\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Setting up the CUCM on an ESXi Server

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Author*:

Gabriel Rosas



Purpose

When I want to talk to an English friend remotely from Seattle, I can easily call them and receive near instantaneous audio transmission, if I am connected to the internet. *Voice over Internet Protocol* (VoIP) allows voice to transmit over the internet instead of the old-fashioned copper phone lines. With the internet expanding ever so vastly into the average person’s life, especially during the pandemic, it only seemed natural to take a gander behind the scenes of VoIP.

Background Information

Why do we use VoIP?

Originally, copper landlines were used as the media transmitting incoming and outgoing calls. Since then, advancements have been made in networking allowing voice to be sent as packets over the internet. Some might relate VoIP to on-premises hardware – the stereotypic, bulky physical phones used in offices for marketing – though modern software, such as Skype or Discord, use VoIP to manage calls.



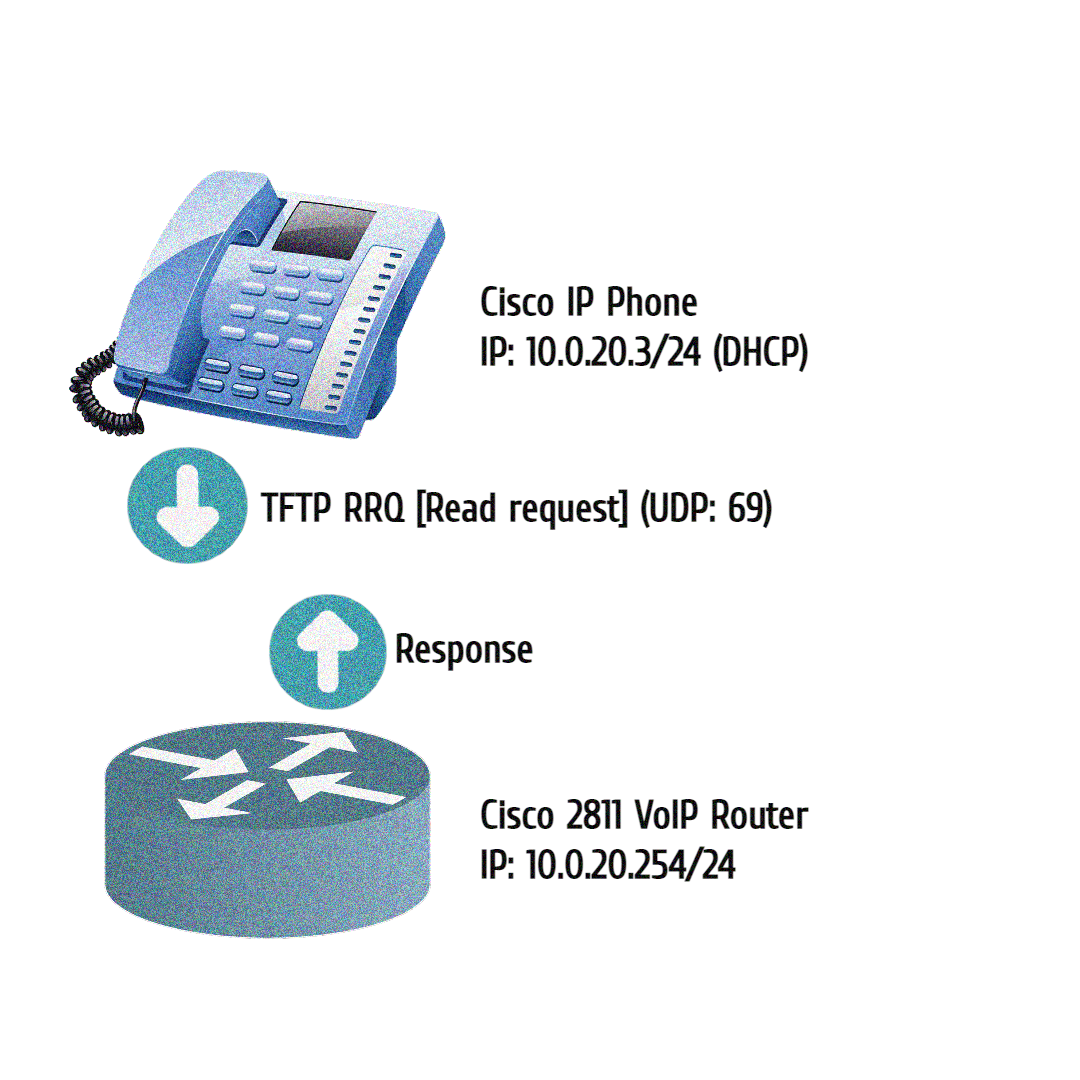
People opt for VoIP because a phone call only requires a connection to the internet. Other infrastructure, such as extra copper phone lines, is not necessary. Why go through the trouble of installing a landline-like phone when VoIP is so much simpler, capable of high-definition phone calls, and only requires 100 kbps of bandwidth?

What is the Cisco Unified Communications Manager?

The Cisco Unified Communications Manager (CUCM) is an IP-based communications system that allows users to contact their coworkers, customers, or friends through audio or video regardless of physical location. All calls begin, are maintained, and end with the CUCM. While the CUCM can manage calls, it also hosts files for IP phones on the network for when they initially boot up. There are two ways of configuring VoIP with the CUCM: with the full CUCM that runs on a server, or the CUCM Express (CUCME) which runs on certain Cisco routers. The full CUCM has many services, such as call manager, dhcp service, tftp, service analyzers, and others, granting all sorts of unique configurations for different purposes. The CUCME is ported down to be runnable in a router. In this lab, I host a call between two IP phones using the CUCM running on an ESXi server.

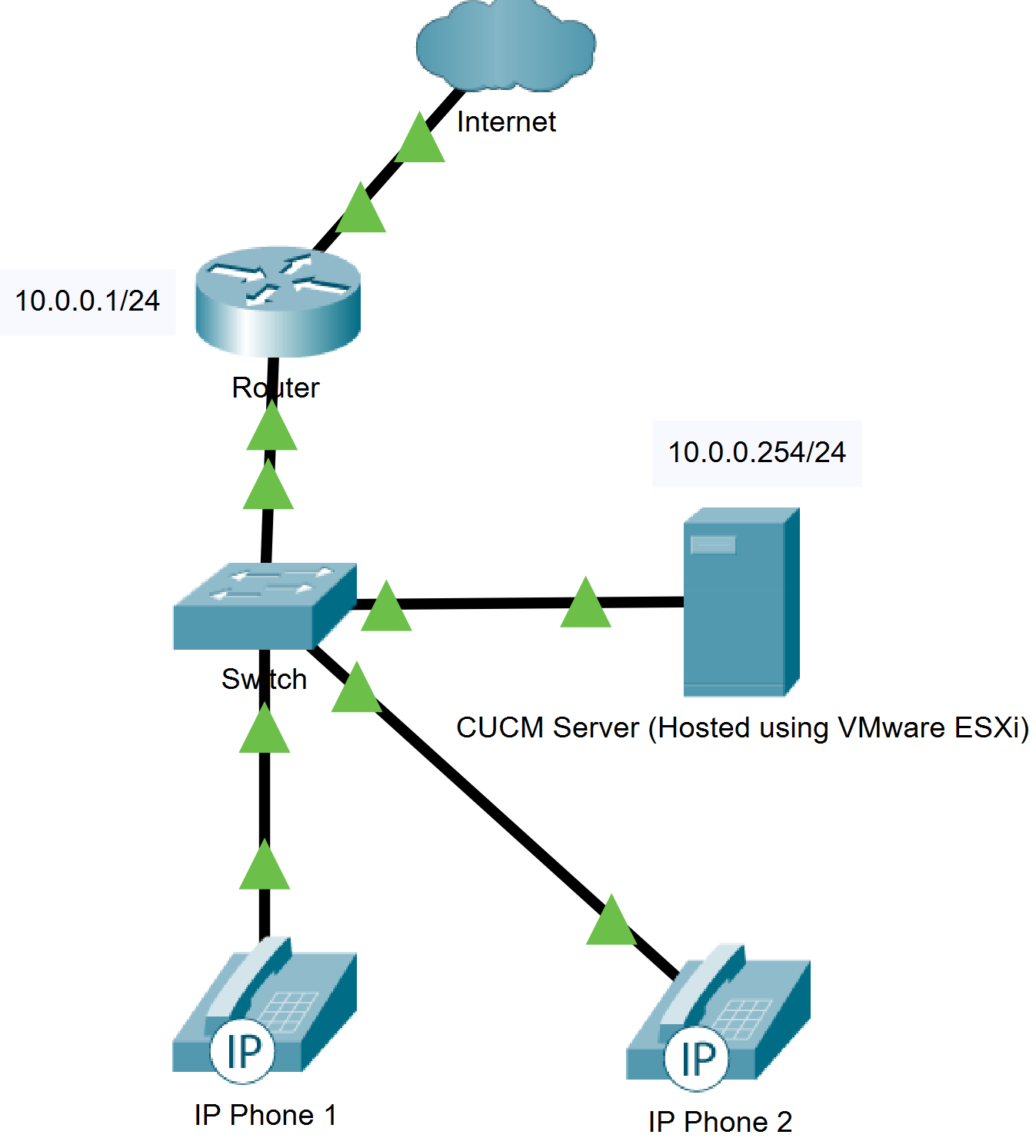
Trivial File Transfer Protocol

Trivial File Transfer Protocol (TFTP) allows a client to obtain a file from or upload a file to a remote host. Unlike File Transfer Protocol (FTP), which uses TCP ports 20 and 21, TFTP uses UDP port 69 to establish network connections. While some may question the application of UDP (the faster but less secure method of data communication) the reasoning becomes clearer if one considers the *triviality* of TFTP. TFTP is much simpler than FTP. Files hosted by a TFTP server are generally quite small; you are more likely to find TFTP used on a LAN while FTP is implemented on a WAN; network engineers use TFTP to distribute software across a LAN because packet loss is unlikely.



Why do we care about TFTP? Cisco SCCP IP phones attempt to download software on boot that we need to host. This process is known as bootstrapping. Bootstrapping allows machines that are new to the network, or machines that have lost everything, to download software files which gets them up and running. We can use the TFTP service provided by the CUCM to host these files for IP Phones.

Network Diagram



Process

It took a while for me to breach the surface of this project since I was not clear on what I needed to do. Much of the struggle in the beginning could have been avoided if I asked for clarifications on certain things, such as a topology. Eventually I did. Having worked with the CUCME, the express version on a Cisco 2811 service router, I had insight on configurations that I likely needed to get working. For example, dial numbers and phone profiles. However, before I could do that, I needed to install the CUCM.

Installing the CUCM in VMware ESXi

The CUCM is an ISO, an operating system image, so you either need a physical device to boot it on or a Virtual Machine (VM) service such as ESXi. I chose to run the CUCM on ESXi to familiarize myself with the service (and conveniently we already had a server running it). I downloaded the ISO on the local ESXi server using TFTP and hastily created a VM.

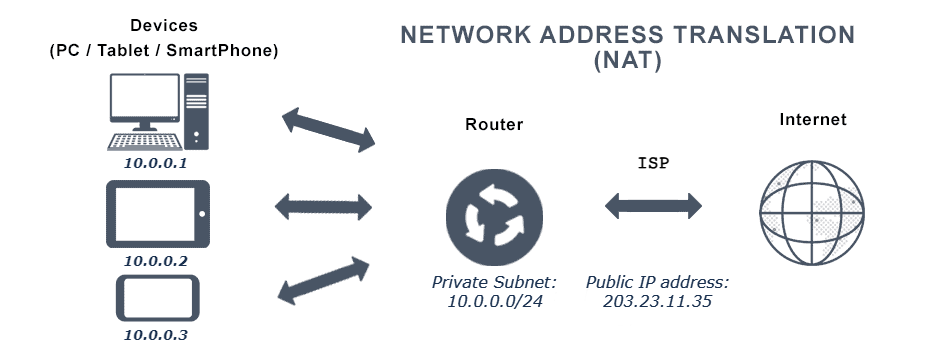
When creating a VM, there are several configurable options, such as type, RAM, and storage, but I left them default. The CUCM VM booted, shortly running into a “critical error”: *there are no deployments provided by this software that support this hardware. Installation will now halt.* The solution was not too hard to find, I just needed to allocate more RAM and storage.

Initial Setup

With the CUCM as a runnable VM, my next step was to complete the initial setup: configuring settings such as addressing and defining administrative accounts. I left most settings as default. The trouble came a couple of pages later, when I was forced to define a *Network Time* (NTP) server.

I decided to configure my router as an NTP server, so I did a little research and found a command that should do just that: *ntp master*. Theoretically, *ntp master* is the only command needed for a router to forward its local clock, but the CUCM was not cooperating. Perhaps it was that my router had a high “stratum” number, and the CUCM would only accept responses from a more credible source. In NTP, “stratums” are a form of hierarchy amongst NTP servers where lower stratums send more accurate versions of time. For example, stratum one servers typically have 10 microseconds of accuracy. Stratum two servers get their time directly from stratum one servers but have more delay. Stratum three get theirs from stratum two, and such. Higher stratum servers are more common but have more delay.

My solution was to forget about creating a local NTP server and instead use one from *pool.ntp.org*, a free timeserver host with many pools across the world. However, the CUCM was in an internal part of my network, unable to access the NTP servers hosted on the internet. The fix here was simple: *NAT*. *Network Address Translation* (NAT) is a process that enables a unique IP address to represent an entire group of devices. Typically, this IP address is “Public”, administered by one’s ISP.



In this example, the public IP, *203.23.11.35*, is representing the *10.0.0.0/24* subnet. There can be multiple networks that use the same private subnet – what matters is that the public IP is unique. With so many devices nowadays, the total IPv4 address pool is running out. NAT helps conserve these addresses by translating multiple private IPs through one public IP. This is known as dynamic NAT.

Using my experience from previous labs, I was able to set up NAT relatively quickly with no issues. Once implemented, the CUCM could access the internet and request time from legitimate NTP servers. Now I could move on and finalize the initial setup. I left the CUCM to finalize over the weekend, as it takes a couple of hours to prepare after the initial setup.

Configuring the CUCM

Much like other labs, I compiled steps from tutorials online on how to configure what I want. In this case, it was registering two IP phones and starting a call. There was a tutorial online that provided a lot of steps, so I wanted to follow that through. Many of these steps were unnecessary for my purpose, but were good practice, so I decided to try and implement all of them. Here are the steps I compiled:

* Enter the CUCM Web Interface
* Under "Navigation" select *Cisco Unified Serviceability*
  + Under the "Tools" tab, select "Service Activation"
  + Activate all Services but DHCP (will activate this later)
  + [Save]
* Under "Navigation" select *Cisco Unified CM Administration*
  + Under the "System" tab, select "Server"
  + Click "Find" (Should find a Server) and select that one
    - Change the Host name to the CUCM IP address
    - [Save]
* Under the "System" tab, select *Enterprise Parameters*

(Note that you can use "CTRL+F" to search for these parameters)

* + Change "Auto Registration Phone Protocol" to "SIP"
  + Find Phone URL Parameters and Secure Phone URL Parameters (There should be a bunch of URLs)
  + Change the Domain name of each URL to the IP of your CUCM
    - (For example, I would change the URL "http://The-real-CUCM.lol:8080/ccmcip/authenticate.jsp" to "http://10.0.0.254:8080/ccmcip/authenticate.jsp")
  + [Save]
  + [Reset]
* Under the "System" tab, select *Service Parameters*
  + Select your Server
  + Select the service "Cisco CallManager"
  + Change the delay of the "T302 Timer" to 3 seconds
    - (Note the unit is milliseconds)
  + Change the delay of the "H225 T302 Timer" to 3 seconds
    - (Note the unit is milliseconds)
  + Change each "Stop Routing" parameter to "False"
  + Change the "Default Interregion Max Audio Bit Rate" to “16 kbs (iLBC, G.728)”
  + Change "Automated Alternate Routing" to "True"
  + Change "Enable Enterprise Feature Access" to "True"
  + Change "Enable Mobile Voice Access" to "True"
  + Change "Matching Caller ID with Remote Destination" to "Partial Match"
  + Change "Number of Digits for Caller ID Partial Match" to "7"
  + [Save]
* Under the "System" tab, select *Phone NTP Reference*
  + Click "Add New"
  + Define an NTP server
  + Set the "Mode" to "Unicast"
* Under the "System" tab, select *Date/Time Group*
  + Click "Find"
    - (A default one should appear, likely named "CMLocal")
  + Configure the Name, Time Zone, and Date Formats
  + Select "Add Phone NTP References"
  + Add the NTP reference previously made
  + [Save]
  + Click [Copy] and repeat steps above for each Date/Time Group desired
* Under the "System" tab, select *Region Information/Region*
  + Click "Find" and select "Default"
  + Create new regions to match the ones in your Date/Time Group
* Under the "System" tab, select *Location Info/Location*
  + Click "Find"
  + Select create new Locations for each Date/Time Group made
* Under the "Call Routing" tab, select *Class of Control/Partition*
  + Click "Add New"
  + Separating each partition with a new line, enter the partitions: INTERNAL, PSTN, STRIP, CM-ANI-IN, CM-DNIS-OUT, and a partition for each Date/Time Group
  + [Save]
* Under the "Call Routing" tab, select *Class of Control/Calling Search Space*
  + Click "Add New"
  + Name: "INTERNAL", selected partitions: "INTERNAL"
  + [Save]
  + [Copy]
  + Name: "CM", selected partitions: "INTERNAL", "CMLocal"
  + [Save]
  + [Copy]
  + For each region, create partitions with "INTERNAL" and \*Region
  + [Copy]
  + Name: "PSTN", selected partitions: "PSTN"
  + [Save]
  + [Copy]
  + Name: "CM-DNIS-OUT", selected partitions: "CM-DNIS-OUT"
  + [Save]
  + [Copy]
  + Name: "CM-ANI-IN", selected partitions: "CM-ANI-IN"
  + [Save]
* Under the "Media Resources" tab, select *Media Resource Group*
  + Click "Add New"
  + Name: "CM", selected media resources: \*all of the available
  + [Save]
  + [Copy]
  + For each region, create a media resource group with all available media resources
* Under the "Media Resources" tab, select *Media Resource Group List*
  + Click "Add New"
  + For each media resource group, make a list
  + Example: Name: "CM", selected Media Resource Groups: "CM"
  + Under the "System" tab, select "Device Pool"
  + Click "Find" and select the default
  + Change "Device Pool Name" to "CM"
  + Change "Media Resource Group List" to "CM"
  + Change "Location" to "CMLocal"
  + [Save]
  + [Reset]
  + [Copy]
  + Repeat steps above for each Region previously created
* Under the "User Management" tab, select *End User*
  + Click "Add New"
  + Define a User ID
  + Define a Password
  + Define a PIN
  + Define Names (Last, First, Display names)
  + [Save]
  + Click "Add to Access Control Group" (Scroll down)
  + Click "Find" and select "Standard CCM End Users", "Standard CTI Enabled"
  + Click "Add Selected"
  + [Save]
  + Follow the steps above to create another end user
* Under the "Device" tab, select *Phone*
  + Click "Add New"
  + Enter the type of your IP phone
  + Enter "SIP" for the device protocol
  + Change "MAC Address" to match your IP Phone
  + Change "Device Pool" to "CM"
  + Change "Phone Button Template" to match your IP Phone
  + Change "Calling Search Space" to "CM"
  + Change "Owner User ID" to match a profile you created previously
  + Change "Device Security Profile" to match your IP Phone with SIP
  + Change "SIP Profile" to "Standard SIP Profile"
  + [Save]
  + A new section called "Association" should appear. Click "Line [1] - Add a new DN"
    - Define a Directory Number
    - Change "Route Partition" to "INTERNAL"
    - Add a Description, Alerting Name, ASCII Alerting Name
    - Add a Display (Caller ID), ASCII Display (Caller ID), External Phone Number Mask
    - Change "Busy Trigger" to "1"
    - Click "Associate End Users"
    - Click "Find"
    - Add an end user
    - [Save]
  + Follow the steps above to create a second phone

Registration Rejected

After configuring the CUCM using the steps above, I was ready to launch the IP phones. Nothing happened. The IP phones got addresses but failed beyond that step. In my previous lab with the CUCME, I set up a DHCP pool that points the IP phones to a TFTP server (in this case the CUCM) for their bootstrap files. I forgot to configure that option, *option 150*, on my new DHCP server (the router). I quickly added in this line: *option 150 ip 10.0.0.254*, pointing the IP phones to the CUCM. Now that the IP phones were directed to the CUCM, they should be able to access their files. But they still could not, which became a problem that would haunt me for the next couple of weeks: trying to solve the error “*Registration Rejected*”. How very descriptive.

My first thought was to add a DNS pointer in the DHCP pool so the IP phones could use DNS if they needed it. I do not think they did, so I began researching other solutions. The Cisco IP phone I used, the 7940 model, has a small display that provides information on the booting phase. I had not noticed before, but another message would blink briefly after the *Registration Rejected* error: “*Version Error*”. This prompted me to two conclusions: the CUCM version was outdated, or the IP phones were outdated. I really did not want to reinstall the CUCM, so I opted to try and fix the phones. I reset both to factory default in hope that they would download their new model files from the CUCM, though it had no impact.

At this point a nagging dread was creeping up in me – did I compile the steps I wrong? There was always a thought in the back of my mind to start over and use the very minimal configs to get IP phones communicating. That, however, would undermine the good practices I had implemented, such as the various groups. I did not want to restart without these configurations, but I was coming up with less and less other solutions.

I tried enabling *auto-registration*, so that the IP phones might still register even if I configured their profiles incorrectly. I tinkered with TFTP service parameters in the CUCM, phone profiles, even different physical IP phone models. All methods ended with the same “*Registration Rejected*” message. I tried changing the default *auto-registration* profile to SCCP, the simpler protocol for communicating with the CUCM. Eventually I decided to take a break and work on a different project for a distraction, then come back to the CUCM after a little time away.

A weekend later, I accepted that I needed to start from the beginning. I reverted to the initial state of the CUCM before any configurations and started again. Though I skipped a lot of the good practices, I managed to get the IP phones communicating with basic configurations, using a different resource than the first time. Having got the IP phones connecting, I went back to root out why they failed the first time. I believe the issue was in using SIP: the working configurations used SCCP. I tried changing the protocol in my troubleshooting, but not every setting relating to it. Perhaps if I enabled more settings related to SCCP the phones would have received their files.

Lab Commands

|  |  |
| --- | --- |
| **Command** | A statement necessary for a configuration to work, denoted in bold |
| **[*Argument*]** | An argument necessary for a command to function, denoted in bold italics. |
| *Optional-Statement*  *<Optional Argument>* | An optional argument or statement, not necessary for a command to function, denoted in italics |

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*initial ip*]** <*final ip*>

* Set an IP or range of IPs to exclude from the pool

*If the network administrator so chooses to exclude a range of IP addresses, the range would be from the* Initial IP *to the* End IP*, inclusive. The* End IP *argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.*

Router(config)# **ip dhcp pool [*pool name*]**

* Creates a pool for distributing routing information

*Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. A DHCP pool is used to define the range of IP addresses that the server will divvy out to clients.*

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

* Configures a pool that distributes the specified subnet

Router(dhcp-config)# **default-router [*ip*]**

* Sends the specified default gateway to clients

Router(dhcp-config)# **option 150 ip [*call server address*]**

* Advertises the address of the TFTP server where IP phones will request files

*Cisco IP phones download their files from a TFTP server. Once an IP phone receives DHCP information, gaining an IP, it will then attempt to download the files.*

// Dynamic NAT Configuration

Router(config-if)# **ip nat [*inside/outside*]**

* Configure an interface to be internal or external

Inside interfaces are translated through the outside interface.

Router(config)# **access-list [*#*] permit [*network address*] [*wildcard mask*]**

* Create an access list that permits a subnet

The subnet specified here should be on an *inside* interface. NAT will translate the subnet out the *outside* interface. Keep note of the [*#*] defined, for that will be used in a later command.

Router(config)# **ip nat inside source list [*#*] interface [*id*] overload**

* Enable the translation of an access list through an outside interface

The source list [*#*] should be an access list created earlier; the *interface id* should be the outside interface.

Configurations

Router title

Router# show running-config

service timestamps debug datetime msec

service timestamps log datetime msec

platform qfp utilization monitor load 80

platform punt-keepalive disable-kernel-core

hostname Router

boot-start-marker

boot-end-marker

vrf definition Mgmt-intf

address-family ipv4

exit-address-family

address-family ipv6

exit-address-family

no aaa new-model

ip dhcp excluded-address 10.0.0.1

ip dhcp excluded-address 10.0.0.254

ip dhcp pool PHONES

network 10.0.0.0 255.255.255.0

default-router 10.0.0.1

option 150 ip 10.0.0.254

dns-server 8.8.8.8

login on-success log

subscriber templating

multilink bundle-name authenticated

crypto pki trustpoint TP-self-signed-187689846

enrollment selfsigned

subject-name cn=IOS-Self-Signed-Certificate-187689846

revocation-check none

rsakeypair TP-self-signed-187689846

license udi pid ISR4321/K9 sn FLM2407014H

no license smart enable

diagnostic bootup level minimal

spanning-tree extend system-id

redundancy

mode none

interface GigabitEthernet0/0/0

ip address 10.0.0.1 255.255.255.0

no shut

ip nat inside

negotiation auto

interface GigabitEthernet0/0/1

ip address dhcp

no shut

ip nat outside

negotiation auto

interface GigabitEthernet0

vrf forwarding Mgmt-intf

no ip address

shutdown

negotiation auto

ip forward-protocol nd

ip http server

ip http authentication local

ip http secure-server

ip tftp source-interface GigabitEthernet0

ip nat inside source list 1 interface GigabitEthernet0/0/1 overload

access-list 1 permit 10.0.0.0 0.0.0.255

control-plane

line con 0

transport input none

stopbits 1

line aux 0

stopbits 1

line vty 0 4

login

end

Conclusion

I learnt a lot about working with ESXi and the CUCM, along with the major concepts of VoIP. In the future I would like to revisit this lab and try to implement more features, such as ringtones, video, and paging. Though I ran into a lot of problems, I’m glad I persevered to reach a working solution. As a wise entity once said, *it’s about the journey, not the destination*.