DISTRIBUTED SDR APPLICATIONS FOR DISTANCE LEARNING

Carlos R. Aguayo Gonzalez (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; caguayog@vt.edu);
Carl Dietrich (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; cdietric@vt.edu); and
Jeffrey H. Reed (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; reedjh@vt.edu).

ABSTRACT

Software-defined radio (SDR) has become a key technology in the development of wireless communications systems because of the flexibility it provides and its potential to enable exciting applications such as cognitive radio (CR). In order to maintain the innovation pace in this area, it is necessary to train future generations of engineers in the techniques, and crafts, of SDR development. This can only be achieved by complementing theoretical coursework with hands-on, practical labs, exposing engineering students to the latest SDR standards and development practices. To facilitate the acquisition of lab equipment for SDR training and reduce its cost, we present a distributed platform that allows remote users and distance learners to share key hardware resources, allowing them to perform complete experiments at a fraction of the cost.

This platform is based on OSSIE, an open-source implementation of the Software Communications Architecture (SCA) developed at Virginia Tech, and the Universal Software Radio Peripheral (USRP), developed as part of the GNU Radio project. In this approach, a computer with the USRP hardware connected is made available for remote access over the internet to students so they can use it to acquire real-time, over-the-air data and feed it into their waveforms, taking advantage of the inherent distributing capabilities of the SCA. This way, experiments and labs can be performed by a large group of students, local or remote, without the need to provide each student with physical access to the RF hardware, reducing costs and facilitating experiment setup.

In this paper, we describe the operation of the platform and the logistics involved in its operation. An example of the experiments that can be performed in this platform is provided, along with initial performance and usability evaluations.

1. INTRODUCTION

Software-defined radio (SDR) has become a key technology in the development of wireless communications systems. This is because of the flexibility it provides and its potential to enable exciting technologies such as cognitive radio (CR). In order to maintain the innovation pace in this area, it is necessary to train future generations of engineers in the techniques, and crafts, of SDR development. This can only be achieved by complementing theoretical coursework with hands-on, practical training, while exposing students to the latest SDR standards and development practices. Unfortunately, equipment procurement to provide such learning environment can be expensive. In order to facilitate the acquisition of lab equipment for SDR training and reduce its cost, we present a distributed platform that allows remote users and distance learners share key hardware resources, allowing them to benefit from practical education at a fraction of the cost.

SDR is a complex, multidisciplinary field. In oder to provide students with the knowledge and skill required to ultimately build an SDR, it is necessary to cover elements from communications theory, software engineering, computer engineering, etc. [1]. Students need to try and experience first-hand the effects of real-world implementation of this kind of communication systems.

Providing this type of education requires specialized equipment. While SDR platforms come in all sizes, capacities, and price points, even low-end platforms can get expensive when considering procuring for a large class size. This may get prohibitively expensive for institutions with limited budgets. Such limitations require SDR instructors to acquire a small set of critical hardware that is then shared by the students. Instructors also have to decide if they will hand out the platforms to the students or leave them in a specific location, with the latter option usually preferred. While this is a viable solution, it goes at expense of the students' convenience, who need to be physically present at that location at a prescheduled time. Furthermore, both approaches severely limit the ability of distance-learning students to participate of the experiments and benefit from the hands-on approach.

It is the desirable to provide an environment where SDR platforms can be shared among students in a remote way, enabling distance learners to participate and increasing convenience for local students. This approach can make the best possible use of the platforms such that a small number of them can support a large number of students. Remote access is convenient and reduces the need for strict scheduling.

In this paper we describe how to enhance SDR curricula with hands-on training without incurring excessive costs, by developing and deploying distributed applications using OSSIE and the inherent distributed capabilities of the SCA. This approach allows concurrent remote access to data captured by a USRP. We address and explain the technical and management issues of adopting such an approach and present the results of initial feasibility experiments involving the deplyment of a simple AM receiver waveform.

2. ENABLING TECHNOLOGIES

SDR has captured the attention of researchers and developers for quite a few years now. Significant technological advances have taken place in SDR and supporting areas that have had

a great impact not only on SDR implementation, but also in its education.

One of the simplest approaches to share hardware resources and provide remote access to enhance distance learning, is to set up a remote computer that has Secure Shell (SSH) access, which provides a secure channel to share data between computers. This way, a single piece of hardware, such as the USRP, can be accessed by multiple users without requiring physical access to it, as shown in Figure 1. This approach has many advantages such as ease of setup and low cost. However, this approach only allows one user at a time and data streams can be difficult to access for audio output and debugging. While there are ways to workaround these limitations, we will focus on other technologies that enable distributed processing and better resource sharing.

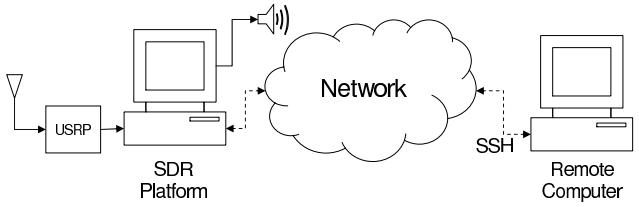

Fig. 1. Remote access to SDR HW using SSH

2.1. The Software Communications Architecture

The Software Communications Architecture (SCA) is an open architecture developed by the Joint Tactical Radio System (JTRS) program of the US Department of Defense (DoD). The SCA provides a framework for the design, development, deployment, and management of SDR applications, or waveforms. It was designed for scalability and waveform reuse, which is achieved by decoupling software and hardware by establishing standard interfaces and a common operational environment (OE). The OE relies on COTS components and well-known design patterns [2] to describe, deploy, configure, and manage components and waveforms.

The OE includes interfaces to control individual waveform components (the "Resource" interface), interact with physical hardware ("Base Device Interfaces"), and manage and control the radio domain ("Framework Control Interfaces").

At the core of the SCA there is the assumption of a modular, reconfigurable platform, which requires the use of sophisticated middleware to deliver data and control information. The SCA relies on the Common Object Request Broker Architecture (CORBA) to provide this functionality. We explain CORBA operation a little bit more in the next section. For now, we need to point out that CORBA provides the SCA with the ability to distribute applications seamlessly, allowing components to be deployed on different processors, boards, computers or networks and yet appear as if they were local to everyone else in the application.

2.2. Common Object Request Broker Architecture

The SCA dictates the use of minimum-CORBA, as standardized by the Object Management Group (OMG) [3], to provide transparent exchange of information across different components. CORBA provides a layer of abstraction between the application and the specifics of the operating system and lower layers enabling a flexible flow of information across the software bus.

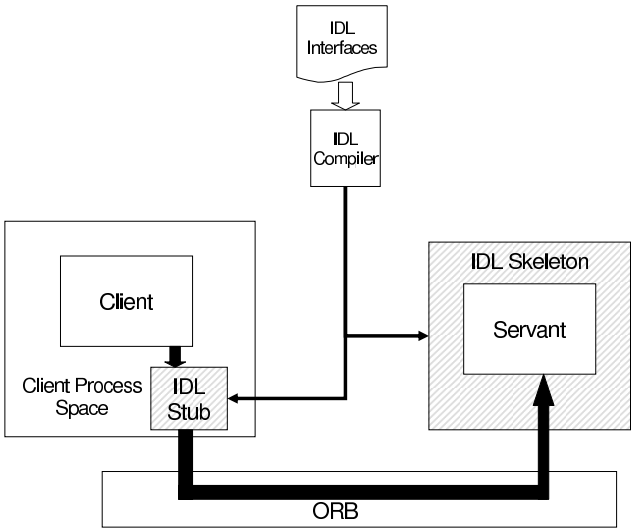

Fig. 2. Communication Between CORBA Components

An in-depth description of CORBA is beyond the scope of this paper, but we will briefly describe its operation from the SCA perspective. First, a CORBA component is defined by describing its interfaces in the CORBA Interface Description Language (IDL), including methods and attributes. Once the interfaces are defined, they are passed through an IDL compiler which creates bindings to the specific implementation language (e.g. C, C++, java, Python, etc.). Two different sets of bindings are created: one skeleton for servants (the components providing a service) and one stub for clients (the ones using a service). From here, a developer has to populate the skeletons implementing the actual functionality.

At runtime, the components register with the ORB and get a unique reference assigned to them. When a client requires a service, it takes the servant's reference, usually obtained from a naming service, and makes the request using the local stub as if both, client and servant, were collocated. This process is shown in Figure 2. Once the request has been sent, the ORB takes control. It finds the servant, delivers the messages, and returns the results – all performed transparently from the client's perspective.

2.3. OSSIE

The Open Source SCA Implementation::Embedded (OSSIE) Project [4] is an initiative by the Mobile and Portable Radio Research Group (MPRG) of Wireless@Virginia Tech to provide an open-source implementation of the SCA. OSSIE is written in C++ and uses omniORB and the Xerces XML parser, both of which are openly available. OSSIE is developed for Linux and specifically supports the Fedora distribution. Cygwin or VMWare can be used to run OSSIE on Windows or other platforms. It only implements the

minimum elements of the architecture required to operate. It also makes some basic assumptions that simplify the development (e.g. only one implementation per component, a DeviceAssignmentSequence is provided when deploying a waveform), making it suitable for efficient, flexible implementations.

Although OSSIE is not a certified implementation of the SCA, it provides an excellent framework for SDR education and experimental SCA development due to its simplicity, shallow learning curve, and free distribution. Currently, OSSIE has been used to teach SDR classes in Virginia Tech and the Naval Postgraduate School (NPS) and several other research projects.

Besides providing an implementation of the SCA core framework, OSSIE includes a select set of signal processing modules and tools that support the development of waveforms. The OSSIE Waveform Developer (OWD) allows users to easily assemble waveforms by creating the architectural code compliant with the SCA and all the XML descriptors required. Another very useful tool is ALF[1], which provides a visual environment to manage SCA waveforms. ALF allows the deployment of waveforms, locally or remotely, and the visualization of data being passed between components, thanks to a set of plug-in tools.

In order to facilitate the interaction with RF and digital conversion hardware, OSSIE also provides a logical device to that serves as a proxy to integrate the Universal Software Radio Periferal (USRP) into the SCA.

2.4. Universal Software Radio Peripheral

The USRP is a open-source, low-cost digital conversion board designed to allow general purpose computers to capture a wide spectrum band. It was developed for the GnuRadio Project and can be coupled with several RF daughter boards that allow access to different spectrum bands [5].

The USRP has 4 high-speed analog to digital converters (ADC) and 4 high-speed digital to analog converters (DAC), 12 bits per sample and 64MSamples/sec, and 14 bits per sample and 128MSamples/sec, respectively[2]. All ADCs and DACs are connected to an Altera Cyclone EP1C12 FPGA, which performs all of the high-speed general purpose operations(e.g. digital up and down conversion, decimation and interpolation). The interface with the computer is done via USB2.

One of the biggest advantages of the USRP is its low cost. This has made the USRP a very popular device within the SDR community. However, at a base price of $700[3], even the USRP can be expensive when considering procurement for a large SDR class. Hence the need for some institutions under budget constraints to share this kind of hardware platform.

[1]ALF stands for "Alien Life Form", which was the initial project name.

[2]Specifications at the time of submission. However, the upgraded USRP2 will soon enter the market.

[3]List price at the time of submission.

3. DISTRIBUTED WAVEFORMS FOR REMOTE RESOURCE SHARING

In an effort to faciliate remote resource sharing and access to critical hardware even for distance learning, we propose an approach that leverages the intrinsic characteristics of the SCA to abstract the underlaying platform without affecting the application's logic.

The basic premise is that a group of students, or users, will share a specific piece of hardware that is too expensive to buy for each user but critical for SDR education, the USRP for example. The basic approach is to have a distributed platform that comprises a main node (U_Node), the one with the USRP connected, and multiple remote nodes (R_Nodes). Within this super platform, there is a proxy in the U_Node between the USRP and the rest of the domain, the "USRP Device", which is in charge of configuring the USRP and translating data into the CORBA domain. Notice that "USRP Device" can support multiple data connections.

Remote nodes are all part of a single SCA domain controlled by the DomainManager. Basically, every time a new user wants to get access to the USRP, it registers her own computer as a remote node, technically adding more processing capability to the whole radio domain. Then, she deploys a waveform in which the majority of the components reside in her R_node, but gets the data from the USRP device in the U_node. This setup is depicted in Figure 3. The only requirement to achieve this deployment configuration, is that, at build time, the right components are paired with the right devices in OWD. This will generate a DeviceAssignmentSequence that tells OSSIE where to deploy each component.

The U_Node is always running and is configured and managed by the instructors. It also contains an instance of the GPP device, which allows certain fast or latency-sensitive components to be deployed in the same hardware where the USRP resides. A common example of this would be a Decimator. The U_Node contains the "DomainManager" and a "DeviceManager", which launches and configures the devices in the node. On the other hand, the R_Node only contains an instance of "DeviceManager" to launch and configure the logical devices residing in it (e.g. GGP and SoundCard devices).

In order to get access to the U_Node, the remote user would have to configure omniORB to use the IP address of the U_Node as the initial reference for the ORB. She would then launch the remote node, using OSSIE nodeBooter, effectively making the new node available to the domain. When this step is completed, she can use ALF to show and install the available waveforms and components in the radio domain from the remote computer. She can pick her desired waveform, install it, display it, and debug it using ALF, with all the signal processing components running locally in her computer, while getting real-time data from the remote USRP.

When different users try to get access to the USRP, they follow the exact same procedure letting OSSIE and CORBA handle the multiple connections to the USRP. This

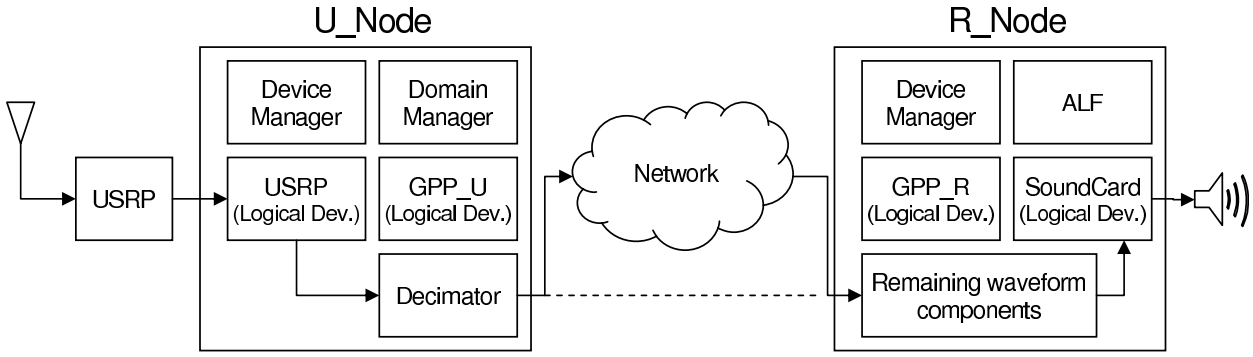

Fig. 3. Distributed receiver architecture

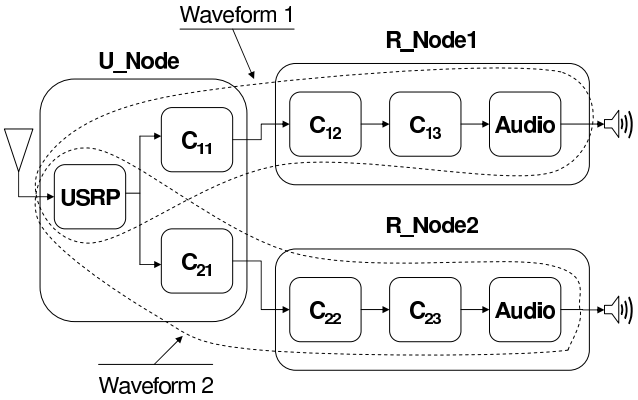

Fig. 4. Logical waveform deployment

way, simultaneous waveforms are deployed which share the USRP data, as shown in Figure 4. CORBA interfaces can support, in theory, an infinite number of fan-out connections. This allows the same data received by the USRP to be broadcasted to every waveform. In reality, there is a limit to the number of connections which depends of the specific CORBA implementation. The limiting term, however, is the latency increase that happens every time a new connection to the USRP is added. We get into more details about this in the results section. In order to limit the maximum number of connections allowed by the USRP, a capacity model is required, where the USRP connection capacity is established. When a new waveform includes the USRP, part of this capacity gets allocated to it at deployment time. When the waveforms are uninstalled, the capacities are returned. Note that this is the expected behavior for logical devices as specified by the SCA.

4. MANAGEMENT APPROACH

In setting up this approach, there are many important aspects that must be considered. One of them is the kind of labs or experiments that can be performed. The remote approach we are proposing is better suited for simple broadcast reception, where a single antenna,or antenna array, can feed a multitude of receivers and where latency is not a critical issue. Receivers for AM, FM, or digital packet radio broadcasting can be setup this way. While it may seem limited, these options are usually enough for a one semester SDR class. It is a bit more difficult to set up experiments where transmission is required. The setup of the USRP and the multiple access to its resources by multiple users is more involved. We can envision, however, a frequency-division multiple access system implemented in the USRP where users get assigned a specific transmit channel as a possibility to enhance this approach. A description of this approach lays beyond the scope of this paper and will be addressed in a future publication.

Another important aspect to be considered when implementing this approach is the security of the U_Node and its local network. Because it is necessary to open some TCP ports in the firewall to allow remote CORBA connections, possible vulnerabilities are created. There are many ways to address these issues, from setting up SSH tunneling to a full-fledged VPN. Close collaboration with the IT staff will be required when setting up the labs.

The course instructor needs to setup the U_Node with a USRP and configure it to the appropriate frequency and decimation rates. Because of the experimental nature of the waveforms being developed by students, they are expected to have errors. Therefore, the ports of the “USRP device” need to be robust enough to support the occasional connection failure. This issue has been addressed for the most part on the current OSSIE repository, but there are always situations that are not foreseen and result in unexpected behaviors. This requires the instructor to occasionally improve the device code and, more importantly, to have a way to remotely restart the “USRP device”

4.1. Resource Requirements

In order to set up the right number of U_nodes to provide an adequate service to the students, it is necessary to collect some statistics about the access behavior of the class. Once the statistics are available, some basic concepts from queueing theory can be applied to estimate the number of USRP ports that are required. It is difficult to have traffic statistics for the very first time this approach is executed, so some assumptions are necessary.

Assume a class of 25 students who are performing an experiment that takes in average two hours to complete. Let’s

further assume that students work individually and only from 5:00 to 10:00 PM for four days a week (before that time they are in class, after they are asleep, and they take Fridays and weekends off). That leaves us with a total of 20 working hours during which they are expecting access to the USRPs. If the remote log-ins are uniformly distributed across all working hours, then there are in average 2.5 log-ins per hour. That is our arrival rate. Because the experiment takes two hours to be completed that leaves the busy hour traffic, E, as 5 Erlangs. We can use this and the Erlang-B equation (1) to estimate the blocking probability, P_b, for a student who tries to use one USRP port remotely for a given number of USRP ports available, m.

$$P_b = \frac{\frac{E^m}{m!}}{\sum_{i=0}^{m} \frac{E^i}{i!}} \quad (1)$$

Figure 5 shows the resulting blocking probabilities for different number of available USRP ports under the current scenario. If we setup a target blocking probability of 0.05, it can be achieved with only 9 USRP ports. If we allow three logical connections per physical USRP, then we only need three USRPs and U_Nodes to serve the whole class. This approach can greatly reduce the budget requirements to run a hands-on SDR class. There are other things that can be done to further reduce hardware requirements, organize students by team for example, without much impact on the quality of the training. Direct physical access to the hardware could also be supported with three USRPs, but only with strict scheduling, and with no ability to support distance learning students.

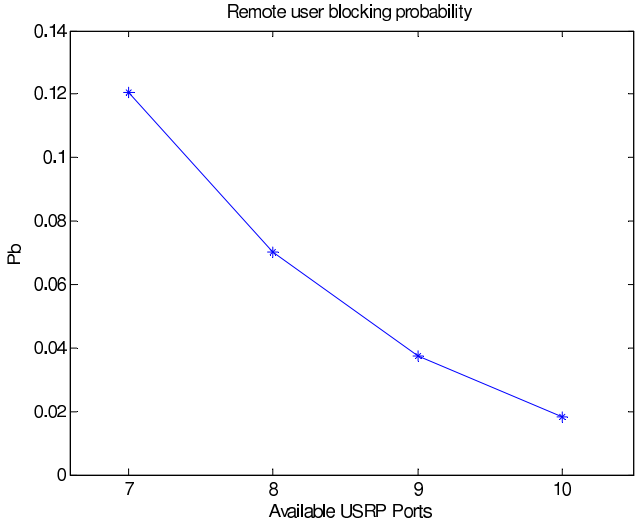

Fig. 5. Remote user blocking probability as a function of available USRP ports

5. RESULTS

In order to validate the feasibility of our approach, we set up a U_Node and several R_Nodes and verified that the infrastructure was capable of supporting it. We chose a simple AM receiver waveform to deploy across all different R_Nodes. The basic AM receiver waveform application, depicted in Figure 6, is currently assigned to the students during the SDR class in Virginia Tech and the NPS. It consists of a simple AM demodulator that receives data from the USRP through a decimator. The output goes out to the sound card. In our case, instead of sending the output to the speakers, we used the "speaker" utility of ALF to confirm the correct reception of the AM signals captured over the air.

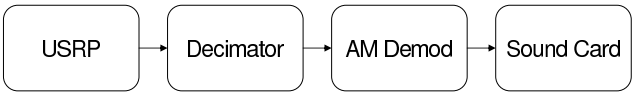

Fig. 6. Sample logical application (AM Receiver)

A total of five R_Nodes were created and deployed over an Ethernet network to verify the feasibility of the approach and identify its limitations and shortcomings. One of the most important issues is the maximum number of waveforms that can be supported by a single USRP. In this experiment, a total of five waveforms were deployed without significant degradation of the sound quality at the output. Because of the use of CORBA, the actual number of connections supported depends on the implementation, but it is much larger than what we can support due to latency. In OSSIE, "provides" ports literally cycle through all the connections and send the same CORBA message to each one them. If the port takes too long to send a specific packet, data coming from the USRP can overwrite older samples, causing a Rx Overrun.

Fully characterizing latency in a TCP/IP network is an involved and complex process, and assessing middleware performance under it is an important research topic, but it is beyond the scope of this paper. Here, we are mostly concerned with assessing feasibility and we limit our results to characterizing the performance penalty due to extra connections to the USRP. To this end, we measure the time the USRP takes to send a packet across multiple waveforms. We do this by profiling the pushPacket function in the USRP logical device implementation. Inside this call, the message is fanned out to all active connections, in our case, to all waveforms using it.

We profiled pushPacket under three different scenarios. In the first one, the U_Node and all the R_Nodes are collocated in the same processor. This is possible, because the distinct nodes are logical representations and can be deployed anywhere without affecting the waveform's logic. This scenario gives us a baseline reference to evaluate the impact of distributing the waveforms across different computers. In the next two scenarios the U_Node and R_Nodes are deployed on two different computers connected via a TCP/IP LAN. In the first one, the decimator component is deployed in the U_Node, technically leaving the same connection layout as in the previous scenario, because the USRP would send data to components that are collocated. For the last scenario, the decimators were deployed in the R_Nodes.

Figure 7 shows the average delays for all three scenarios. We can see that both cases where the USRP and decimators

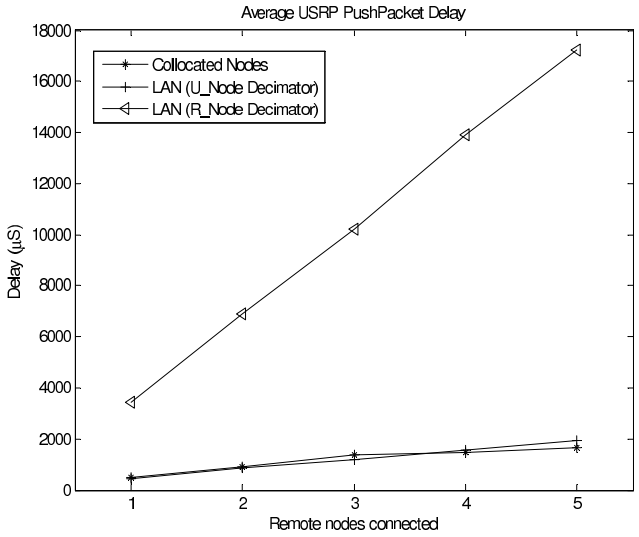

Fig. 7. USRP PushPacket Average Delay

are collocated have similar delays that depend linearly on the number of nodes connected, as expected. When the USRP has to send data through the LAN before the sample rate is reduced by a decimator, however, we can see a significant increase in latency over the previous two scenarios. Again, latency is linearly dependent on the number of remote nodes connected. Using Figure 7, we can make a ball-park estimation of the latency overhead incurred by distributing waveforms. For our system, every time there is a network connection, a minimum of 3.5 ms need to be added to the latency, plus 0.5 ms for collocated connections and all the processing time. Needless to say, there are many factors that can impact these results: processor speed, network data rate, traffic, etc. These have to be considered and more experiments need to be performed before deploying this approach.

6. CONCLUSIONS

In this paper we described how to use OSSIE, the inherent distributed properties of the SCA, and the USRP to provide a practical, inexpensive environment to support SDR hardware resource sharing to enhance SDR education. This approach not only presents convenient remote access by the students to the SDR platforms, but also enables distance learners to benefit from practical experiments and labs.

This approach was demonstrated by implementing a simple AM waveform that was distributed across multiple nodes. A total of five nodes were deployed through a TCP/IP LAN and five waveforms were able to share the USRP without degrading the received sound quality.

ACKNOWLEGMENTS

This work was supported by Texas Instruments, SAIC, and Wireless @ Virginia Tech Affiliates, and by the National Science Foundation under Grant No. 0520418. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] F. Kragh, J.H. Reed, C.B. Dietrich, and D. Miller. *Education for Software Defined Radio Design Engineering*. ASEE National Conference, June 22-25, 2008.

[2] E. Gamma, R. Helm, R. Johnson, and J Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[3] Common Object Request Broker Architecture (CORBA/IIOP) Specification. Available at: *http://www.omg.org*.

[4] Open-Source SCA Implementation::Embedded. Available at: *http://ossie.mprg.org*.

[5] Ettus Research Website. Available at: *http://www.ettus.com*.