ReadyComm: A Hands-on Practice for VoIP Network

Sean Guillen, Chris Fernandes, Mira Yun, Chen-Hsiang Yu, and Magdy Ellabidy

Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{guillens, fernandesc, yunm, yuj6, ellabidym}@wit.edu

Abstract

Compared to traditional landline phone services, Voice over Internet Protocol (VoIP) technology has emerged as a competitive alternative in the communication field in 1973. Studies have shown that VoIP technology has a high adaptability and portability, along with its low cost and robustness. In higher education, although we can teach theoretical knowledge in the classroom, it is not easy to provide a hands-on practice to verify learned knowledge. The research question we have is: how to design a hands-on practice for the students to verify learned knowledge of VoIP technology. In this paper, we present *ReadyComm,* which a project designed to help the students create and manage their own VoIP network. The current results indicate that *ReadyComm* not only helps understand VoIP technology and wireless mesh network, but it also provides a hands-on practice to strengthen the understanding. We firmly believe this is a good example that should be shared to other similar teaching institutes.

Keywords – VoIP, Wireless Mesh Network, SIP, Hands-on Practice.

I. INTRODUCTION

The first landline phone service was constructed in 1877-78 and it has dramatically changed the people's life in communication [1]. However, this kind of landline phone service suffers from its inherited nature, i.e. easily impacted by natural disasters. For example, Hurricane *Maria*, which struck Puerto Rico in late 2017, was the second longest recorded blackout in world history [2]. This incident seriously affects landline phone services. From technology point of view, it is not acceptable due to the amount of resources we have in the 21st century.

Voice over Internet Protocol (VoIP) is a methodology starting in 1970s for enabling voice communications over Internet Protocol (IP). Since voice and multimedia data can be transmitted over the IP layer, a VoIP system can be considered as a possible solution to areas affected by natural disasters. In higher education, although we can teach methodology and theory of VoIP and wireless mesh network in a traditional classroom in higher education, we have a difficulty to provide a hands-on practice for the students to verify learned knowledge. This difficulty motivates us to design a hands-on practice for VoIP Network.

We propose *ReadyComm*, which is hands-on practice for VoIP network. We construct a Wireless Distribution System (WDS) with 4 Linksys access points (APs). The cost of each AP is between $35 and $50. As for the VoIP server, we adopted Raspberry Pi Model 3 B that is around $35. Raspberry Pi, produced by Raspberry Pi Foundation, is an affordable, lightweight, single-board computer that has roughly the size of a credit card and can serve as a VoIP server. Once given a proper software, the Raspberry Pi can process the routing of calls based on the dial plan architecture. We used two separate smart phones as clients, one is a Samsung Android Galaxy Note 8 and another is an iPhone 7 Plus. In order to enabling communication between the VoIP server and its clients, Asterisks, an open-source public branch exchange (PBX) software, was installed on the Raspberry Pi along with FreePBX, an open-source graphical interface, to provide easy configuration. The service will be able to run without relying on any external service or specialized hardware beyond the

Raspberry Pi, APs, and devices running Session Initiation Protocol (SIP) softphone applications. To enhance security of the service, we also run an authentication server on an Ubuntu Virtual Machine (VM).

In the following of the paper, we will introduce the design of *ReadyComm* in details, following by the implementation and testing of the system. At the end, we will conclude our study and explain future work.

II. READYCOMM

The *ReadyComm* network consists of two servers: the VoIP server running on the Raspberry Pi and the authentication server running on an Ubuntu VM. Each of these servers is connected to an AP and from here clients can be added to our network as necessary as long as they are within range of the APs. These APs come together in a WDS that acts as a mesh network. Figure 1 shows a diagram of our network architecture.

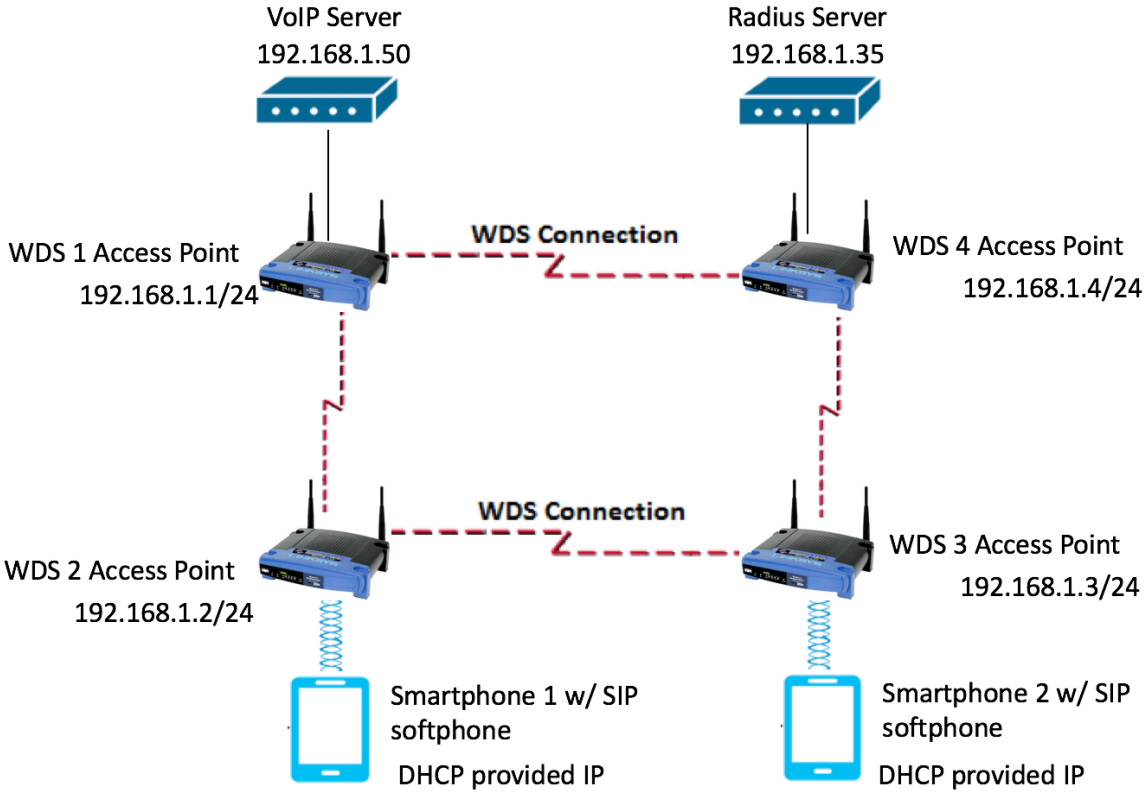

Fig 1. ReadyComm System Architecture

2.1 VoIP Server

To configure our VoIP server on the Raspberry Pi which operates on the Raspbian operating system, Asterisks 13 was installed as well as FreePBX. After the initial install, the FreePBX web-based graphical interface is ready for use. The first step in securing the system was to move into the first point of entry which is to secure the management interface of both the Pi and FreePBX by changing the usernames and passwords from default. Next, the extensions were set by utilizing the *Add Extension* feature, which allows a user extension to be created through a web form by going through 2 steps. The first step of the process involves selecting an

extension type, extension number, display name for the user, password, and optionally the *Outbound Caller ID*, and *Email Address*. For *Type*, PJSIP is selected, which is a new SIP channel driver for Asterisk and is built on the PJSIP SIP stack. As shown in Figure 2, the *Extension Number* and *Display Name* can be entered to match any numbering and naming convention desired. The *Password* must be shared with the user and will be used on their mobile device to register with the VoIP server. The second step simply asks if the user will be a user manager, meaning the user has center privileges to control their extension. Once finished, *Apply Changes* must be click in on in order for new configurations to take effect. That completes the process to add an extension. There is also a *Quick Create Extension* option which removes some of the optional fields and streamlines the extension process. Once added, the extension will appear in FreePBX's dashboard, as seen in Figure 3.

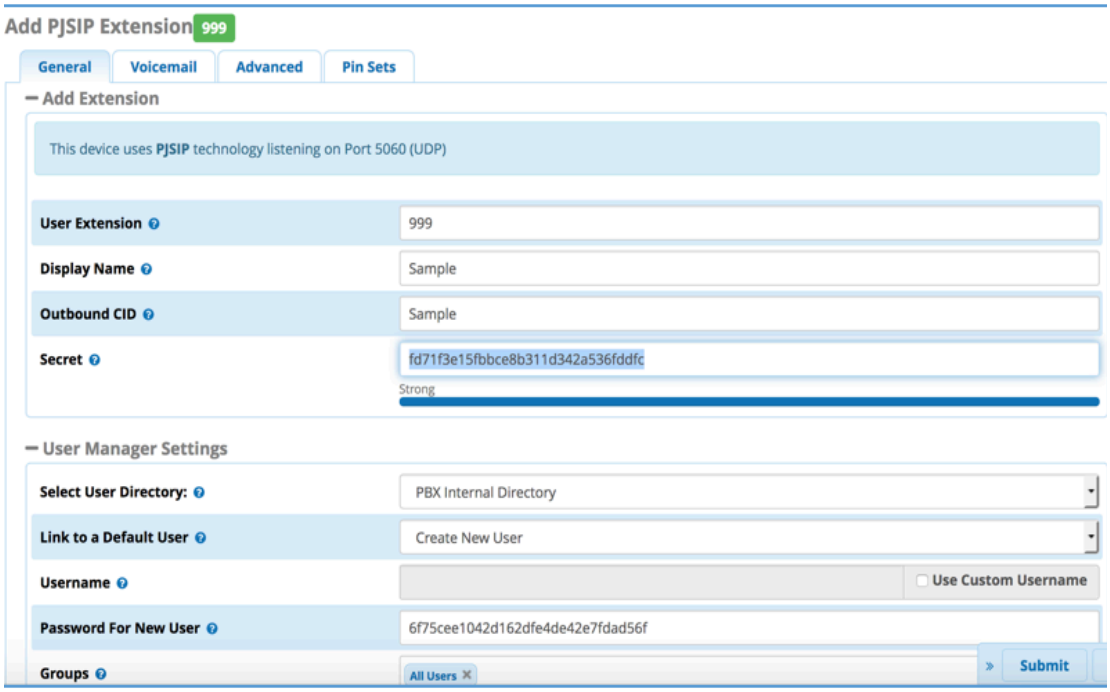

Fig 2. Adding Extension to Server

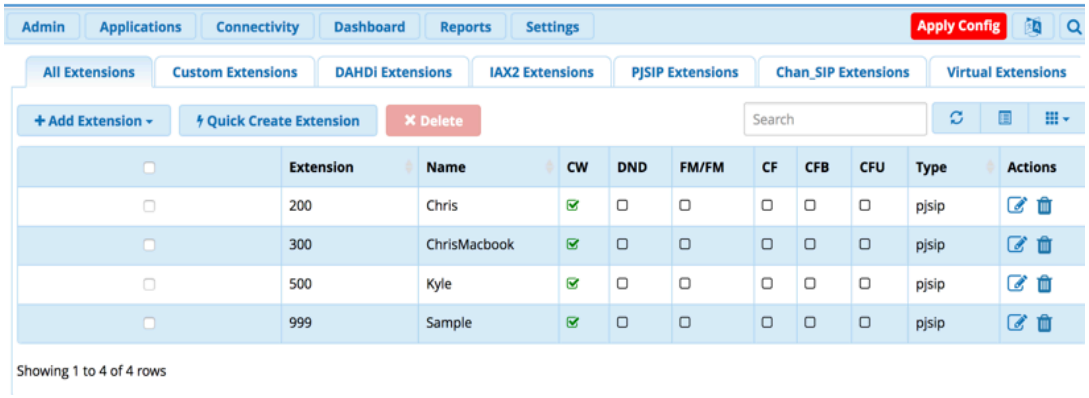

Fig 3. Dashboard of added extensions

This started the base of our VoIP system as the server will be the hub that contains all the information necessary to facilitate the necessary functions for VoIP. With the Raspberry Pi setup, it was time to complete the network that acts as the communication bridge for server and clients.

2.2 Wireless Distribution System

The mesh network itself began with flashing each AP and installing the Tomato firmware [3]. The correct firmware version must be selected to match the AP model. Tomato is a small, lean and simple replacement firmware that works in junction with our Linksys WRT54GL routers. Tomato features an easy to use GUI, bandwidth usage monitor, more advanced QoS and access restrictions, and most importantly is capable of wireless features like WDS and wireless client modes [3]. With Tomato installed our wireless mesh can begin to be built.

The concept behind mesh networks is to create a truly wireless network that is extremely scalable. In a wireless mesh network, the network connection is spread out among dozens or even hundreds of wireless mesh nodes, in our case the Linksys WRT54GL routers, which communicate to each other to share the network connection across a large area [4].

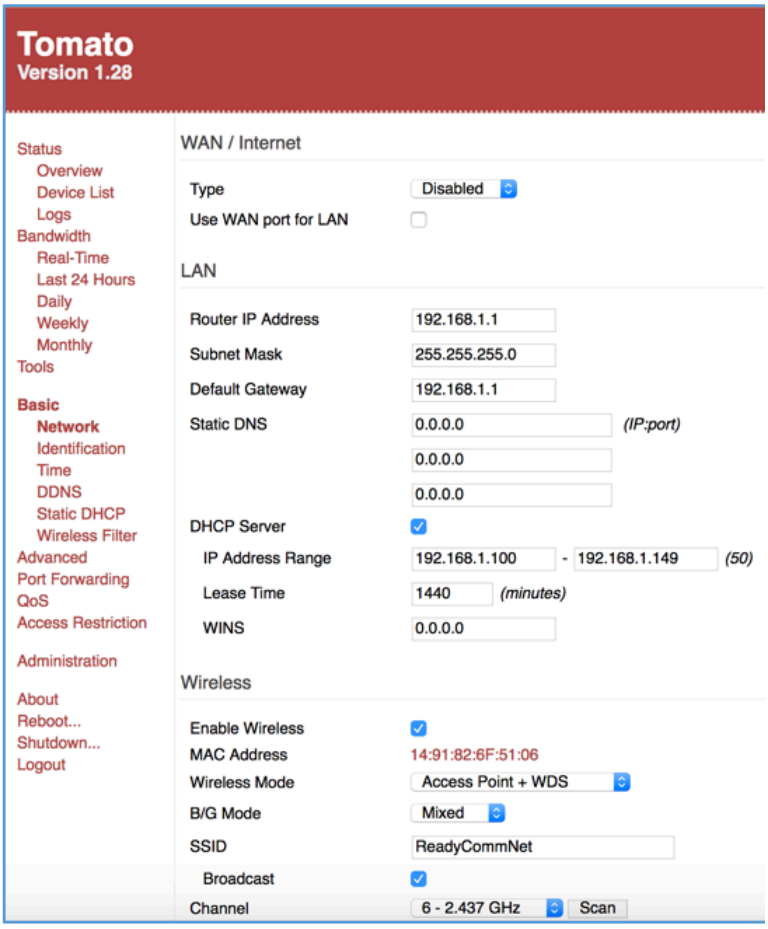

Fig 4. Basic Network Setting of WDS Main AP

Once the AP is flashed and rebooted the WDS configurations can begin. The WDS configuration begins with the setup of a main AP, which is no different than the others except for the fact that it provides DHCP services to the network and will serve as the default gateway. The first step in configuring the main AP is to disable the WAN interface by selecting disable under *WAN Type* in the Basic Network configuration menu as shown in Figure 2. Then we configure the IP and IP services. The AP's *Router Address* is set to be 192.168.1.1 with a *Subnet Mask* of 255.255.255.0, which gives 254 usable IPs for hosts. The checkbox next to *DHCP Server* must be checked to enable DHCP services then we assign a range of 192.168.1.100 – 192.168.1.149 in the *IP Address Range* box that can be handed out to hosts as they connect to the network. This DHCP range can be set to any range of IPs within the 192.168.1.0 network, but for testing purposes the range was kept small. The next step is to ensure *Enable Wireless* is checked and from there the *Wireless Mode* is set to Access Point + WDS. The *SSID* is set to any desired ID and can optionally have *Broadcast* disabled by checking. The *Channel* setting can be left default, however if there are interference concerns it can be changed. As shown in Figure 3, in the *Security* field, select WPA/WPA2 Personal then select AES for *Encryption* and enter a desired *Shared Key*. In the *WDS* option select Link With… and then enter the Mac address of the other non-main APs. As additional APs are added their Mac addresses must be entered here. For changes to take effect *Save* must clicked on at the bottom. All these settings can be found below in Figure 4.

For the other APs all settings are virtually the same except for the Router IP, which must be unique, the *DHCP* check box must be unchecked, and the *WDS* option must be set to Automatic. To verify WDS links select Device List under Status on the left hand side menu and check for the WDS peering in the list of connected devices.

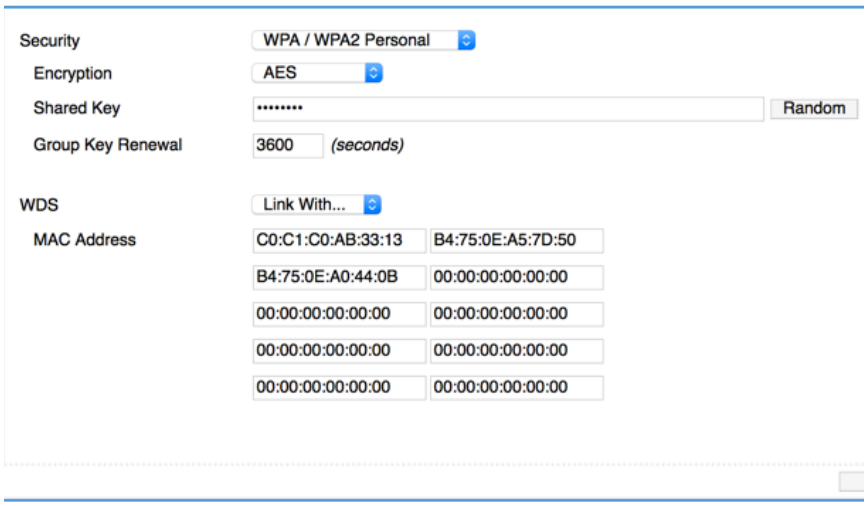

Fig 5. Encryption on the APs for WDS network

2.3 Authentication Server

We need to consider some further security within our WDS network. As shown in Figure 5, WPA2-Personal was the security protocol in place for this network. WPA2 uses Pre-shared Key Authentication (PSK) in order to accept devices into the network. PSK is essentially a secret value manually entered on both the AP and each wireless device. This creates several areas for weaknesses. For one, the key needs to be kept secret and manually updated, and the key itself may be weak.

For example, WPA2 can be cracked quite easily using tools supplied by the Kali Linux operating system. The Aircrack tool can help some with malicious intentions into picking up all available traffic within its vicinity, capturing a packet of a user using WPA2 to join a network, and ultimately using a dictionary attack or whatever means preferred for cracking the password that is the packet [5]. To prevent this, an additional server, an authentication server, was put in place to prevent unauthorized access into our network.

```
client wds1 {
ipaddr = 192.168.1.1
secret = testing123
}

# IPv6 Client
client localhost_ipv6 {
    ipv6addr    = ::1
    secret      = testing123
}

# All IPv6 Site-local clients
#client sitelocal_ipv6 {
#   ipv6addr    = fe80::/16
#   secret      = testing123
#}

#client example.org {
#   ipaddr      = radius.example.org
#   secret      = testing123
#}
```

Fig 6. Adding APs to FreeRADIUS

FreeRADIUS was used as our authentication server and it was implemented running on an Ubuntu VM. Installing FreeRADIUS is as simple as *sudo apt-get install freeradius* from the terminal. With FreeRadius installed, we added all the APs into the *clients.conf* file by adding its name, IP address, and secret key. This way, our network will only operate with the APs we specifically define. The edit to this file can be seen in Figure 6.

Statically adding the APs in our topology provides a huge security benefit. Having to authenticate routers prevents the ability of a man-in-the-middle attack. Without this implementation, a person with malicious intent could crack WPA2, join the network, and add their own AP into our network without any of the users knowing. If a user were to have their phone automatically connect to our network as soon as they are in range, they would automatically be funneling their traffic through the hacker's AP without being aware.

This same logic can be applied to end devices that are a part of our network, so FreeRadius can again help add another layer of security. User accounts can be created with a simple *sudo vi username* to add create a base user. From there a passkey can be created for that user. In our case, we wanted to further the security implementation with this passkey by first running the password through a hashing function to create a ciphertext so the password is not stored in cleartext. Figure 7 shows the example of inserting hashed password for user. The only additional libraries needed are the *passlib.hash* in order to encrypt the password.

```
chris Crypt-Password := "42CDD33820BCF8FF4BEEE575017DE1A1F06D4D39C
#
# The canonical testing user which is in most of the
# examples.
#
bob Cleartext-Password := "hello"
#   Reply-Message := "Hello, %{User-Name}"
#
```

Fig 7. Example of inserting hashed password for User

This, in a way, creates a dual-authentication in order to functionally use our VoIP services, as we require authentication to access the network as well as access in order to sign into the VoIP service through Zoiper. With all this in place, we ultimately have a fully functional VoIP server via a wireless mesh that has its own authentication server as an additional layer of security to keep out intruders.

2.4 VoIP Client

After creating the VoIP server on our Raspberry Pi and the creating of the WDS network via our Linksys APs, we add the clients to make a functional service.

To configure the mobile devices, the Zoiper app was downloaded from the respective phone's App Store. Once the application is downloaded and installed the device is put into airplane mode (optional) then connected to the *ReadyComm* network by finding the SSID configured on the APs and attempting to join by entering the configured password for the WDS network. The next step is to have the device register with the VoIP server by opening the Zoiper application and going into *Settings***.** In *Settings* set the *Account Name* to any name desired, the enter the VoIP server's IP in the *Domain* field (192.168.1.50). The *Username* and *Password* must be entered to mirror what was configured on VoIP server during the extension setup steps. Once all required fields are filled in tap on *Register* and if successful the *Registration Status* will change to OK. This entry can be seen in Figure 8.

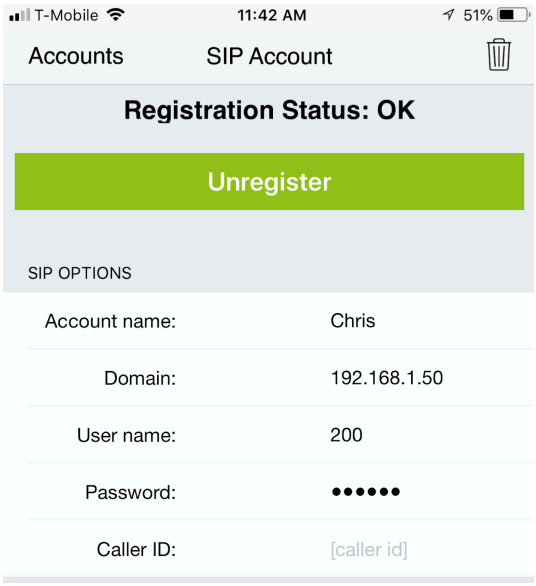

Fig 8. The SIP registration process on an iPhone using Zoiper

III. TESTING *READYCOMM*

In order to verify the functionality of the system, several tests were conducted and compared against the project's goals. FreePBX's testing capabilities proved helpful for this section.

Calls were tested following successful SIP registration. This was done by calling the configured extensions several times and checking for quality of the calls utilizing FreePBX to verify the metrics. The call performance of the system provided 100% call completion and 100ms or less of latency. The only call issues were observed when the APs were placed too far from one another causing latency and packet drops.

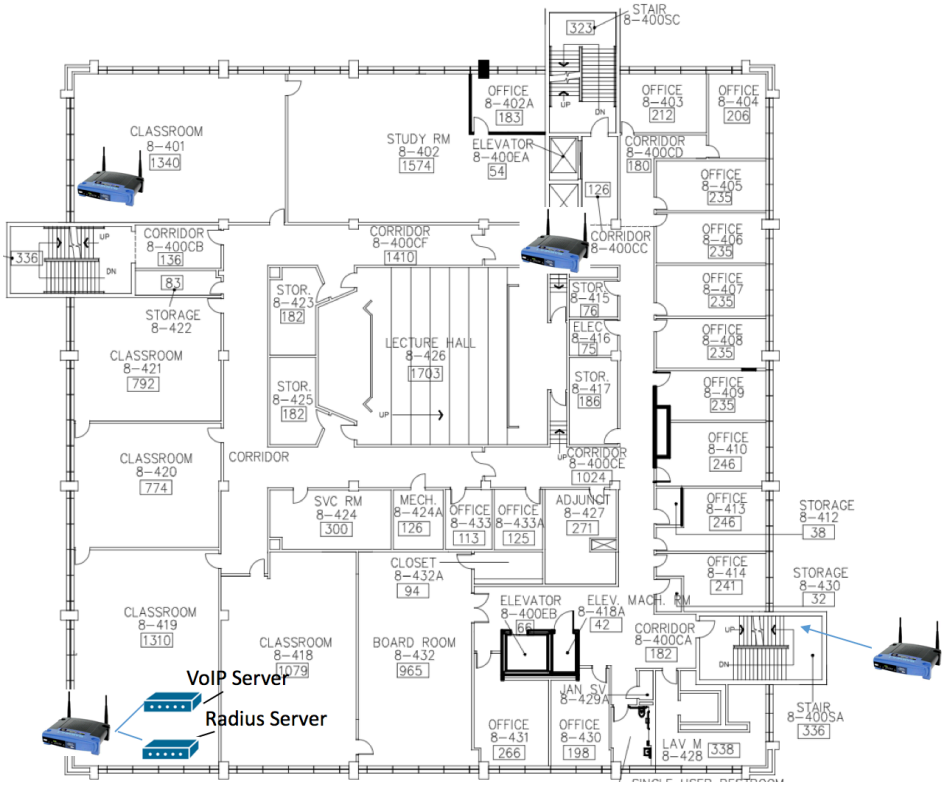

Fig 9. A floor plan of 4th floor in Beatty Hall

These test calls took place on the fourth floor of Beatty Hall at Wentworth Institute of Technology. As shown in Figure 9, APs were placed throughout the entire floor of the building. The call was placed from Classroom 8-419, the room hosting the servers, and was received in the corner Office 8-404 diagonally across the span of the room.

The amount of calls that can be simultaneously held were to be measured. Upon testing for max amount of concurrent calls while maintaining stability it was determined the system can handle 10 calls at once. This is due to the Raspberry Pi's processing limitations. Call encryption was tested to ensure the call data is encrypted as the raw data was encrypted as it passed through the WDS network using AES.

The WDS network was tested by placing calls using each access point and monitoring call performance from different distances. It was determined each AP should be placed roughly 35 feet from one another for optimal performance. This can differ based environmental variable such as building materials. As mentioned earlier, our success rate for calls was 100% of the ten test calls that took place. More detailed results of these tests can be seen below in Table 1.

Table 1. Call Rate Averages

Average Statistics of 10 Calls	
Statistic	***Rate***
Average Received Bitrate	83178.8 bit/s
Average Sent Bitrate	85 bit/s
Current Received Bitrate	83310.8 bit/s
Current Sent Bitrate	85 bit/s
Sent Bytes	308974.4 bytes
Sent Bytes Payload	236344 bytes
Received Packets	2943.8 packets
Sent Packets	2914.2 packets
Received Bytes	548.2 bytes
Received Bytes Payload	468.2 bytes

The authentication server was tested using *radtest*, which tests the service by sending queries requesting to join the network after passing in parameters of the username and password. This ran internally but we also ran tests attempting to join the network. We ran tests with two different accounts, *Chris* and *Bob* and their successful connection can be seen below in Figure 10.

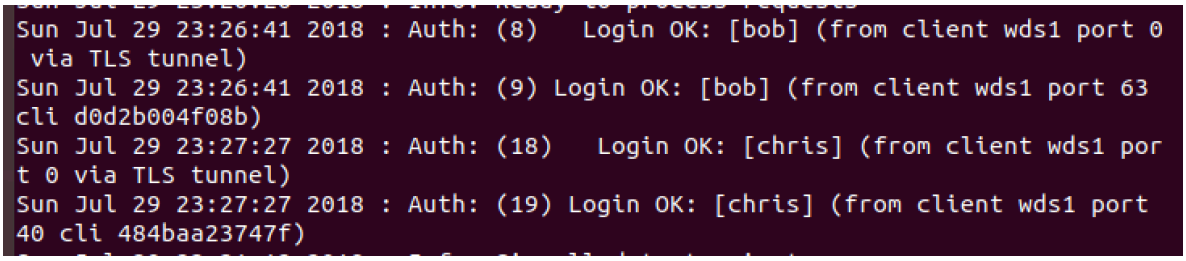

Fig 10. Successful connection to network with

IV. CONCLUSION AND FUTURE WORK

In higher education, learning VoIP network in a traditional classroom is dry and there is no standard hands-on practice for verifying learned knowledge. In this paper, we present *ReadyComm*, which is an effort of creating a hands-on project for undergraduate students. The project creates a VoIP server from scratch to strength the understanding of fundamental knowledge. The proposed idea is not only adaptable for business with different sizes, but it also saves money in a long term. The demonstrated implementation particularly is very scalable. It is easy to add more APs to the network to supply a large number of clients, such as different smartphones and tablets. In addition to the scalability, the cost of the design can be remained reasonable for added licenses.

In the future, we are considering including more challenges in the design to mimic a real usage. For example, creating extension APIs for the developers to add more SIP compatible services, creating security holes for vulnerability learning, standardizing the project to make it as a regular practice for networking classes, etc. The current result not only presents a good example of learning by doing in VoIP technology, but it also provides a flexibility to include more interesting topics to extend the students' learning in computer networking domain.

REFERENCES

[1] 1870s-1940s-Telephone: http://www.elon.edu/e-web/predictions/150/1870.xhtml

[2] Goodkind, Nicole. “Six Months after Hurricane Maria, Puerto Rico's Power Outage Is the World's Second Largest Blackout Ever.” *Newsweek*, 12 Apr. 2018. https://www.newsweek.com/puerto-rico-power-hurricane-maria-blackout-882549

[3] Tomato Firmware: http://www.polarcloud.com/tomato

[4] Mira Yun, Magdy Ellabidy, and Bowu Zhang, "Project-based Learning Example: Wireless Mesh Networks for Undergraduates", The Journal of Computing Science in Colleges, Vol 30:2, pp.52-59, December 2014 .

[5] Roos, Dave, "How Wireless Mesh Networks Work.",HowStuffWorks, 20 June 2007