VoIP: Voice over Internet Protocol for Small Single-board Computers

Final Report of Term Project

Students:
Trupeshkumar R. Patel
trpatel2@crimson.ua.edu
David Coleman
dmcoleman1@crimson.ua.edu
Xiaoming Guo
xguo29@crimson.ua.edu

Supervisor:
Dr. Xiaoyan Hong
hxy@cs.ua.edu

Department of Computer Science

VoIP: Voice over Internet Protocol for Small Single-board Computers

(Final Report)

Trupesh R. Patel*, David Coleman*, Xiaoming Guo*
*Department of Computer Science
College of Engineering
University of Alabama
Tuscaloosa, AL 35487
{*trpatel*2, *dmcoleman*1, *xguo*29}@crimson.ua.edu

Abstract— **Asterisk PBX [1] is an implementation of IP-PBX software, a digital version of Private Branch Exchange (PBX) technology, meant for IP networks instead of traditional telephone networks. IP-PBX software allows for the setup of VoIP (Voice over Internet Protocol) communications on computers and mobile devices, and can be run on less powerful devices such as Raspberry Pi. However, using smaller single-board devices come with the limitation of computing power and storage capacity. As a result of these limitations, packets can be dropped at a great rate, cutting off VoIP callers when a Pi needs to handle a significant number of calls simultaneously. In this project, the authors provides test the Raspberry Pi's suitability for providing IP-PBX services by analyzing the packets delays, network jitters, end-to-end network latency, and bandwidth between an Asterisk PBX system installed on Raspberry Pi 4 [2] and IP clients that are using VoIP as service. This analysis is performed by testing the system with clients at varying distances from the Raspberry Pi.**

I. INTRODUCTION & BACKGROUND

Voices can have multiple functions among the information flowing at the edge of the IoT network. They can carry valuable content, reflect the conditions of an environment, and can be used to command other entities through acoustic actuators and phone calls. However, implementing voice systems at the edge of such networks typically faces challenges, namely that edge devices are always constrained by computing power, bandwidth contention, and energy consumption. Therefore, it is not feasible to implement a complete TCP/IP stack on each node and give all nodes the ability to connect to remote entities outside the local network.

IP-PBX (IP-based Private Branch Exchange) provides a comprehensive solution to address the aforementioned issues. As the analog equivalent of IP PBX, the design of the traditional PBX system is to serve a private organization, in which both the geographic area and the communication connection are limited to a specific scope. Connections between internal phones are without cost, while only central office lines provide connections to the public switched telephone network (PSTN). This scheme meets our expectations for edge IoT communication – internal communication does not occupy egress bandwidth, and some switcher servers still reserve the communication

egress to the outside. Leveraging VoIP technology, IP-PBX has ported the PBX scheme to the Internet, replacing telephone lines with packet-switching networks. The IP-based paradigm offers better scalability and lowers the cost same as the Internet brings to other domains.

Several mature implementations of the IP-PBX paradigm are available, including 3CX [3] and Asterisk PBX [1]. Asterisk is an open-source software package that can run all PBX functions, along with some other functions, usually on a Linux operating system platform. Voicemail services, conference calling, interactive voice response, and call queuing are provided by Asterisk, along with the essential telephony services. It also provides multi-party calling and displaying of caller ID (display calling number). To interact with digital telephone equipment and analog telephone equipment, Asterisk needs the support of PCI hardware, the most famous of which is provided by the Digium platform.

From the architecture perspective, Asterisk serves as a middleware function, connecting the underlying telephony technology and the upper-level telephony applications. Both PBX and IVR (Interactive Voice Response) functionalities are integrated within Asterisk. Using compatible PCI hardware, Asterisk supports traditional telephone lines, including TDM (Time Division Multiplexing), TI/El PRI/PRA&RBS (Robbed Bit Signal) mode, analog telephone line/analog telephone (POTS), ISDN (Integrated Services Digital Network) and BRI (Basic Rate) and PRI (Primary Rate). Also, due to the PCI hardware support feature, Raspberry Pi can be used to implement the Asterisk instance.

By using small single-board computers such as Raspberry Pi, any types of businesses can deploy the cost affordable IP based PBX system. Raspberry Pi is small and light weight device, and runs on low power consumption. Having it for single tasks like allowing for VoIP by running Asterisk PBX is a very easy and economical. However, because this device is smaller and lighter, it can only handle a limited number of tasks. This paper will analyze the boundaries of the Raspberry Pi by packets delays, network jitters, end-to-end network latency, and bandwidth of calls made at varying distances from the Pi to determine how effectively IP-PBX services can be run on such devices.

This paper is organized into specific sections. Section II lists the required hardware and software involved with the experiments performed here. Section III explains the obstacles faced when setting up for the tests in this experiment. Section IV explains the protocols that Asterisk PBX is using to connect devices and forward calls, how the calls are impacted by different network issues, and what topology IP-PBX services utilize to connect devices. Section V shows detailed description for each experiment performed. Section VI goes over different ways the material of this paper can be expanded upon in the future. Finally, Section VII concludes the paper, giving an overview of the results seen thus far.

II. SOFTWARE, HARDWARE & INTERNET REQUIREMENTS

To reproduce or memic the process of this process you will need to satifiy following requirements, and follow the installation steps mentioned in next sections.

A. Hardware Requirements

1) Raspberry Pi 4 from CanaKit (32 GB EVO+, 4GB RAM)
2) Cell Phone (iPhone/Android)
3) Desktop/Laptop (MacBook/Windows/Linux)

B. Software Requirements

1) Raspberry Pi OS with desktop:
   - System: 32-bit
   - Kernel version: 5.15
   - Debian version: 11 (bullseye)
2) Asterisk:
   - Version: 18.11.1
   - Releases: Long Term Support (LTS)
   - Build: From source http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-18-current.tar.gz
3) ZoiPer
   - Version: 5
   - Use: non-commercial
   - For: Mac, Windows, Linux, iOS, Android
4) MicroSIP
   - Version: 3.20.7
   - Use: non-commercial
   - For: Windows
5) Telephone
   - Version: 1.5.2
   - Use: non-commercial
   - For: Mac
6) Wireshark
   - Version: 3.6.3
   - For: Mac, Windows, Linux

C. Internet Requirements

1) eduroam
   - Authority: The University of Alabama
   - Login: <crimson-ID> and <crimson-passphrase>
   - Script: "config_eduroam.sh"

III. OBSTACLES

A. Local Network Issues

An initial problem found when working with PBX software was configuring connections for the device hosting it and for the devices that are connected with it (e.g. smart phones and computers). As IP-PBX uses IP networks for its functionality, a single local network was used for testing the functionality of the Raspberry Pi, which the Pi itself and all machines running VoIP software were connected to, Eduroam. Eduroam is the primary wireless network used within the University of Alabama campus, with access points in each building, allowing devices across buildings to stay on one local area network.

This provided a few benefits to the experiment, most prominently allowing for VoIP calls to be made from greater distances (i.e. from a different building) while simplifying the process of connecting devices to the Raspberry Pi. However, an issue found with this was that Eduroam was not initially allowing the Pis to connect. An additional script was required to sign into the network as students. After the script was ran and the Pis rebooted, they could freely join the network afterward.

```
1   #!/bin/bash
2   ifconfig wlan0 down
3   ifconfig eth0 down
4   killall wpa_supplicant
5   echo "
6
7   allow-hotplug wlan0
8   iface wlan0 inet manual
9           wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
10
11  iface wlan0 inet dhcp
12
13  " >> /etc/network/interfaces
14  read -p 'Username(email-address): ' username
15  read -sp 'Password: ' password
16  echo "
17
18  network={
19          ssid=\"eduroam\"
20          key_mgmt=WPA-EAP
21          eap=PEAP
22          identity=\"$username\"
23          password=\"$password\"
24          phase2=\"auth=MSCHAPV2\"
25  }
26
27  " >> /etc/wpa_supplicant/wpa_supplicant.conf
28
29  echo "Testing conntion with \"eduroam\""
30
31  timeout 20 wpa_supplicant -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf > test_config_eduroam.txt
32
33  test=`grep "authentication completed successfully" test_config_eduroam.txt | wc -l`
34
35  if [ $test -ge 1 ]
36  then
37          echo "\"eduroam\" Successfully connected!!!"
38  else
39          echo "\"eduroam\" Conection Failed :("
40  fi
41
42  ifconfig wlan0 up
43  ifconfig eth0 up
44
45  /etc/init.d/dhcpcd restart
46
47  reboot
```

Fig. 1: Script used to join Eduroam network

Another problem more central to the use of IP-PBX software is the complication of connecting remote devices to the Raspberry Pi. Within Eduroam, VoIP software only required the domain of the Raspberry Pi within the local area network to register on the IP-PBX software the Pi was running. However, if the device was on a different network,

it proved incapable of finding the Pi as easily. There are a few possible explanations for this, such as there being a firewall blocking most traffic outside of Eduroam or NAT changing the IP of the Pi from outside Eduroam. Without the ability to make calls from different networks, making long distance VoIP calls proved to be quite limited, and the available tests for the Raspberry Pi were limited. While this issue may have been resolved with further research, it was ultimately decided that having all devices connect to Eduroam still allowed for enough trials to analyze the Pis functionality in running IP-PBX software.

B. Complexity of IP-PBX

IP-PBX proved to be quite complex in setup and configuration. Asterisk PBX, a mature open-source implementation, was utilized. Initial compilation of Asterisk showed that there were many options on modules to be used, many of which were extraneous. After setup was done, the PBX needed to be further configured to allow for distinct users to be registered, to denote what should be done if a call was made to a certain number, and to dictate what kind of channel to use for SIP. Determining what configuration files needed to be used and changed and in what ways was a major obstacle in beginning the testing of the Raspberry Pi.

Fig. 2: The configuration files for Asterisk PBX. Note that the majority of edits needed to be made to sip.conf and extensions.conf

C. Abundance of Third Party Software

Many pieces of third party software needed to be utilized to conduct these tests, each adding more setup time and complication to the experiments. Asterisk PBX needed to be downloaded onto the Raspberry Pi to allow the Pi to provide IP-PBX services. Zoiper was needed to make VoIP calls and register devices on the Pi for IP-PBX services, with MicroSIP and Telephone needed to make additional calls, as Zoiper limits the number of free users. Wireshark was required to gather more information on the packets sent to and from the Pi while hosting calls to analyze its performance. Additionally, other services, such as SIP.US (used for SIP trunking) and Clumsy (used for network emulation) were considered for use until they were deemed unnecessary.

The number of extraneous services utilized exacerbated the complexity of performing these experiments. enefits to the experiment, most prominently allowing for VoIP calls to be made from greater distances (i.e. from a different building) while simplifying the process of connecting devices to the Raspberry Pi. However, an issue found with this was that Eduroam was not initially allowing the Pis to connect. An additional script was required to sign into the network as students. After the script was ran and the Pis rebooted, they could freely join the network afterward.

IV. ANALYSIS & METHODS

Due to the inherent characteristics of the VOIP (Voice over IP) application, a series of protocols and mechanisms have been designed to ensure the quality of VoIP communication. One task of the experiments is protocol research, including analyzing the collaboration between the protocols included in the process and finding out the principles of the VoIP application design.

According to the "brunch exchange" scheme, the first step is building the link, of which the responsibility is taken by the Session Initiation Protocol (SIP).

A. Basic SIP Analysis

The Session Initiation Protocol (SIP) is an application layer control protocol for establishing, changing and terminating multimedia sessions, where the sessions can be IP telephony, multimedia sessions or multimedia conferences. SIP is the core protocol of many IP-based PBX applications, including Asterisk.
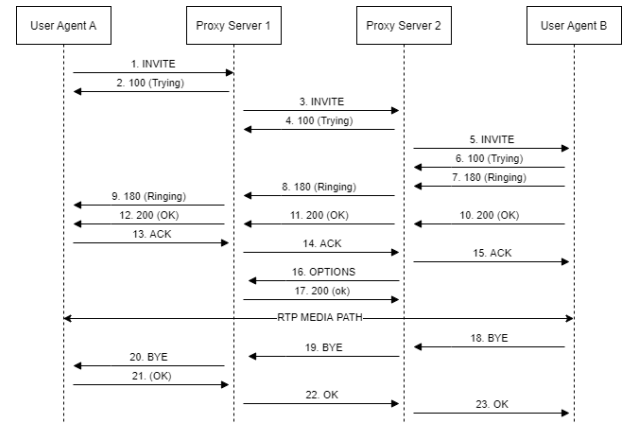

Fig. 3: Session Initialization in SIP model

As shown in Fig.3, the SIP process starts with registration. All SIP terminals as User Agents should register with the registration server to inform their location, session capability, and other information.

Usually, when the SIP terminal (User Agent) is powered on or configured to perform a registration operation, a registration request message (REGISTER) is sent to the registration server, carrying all the information that needs to be registered. After receiving the registration request message, the registration server sends a response message

to the terminal to inform it that the request message has been received. If the registration is successful, it will send a "200 OK" message to the terminal. However, the most common response is a "401 Unauthorized" message because authorization is required in most of the implantations for security purposes.

The SIP protocol adopts the Client-Server pattern, in which the calls are established between User Agents through the proxy server.

Upon the link built by SIP, the data transmission tasks are carried out with RTP/RTCP protocol.

B. RTP

RTP is a real-time protocol for transmitting audio and video streams over IP networks. For telephony applications, the data consists of voice spurts, instead of being a stable data flow. So each RTP packet only contains a small sample of voice messages, whose length is usually 20 to 30 ms.

RTP introduces two fields into its hearder format, the sequence number and the timestamp, generated by the sender devices.

C. RTCP

RTCP is a control protocol accompanied by RTP, providing the report of the RTP stream received by the destination devices. So RTCP packets always head in the reverse direction of RTP streams. After checking the RTCP report, the sender updates the transmission strategy to adapt to the network condition.

D. Jitter and Pakcet loss

VoIP applications are sensitive to jitter. In VoIP calls, the voice stream consists of alternating talk spurts and silent periods. Since silent periods separate the discrete valuable voice messages, the timing and the sequence of the packets' arrival are very important. Voice packets that are out of order may not be decoded correctly. Jitter has a large impact on the sequence of the arriving RTP packets. If the packets arrive at even intervals in the correct order, the jitter is low. Reversely, if the packets arrive in bursts or out of order, the jitter is high.

Compared with jitter, packet loss can be tolerant because it does not change the sequence and time of the packet. So the packet loss that is under a threshold, usually under 3% can be accepted.

In the experiment, jitter has been applied as an important indicator to evaluate the quality of the VoIP calls.

E. Basic Network Topology

To demonstrate the main components of the SIP communication, a simplified scenario is implemented, as shown in Figure Figure 4. Two softphone applications are running on the same computer, and an IP-based PBX application (Asterisk), running on a Raspberry Pi 4, is serving as the PBX server.

To trace and see all types of packets tramissions, author used Wireshark open-souce software. Figure 5
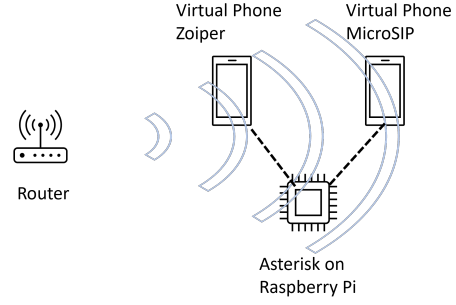
Fig. 4: Simplified Scenario

is a flow sequence of the one single call from Zoiper5 user (IP: 192.168.60.106:63771) to MicroSIP user (IP:192.168.60.106:53493). In this, senario their are actual two connections, first from Zoiper5 user to Asterisk (shown in left-side of Figure 5) and second Asterisk to MicroSIP (shown in right-side of Figure 5).
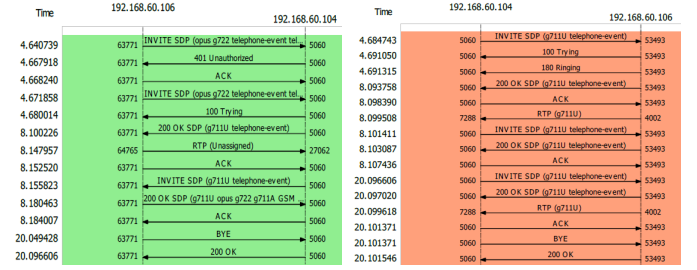

Fig. 5: SIP packet trasmission from Zoiper to Asterisk(left-side), Asterisk to MicroSIP (right-side)

V. EXPERIMENT DESIGN

In this paper, author have consider five different senarios to see how well the communication is between two callers (peers).

In the senario 1 (Figure 6), peer 'A' (IP::10.127.10.203) and peer 'B' (IP::10.127.204.227) are calling from same lab/office. Which let them connect at the same access point. Now, since both callers are at the same location we expect jitter to be very minimal and same send/receive loss at the both peers. In the Figure 7 we can see the sending jitter at both peers have very same pattents. However there jitter vaules heigher then other. And this could happen due to the device that are been used are different.

In the senario 2 (Figure 8), peer 'A' (IP::10.127.10.203) and peer 'B' (IP::10.127.204.227) are calling from different loaction. Peer 'A' is staying at same lab/office and peer 'B' is continuously moving into different locations but within the same floor as peer 'A'. This can let peer 'B' change it's access point within same local area network. Now, here we can expect that peer'B' have more jitter then peer'A', since peer 'B' has to connect and re-connect to different access points. Figure 9 we can see the sending jitter at both peers have
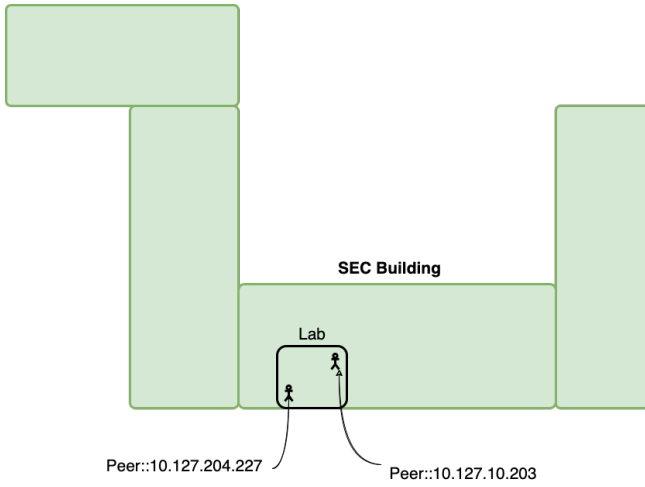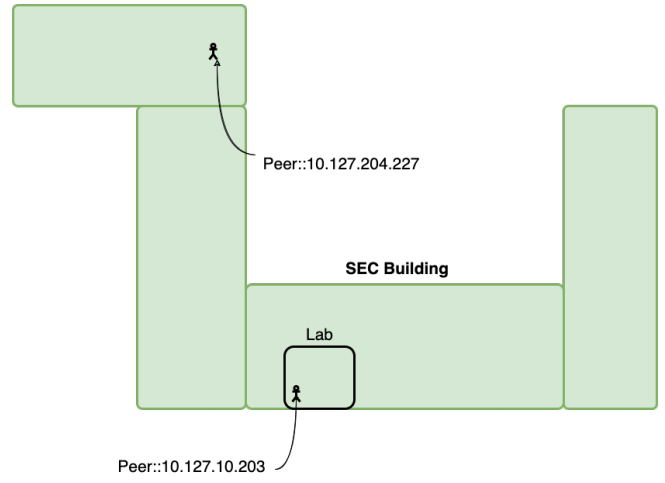
4

Fig. 6: Senario 1: Two peers within same office

Fig. 8: Senario 2: Two peers within same floor

almost same patternt and differenc as Senario 1. However, since peer 'B' is moving one location to another we can see there are two heigher picks then peer 'A', and which could be the connection of new access point and/or re-connection of previous access point.

In the senario 3 (Figure 10), peer 'A' (IP::10.127.10.203) and peer 'B' (IP::10.127.204.227) are calling from different loaction. Peer 'A' is staying at same lab/office and peer 'B' is continuously moving into different locations but within same building and never on same floor. Now, here we can expect that peer'B' have more jitter then peer'A', since peer 'B' has to connect and re-connect to different access points. Figure 11 we can see the sending jitter at both peers have

almost same patternt and differenc as Senario 1. However, since peer 'B' is moving one location to another we can see there are many heigher picks then peer 'A', and which could be the connection of new access point and/or re-connection of previous access point.

In the senario 4 (Figure 12), peer 'A' (IP::10.127.10.203) and peer 'B' (IP::10.127.204.227) are calling from different loaction. Peer 'A' is staying at same lab/office and peer 'B' is continuously moving into different locations, which is outside of the same building. This will allow peer 'B' to be in same access point ranges. Now, here we can expect that peer'B' have more jitter then peer'A', since peer 'B' has to connect and re-connect to different access points. Also at
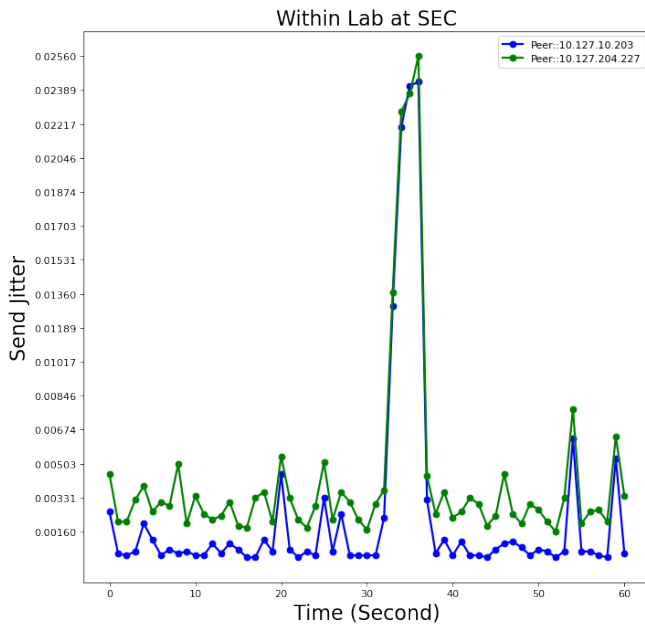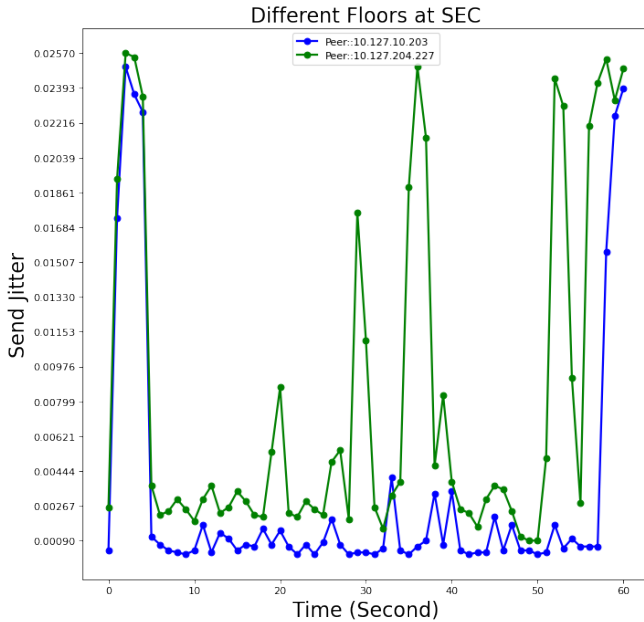

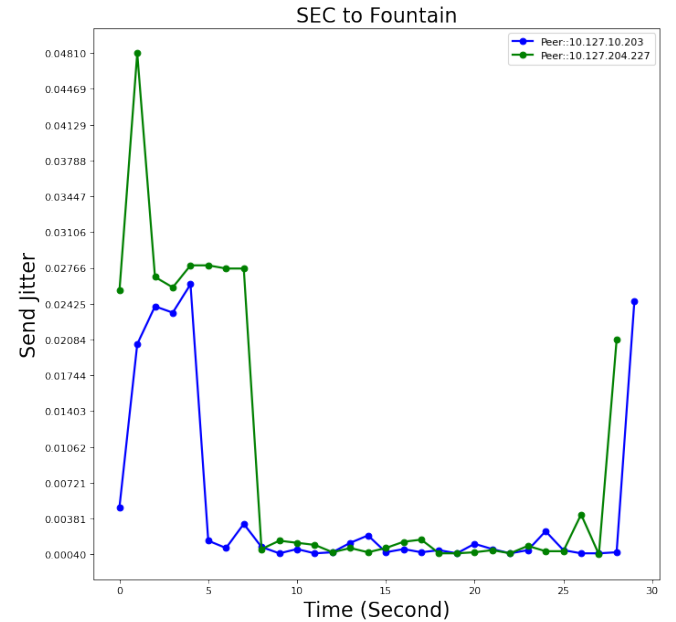Fig. 7: Jitter values for the time spend by callers (for Senario 1)
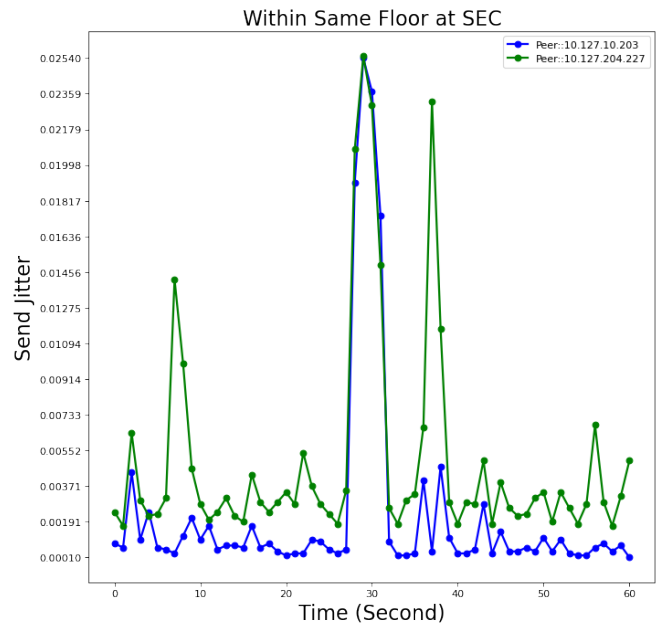
Fig. 9: Jitter values for the time spend by callers (for Senario 2)
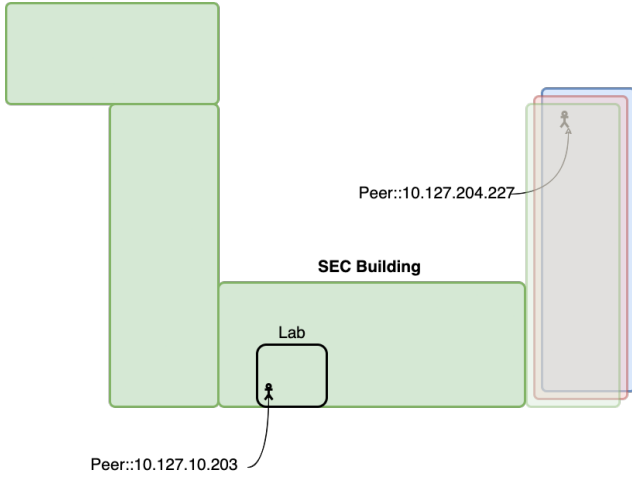
Fig. 10: Senario 3: Two peers at different floors but within building
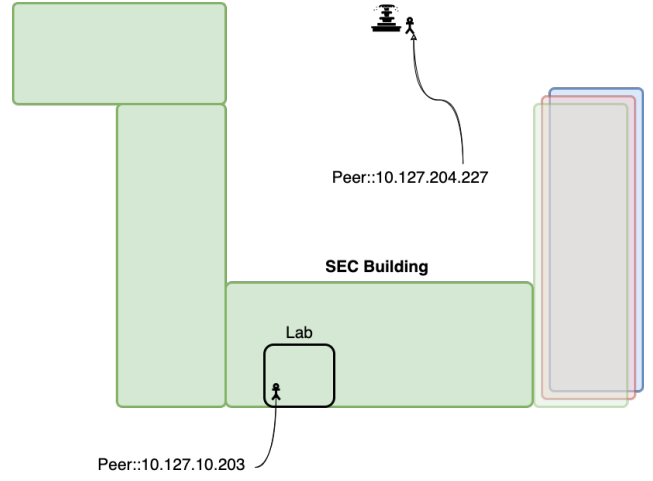
Fig. 12: Senario 4: Two peers are at different locations but close access points

one point peer 'B' totally disconnected from the network, which led to record only few point in this experiment. In Figure 13 we can see the sending jitter at both peers have almost same patternt and differenc as Senario 1. However, since peer 'B' is moving one location to another we can see there are many heigher picks then peer 'A', and which could be the connection of new access point and/or re-connection of previous access point.

In the senario 5 (Figure 14), peer 'A' (IP::10.127.10.203) and peer 'B' (IP::10.127.204.227) are calling from different loaction. Peer 'A' is staying at same lab/office and peer 'B' is continuously moving into different locations but within

same local area network. Now, here we can expect that peer'B' have more jitter then peer'A', since peer 'B' has to connect and re-connect to different access points. In Figure 15 we can see the sending jitter at both peers have almost same patternt and differenc as Senario 1. However, since peer 'B' is moving one location to another we can see there are many heigher picks then peer 'A', and which could be the connection of new access point and/or re-connection of previous access point.

VI. FUTURE WORK

While this may act as a basic foundation for proving the efficacy of the Raspberry Pi as a cheap means to implement

Fig. 11: Jitter values for the time spend by callers (for Senario 3)

Fig. 13: Jitter values for the time spend by callers (for Senario 4)
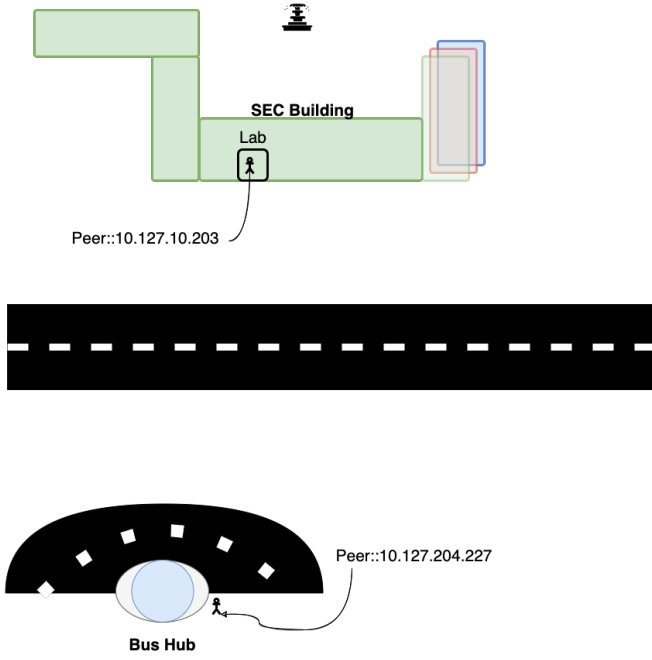
Fig. 14: Senario 5: Two peers are at different locations and different building access point
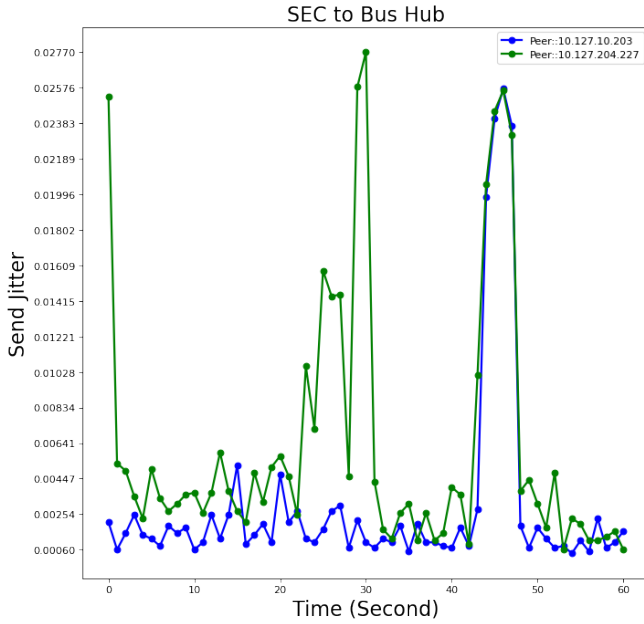
Fig. 15: Jitter values for the time spend by callers (for Senario 5)

IP-PBX software, there are several means of expanding upon this work.

A. Further Testing

More experiments can be performed to determine the limitations that Raspberry Pis can prove to have when running IP-PBX software. A major test would be stressing the Pi by seeing how many calls it can handle in situations where bandwidth is plentiful, with the CPU and RAM of the Pi being the major bottlenecks at that point. This same test could be performed using different models of the Pi to see how the difference in CPU and RAM can affect VoIP calls. Analyzing this data could provide a way to determine the best board to purchase based on how many calls are needed to be handled.

A series of tests could be made while the Raspberry Pi is under a set of different network conditions. A comprehensive method to perform these tests is to conduct them while the Raspberry Pi is running network emulation. Network emulation allows for the Raspberry Pi to deliberately drop and corrupt packets, and to add a delay to them with variability, inducing jitter. Performing such tests will allow for a better understanding of the quality of calls made while the Raspberry Pi is in a network with low bandwidth or heavy traffic conditions. While such analysis could be performed within this paper, it would not be as accurate or precise as with experiments performed with network emulation.

A limitation of data for outside network calls restricts current analysis. Future studies could attempt to have separate Raspberry Pis running PBX software on different networks, while allowing VoIP calls to be made between them. Data could then be collected based on test calls between these IP-PBX instances. This would prove useful in analyzing the effectiveness of the Pi in making outside network calls.

B. Horizontal Scaling

A powerful characteristic of cheap computers such as a Raspberry Pi is their suitability for horizontal scaling. If an institution grows large enough that one Pi cannot handle the all of the institution's VoIP calls, there are two methods of scaling up the memory and processing power available for IP-PBX services. The first is to scale the system vertically; replace the Pi with a single, more powerful computer with more memory, and either sell the Pi or reuse it for some other purpose. The second is to scale the system horizontally; connect more Raspberry Pis with the original and have them share the workload of the VoIP calls. As Raspberry Pis are inexpensive, this allows for a cheaper alternative to buying a more powerful computer. Further works can delve into whether this is a possible technique to implement with IP-PBX software, and whether or not horizontal scaling provides many benefits for IP-PBX services.

VII. CONCLUSION

A single Raspberry Pi with 4GB of RAM is quite capable of running IP-PBX software without any noticeable issues. However, configuring the software to allow for VoIP calls can be quite complicated, and requires a great deal of outside instruction to fully understand. As well, a vital component of using the Raspberry Pi for these purposes is considering the network involved. If the Pi or VoIP devices are in areas

with weak signal strength, jitter can grow rapidly, packet loss can begin to noticeably occur, connections can be dropped abruptly, and delays can become much more noticeable. If these effects occur, either due to weak signal strength or due to other possible issues, call quality can suffer dramatically. Consideration is required to install a usable and useful IP-PBX system for VoIP calls.

REFERENCES

[1] S. Technologies, "Asterisk 18 documentation." [Online]. Available: https://wiki.asterisk.org/wiki/display/AST/Asterisk+18+Documentation
[2] C. Technologies, "Raspberry pi 4 model b - 4gb." [Online]. Available: https://www.cytron.io/p-raspberry-pi-4-model-b-4gb
[3] 3CX, "3cx. pbx. live chat. video calling." Apr 2022. [Online]. Available: https://www.3cx.com/

APPENDIX

- **Installation of Raspberry Pi OS** To install Raspberry Pi OS on "Raspberry Pi 4 from CanaKit", you will need to download **'Raspberry Pi Imager'** from the website. here. The **'Raspberry Pi Imager'** is supported by all OS (Mac/Windows/Linux).

- **Installation of Raspberry Pi Imager on Mac**

  Step 1: Download Raspberry Pi Imager from here
  Step 2: Open/Install "imager_x.x.x.dmg"
  Step 3: Run application "Raspberry Pi Imager"
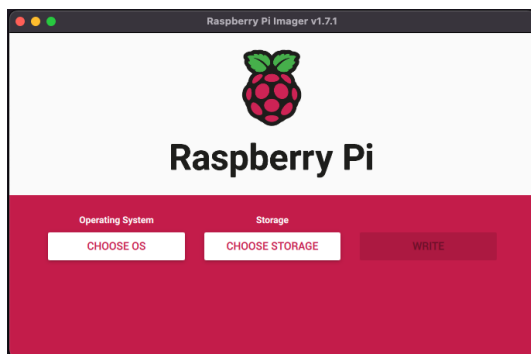
  Step 4: Click on "CHOOSE OS" and Select **"Raspberry Pi OS (32-bit) Desktop"**
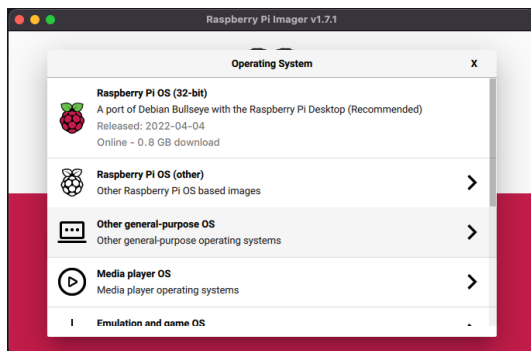
  Step 5: Click on "CHOOSE STORAGE" (**NOTE:** Please make sure you select correct SD card with is in Raspberry Pi 4)
  Step 6: Click on "WRITE"
  Step 7: Wait untill writting is complete.

- **Login to Raspberry Pi OS** At the begining the user name of the OS is **"pi"** and password is **"raspberry"**. (**NOTE:** For this project we did **NOT** change any default user and password.)
- **Accessing Raspberry Pi** There are multiple ways you can access the **"pi"** user.
  - Accessing through SSH
    To access **"pi"** user using SSH connection you need to make sure that your computer and Raspberry Pi are in same Internet connection. In this project it is "eduroam" connection. To setup "eduroam" connection into your newly install Raspberry Pi OS, please follow the instruction from Section A.
    Step 1: Open Terminal in your local computer
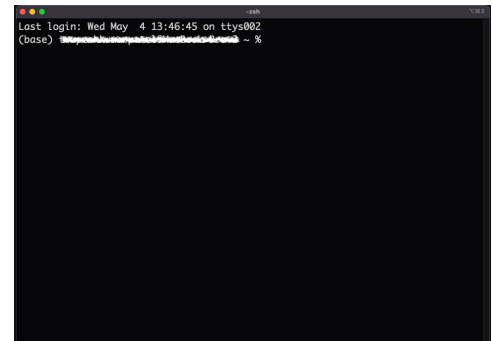
    Step 2: Type command "ssh pi@10.127.227.112" <– here '10.127.227.112' is IP address of Raspberry Pi, in your case might be different.
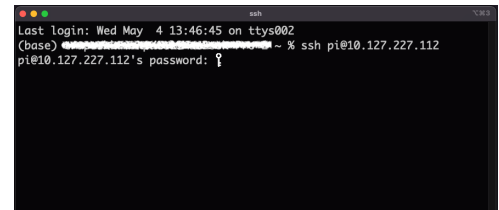
    Step 3: Enter the password "raspberry". Please enter the password that you set if you have, else it is going to be same as the authors.
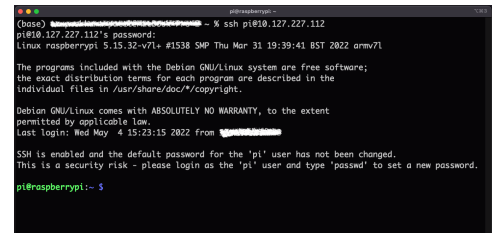
- **Connect to eduroam** To re-run the project, you have to connect your Raspberry Pi with "eduroam" network at The Univeristy of Alabama. Now, if you have freshly install the Raspberry Pi OS, then you need to run the script called **"config_eduroam.sh"**, which can be found here.
- **Configure Raspberry Pi for eduroam**

  Step 1: Login to the Raspberry Pi as user "pi"

Step 2: Type command "`git clone https://github.com/TrupeshKumarPatel/IoT_RaspberryPi.git`"

Step 3: Type command "`sudo su`" ⚠**NOTE** This command to get sudo access to Raspberry Pi ⚠

Step 4: Type command "`chmod +x IoT_RaspberryPi/source_code/eduroam_config/config_eduroam.sh`"

Step 5: Type command "`./IoT_RaspberryPi/source_code/eduroam_config/config_eduroam.sh`"

Step 6: Enter your crimson email address

Step 7: Enter your crimson password

Step 8: Now wait for your Raspberry Pi to restart

- **Installation of Asterisk** To install Asterisk same as this project authors, please follow below instructions: (**Note:** Here author installing Asterisk from source code. You can also follow the installation video from InnovationAsterisk)

- **Asterisk Installation from the Source Code**

    Step 1: Type command "`wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-18-current.tar.gz`"

    Step 2: Type command "`sudo apt-get update`"
    Step 3: Type command "`sudo apt-get upgrade`"
    Step 4: Type command "`sudo apt-get install ntp`"
    Step 5: Type command "`sudo apt-get install speex speex* libspeex-dev libspeexdsp-dev`"

Step 6: Type command "`sudo apt-get install libspeex-dev libspeexdsp-dev speex speex-doc`"

Step 7: Type command "`sudo apt-get install xmlstarlet libopus-dev libopusfile-dev`" <– (**NOTE:** This optional if you don't want browser-base telephony, here author did not do this)

Step 8: Type command "`tar -xvf asterisk-18-current.tar.gz`" <– This command will extract all the source file into folder called "asterisk-18.x.x" (**Note:** here 'x' could be any number based of current release/download)

Step 9: Type command "`cd asterisk-18.x.x`" <– (**Note:** here 'x' could be any number based of current release/download)

Step 10: Type command "`sudo contrib/scripts/install_prereq install`" <– This will check all the prerequisites of Asterisk. This command will ask few promps to set configurations

  1) Telephone code : Type "1" for US

Step 11: Type command "`sudo ./configure -libdir=/usr/lib -with-pjproject-bundled`"

Step 12: Type command "`sudo make menuselect`" <– Please watch this video) from InnovateAsterisk for more details.

Step 13: Type command "`sudo make`"

Step 14: Type command "`sudo make install`"

Step 15: Type command "`sudo make samples`" <– This will create all default Asterisk configuration files ⚠**NOTE: Step 16-17** is only required if you haven't downloaded "IoT_RaspberryPi" GitHub repository from here ⚠

Step 16: ⚠Type command "`cd`"

Step 17: ⚠Type command "`git clone https://github.com/TrupeshKumarPatel/IoT_RaspberryPi.git`"

Step 18: Types command "`sudo cp -R /IoT_RaspberryPi/source_code/asterisk_config/* /etc/asterisk/`"

- **Command of Asterisk**

Now, after following instruction from Section A, you should be able to run asterisk commands.

- To check the status of Asterisk
  Type command "`sudo service asterisk status`"
- To start the Asterisk
  Type command "`sudo service asterisk start`"
- To restart the Asterisk

Type command "sudo service asterisk restart"
- To stop the Asterisk
  Type command "sudo service asterisk stop"
- To access CLI of the Asterisk
  Type command "sudo asterisk -r"

- To see all peers
  Type command "sip show peers"

- To see all users
  Type command "sip show users"

There are many Asterisk commands are available in their document offical website here and external soure here, use it as you need.

• **Helper Functions of Asterisk** Author has created helper function to ease Asterisk configuration. Currently only one function is created, but in future their might be more.

- To add new user/peer use the bash script called "**add_user.sh**", this script can be found here.

  Step 1: Type command "cd IoT_RaspberryPi"
  Step 2: Type command "sudo su"

  Step 3: Type command "bash source_code/asterisk_helper_scripts/add_user.sh"
  Step 4: Enter extention number(must be 4 digits) <– for now it only accepts 4 digits, you can change the bash script if you want.
  Step 5: Enter caller ID
  Step 6: Enter Password