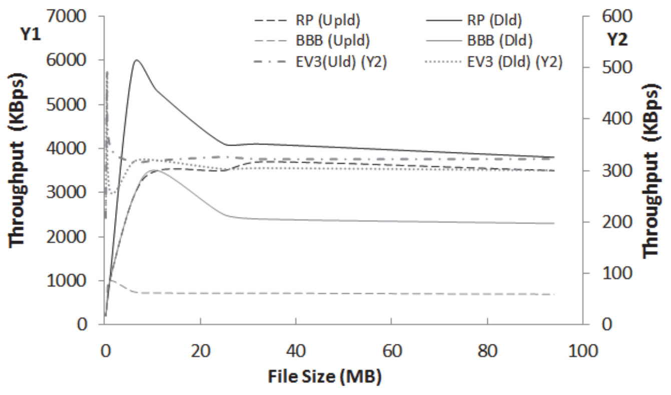

Fig 3. The throughput capacities of the MCUs

(with a custom Java firmware) the SCP command was used from PC to transfer files. For an absolute bandwidth test for BBB and RP, the IPERF tool was used and it reported the maximum of 90-95 Mbps. The results of the test are shown in Fig 3. It is observed that for sending small amount of information (≤ 1MB), time taken is very low as the throughput decreases with larger files and transmission time. Both the RP and BBB can achieve download speeds of more than 2.5 MBps in a LAN. The outgoing speeds of RP are on average 3.5 MBps and for BBB, 700KBps. The EV3 registered average speed of 300 KBps for outgoing and 320 KBps for download. These values are sufficient for any RAL activities. For Arduino UNO, a web server was used to upload and download files. The speeds were below 10 KBps. This may vary a little with different Arduino boards and implementations, but due to limited computational capacity the speed will remain significantly smaller than the others.

To send video feedback, webcams are used that have in-built encoding mechanisms such as support for high resolution with hardware based H.264 encoding. This encoded stream is directly fed to the MCU which then transmits it over the network. The BBB transmits video at 900 Kbps with the 320x240 resolution. Other devices like IP cameras can be used along with MCUs that do not support video streaming.

B. FPGA vs MCU

Field programmable Gate Arrays (FPGA) are capable of delivering the same flexibility required for implementing the modularity and scalability required in the P2P RAL [40]. FPGAs can be used to control other sensors and actuators connected via ports and can be network enabled. For the context discussed in this paper, they are unnecessary complex and expensive than MCUs.

FPGAs are different from the MCUs in the sense that MCUs have fixed processor architecture while using an FPGA can allow the user to create a variety of processing (controller) devices. However, the aim of an experiment design in RAL (centralized or distributed) is not to design a controller for the rigs but using a controller for operating a rig built with actuators and sensors. The functional requirements of the controller component of the rigs in experiment design are fixed i.e. IEL. As such MCUs provide enough flexibility for operating any kind of rigs.

To write a program for a MCU requires knowledge of common programming languages like C, C++ or JavaScript. Programming a FPGA requires to first create a VHDL programming which is more difficult. As the aim is to allow '*individuals*' rather than '*organizations*' to create experiments, a low cost alternative is required.

C. Operational and Communication Standard

A particular communication and control standard ensures consistent functioning, integrity and compatibility of the devices used for RAL. But unlike FPGAs, the MCUs proposed to be used as part of this architecture does not support any uniform standard for communication and/or control such as IEEE1451.x [40] or LXI [8]. Moreover, these protocols do not allow implementing a flexible programming logic required for creating and running variable rig designs. Thus, the protocol presented here is an abstract model, implemented in this paper to investigate the networking and control characteristics of the distributed architecture. This lightweight protocol covers a basic set of commands that are used to control the rig and may be extended as a standard for the P2P RAL.

VI. Components of the Modular Design

The distributed systems consists of three modules and each of these modules features a number of components - the experiment module has a programming interface, messaging protocol flow control and queuing methods, the RLMS module has a authentication and scheduling in a relay server and the user module has just the user interface.

A. The Messaging Protocol

To exchange information about control, a set of messages must be defined that are issued by and interpreted at the nodes. Such messages are unidirectional i.e. an experiment end-node can issue an ACK or EVENT message but not INSTRUCTION message while an users node can only send INSTRUCTION messages. Messages must contain final destination information for routing, as an overlay network will be used for establishing the communication. The messaging protocol in this study implements the most basic requirements to control a RAL experiment. The main aim in designing the messages was to keep them small.

Network protocols for controlling robots can operate in either *object oriented manner* by associating specific functions with the devices or *event-oriented manner*. For example, the software architecture of SNRP (Simple Network Robot Protocol) as described in [35] is an *object oriented* approach. In case of RAL, a *functional and event-oriented programming* is used where the users' action generates messages that represent the event, i.e. a read operation to get the status of the rig or a write operation to change of state of the rig. Messages are executed on the experiment node.

Two message types based on the most basic operations on instruments - *READ and WRITE* have been discussed in [34]. The client sends a series of instructions and acknowledgement messages to communicate between the MCU and the client. These messages may be improved by using a variable number

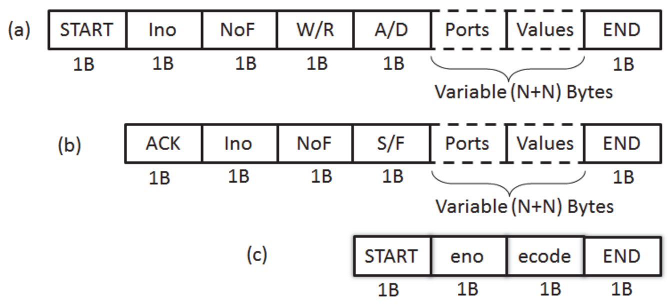

Fig 4. (a) Instruction message from User to Provider (b) Acknowledgement message from Provider to User (c) Error message from Provider to User

of fields by specifying multiple pin numbers in one message. This may be done by introducing a NoF (Number of Fields) Block (to specify number of values) and a variable length message (see Fig 4). All of these messages can be built with a few bytes (7-8 bytes for a single message representing a single event) making the information sent very small. For Scenario 2, where only auxiliary variables are used to change rig behaviors and not port is directly accessed form the UI, the instruction messages can send the values of these auxiliary variables instead of ports. The protocol may be extended to include more complex elements to support the different types of ports available on MCUs such as I2C and CAN ports. These can be used directly to control the external peripherals. The "Start" field can include information about the final destination of the message which can be used for routing it through the overlay network on the internet.

For further control a new type of message – the EVENT message is used that originates from the MCU. The Event message may send any information regarding an event that has occurred at the providers' side. The user side may not have asked for this information, e.g. when an instruction is not received, the battery power getting very low or a port suddenly stops operating due to structural failure. The EVENT message starts with a static START block followed by event number (for the client to keep track of events), *ecode* stating a predefined numerical value for the event and finally the END block.

B. *Flow Control of the messages*

The instructions sent to control the rig must be executed in the exact order and time interval, hence each message is numbered in a session between users and the experiment to maintain synchronization. However, it is observed that due to the low computational capacities, the MCUs can lose messages i.e. not process every incoming message and skip to the next one, if the messages are sent more frequently. Even if the messages are delivered at the network level, they are dropped by the MCU.

Messages originate and are sent in a particular order. An experiment session can have a set of commands $C = \{c_1, c_2 ...\}$ dispatched at intervals $J = \{\delta_1, \delta_2 ...\}$ where $\delta_l < \delta_i < \delta_u$ such that the total time of the entire session $T = \Sigma\delta_i$. δ_l and δ_u are the lower and the upper bounds of the intervals. These values are dependent on the nature of the experiment design. For any rig, the commands c_i changes the rig position from one state to another where the change is always deterministic. This is done in unit operations according to c_i, for example, a servo will always be issued commands with moving up to a certain degree which guarantees that the behavior of the servo is entirely depended on the input given to it.

If the MCU detects that a message is missing, then an EVENT message is sent back requesting the missing message. But if the client has to wait for each acknowledgment before sending the next instruction, then it could cause significant delays. Assuming that, in general, the underlying network will be reliable, i.e. most messages will be delivered correctly, all messages may be sent directly without processing the acknowledgement in the IIL but rather an intermediate service. This service keeps track of all sent and acknowledged messages. If some request is not received at the MCU then the MCU sends an EVENT message to the IIL, which is paused and the service resends the messages again. This method in general follows the 'Go-back N' protocol. The service also keeps record of the time at which each message was dispatched and when resending them maintaining the exact time intervals.

The flow control is more useful in the Scenario 1 where the number of messages is high as the actual logic or origin of instructions is on the users' nodes resulting in higher message loss compared to Scenario 2. Scenario 3 is a hybrid form of Scenario 1 and performs in a similar manner but for multiple sites. In the Scenario 3, the number of messages could be dependent on the number of different sites connected together, if multiple sites are used. The number of messages will be linearly proportional to the number of such sites.

C. *Queuing the messages*

Since the MCUs tend to induce delay in the processing of request, it is desirable to send as many commands in one instruction message as possible. However, the message from the UI may occur at random time thus simply waiting for a specific number of commands is not feasible. Thus a queuing methodology is required to optimize the waiting time and the number of the commands to be sent in an instruction.

A simple time-stamp method is used in this study. Every action in the UI generates one or more new message(s) to be sent over the network. Due to temporal locality, it is expected that during state change on the rig, a number of independent instructions will be executed simultaneously to create the action. These instructions originate with a very small time gap between them. As a new message is created it is associated with a timestamp immediately. Depending upon the nature of the experiment, a message may be delayed only by a certain amount of time (*t*). If any message is created within this time frame, it is joined with the earlier message to create a new combined message. This message retains the timestamp of the first component message. The messages are combined by putting the new port and value combinations into the earlier message and increasing the NoF field. The combined message is then dispatched as soon as the delay reaches the value *t*, a small value (<10ms) that does not alter the time intervals of the inputs. This way a lot of messages can be accumulated together and the actual number of transmission can be reduced maintaining the order of instructions. The size of each message varies with the frequency in which the messages are generated. A message may also be dispatched if its size becomes equal to a maximum size allowed. However, since the individual message sizes is of 8 bytes, combining them will still not create a large sized message as the time gap t is small and the number of instructions generated during the interval will be low for all practical purposes. This should help in situations where multiple events occur simultaneously and the instructions for each event can be combined together.

With queuing, none of the interval is increased by more than t and δ_i has no impact on δ_{i+1}. The message creation times and departure times are independent of the next messages. Hence the entire session time T does not increase by more than the value of t which is negligible. Considering that there is γ % intervals in J where $\delta_i \leq t$, the entire command set may be reduced by γ % at most in the best case if all such message appear after $\delta_j > t$. In the worst case, if there is a single sequence of all $\delta_i = t$, then the number of message can be reduced by $\gamma/2$ per cent.

A similar approach has been used in previous work to reduce the transmission load by withholding information from the nodes if the previously transmitted data are within a tolerable range [36-37]. These approaches however considered closed-loop control systems, but the proposed distributed RAL is not closed loop as it does not operate directly on any feedback from the rigs. The rig may gather data from its sensors which is sent to the UI and can influence the next decisions, but a human user actually gives the inputs to the rig. An alternate way to deal with lossy or reduced network traffic will be to use a predictive system that can estimate the missing instructions according the expected behavior of the rig. The MBPNCS (Model Based Predictive Networked Control Systems) is a NCS mechanism that can overcome adverse conditions of data loss and delay in a network by predicting the future control behaviors based on a model of the end user control system [38]. This is however difficult to implement in the current context as the rigs are designed without following any pattern by different users who will require additional tools for creating a model.

D. Programming and User Interface

To run an activity, the experiment setup has to be programmed to operate and respond to users inputs. The experiment side should also have the user interface which does not carry any operational code, but an array of controls linked to different aspects of the experiment. The programming interface has to be easy enough for novice users to be able to program them yet have the functional features of high level languages such as programming constructs, multi-threading and GUI-based system interface design. Certain characteristics that are highly required includes a drag-and-drop visual programming style like LabVIEW, combined UI design with basic programming constructs and a library of codes that makes only the simple commands to be enough to create a program. The user interface is created based on the functionality of the rig. It has to map each instruction commands to their corresponding component on the interface.

Operational safety of instruments is a necessary part of RAL [15]. The majority of MCUs are very sensitive to external voltage and the amount of current drawn from its port (with the notable exception of Lego Systems which has inbuilt mechanisms to control power). Any instruction before being executed requires to be validated on the MCU to check that it will not cause any physical harm to the existing pin setup. Safety of the whole instrument setup, however, has to be taken care from the program (IIL) of the experiment.

E. Relay and Remote Laboratory Management System Server

All users will be interacting through the internet. Hence, the network between the user and the experiment may be heterogeneous with several layers of firewall, Network Address Translators (NAT) and proxies. To overcome these limitations in connectivity, relay(s) or server(s) may be used to relay the messages between the user interface and the experiment program. The relay server could be placed as

- *Part of the RLMS* - In a small network with low latency (such as within a city), a central relay server (possibly the RLMS server itself) can be used as a relay node. Messages are just passed without any modification. However, in the *Scenario 3* described in previous section, a dedicated relay server is compulsory for issuing instructions on behalf of the user interface.
- *Part of a broader network of user* - The proposed RAL system is of P2P architecture. Thus multiple users will be accessing the system and some of these can act as relay servers depending upon their networking and processing capabilities. This architecture may be similar to the Skype, JXTA protocol or overlay network systems [39] covering a large region with varying latency.

Regardless of the relay mechanism a central module as the RLMS is required. This includes the management of MCUs, like assigning unique global identification numbers to the MCUs and associating them with the users' accounts. This should also maintain a scheduling scheme for deciding which client can access the experiment and when and authentication components for verifying that all messages being transferred in a session originates from a valid source and transmitted to a valid destination.

VII. An Implementation – Results and Analysis

The evaluation focuses on the technical feasibility and constraints of the modular approach for a P2P RAL. This section discusses the setup of experiments, their operation, network characteristics and the requirements for the users.

A. Test-bed Configurations

The network architecture for peer-to-peer RAL can be implemented in a number of ways. The simplest way is to setup a TCP socket between MCU and user. But, whilst TCP socket can offer the optimal performance, due to different NAT structures in the internet it requires additional network mechanisms like UDP hole punching or STUN methods, both of which are not guaranteed to work in every environment. Another solution is to use a VPN that will allow connecting all peers to other peers directly. But this method is not scalable due to the lack of homogenous support for VPN on MCUs.

Hence, the WebSockets are used for communication in the experiments in this paper as they can traverse NATs and firewalls and can be implemented in all MCUs. They can provide almost the same performances as that of the binary sockets. Using WebSockets also guarantees platform independence. WebSockets run on most computing devices including portable devices and can run from inside Web Browsers. Figure 5 shows the basic network architecture of the system with the users (client) side and providers' side being connected by a cloud based relay server which can handle scheduling and authentication.

For the tested configurations, the host or provider's side consists of a MCU and the actual rig with sensors and actuator. When plugged in to the network with Ethernet, the MCU immediately registers itself with the RLMS, which is a centralized server machine (for these tests) also acting as the relay server and opens a WebSocket link with it. The user side consists of a PC that can run the user interface which upon initiation also connects to the RLMS in a similar manner. In reality the relay server may be replaced by an overlay network on the internet. The network also consists of a server machine with Wanem 3.0 running on it. The Wanem is a Linux based network emulator to create a simulated network environment with different round trip time between the devices and also other network properties such as bandwidth. The MCU IEL (or IIL) to execute incoming instructions was written in their native languages for Arduino UNO (C++) and BeagleBone Black (boneScript based on NodeJS).

The communication mechanisms discussed earlier are all managed by the UI and the IIL. The users should not be burdened with additional programming regarding send/receiving and encoding/decoding messages. For this, a high level graphical language such as Scratch is the best option. The Scratch program was used for the providers programming [34] to create the program logic and UI. The Scratch program communicates with a Java Intermediate Service/Daemon (JIS) which is the real interface to the network from the users' side and maintains the WebSocket link to the RLMS. Messages are generated by the UI which is then passed to the JIS which sends the message to the RLMS. The RLMS or the relay server network sends these messages to the MCU and the acknowledgments follow the opposite direction.

The Scratch program follows some guidelines to communicate with the JIS: (1) First it sends two messages to specify the sensors and actuators ports, (2) Sets initial values for all actuator ports, (3) Sends a message to Java Program for values of Sensor Ports, (4) Sets a PAUSE variable that is globally used to check if the state of the program is paused. Whenever a value is requested, the program is *paused*, until the value is received (5) Sets actuator values and asks for sensor values on ports when required.

B. *Evaluation and Results*

A traffic light activity was chosen for testing. The aim was to create a rig consisting of 4 LEDs (Red, Green, Yellow and Blue for pedestrians). This activity consists of an infinite loop that turns on the red, green and yellow at definite intervals and checks the users inputs. The user can press a button for pedestrian crossing on the UI that will turn on the blue LED when the red light is on next time. First this activity was run with Scenario 1 by sending all instructions of operating the LEDs to the MCU from Scratch on client computer and then Scenario 2 by creating the program with a iterative loop for Red, Green and Yellow on the MCU and the UI sending only the 'button pressed' instruction.

To do this four port variables are required for each LED and an auxiliary variable for storing the request status. For operating the LEDs, 2 messages are required per led in one iteration i.e. compulsory 6 messages per iteration along with 2 messages optionally for the Blue LED.

Using Scenario 2 however, only one message is transferred and that too in the event a user initiates the 'button pressed' procedure, thus reducing the traffic to 12% compared to Scenario 1. However, this method requires additional processing on receiving the auxiliary variable values and includes extra programming statements to update it on the IIL. It also requires for the user to create two programs, one for the UI and the other for the IIL. Moreover, this method only works if the inputs and outputs of an activity are not synchronous. Also, in terms of distributed computing, attaining synchronous states in multiple nodes will need additional overhead messages. Thus this method, although

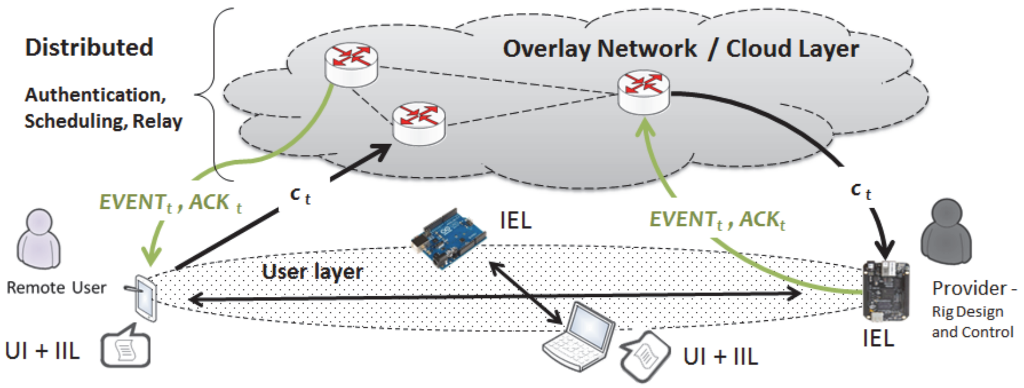

Fig. 5. The distributed network architecture consisting of the user sites and the experiment sites.

efficient, is applicable only to compatible activities. Experiments may be designed with either of the two architectures according to the nature of the experiment.

C. Latency Measurement with WebSockets

To test the latency induced in the communication process, the JIS was used to measure the time between the dispatching a message and its acknowledgment. Figure 6(a) shows the percentage of increase in round trip time (RTT) and the message drop rate for the random sequence of messages (with 95% confidence). A total of 8999 random integer values were generated in the JIS to represent the delay between 9000 instruction messages (δ_i). The value for δ_i was kept between (δ_l, δ_u) - 90-100ms, 190-200ms, 290-300ms, 390-400ms and 490-500ms with randomly introducing values between less than 10ms. The final sequence contained 38.9% instructions generated within 10ms of the last one. This represents a typical scenario of a rig with multiple or quick control. An average RTT is recorded with and without the queuing mechanism. The architecture follows the Scenario 1 with the BBB. The delay is caused by the propagation delay and the processing delay at the MCU. The time taken for actual read write operation on Arduino and BeagleBone is in the order of 1-10 µs which is negligible.

The BeagleBone is the most suitable platform for the purpose of the P2P RAL architecture based on the high processing, networking and control capabilities and thus primarily used in this study. The BBB was able to reply to all messages in both cases. With queuing the total number of messages was reduced to 67.32% of the initial 9000 messages. The average RTT are closely same for $\delta_u < 400$ ms beyond which there is a sharp fall of 74.07 ms for $\delta_u = 500$ms in the average RTT. This is due to the fact that with queuing, there is less message or traffic in the network. Indirectly, this also reduces the probability of losing and resending message on the internet. The overall session was not delay beyond 10ms.

To test the effects of flow control, a random sequence of 999 integer values representing the time difference δ_i was generated and a total of 1000 messages were sent with each message was sent after waiting for each δ_i with (δ_l, δ_u) = 50-200ms, 100-250ms, 150-300ms and 200-350ms. The RTT between the user node and the Arduino is set at 300ms and the window size was set to 5. Figure 6(b) shows the result of this test. Without any flow control, 30% of the messages were lost or missed by the Arduino.

The flow control algorithm with '*go back n'* was able to send all messages through to the Arduino UNO. However the resultant time taken to send a sequence of 1000 messages was 318 % of the total sum of all intervals (*T*) i.e. the actual time that should have been taken between the first and the last message. With the flow control 77.5% of the messages were missed at least once by the Arduino UNO i.e. they had to be resent at least once. The average net delay of a message i.e., the time between it is sent the first time and it's acknowledgment received, is 590 ms with standard deviation of 255.1 ms and the average gross delay i.e. time between it is sent the last time sent and acknowledgment received is 196.0 ms with standard deviation of 79.93 ms. Changing the delay bounds (δ_l, δ_u) from between 50-200ms to 200-350ms reduces the average net delay to 450.0 ms, the average gross delay to 162.7 ms and the overall increase in total time consumed *(T)* to 125%, without affecting percentage of messages that were missed at least once.

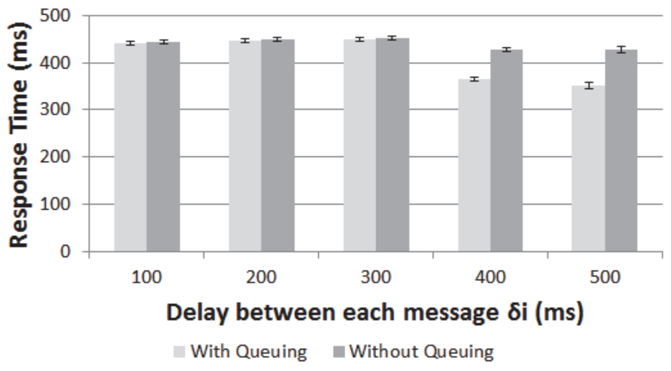

Fig 6(a). Queuing reduces traffic and response time

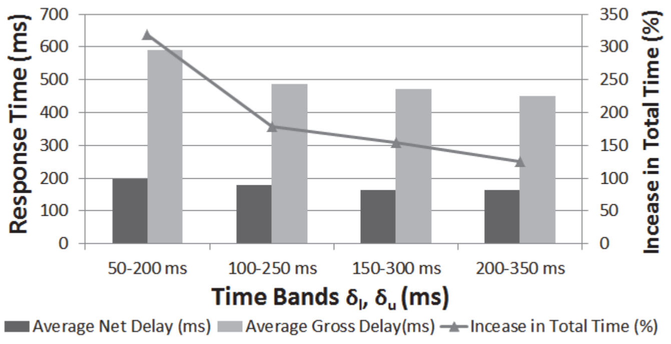

Fig 6(b). Flow control increases the session time.

None of these values are absolute for an MCU as these will differ with experiment and network environment. The observations may be summarized (for a given experiment with consistent characteristics, such as δ_i) as follows:

a) Queuing is able to reduce the volume of messages sent and thus reduce the delay per message in most cases. Thus a queuing mechanism improves performance of an experiment session.

b) Applying flow control is able to send all messages through, but it can considerably increase the total time consumed in a session for a slow MCU or low values of δ_i. Thus a flow control mechanism may not always improve the performance and should not be heavily relied upon.

c) Increasing the frequency of messages increases the delay per acknowledgment of messages.

This shows that all MCUs are not suitable for all experiments due to difference in capacity. The distributed RAL system must identify the type of the experiment node and follow corresponding flow control and queuing controls methods. It is however the maker who must decide the suitability of an MCU for an experiment.

D. Evaluation with Users

The proposed P2P RAL system is aimed to be used by students and teachers. It is necessary that they understand the concepts of constructing and programming a rig. To get a better handle on the conceptual understanding of potential users, a group of students took part in a trial to create a traffic light experiment and access it online on the network.

Although users do not have to manage the underlying messaging structures and data exchange, there were two main issues that users need to be aware of whilst creating their programs:

1. *Ports*: these are implemented and used in the program as variables that can then be used to create the logic that drives the rig. This required that users understand the linkage between the physical connections and the software variables.
2. *Delays*: Despite implementing queueing and flow control at the IIL and UI, the users may need to arrange their program logic so that a forced delay is induced between command instructions for the experiment. For instance, in the traffic lights activity, the lights must change state after specific periods of time. As such, the user must be able to produce an acceptable program outcome when creating time sensitive program implementations. Time critical programming however would not be typically required for basic data acquisition situations of standard rigs.

The evaluation of the participants focused on these aspects. A total of 10 participants took part in the activity. In order to ascertain the impact of the delays and port variable linkage, survey questions were asked of the participants. The results were:

1. All participants were able to assemble the experiments and plug them into the internet within 10 minutes and then complete the activity as per the instructions provided.
2. They were able to change the values of the ports to make the necessary changes to the state of the peripheral devices connected.
3. They were able to change the 'delay' or 'wait' values to modify the behavior of the rigs i.e. slow down or speed up the rigs operation.

Whilst it required an extra compilation step to execute the code to communicate with the target MCU (a BeagleBone Black), this extra procedural step was easily understood and able to be performed without any interjection by staff. Collaborative team work with the activity was clearly evident as participants used differing pre-existing knowledge bases to satisfy overall knowledge and skill requirements to complete the activity. One Windows 7 PC (as the host for designing the UI) had the Scratch and the JIS running in the background where the program was created or altered and tested. The final Scratch program was uploaded to a server where it was converted to an exe file and downloaded into a remote PC. The remote PC had only the JIS running in the background. The rig could be communicated through the network from both the host and remote PCs. For linking the two parts of the activity together (UI and IIL), having a team was also particularly useful. Participants also spent a considerable amount of time reverse engineering the activity to ensure that connections were attached correctly.

Conclusions

A distributed framework for RAL suitable for peer to peer instrumentation and different paradigms of designing it has been presented. The microcontrollers are the key components of this framework and it has been shown that they are capable of delivering acceptable performance with support for adequate bandwidth and latency relative to the experiment. Several ways to create an experiment setup suited to their nature of generating instructions were discussed. This system can be used to create a RAL environment that is highly scalable with users being able add their own rigs and sharing them with peers. The key to real time control by users is the flow control and queuing of instructions and their effects were studied here. The communication protocol discussed covers only the basic elements. Advanced queuing methods and flow control may be used by creating tools for producing a model for individual experiment rigs.

The distributed RAL architecture can enable users to connect to each other and share their rigs. They can design and deploy with high flexibility. However, when applied to remote laboratories, these collaborations may be not as regular as centralized RAL, but can aid the teachers and students by allowing a larger cohort of users to contribute to the RAL with resources that might not be available in centralized locations.

References

[1] L. Gomes and S. Bogosyan, "Current Trends in Remote Laboratories," *IEEE Trans. on Ind. Electron.*, vol. 56, pp. 4744-4756, Dec 2009.

[2] A. Maiti, A. D. Maxwell and A. A. Kist, "An Overview of System Architectures for Remote Laboratories," in *IEEE Int. Conf. on Teaching, Assessment, Learning and Education*, pp. 661-666, 2013.

[3] M. Tawfik, E. Sancristobal, S. Martin, G. Diaz, J. Peire, and M. Castro, "Expanding the Boundaries of the Classroom: Implementation of Remote Laboratories for Industrial Electronics Disciplines," *IEEE Ind. Electron. Magazine*, vol. 7, pp. 41-49, 2013.

[4] T. C. Yang, "Networked control system: a brief survey," *Control Theory and Applications, IEE Proceedings*, vol. 153, pp. 403-412, 2006.

[5] C. Mergl, "Comparison of Remote Labs in Different Technologies," *Int. Journal of Online Engineering (iJOE)*, vol. 2(4), pp. 1-8, 2006.

[6] G. Farias, R. De Keyser, S. Dormido, and F. Esquembre, "Developing Networked Control Labs: A Matlab and Easy Java Simulations Approach," *IEEE Trans. on Ind. Electron.*, vol. 57, pp. 3266-3275, 2010.

[7] J. Garcia-Zubia, P. Orduna, D. Lopez-de-Ipina, and G. R. Alves, “Addressing software impact in the design of remote labs,” *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4757–4767, Dec. 2009.

[8] J. García-Zubia, U. Hernandez-Jayo, et al., "LXI Technologies for Remote Labs: An Extension of the VISIR Project," *iJOE*, vol. 6, 2010.

[9] V. J. Harward, et al., "The iLab shared architecture a web services infrastructure to build communities of Internet accessible laboratories," *Proc. of the IEEE*, vol. 96, pp. 931-950, Jun 2008.

[10] D. Lowe, S. Murray, E. Lindsay, and D. K. Liu, "Evolving Remote Laboratory Architectures to Leverage Emerging Internet Technologies," *IEEE Trans. on Learning Technologies*, vol. 2, pp. 289-294, 2009.

[11] A. Gardel Vicente, I. Bravo Munoz, et. al., "Remote Automation Laboratory Using a Cluster of Virtual Machines," *IEEE Trans. on Ind. Electron.*, vol. 57, pp. 3276-3283, 2010.

[12] I. Santana, M. Ferre, E. Izaguirre, et al., "Remote Laboratories for Education and Research Purposes in Automatic Control Systems," *IEEE Trans. on Ind. Informatics*, vol. 9, pp. 547-556, 2013.

[13] A. A. Kist, A. Maiti, A. D. Maxwell, et. al., “Overlay Network Architectures for Peer-to-Peer Remote Access Laboratories”, *in Int. Conf. on Remote Engineering and Virtual Instrumentation (REV)* 2014, 26-28 Feb. 2014, Porto, Portugal, pp.274-280.

[14] M. Auer, A. Pester, D. Ursutiu, and C. Samoila, "Distributed virtual and remote labs in engineering," *IEEE Int. Conf. on Ind. Technology*, pp. 1208-1213 Vol.2, 2003.

[15] K. Henke, S. Ostendorff, H.-D. Wuttke, and S. Vogel, "A grid concept for reliable, flexible and robust remote engineering laboratories," *Int. Journal of Online Engineering (iJOE)*, vol. 8, pp. 42-49, 2012.

[16] J. García-Zubia, P. Orduña , I. Angulo, et. al., "Towards a Distributed Architecture for Remote Laboratories," *Int. Journal of Online Engineering (iJOE)*, vol. 4(S1), pp. 11-14, 2008.

[17] M. Diponio, D. Lowe, and M. de la Villefromoy, "Supporting Local Access to Collections of Distributed Remote Laboratories," in *AAEE* 2012 Conference, Melbourne, Australia, 2012.

[18] H. Wenshan, L. Guo-ping, and Z. Hong, "Web-Based 3-D Control Laboratory for Remote Real-Time Experimentation," *IEEE Trans. on Ind. Electron.*, vol. 60, pp. 4673-4682, 2013.

[19] M. Tawfik, C. S., D. Gillet, D. Lowe, et. al., "Laboratory as a Service (LaaS): a Model for Developing and Implementing Remote Laboratories as Modular Components", *in Int. Conf. on Remote Engineering and Virtual Instrumentation (REV) 2014*, pp.11-20, 2014.

[20] D. Ursutiu, D. T. Cotfas, M. Ghercioiu, et. al., "WEB Instruments," in *IEEE Education Engineering (EDUCON)* 2010, 2010, pp. 585-590.

[21] A. Melkonyan, A. Gampe, M. Pontual, et. al., "Facilitating Remote aboratory Deployments Using a Relay Gateway Server Architecture," *IEEE Trans. on Ind. Electron.*, vol. 61, pp. 477-485, Jan 2014.

[22] D. Adami, S. Giordano, and M. Pagano, "Dynamic Network Resources Allocation in Grids through a Grid Network Resource Broker," in Grid Enabled Remote Instrumentation, F. Davoli, N. Meyer, R. Pugliese, and S. Zappatore, Eds., ed: Springer US, 2009, pp. 115-130.

[23] C. Kotsokalis, T. Ferrari, et. al., "Grid-Enabled Instrument Representation and Reservation," in *IEEE Fourth Int. Conference on eScience*, 2008, 2008, pp. 16-22.

[24] M. Prica, R. Pugliese, A. Del Linz, et.al., "Adapting the Instrument Element to Support a Remote Instrumentation Infrastructure," in Remote Instrumentation and Virtual Laboratories, F. Davoli, N. Meyer, R. Pugliese, and S. Zappatore, Eds., ed: Springer US, 2010, pp. 11-22.

[25] Z. Nedic and A. Nafalski, "Suitability of SCADA for development of remote laboratories," in *Remote Engineering and Virtual Instrumentation* 2013, pp. 1-4, 2013.

[26] R. Marques, J. Rocha, S. Rafael and J.F. Martins, "Design and implementation of reconfigurable remote laboratory, using oscilloscope/PLC network for www access", *IEEE Trans. on Ind. Electron.*, vol. 55, pp. 2425-2432, June 2008.

[27] G. Patricio and L. Gomes, "Smart house monitoring and actuating system development using automatic code generation," in *IEEE Int. Conference on Ind. Informatics* 2009, pp. 256-261, 2009.

[28] P. Kaushik, A. K. M. Azad and K. C. Vakati, "Customizing Household Mobile Robot for Remote Laboratories", in *Int. Conf. on Remote Engineering and Virtual Instrumentation 2014 (REV)*, pp.144-150.

[29] J. B. da Silva, W. Rochadel, R. Marcelino, et. al., "Mobile remote experimentation applied to education", IT Innovative Practices in Secondary Schools: Remote Experiments, Olga Dziabenko and Javier García-Zubía (eds.), University of Deusto, pp.281-302, 2013.

[30] A. Maxwell, et. al.., "Robot RAL-ly International – Promoting STEM in elementary school across international boundaries using remote access technology," presented at the *in Int. Conf. on Remote Engineering and Virtual Instrumentation (REV) Conference 2013*, 2013, pp 1-5.

[31] Y. Wei-feng, S. Rong-gao and W. Zhong, "Distributed Remote Laboratory using Web Services for Embedded System," in Proceedings of the 3rd WSEAS Int. Conf. on Circuits, Systems, Signal and Telecommunications (CISST'09), pp. 56-59, 2009.

[32] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, pp. 72-93, 2005.

[33] K. Baraka, M. Ghobril, S. Malek, et al., "Low Cost Arduino/Android-Based Energy-Efficient Home Automation System with Smart Task Scheduling," in *CICSNC 2013*, pp. 296-301, 2013.

[34] A. Maiti, A. A. Kist and A. D. Maxwell, "Using Network Enabled Microcontrollers in Experiments for a Distributed Remote Laboratory," *in Int. Conf. on Remote Engineering and Virtual Instrumentation (REV) 2014 Conference*, 26-28 Feb. 2014, Porto, Portugal, pp.180-186.

[35] R. Marin, G. Leon, R. Wirz, et al., "Remote Programming of Network Robots Within the UJI Industrial Robotics Telelaboratory: FPGA Vision and SNRP Network Protocol," *IEEE Trans. On Ind. Electron.*, vol. 56, pp. 4806-4816, 2009.

[36] P. G. Otanez, J. R. Moyne, and D. M. Tilbury, "Using deadbands to reduce communication in networked control systems," in *American Control Conference*, 2002, pp. 3015-3020 vol.4.

[37] D.-S. Kim, Y. S. Lee, W. H. Kwon, and H. S. Park, "Maximum allowable delay bounds of networked control systems," *Control Engineering Practice*, vol. 11, pp. 1301-1313, 2003.

[38] A. Onat, T. Naskali, E. Parlakay, and O. Mutluer, "Control Over Imperfect Networks: Model-Based Predictive Networked Control Systems," *IEEE Trans. on Ind. Electron.*, vol. 58, pp. 905-913, 2011.

[39] S. A. Baset and H. Schulzrinne, "An analysis of the Skype peer-to-peer Internet telephony protocol," *Proc. of the INFOCOM* '06, 2006.

[40] R. J. Costa, G. R. Alves, and M. Z. Rela, "Embedding Instruments & Modules into an IEEE1451-FPGA-based Weblab Infrastructure," *Int. Journal of Online Engineering*, vol. 8, 2012, pp 4-11.

[41] E. Bogdanov, C. Salzmann, and D. Gillet, "Widget-Based Approach for Remote Control Labs," *IFAC Symposium on Advances in Control Education*, 2012.

[42] D. Gillet, "Personal learning environments as enablers for connectivist MOOCs," in *Information Technology Based Higher Education and Training (ITHET), 2013 Int. Conf. on*, 2013, pp. 1-5.

[43] A. Maiti, A. D. Maxwell, A. A. Kist, and L. Orwin, "Integrating enquiry-based learning pedagogies and remote access laboratory for STEM education," in *IEEE Global Engineering Education Conference (EDUCON)*, 2014, pp. 706-712.

[44] J. Garcia-Zubia, I. Gustavsson, U. Hernandez-Jayo, P, et al., "Using VISIR: Experiments, Subjects and Students," *Int. Journal of Online Engineering (iJOE)*, vol. 7, 2011.

[45] C. Salzmann and D. Gillet, "Smart device paradigm, Standardization for online labs," in *IEEE Global Engineering Education Conference (EDUCON)*, 2013, pp. 1217-1221.

Ananda Maiti is a PhD student at the University of Southern Queensland, Toowoomba, Australia. His research area consists of establishing specifications for network infrastructure, end node architecture and interface design for remote control. His research interests are in algorithm design and analysis, networking and remote access laboratories.

Alexander A. Kist received the Ph.D. degree in Communication and Electronic Engineering from RMIT University, Melbourne, Australia in 2004. He is a Senior Lecturer (Telecommunications) and the School Coordinator (Learning and Teaching) in the School of Mechanical and Electrical Engineering in the Faculty of Health, Engineering and Science at the University of Southern Queensland, Toowoomba, Australia. His research interests include teletraffic engineering, performance modelling, remote access laboratories and engineering education. Dr Kist has authored more than 90 scientific articles and from 2009 until 2011 he has lead the Faculty of Engineering and Surveying Remote Access Laboratory initiative. He is an elected executive member of the International Association of Online Engineering (IAOE) and the Global Online Laboratory Consortium (GOLC). He is a Steering Committee member of the annual Australasian Networks and Applications Conference (ATNAC).

Andrew Maxwell received the BEng degree in electrical engineering in 1994, and in 2002 received the PhD Degree in electrical engineering from the University of Southern Queensland, Australia.
He has been a Research Engineer since 2000, and an Academic since 2009. Since 2014 he has been a Senior Lecturer, in Electronics and Communications Engineering. His research interests include measurement science, biomedical engineering, remote access labs, and engineering education.