Using Node-HTTP-Proxy for Remote Experiment Data Transmission Traversing Firewall

http://dx.doi.org/10.3991/ijoe.v11i2.4443

Ning Wang[1], Xuemin Chen[2]*, Gangbing Song[1] and Hamid Parsaei[3]
[1]University of Houston, Houston, USA;
[2]Texas Southern University, Houston, USA;
[3]Texas A&M University at Qatar, Doha, Qatar

Abstract—**In this paper, a novel real time experimental data transmission solution based on a new web server software architecture that allows the traversing of network firewalls is proposed. With this new software architecture, the public network port 80 can be shared between Node.js and the Apache web server software system. With this new solution, the Apache web server application still listens to the public network port 80, but any client requests for the Node.js web server application through the port will be forwarded to a special network port which Node.js web server application is listening to. Accordingly, a new solution in which both Apache and Node.js web server applications work together via HTTP proxy developed by the Node-HTTP-Proxy software package is implemented on the server-side. With this new real time experiment control and data transmission solution, the end user can control remote experiments and view experimental data on the web browser without firewall issues and without the need of third party plug-ins. It also provides a new approach for the remote experiment control and real time data transmission based on pure HTTP protocol. The solution will significantly benefit the development of remote laboratory technology.**

Index Terms—**Remote Laboratory, Traversing Firewall, Node-HTTP-Proxy, HTTP-Proxy, Node.js, Apache**

I. INTRODUCTION

The application of remote laboratory technology for enhancing engineering education has attracted much attention in the last decades due to its flexible accessibility and resource sharing [1][2]. The increased use of remote laboratories for online education has made network security issues ever more critical. In order to defend against numerous new types of potential attacks, the complexity of network firewalls has been significantly increased. As a result, the use of multi-leveled firewalls (e.g., one for local area network access, one for each of servers, etc.) to protect a computer network has become a common practice. In addition to the firewalls, the servers for remote experiments are configured to promptly block and report suspicious activities. Consequently, the network firewall will inevitably limit the real time remote experimental data transmission, for example limit data transfer rate because of the filter data [3]. However, real time experimental data transmission is an essential function in remote laboratories. Thus, in the design and implementation of the real time remote experimental data transmission, two critical challenges must be met: 1) how to cross the limitations of network security management; and 2) how to achieve high-performance real time experimental data transmission without a need for any extra plug-ins. To the best of the authors' knowledge, being able to provide a stable and high-performance data transmission for remote experiments through port 80 without firewall issues or extra plug-ins remains a critical issue.

Most early stage remote laboratories relied on the Client-Server architecture for achieving high-performance real time experimental data transmission [4][5]. Examples include the remote panel provided by the National Instrument's LabVIEW (Laboratory Virtual Instrument Engineering Workbench) [6], a digital-signal-processor-based remote control laboratory at University of Maribor [7], the Distance Internet-Based Embedded System Experimental Laboratory (DIESEL) at the University of Ulster [11] and others. Later on, Client-Server architecture based on Web services and .NET remote services were developed and deployed for remote laboratories [8]. However, almost all of these systems use special network ports, usually TCP/UDP ports for socket protocol, to traverse the network firewall.

With the continuing improvements to computer performance, the technology supporting the Browser-Server architecture is becoming increasingly more stable and suitable for cross-platform system design. Meanwhile, a large number of new technologies have been developed to support more complex web browser-based internet applications. Consequently, more and more remote laboratory software systems have selected to use the Browser-Server architecture technology. Examples include iLab [9], WebLab-Deusto [10], the Networked Control System Laboratory (NCSLab) [12], the improved NCSLab 3-D [13], the eComLab at UTSA [14] among others. In particular, the eComLab, which was designed and developed in the University of Texas at San Antonio, used SSH tunneling via HTTPS port 443 to address data transmission security while not being blocked by firewall. However, the SSH tunneling has the drawback of decreasing the performance of the system. Therefore, in these systems, the real time experiment data transmission across network firewall issue still has not been solved satisfactorily. Meanwhile, other software plug-ins, such as java applet plug-ins, flash plug-ins, etc, also were used to resolve the drawbacks issue to achieve the high-performance real-time experiment data transmission in these systems.

To solve the challenge of achieving high-performance real time remote experimental data transmission through a Browser-Server architecture, the authors proposed and developed a unified remote laboratory framework [15][16]. The primary goal of this unified framework was

to allow online experiments to be accessed by any Internet browsers without the need for any extra plug-ins. For achieving high-performance real time data transmission, a new web socket protocol implemented through the Socket.IO was also created [17]. Although the Socket.IO solution performs well, the real time data transmission still required a special network port (TCP port 1029).

Consequently, to address this critical issue, a novel approach is required to implement the high-performance real time experimental data transmission across the network firewall only via the public network port 80. A stable and efficient solution to this problem will be an essential improvement for the remote laboratory development and help with future developments.

The rest of the paper is organized as follows: Previous works and methods research for Browser-Server architecture remote laboratory software are summarized in Section II. In Section III, the detail working process of the new experimental data transmission solution is presented. In Section IV, the new solution implementation process is presented. Concluding remarks are drawn in Section V.

II. PREVIOUS WORKS AND METHODS SEARCH

Previous work [14] focused on the fundamental design and development of a unified remote laboratory framework was designed. The subsequent iteration of the design resolved challenges of developing cross-browser and cross-device web user interface as an improvement to the unified framework [15]. This unified framework was used to implement remote control engineering experiments. As an example, the new Smart Vibration Platform (SVP) remote experiment is now used to teach students in mechanical engineering courses at the University of Houston. The SVP offered students hands-on experience on structural vibration control by using a Magneto-Rheological (MR) and Shape Memory Alloy (SMA) braces to control the vibration of a one story model [17].

Nowadays, with the fast development and improvement of the network technology, Node.js has become one of the more prevalent server-side technologies that are revolutionizing the web. It is also a free-to-use software system for server-side application development. Node.js contains a built-in HTTP server library, thus allowing more control of the web server by making it possible to run a web server without the use of external software systems, such as Apache or Microsoft IIS [18]. Node.js enables web developers to create an entire web application in JavaScript which are both server-side and browser-side. In the Node.js server-side software system, Socket.IO is a JavaScript library used to support real time web applications development [19][20].

With the previous improved unified remote laboratory framework, a Comet solution based on Node.js and its Socket.IO package was implemented in the server-side and experiment control workstation. For improving the experimental data transmission, a new application transmission protocol based on the Socket.IO was designed and implemented. However, with the Comet solution, the real time experiment data was transferred via network port 1029, which is the special network port, to traverse the network firewall. The reasons are as follows. The web server was built on the Apache HTTP web server software engine. By default, the server occupied the public network port 80. Consequently, the Node.js web server software system has to use the other network port (such as port 1029.) for real time experimental data transmission. Currently, in the network system, the network firewalls are designed for network security in all colleges and universities. Thus, in most academic institutions, only several network ports, including public network port 80, are opened and most other network ports are either blocked or limited. The user only can remotely conduct the experiment using the web browser without installing any plug-ins while on-campus. Accordingly, when the user hopes to conduct the remote experiment off-campus, they also need to use the VPN. Consequently, the issue of traversing the network firewall remains, which was the essential issue not to be resolved in previous works. In order to resolve this essential transmission issue, a new approach must be found to transfer the real time experimental data only via the public network port 80.

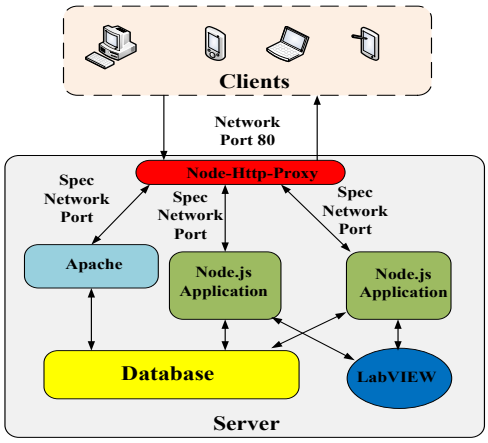


Figure 1. The architecture of the novel real time experiment control commands and data transmission

This paper presents the detailed working process regarding a novel real time experiment control command and data transmission approach proposed for the remote laboratory development. This experimental data transmission approach via HTTP proxy gets both Apache web server application and Node.js with its Socket.IO library working together. Figure 1 illustrates the architecture of the novel real time experiment control command and experimental data transmission solution. The new solution contains the following three parts: HTTP proxy, Node.js and socket.IO. In addition to the HTTP proxy implementation was the Node-HTTP-Proxy, which is a free software package developed by the Node.js development team.

III. PROPOSED SOLUTION

The main purpose of this new solution is to transfer the experimental control command and experimental data between client and server across network firewall. The ability to transfer across the network firewall will be a great improvement for the remote laboratory unified framework. In order to address the network firewall issue, an HTTP proxy was set up using a Node-HTTP-Proxy in the server-side. The Node-HTTP-Proxy proxy was used to monitor the connections on the public network port 80 in the server-side. The HTTP proxy switches the requests from the client browsers to the proper software applications which run in the server-side. Each application listened for its own special network port based on the host-

name in each request. In the new web server, an HTTP proxy application built up by Node-HTTP-Proxy software package was configured and used to finish the requests for transferring tasks. This proxy application monitored public network port 80 and looked up the appropriate web server application for a request from the client. The proxy also switches the experimental data directly from the web server applications to the client web browsers. Currently in the server-side, there are one Apache web server application and one Node.js web server application which all worked for the remote SVP experiment. The Apache web server application was used to generate the user interface framework, and the Node.js web server application was used to handle the experimental data and equipment control commands.

Figure 2 illustrates the new data transmission solution working process. As seen in Figure 2, Node.js web server software should be set up in server-side. Furthermore, there is also a need to compile and install the Node-HTTP-Proxy software package and reconfigure the web server. Meanwhile, the web server must include the Apache and Node.js web engines in order to provide experimental control commands and experimental data real-time transmission environment.

A. The Node-HTTP-Proxy for the network port sharing

In the previous iteration of the unified remote laboratory framework the Apache and Node.js web server applications could not be set to listen to the same public network port 80 at the same time. Additionally, the Node.js web server application could not be run without first activating the Apache web server application.

A solution to the above problem is to set up one HTTP proxy using the Node-HTTP-Proxy software package. This new approach can get both Apache and Node.js working together without problems while also at the same time share the same public network port 80.

Node-HTTP-Proxy is a full-featured HTTP proxy which uses Node.js web server software system. The proxy is also the free and open source software package. Node-HTTP-Proxy was designed as a middleware concept and middleware is a simple way to add new features to the application stack. Nevertheless, the problem here is that the extremely popular middleware style of web application development is often incompatible with high performance real-time data transmission capability. For example, if a body decoder middleware was used, every request will need to be buffered in full before the body can be properly decoded. This approach will increase memory usage and reduce system performance. This process can be compared to simply relaying unaltered requests and responses to other destinations. All the tasks of the middleware must be waited for and read into memory, possibly altered, then sent again, thus adding to both the time and resource cost of each request.

However, when the Node-HTTP-Proxy was designed and developed, its purpose was for high performance real-time data transmission capability. The concept of a stream in Node.js lends itself well to working with HTTP requests. One of Node.js strengths is its ability to stream data. Meanwhile, a great deal of performance can be achieved simply by piping the request and response streams to other destinations and then back again. More importantly, if any subsequent task in the Node-HTTP-Proxy task chain tries to listen for 'data' or 'end' events, or otherwise treat the current HTTP connection as a stream, it just doesn't work. As a buffered request exits a stream, the request simply becomes a data object.

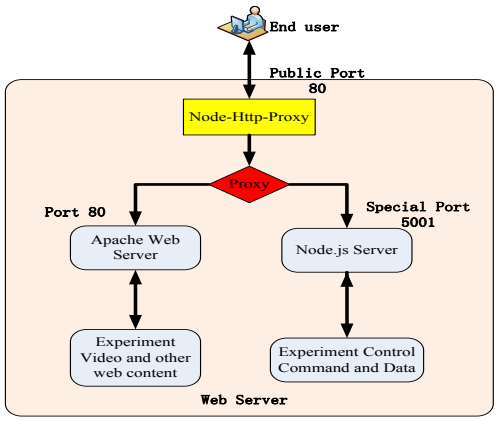


Figure 2. The new data transmission working process

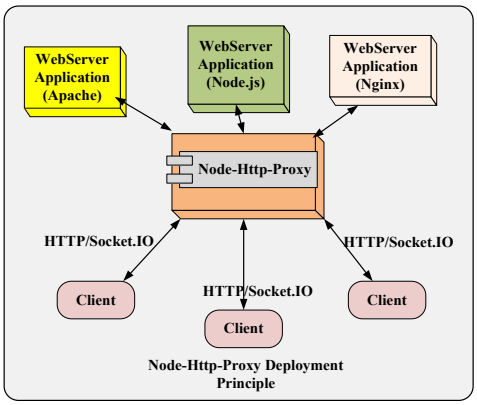


Figure 3. Node-HTTP-Proxy Deployment Principle

Figure 3 illustrates the deployment principle of the Node-HTTP-Proxy. From its deployment principle, it can be seen that the Node-HTTP-Proxy is working as the agent to transfer the request and data between the clients with web server applications via the public network port 80.

The Node-HTTP-Proxy also has a high capacity for the proxy Web Sockets protocol. Therefore, apps can be run independently on different special network ports while simultaneously serving everything to the user over public network port 80, such as the socket.io for the data transmission solution. Example 1 is a segment of a simple code segment for using Node.js with Node-HTTP-Proxy on the public network port 80.

```
var HTTP = require('HTTP'),
    HTTPProxy = require('HTTP-proxy');

HTTPProxy.createServer({
  hostnameOnly: true,
  router: {
    //web-development.cc
    'www.my-domain.com': '127.0.0.1:80',
    'www.my-other-domain.com': '127.0.0.1:5001'
  }
}).listen(80);
```

Example 1: Node-HTTP-Proxy code line

The example code creates the basic HTTP proxy server application and listens for the public network port 80. The code will switch any requests from the clients to two server applications which separately listens for network port 80 and network port 5001.

B. Server-side software system (Node.js)

A stable server-side software engine must be chosen which can support real-time communication web application development in order to allow real time experimental control command and data transmission. Node.js, with its support for real time communications, becomes a prime candidate for use. The most notable distinction of Apache web server software engine is that Node.js is an especially fast and efficient server software system that scales well with application size and scope. The Start Servers and Min-Spare Servers settings of the Apache web server helps keep a specified number of idle Apache servers running in order to bypass the time required to start servers for new connections. In contrast, Node.js instructs the operating system (through e-poll, k-queue, /dev/poll, or select) that it should be notified when a new connection is made, and then it goes to sleep. If any new clients connect, then Node.js executes the callback. Each connection is only a small heap allocation. Figure 4 illustrates the Node.js software system architecture.

Node.js is server-side software designed to write scalable Internet applications and notably setup web servers. Node.js is also a packaged compilation of Google's V8 JavaScript engine, which include the libuv platform abstraction layer and a core library which is primarily written in JavaScript. Meanwhile, Node.js uses an event driven operation mode, asynchronous I/O port to minimize overhead and maximize scalability. The original goal of Node.js was to create web sites with push capabilities as seen in web applications like Gmail. Nevertheless unlike most other JavaScript programs Node.js is not executed in a web browser. Instead Node.js is executed as a server-side JavaScript application. In the server-side, Node.js not only implements multiple common JavaScript specifications, but also it provides a Read–Eval–Print-Loop (REPL)[21] environment for interactive testing.

Node.js also is a JavaScript environment running in Google's V8 JavaScript engine. As depicted in Figure 4, Node.js only exposes non blocking asynchronous interfaces to the programmer. Node.js has very few abstractions. Its power lies in the fact that it stays away from certain undesirable interfaces, such as synchronous I/O. There is no worry about an event completing and taking over while the user is in the middle of another task. Each Node.js is a single thread. To increase the work load, multiple Node.js instances may be started, and the kernel can be relied upon for load balancing. Memory isolation is enforced at the process boundary.

Node.js uses the module architecture to simplify the creation of complex applications. Modules are akin to libraries in the C language, or units in the Pascal language. Each module contains a set of functions related to the 'subject' of the module. For example, the Node HTTP proxy module contains functions specific to HTTP Proxy. Node.js provides some core modules out of the box to help you access files on the file system, create HTTP Proxy and Socket.IO, and perform other useful functions.

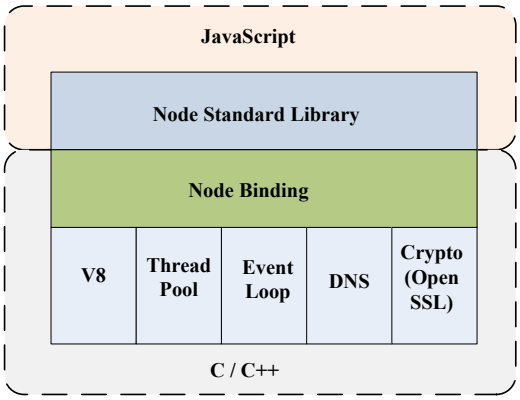


Figure 4. Node.js Architecture

```
var comunicate_ io = require('socket.io');
var HTTPProxy = require('HTTP-proxy');
```

Example 2: Node.js example Code in Server

As can be seen in the simple code segment of Example 2, the '*require()*' function can be used as a simple way to include modules in the program. Node.js is a promising technology and an excellent choice for high load applications. Node.js has been proven by corporations, like Microsoft, eBay, and Yahoo [22].

C. Real time web application with Socket.IO

The WebSocket protocol was used to support real time communication for web application development. The WebSocket protocol is most often used to describe as the "Real-time Web." [23]Basically, the purposes of WebSocket are to break the limitations of the request/response protocol of HTTP. Instead, with event-handling implemented at each end, it creates a socket connection directly between the browsers with the back-end. This makes it possible for events to be triggered in the back-end of the user's browser without the browser having to poll to see whether anything has happened at the back-end. When a message is sent via the WebSocket protocol connection to the browser, this triggers a pre-defined event and the contents which are associated with the event immediately shown in the browser. As WebSocket protocol is bi-directional, the events occurred in the browsers also can generate messages which are sent through the WebSocket protocol connection to the back-end. The WebSocket protocol also effectively provides an alternative to use Ajax over HTTP protocol.

As mentioned previously, the real time communication application needs to be based on the WebSocket protocol. In the interest of adapting the server-side architecture, Socket.IO, which is the module of Node.js, was also used. The WebSocket protocol was also used to connect the client web pages with the web server.

Socket.IO is also supported by other software packages other than WebSocket. If there are additional requirements from the user, Socket.IO can fall back on other methods, such as Adobe Flash sockets, JSONP polling, and AJAX long polling, etc, while continuing to provide the same interface. Although Socket.IO can be used as simply a wrapper for the Web Socket protocol, it provides many more features including broadcasting to multiple sockets,

storing data associated with each client, and asynchronous I/O. Figure 5 illustrates the Socket.IO working process.

Socket.IO enhanced the WebSocket protocol by providing built-in multiplexing, horizontal scalability, automatic JSON encoding/decoding, and more. Socket.IO also uses Long Polling. Long Polling addresses the weakness of traditional polling methods by asking the server for new information a certain intervals and keeping the connection open even if the server has nothing new to process. This technique dramatically decreases latency and network traffic, which means that it efficiently disguises itself as a server-push technique.

Socket.IO is a JavaScript library. Through blurring the differences between the different transmission mechanisms, it is possible that the Socket.IO supports the real-time web applications in every browser and mobile device. Socket.IO mainly includes two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it also uses event-driven operational principle. Example 3 and Example 4 show sample codes of these two parts.

```
var io = require('socket.io').listen(80);

io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```

Example 3: Socket.IO example Code in Server

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('HTTP://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
</script>
```

Example 4: Socket.IO example Code in Client

IV. SOLUTION IMPLEMENTATION

A. Technical Characteristics of the novel data transmission solution for unified remote laboratory framework

The architecture of the novel real time experiment control commands and data transmission solution, which is described in the previous section, has been used for the new Smart Vibration Platform (SVP) remote experiment. The SVP remote experiment was developed based on the unified remote laboratory framework. As the novel unified framework is based on Web 2.0 Technology, which include HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and jQuery/jQuery-Mobile JavaScript libraries, the whole framework was built directly on top of MySQL database. The PHP, which stands for Hypertext Preprocessor and MySQL database driven data-streaming solution, eliminated the need for the client side LabVIEW

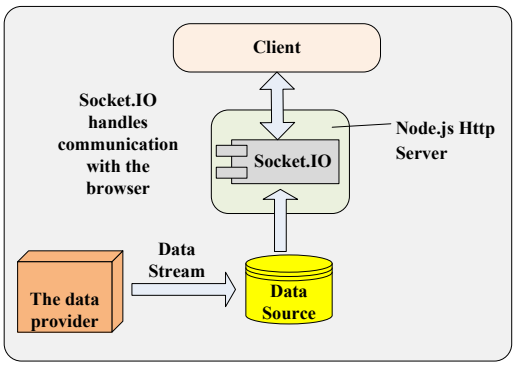


Figure 5. Socket.IO Working Process

TABLE I.
TECHNOLOGY/PROTOCOL/SOFTWARE LIST FOR THE NEW SOLUTION

	Name	Technology/Protocol/Software	remark
1	HTTP Proxy	Node-HTTP-Proxy	Part of Node.js
2	Data Protocol	Socket.IO	Part of Node.js
3	Database	MySQL 5.5	
4	Server - Web Service	Apache(2.2), Node.js(V0.10.20),JSON	LtoN (LabView to Node.js)
5	Equipment Control	LabView(2012)	

(Laboratory Virtual Instrumentation Engineering Workbench) plug-in. Three technologies were mainly used for the system implementation, including the LabVIEW-to-Node.js technology for experimental data and experiment equipment control commands transmission, the novel video transmission approach based on HLS protocol for real-time system monitoring, and Mashup technology for user interface implementation. Table I lists the detail technology, protocol and software package which were used in the new data transmission solution.

B. Node-HTTP-Proxy configuration

In the interest of allowing real time experiment control command and data transmission via public network port 80, a new architecture was required in the web server to get the Apache 2.0 web server application working well together with Node.js V0.10.20. Configuration of the new architecture to match the Node-HTTP-Proxy requirement was also needed.

In order to achieve this aim, the following tasks must be accomplished. Firstly, The Express and Node-HTTP-Proxy software package need to be installed in the server-side. Accordingly, the virtual hosts which need to be set up normally on Apache web server software system. Secondly, all virtual hosts should be placed in a common directory (for example: "/localhost"). Then the Apache web server application would listen for the public network port 80. The next step is to configure the port in file "httpd.conf" (such as changed "Listen 80" to "Listen other special port"). Finally, all virtual hosts must be fixed. As defined in "extra/HTTPd-vhosts.conf", all virtual hosts were set to an IP based name Virtual Host (such as HTTP://127.0.0.1) instead of using a port (such as HTTP://vr-lab.engineeringtech.tsu.edu:80). On the Node.js server-side software system, the application/server (Node.js virtual host) that listened for the spe-

cial network port (such as port 5001) which is different with Apache port (arbitrary choice of port number) was created. Then in the "/localhost" directory, a file called "nodeHTTPProxy.js" was created. The following code segment, Example 5, shows sample code of the Node-HTTP-Proxy in our server. For using node-HTTP-proxy, a proxy server case which is listening for the public network port 80 also was created in nodeHTTPProxy.js.

```
// Module dependancies
var HTTPProxy = re-
quire('/usr/local/lib/node_modules/HTTP-proxy/lib/node-
HTTP-proxy')
, express = re-
quire('/usr/local/lib/node_modules/express/lib/express');
// HTTP proxy-server
HTTPProxy.createServer(function (req, res, proxy) {
   // Array of node host names
   var nodeVhosts = [
      'vhost1'
      , 'vhost2'
   ]
   , host = req.header('host')
   , port = nodeVhosts.indexOf(host) > -1
      ? 80
      : 5001;
   // Now proxy the request
   proxy.proxyRequest(req, res, {
      host: host
      , port: port
   });
})
.listen(80);
```

Example 5: NodeHTTPProxy.js example code

After solving all of the issues mentioned above, node-HTTPProxy.js just need to be run. Each of the Node.js applications has a map file that contains the port that the application is listening to as well as a map that indicates the expected path which the application is being served on.

C. *LabView to Node.js via Socket.IO*

In order to implement real time communication between the client application and the server application, a new Socket.IO based application transmission protocol was designed and developed. This new application communication protocol includes two parts, a client part that runs in browsers and a server part that runs in the web server. The two parts were developed with JavaScript language and also enhanced the real-time communication by the new Socket.IO based application transmission protocol.

In this new application transmission protocol, a custom defined special communication instruction set was used. With the new instruction set, communication security can be guaranteed during the process of con

ducting the remote experiment. In the new protocol, brief instructions were used to control the experiment to improve data transmission performance.

For the protocol implementation, two JavaScript program files were required: one for client application (runs in web browsers) and the other for server application (runs in web server). Then a new Node.js task was created to run the protocol in the server-side, and this server-side application must hold running status indefinitely. Because only the server-side protocol application holds the active status, it can ensure the normal real time communication. In client-side, there is a configuration file which defines and describes the communication instruction functions to support the client application normal operation in web browsers.

With the resolution of the issues mentioned above, the server-side protocol was run using 'forever start' command of Node.js in server-side. The server-side protocol entered into the active status, and the real time communication will be present during normal operation. We implemented the experiment data record function using our new data transmission protocol. The downloaded experiment data for the real displacement and the displacement reference in remote SMA experiment was plotted in the same figure by using MATLAB. Figure 6 and Figure 7 depict the results of the displacement change of SMA in two tests from plug-in free remote laboratory.

D. *Sample Paradigm of the new solution*

Table II shows the comparison of the sample paradigms, which show the terminal user interface, which was developed under the novel unified frameworks within different terminal devices.

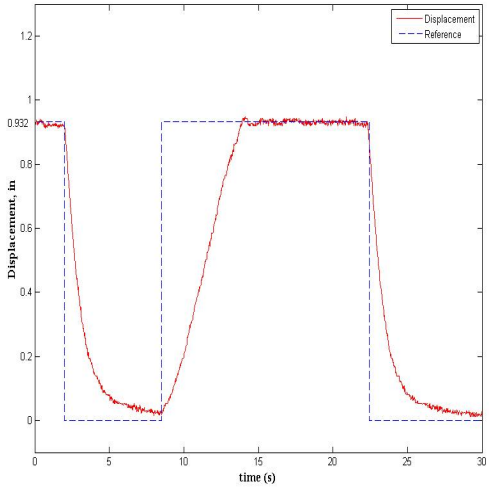


Figure 6. Arbitrary displacement control (with amplitude of 0.932in) in remote SMA experiment

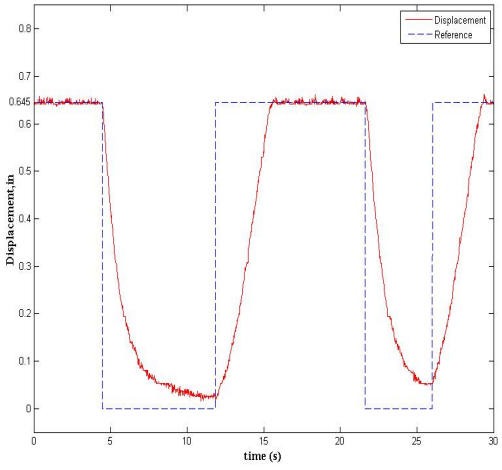
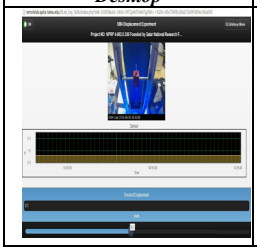
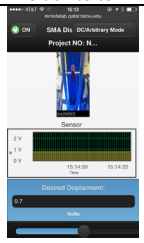
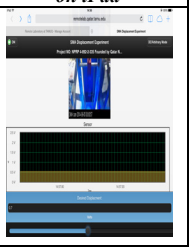


Figure 7. Arbitrary displacement control (with amplitude of 0.646in) in remote SMA experiment

TABLE II.
SAMPLE PARADIGM OF THE NEW SOLUTION COMPARISON IN DIFFERENT DEVICES

Sample Paradigm on Desktop	*Sample Paradigm on iPhone*	*Sample Paradigm on iPad*
		

This user interface can be run in any terminal devices without the installation of any plug-ins or software. As an example, the SMA remote experiment was used to depict the experiment operation method from the user interface. When the switch on the left corner of the web page is turned on, the user can move the slide bar to control the Desired Displacement value. The sensor output, which indicates the desired displacement volt value, is shown in the middle of the web page. The real-time video is displayed on the top of the web page. With our novel data transmission solution, the real-time experiment data can be transferred to different terminal interfaces across the network firewall. Meanwhile, end users can also use any terminal devices to send the commands from client web page to the remotely located workstation as the novel data transmission solution now allows communications across the network firewall.

The remote laboratory system based on the novel unified framework is able to take full advantage of the hardware's potential. With the new real time command and data transmission solution, the improved remote experiment provides better control performance for a wider range of terminal equipment.

V. CONCLUSIONS

The paper presented a novel real time experiment command and data transmission solution with the new web server architecture. To solve the traversing network firewall issue, we implemented the HTTP proxy using the Node-HTTP-Proxy software package in the server-side. To improve the data transmission performance, a new experiment application transmission protocol based on Socket.IO was designed and implemented. The SMA remote experiment was augmented with the new solution as a demonstration of the benefits it provides. End users can now conduct the SMA remote experiment and view the experiment data in real time through web browsers anywhere that has internet connection without any third party plug-in. Consequently, the novel real time experiment data transmission solution gives the unified framework much needed improvement.

REFERENCES

[1] L. Gomes and S. Bosgoyan, *"Current trends in remote laboratories"*, IEEE. Trans. on Industrial Electronics, Vol 56, NO 12, pp. 4744-4756, December 2009, ISSN: 0278-0046.

[2] J. Rodriguez-Andina, L. Gomes and S. Bogosyan, *"Current trends in industrial electronics education"*, IEEE. Trans. on Industrial Electronics, Vol 57, NO 10, pp. 3242-3244, October 2010, ISSN: 0278-0046.

[3] C. Salzmann and D. Gillet *"Challenges in Remote Laboratory Sustainability"* In Proceedings of International Conference on Engineering Education, 2007

[4] S. Li; J. Huai; and B. Bhargava *"Building High Performance Communication Services for Digital Libraries"*, Computer Science Technical Reports Paper 1212, 1995.

[5] D. Gillet, C. Salzmann, R. Longchamp and D. B. Telepresence *"An Opportunity to Develop Real-World Experimentation in Education"*, European Control Conference, Brussels, Belgium, Session WE-M-L 1, July 1-4, 1997.

[6] N. Duro, R. Dormido, H. Vargas, S., Dormido-Canto, et al. *"An Integrated Virtual and Remote Control Lab: The Three-Tank System as a Case Study"* Computing in Science & Engineering, 10(4), 50-59, 2008. http://dx.doi.org/10.1109/MCSE.2008.89

[7] D. Hercog, B. Gergic, S. Uran and K. Jezernik, "*A DSP-Based Remote Control Laboratory*", IEEE. Trans. on Industrial Electronics, Vol54(6), pp. 3057-3068, December 2007, ISSN: 0278-0046.

[8] M.J. Callaghan, J. Harkin, E. McColgan, T.M. McGinnity, L.P. Maguire *"Client–server architecture for collaborative remote experimentation"* Journal of Networks and Computer applications, ISSN: 1084-8045, Volume 30,Issue 4, pages 1295-1308, November 2007

[9] V. J. Harward, J. A. DelAlamo, S. R. Lerman, P. H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbur, P. D. Long, T. Mao, L.Naamani, J. Northridge,M. Schulz,D. Talavera, C.Varadharajan, S. Wang, K. Yehia, R. Zbib, and D. Zych, "The iLab shared architecture: A web services infrastructure to build communities of internet accessible laboratories," Proc. IEEE, vol. 96, no. 6, pp. 931–950, Jun. 2008. http://dx.doi.org/10.1109/JPROC.2008.921607

[10] P. Orduna, J. Irurzun, L. Rodriguez-Gil, J. Garcia-Zubia, F. Gazzola, and D. Lopez-de-Ipina, "Adding new features to new and existing remote experiments through their integration in WebLab-Deusto," Int. J. Online Eng., vol. 7, no. S2, pp. 33–39, 2011.

[11] M. J. Callaghan, J. Harkin, TM. McGinnity and LP. Maguire *"Client-Server Architecture for Remote Experimentation for Embedded Systems"* iJOE International Journal of Online Engineering, ISSN: 1861-2121, 2(4) 2006

[12] W. S. Hu , G. P. Liu & H. Zhou *"NCSLab: A web-based global-scale control laboratory with rich interactive features"*. IEEE Transactions on Industrial Electronics, ISSN: 0278-0046, 57(10), 3253-3265. 2010

[13] Y. L. Qiao, G. P. Liu, G. Zheng & W. S. Hu *"Web-Based 3-D Control Laboratory for Remote Real-Time Experimentation"*. IEEE Transactions on Industrial Electronics, Vol 60(10), pp. 4673-4682, Oct 2013, ISSN: 0278-0046.

[14] A. Melkonyan, A. Gampe, M. Pontual and G. Huang, *"Facilitating Remote Laboratory Deployments Using a Relay Gateway Server Architecture"*, IEEE. Trans. on Industrial Electronics, Vol 61(1), pp. 477-485, Jan 2014, ISSN: 0278-0046.

[15] C. Olmi, B. Cao, X. Chen and G. Song *"A Unified Framework for Remote Laboratory Experiments"*, In Proceedings of ASEE Annual Conference & Exposition, Vancouver, BC, Canada. June 26 - 29, 2011.

[16] C. Omli, X. Chen and G. Song *"A Framework for Developing Scalable Remote Experiment Laboratory"*, In Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2011 (E-Learn 2011), pp. 2045-2050, Honolulu, Hawaii. October 18-21, 2011.

[17] X. Chen, D. Osakue, N. Wang, H. Parsaei and G. Song "Development of a Remote Experiment under a Unified Remote Laboratory Framework" in Proceedings of the World Congress on Engineering Education 2013. H.R. Parsaei and K.S. Warraich, eds.

[18] T. Hughes-Croucher, M. Wilson *"Up and Running with Node.js (First ed.)"*, O'Reilly Media, p. 204, ISBN 978-1-4493-9858-3, April, 2012

[19] D. Herron *"Node Web Development, Second Edition"*, Packt Publishing, ISBN 184951514X, Jul 19, 2013.

[20] R. Rai "*Socket. IO Real-time Web Application Development"*, O'Reilly Media , ISBN 178-2-1607-87, February, 2013.

[21] E. Allen, R. Cartwright, B. Stoler "DrJava: A lightweight pedagogic environment for Java", 33rd SIGCSE Technical Symposium on Computer Science Education, February (2002)

[22] P. Tatade *"Why Node, the fundamental difference between Node and other languages"*, Online, http://www.cuelogic.com/blog/why-node-the-fundamental-difference-between-node-and-other-languages, published February, 5, 2014

[23] Q. Liu and X. Sun *"Research of Web Real-Time Communication Based on Web Socket",* Int'l J. of Communications, Network and System Sciences, Vol. 5 No. 12, 2012, pp. 797-801. http://dx.doi.org/10.4236/ijcns.2012.512083

AUTHORS

Ning Wang is with Department of Electrical and Computer Engineering, University of Houston, Houston TX, USA (nwang@uh.edu).

Xuemin Chen. is with Department of Engineering Technology, Texas Southern University, Houston TX 77004, USA (e-mail: chenxm@tsu.edu).

Gangbing Song is with Department of of Mechanical Engineering, University of Houston, Houston TX, USA (e-mail: gsong@uh.edu).

Hamid Parsaei is with Department of Mechanical Engineering, Texas A&M University at Qatar, Doha, Qatar (e-mail: hamid.parsaei@qatar.tamu.edu).

This work is based upon work supported by the Qatar National Research Fund under Grant No. NPRP 4-892-2-335. Submitted 01 February 2015. Published as resubmitted by the authors 10 March 2015.