**Diagram

Description automatically generated**

**Basic home gateway services: DHCP, DNS, NAT**

[Fraida Fund](https://witestlab.poly.edu/blog/author/ffund/)  
30 MARCH 2017 on [education](https://witestlab.poly.edu/blog/tag/education/), [dhcp](https://witestlab.poly.edu/blog/tag/dhcp/), [dns](https://witestlab.poly.edu/blog/tag/dns/), [nat](https://witestlab.poly.edu/blog/tag/nat/), [routing](https://witestlab.poly.edu/blog/tag/routing/)

Most Internet-connected homes use a home network gateway to connect a local area network (LAN) in the home, to a wide area network (WAN) such as the Internet. In this experiment, we will look at some of the services that are always or often included on these gateways:

* routing between the LAN and WAN,
* DHCP, to provide hosts in the home network with IP addresses,
* DNS, to respond to name resolution queries from hosts in the home network,
* NAT (Network Address Translation), to map one public IPv4 address to internal (private) IP addresses assigned to hosts on the home network.

It should take about 120-180 minutes to run this experiment.

This experiment runs on either the GENI or CloudLab testbed! Whenever there are testbed-specific instructions, follow the ones for the testbed *you* are using.

**GENI-specific instructions: Prerequisites**

To reproduce this experiment on GENI, you will need an account on the [GENI Portal](http://groups.geni.net/geni/wiki/SignMeUp), and you will need to have [joined a project](http://groups.geni.net/geni/wiki/JoinAProject). You should have already [uploaded your SSH keys to the portal and know how to log in to a node with those keys](http://groups.geni.net/geni/wiki/HowTo/LoginToNodes).

**Cloudlab-specific instructions: Prerequisites**

To reproduce this experiment on Cloudlab, you will need an account on [Cloudlab](https://cloudlab.us/), you will need to have [joined a project](https://docs.cloudlab.us/users.html#%28part._join-project%29), and you will need to have [set up SSH access](https://docs.cloudlab.us/users.html#%28part._ssh-access%29). You may want to try [Hello, CloudLab](https://teaching-on-testbeds.github.io/hello-cloudlab/) if you have not yet completed those steps.

* Skip to [Results](https://witestlab.poly.edu/blog/basic-home-gateway-services-dhcp-dns-nat/#results)
* Skip to [Run my experiment](https://witestlab.poly.edu/blog/basic-home-gateway-services-dhcp-dns-nat/#runmyexperiment)

**Background**

A home or residential gateway is a device that sits between a local area network (LAN) in a home, and a wide area network (WAN) such as the Internet.

Diagram

Description automatically generated

At a minimum, it routes traffic between the LAN and the WAN. In many cases, it provides *additional* services that are useful for a home LAN, including three that are the subject of this experiment:

* DHCP, to provide hosts in the home network with IP addresses,
* DNS, to respond to name resolution queries from hosts in the home network,
* NAT (Network Address Translation), to map one public IPv4 address to internal (private) IP addresses assigned to hosts on the home network.

**DHCP**

The Dynamic Host Configuration Protocol (DHCP) is a protocol for dynamically distributing IP addresses to hosts in a network. This eliminates the need for a user or network administrator to configure IP address settings manually on every host. It also allows a "pool" of IP addresses to be shared among many hosts; if a host leaves the network, the IP address it was using can be reassigned to someone else just entering the network.

The process for getting an IP address with DHCP involves four phases known as DORA - discovery, offer, request, and acknowledgement - as shown in the following figure:

Diagram

Description automatically generated

1. First, the client broadcasts a DHCP DISCOVER message.
2. Local DHCP server(s) receive this message, and can respond with a DHCP OFFER that is unicast to the client. This OFFER message "offers" an available IP address to the client. It may also include other "options", such as the address of a gateway or nameserver that the client should use. The client may receive multiple offers, if there are multiple DHCP servers on the LAN.
3. The client responds to *one* offer with a REQUEST for the IP address that was offered to the client. This REQUEST message is broadcast on the LAN, so if another DHCP server also sent an OFFER, it will receive this REQUEST message and consider it as notification that the client has declined that server's offer.
4. The DHCP server whose OFFER was followed up with a REQUEST will acknowledge the client with an ACK.

After these four stages, the client is configured with an IP address. As long as it is still using that IP address, it will renew it at regular intervals (when the "lease" expires), using a REQUEST+ACK. If the client leaves the network, it can send a RELEASE message to alert the DHCP server that it is relinquishing the address, and it may be assigned to someone else.

A typical home gateway will come with a DHCP server built-in, and allow you to select the range of IP addresses it distributes, as well as other DHCP options, e.g.:

Graphical user interface, application

Description automatically generated

**DNS**

For web browsing and similar use cases, hosts on the Internet are referred to by human-readable names, rather than by their IP addresses. The Domain Name System (DNS) is a system that stores records mapping those names to the corresponding IP address, and allows those records to be queried (name resolution).

Many gateways also act as a DNS server, i.e. they can initiate and sequence the queries that ultimately lead to the resolution of a hostname into an IP address.

A DNS server may follow an iterative query procedure, in which it queries a chain of DNS servers. Each server in the chain refers server that initiates the query to the next server in the chain, until it reaches one that can fully resolve the requested name.

For example, consider a DNS server that wants to resolve the name

witestlab.poly.edu.

(Note: all "full" DNS domain names have a dot at the end, although DNS resolvers will still work if you omit the dot. [This answer](http://webmasters.stackexchange.com/a/73946) explains why.)

Suppose the DNS server does not know the IP address associated with this name, and also does not know the address of a DNS server that *does* know the address associated with this name. The following figure describes the iterative procedure it will follow:

A picture containing chart

Description automatically generated

* It will start to query other name servers iteratively, starting with the name servers responsible for the rightmost part of the name, the .. (All DNS servers know the address of some name servers responsible for ., called *root* domain servers.)
* A root server won't know the address of witestlab.poly.edu., but it will respond with a list of name servers that are responsible for .edu., which is a *top level* domain.
* A .edu. name server also won't know the address of witestlab.poly.edu., but it will respond with a list of name servers that are responsible for poly.edu. (a *second level* domain).
* Finally, a name server that is responsible for poly.edu. is likely to know the address of witestlab.poly.edu., a *subdomain* and will respond with an address record.

Thus, any DNS server that knows the address of some root servers can eventually find out the IP address associated with any name.

The hierarchy of the name servers in this example (and examples of other name servers at similar levels) is shown in the following figure:

Graphical user interface

Description automatically generated with medium confidence

**NAT**

Another common feature of residential gateways is NAT, Network Address Translation. This is a method of method of remapping one IP address space into another by modifying source and destination IP addresses in packet headers as they pass through the gateway.

Publicly routable IPv4 addresses, which are scarce, are not generally assigned to every host in a home network. Instead, the Internet Service Provider will provide *one* IP address, which is assigned to the Internet-facing interface of the gateway. Within the LAN, hosts will use a private IP address space, such as the 192.168.0.0/16 space. However, those addresses are not routable on the Internet. Thus, a common NAT use case is to map between private addresses used on a home LAN and the public address associated with the router, so that traffic can be routed between the LAN and the Internet.

The following figure shows how a NAT gateway in this scenario will rewrite the headers of packets that traverse it:

Graphical user interface, text, application, table

Description automatically generated

1. A host on the LAN initiates a connection to a host outside the LAN, sending a packet through the NAT gateway. The packet has as its source address the private IP address of the host.
2. The NAT gateway will add a new entry to a translation table, in which it notes the internal IP address and TCP port number associated with the connection (from the packet header). Then, it assigns this connection a port number. (In the diagram, it says that the WAN port will be randomly selected. This is sometimes, but not always, the case. If the source port number in the packet header is available at the NAT, and not already in use for another connection, it may use that port number. Otherwise, it will assign it a new one from a list of available port numbers.)
3. The NAT gateway rewrites the source IP address and possibly the port number in the packet header according to the translation table, then forwards it on the WAN.
4. When a response arrives at the NAT gateway from outside the LAN, the NAT gateway looks at the destination port in the packet header. Then, it finds the entry in the translation table that has this port number in the "WAN Port" column.
5. The NAT gateway rewrites the destination IP address and possibly the port in the packet header according to the "LAN IP" and "LAN Port" in the relevant entry of the translation table, then forwards it on the LAN.

An entry is added to the translation table only when a packet associated with a new connection arrives from the LAN. If a packet arrives at the NAT from the WAN whose destination port number does not match any "WAN Port" entry in the translation table, it will be dropped, because the NAT does not know how to translate it. Therefore, a device behind a NAT cannot accept connections from the WAN that it did not initiate.

A workaround for this problem, [port forwarding](https://en.wikipedia.org/wiki/Port_forwarding), enables a home user to *proactively* map a port to a particular host on the home network.

For example, consider the home gateway configuration in the following image. Suppose the gateway has 128.238.66.220 as a public IP address. Hosts on the WAN can initiate a connection to 128.238.66.220 on port 443. The NAT will look up the internal IP address in the port forwarding table, rewrite the destination in the packet header from IP 128.238.66.220 and port 443, to IP 192.168.1.2 and port 443, and then forward the packet on the LAN. Similarly, a packet arriving from the WAN with destination IP 128.238.66.220 and port 3389 will be translated to IP 192.168.1.145 and port 3389.

Graphical user interface

Description automatically generated

**Results**

In this experiment, we set up a gateway with clients connected to it, and observe the behavior of DHCP, DNS, and NAT services.

**DHCP**

We see a client requesting and receiving an IP address from DHCP:

15:15:26.554336 02:eb:60:57:10:ad > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17), length 328)

0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 02:eb:60:57:10:ad, length 300, xid 0xc0902a0f, Flags [none]

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Discover**

Hostname Option 12, length 10: ""

Parameter-Request Option 55, length 13:

Subnet-Mask, BR, Time-Zone, Default-Gateway

Domain-Name, Domain-Name-Server, Option 119, Hostname

Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route

NTP

15:15:29.058621 02:e6:fb:b7:f3:a5 > 02:eb:60:57:10:ad, ethertype IPv4 (0x0800), length 342: (tos 0xc0, ttl 64, id 37575, offset 0, flags [none], proto UDP (17), length 328)

192.168.100.1.67 > 192.168.100.157.68: BOOTP/DHCP, Reply, length 300, xid 0xc0902a0f, Flags [none]

Your-IP 192.168.100.157

Server-IP 192.168.100.1

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Offer**

Server-ID Option 54, length 4: 192.168.100.1

Lease-Time Option 51, length 4: 14400

RN Option 58, length 4: 7200

RB Option 59, length 4: 12600

Subnet-Mask Option 1, length 4: 255.255.255.0

BR Option 28, length 4: 192.168.100.255

Domain-Name-Server Option 6, length 4: 192.168.100.1

Default-Gateway Option 3, length 4: 192.168.100.1

15:15:29.060208 02:eb:60:57:10:ad > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17), length 328)

0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 02:eb:60:57:10:ad, length 300, xid 0xc0902a0f, Flags [none]

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Request**

Server-ID Option 54, length 4: 192.168.100.1

Requested-IP Option 50, length 4: 192.168.100.157

Hostname Option 12, length 10: ""

Parameter-Request Option 55, length 13:

Subnet-Mask, BR, Time-Zone, Default-Gateway

Domain-Name, Domain-Name-Server, Option 119, Hostname

Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route

NTP

15:15:29.090184 02:e6:fb:b7:f3:a5 > 02:eb:60:57:10:ad, ethertype IPv4 (0x0800), length 342: (tos 0xc0, ttl 64, id 37576, offset 0, flags [none], proto UDP (17), length 328)

192.168.100.1.67 > 192.168.100.157.68: BOOTP/DHCP, Reply, length 300, xid 0xc0902a0f, Flags [none]

Your-IP 192.168.100.157

Server-IP 192.168.100.1

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **ACK**

Server-ID Option 54, length 4: 192.168.100.1

Lease-Time Option 51, length 4: 14400

RN Option 58, length 4: 7200

RB Option 59, length 4: 12600

Subnet-Mask Option 1, length 4: 255.255.255.0

BR Option 28, length 4: 192.168.100.255

Domain-Name-Server Option 6, length 4: 192.168.100.1

Default-Gateway Option 3, length 4: 192.168.100.1

**DNS**

We see an iterative name resolution with DNS, for the address "website.nat.ch-geni-net.instageni.research.umich.edu" (in this example). First, the DNS server returns a list of root name servers:

. 77637 IN NS e.root-servers.net.

. 77637 IN NS a.root-servers.net.

. 77637 IN NS h.root-servers.net.

. 77637 IN NS j.root-servers.net.

. 77637 IN NS b.root-servers.net.

. 77637 IN NS m.root-servers.net.

**. 77637 IN NS c.root-servers.net.**

. 77637 IN NS i.root-servers.net.

. 77637 IN NS g.root-servers.net.

. 77637 IN NS l.root-servers.net.

. 77637 IN NS f.root-servers.net.

. 77637 IN NS d.root-servers.net.

. 77637 IN NS k.root-servers.net.

. 518071 IN RRSIG NS 8 0 518400 20170411170000 20170329160000 61045 . EhdIynzhYoqVrNs2csvZwCDUhQbDyUs8EujQ+XVKeQ02S2u0ZN39GY9h pBa7XpOyFAlNfYWNvWVLi1nkpJ7JKS+s4Gg/fU0Nw1MXc2hsbtdsduMs x9tNRjq5RduDfGBZLvNEikIoBndMU4esREkLtbSXrXp0TuvRpl3g5w5j +km7IImfv7KrfhZ16w6kKV8O1ye3yrQr9ow7/XBvUlvDnkcLXUphbfE3 M8vOF+5ABzb7KQp28S5WeI6mrC412jZQgGmdkdCVZArJtc+tKBOoLr64 0mwCMM+phVrLTkx9vdnZpnx9QBB7eqdSOqG9x09R7FiHVyaikueWedNs txIotQ==

;; Received 1097 bytes from 127.0.0.1#53(127.0.0.1) in 19 ms

Then it queries one of the root name servers ("c.root-servers.net" in this example), and *that* returns a list of .edu. servers:

edu. 172800 IN NS f.edu-servers.net.

**edu. 172800 IN NS c.edu-servers.net.**

edu. 172800 IN NS g.edu-servers.net.

edu. 172800 IN NS d.edu-servers.net.

edu. 172800 IN NS l.edu-servers.net.

edu. 172800 IN NS a.edu-servers.net.

edu. 86400 IN DS 28065 8 2 4172496CDE85534E51129040355BD04B1FCFEBAE996DFDDE652006F6 F8B2CE76

edu. 86400 IN RRSIG DS 8 1 86400 20170411170000 20170329160000 61045 . RrZnh2YskIX8cXI1YuiVY9mBh79izFT8HuEc0dxU9B8fKbC5ddQZNPrw d2z0Rn/Bhym6yAAbY0u+5j1+RPXNMLcW2qAcktyw4i3wp4UiGowIcQIa dd5PMODvNrFc8IJpahPhwWcCbK8okgyP1AL9dNtWGcgv7+2zVB1e5vWN n5bOVtthU4p8/X4JI5zRZTyP9Mwz7hPkNFTeeAumjm+hlHJkRAQ211Bz 92f5nRgaJyyIfTq1w9AXRJjE9gnbYRVm32Bh0uEPh1wUddmsN788dYmc xzbbNjZDnAksPoopkupQinhvgSBO9Zf9d21dtR3WQLt1w2YoX46likoO rUJnNQ==

;; Received 651 bytes from 192.33.4.12#53(**c.root-servers.net**) in 43 ms

Next, the "c.edu-servers.net." name server returns a list of name servers for the umich.edu. domain:

umich.edu. 172800 IN NS dns.cs.wisc.edu.

**umich.edu. 172800 IN NS dns2.itd.umich.edu.**

umich.edu. 172800 IN NS dns1.itd.umich.edu.

9DHS4EP5G85PF9NUFK06HEK0O48QGK77.edu. 86400 IN NSEC3 1 1 0 - 9N4FQMDDEN9N9I8F7VUF86JHJF574LHS NS SOA RRSIG DNSKEY NSEC3PARAM

9DHS4EP5G85PF9NUFK06HEK0O48QGK77.edu. 86400 IN RRSIG NSEC3 8 2 86400 20170405183649 20170329172649 43544 edu. unQy39EYi/666eEaXxmUQcsEgE/FtqxKCMVuODd4o8NhWp8gaTkukJNS OM1xfxo916xtQyVD2lXaoZ6pJCSENtYo8AUGja7Uvx2zGLpmH2a3gAWK gvZqHpkLchFQrlFcP3Qg6dqm6OJbjBBHDIcWyl8Q4ic5aYM1F15ewzA0 Y0Q=

NOS8G1ANJ5QIKT46L3QM4OLBQD9V72K5.edu. 86400 IN NSEC3 1 1 0 - O9PNA7OPKKO436OEO034N6ANUNH47GJ5 NS DS RRSIG

NOS8G1ANJ5QIKT46L3QM4OLBQD9V72K5.edu. 86400 IN RRSIG NSEC3 8 2 86400 20170405172140 20170329161140 43544 edu. ReOneHxFyyuuqcHaFrBftXfW5VCLGa0XCKWZAan1zW6vlQ3OtATJ0UHr +gR6UDb8nH8C+KAQCCdxanZM5feAOCsEDxl1X6D36nuMfVb86k3q8UZ/ jWy7OqDL2PMzRNjGI0Sdwl49BUx7VhGXjIt9ev87dFa1zHprJJHf8KCe mgU=

;; Received 682 bytes from 192.26.92.30#53(**c.edu-servers.net**) in 30 ms

And then "dns2.itd.umich.edu" returns name servers for the subdomain instageni.research.umich.edu.:

instageni.research.umich.edu. 300 IN NS ns.instageni.research.umich.edu.

**instageni.research.umich.edu. 300 IN NS ns.emulab.net.**

;; Received 141 bytes from 192.12.80.222#53(**dns2.itd.umich.edu**) in 43 ms

Finally, from "ns.emulab.net" we get the address of website.nat.ch-geni-net.instageni.research.umich.edu.:

**website.nat.ch-geni-net.instageni.research.umich.edu. 1 IN CNAME pcvm2-8.instageni.research.umich.edu.**

**pcvm2-8.instageni.research.umich.edu. 30 IN A 192.41.233.62**

instageni.research.umich.edu. 30 IN NS ns.emulab.net.

instageni.research.umich.edu. 30 IN NS ns.instageni.research.umich.edu.

;; Received 195 bytes from 155.98.32.70#53(**ns.emulab.net**) in 39 ms

**NAT**

We see NAT rewriting packet headers. Here are the packets in the LAN:

16:53:13.663719 IP **192.168.100.157**.57962 > 192.41.233.62.80: Flags [S], seq 3245904370, win 29200, options [mss 1460,sackOK,TS val 1688452 ecr 0,nop,wscale 7], length 0

16:53:14.231939 IP 192.41.233.62.80 > **192.168.100.157**.57962: Flags [S.], seq 3992044697, ack 3245904371, win 28960, options [mss 1460,sackOK,TS val 1691466 ecr 1688452,nop,wscale 7], length 0

16:53:14.232738 IP **192.168.100.157**.57962 > 192.41.233.62.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 1688595 ecr 1691466], length 0

And here are the same packets on the WAN:

16:53:14.032389 IP **172.17.3.4**.57962 > 192.41.233.62.80: Flags [S], seq 3245904370, win 29200, options [mss 1460,sackOK,TS val 1688452 ecr 0,nop,wscale 7], length 0

16:53:14.032440 IP 192.41.233.62.80 > **172.17.3.4**.57962: Flags [S.], seq 3992044697, ack 3245904371, win 28960, options [mss 1460,sackOK,TS val 1691466 ecr 1688452,nop,wscale 7], length 0

16:53:14.299903 IP **172.17.3.4**.57962 > 192.41.233.62.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 1688595 ecr 1691466], length 0

**Run my experiment**

First, you will need to reserve resources for your experiment.

**GENI-specific instructions: Reserve resources**

In the GENI Portal, create a new slice, then click "Add Resources". Scroll down to where it says "Choose RSpec" and select the "URL" option, the load the RSpec from the URL: <https://git.io/JThp3>.

This should load a topology onto your canvas with:

* Two "client" nodes, connected to a LAN, and a "gateway" node also connected to the LAN. The "gateway" node has IP address 192.168.100.1 on the interface that is connected to the LAN; the clients have no IP address on this interface. (You can safely ignore the red "!" that alerts you to this.)
* A "website" node. In the RSpec, we request a "Publicly Routable IP" for this node.

Click on "Site 1" and choose an InstaGENI site to bind to, then reserve your resources. Wait for your nodes to boot up (they will turn green in the canvas display on your slice page in the GENI portal when they are ready) and then log in to each node.

Wait for all of your nodes to become ready to log in (they will turn green on the canvas). Then, use the details given in the GENI Portal to SSH into each node, in four terminal windows.

**Cloudlab-specific instructions: Reserve resources**

For this experiment, we will use the CloudLab profile available at the following link: <https://www.cloudlab.us/p/cl-education/home-gateway>.

If you visit this link, you’ll see a brief description of the profile. Click “Next”. On the following page, you’ll see a diagram of your experiment topology, and on the left you’ll be asked to select the “Cluster” on which you want your experiment to run. For this experiment, you'll be asked to select **two** clusters, and they should be two **different** ones! The website host will be on one cluster, and the rest of the hosts will be on the other.

This experiment can run on any pair of two different clusters. However, since CloudLab is a shared resource, on some occasions the cluster you select might not have enough available resources to support your experiment. The status indicator next to each cluster tells you roughly how heavily utilized it is at the moment - green indicates that there are not many users, orange means heavy load, and red means that it is almost fully utilized. You are more likely to be successful if you choose a cluster with a green indicator.

After you select two clusters, you can leave the "Name" field blank, or give your experiment a name - it’s your choice. Also make sure your "Project" is selected. Then, click "Next".

On the last page, you’ll be asked to set the duration of your experiment. At the end of this duration, your resources will be deleted automatically - so make sure to give yourself enough time to finish.

You can leave the start date/time on this page blank if you are ready to work on the experiment right now.

When you have made all your choices, click "Finish" to ask CloudLab to reserve resources according to your configuration.

Once you have successfully instantiated a profile, it will still take some time before your resources are ready for you to log in.

As your resources come online, you’ll see their progress on the CloudLab experiment page. Once each host in your experiment is "green" and has a "✓" icon in the top right corner, it is ready for you to log in!

For *most* CloudLab experiments, you can log in with the terminal in the CloudLab web portal or using a terminal application that is installed on your laptop or PC. For this experiment, however, you should use the terminal in the CloudLab web portal. Right-click on each node in your topology, and open a shell on the CloudLab web portal.

**Reset configuration on the clients**

This experiment mimics clients connecting to a residential gateway and using the network services provided by that gateway (DHCP, DNS, NAT, for example). However, the clients in our experiment are *already* set up to use a university gateway for those services. (Otherwise, they would not have a functional network connection and we would not be able to get in to them over SSH!) To run this experiment, we will tell the clients *not* to use the university gateway for anything except our SSH session.

**GENI-specific instructions: Removing route via university gateway**

On GENI, once you remove the route via university gateway, **you should plan to finish the rest of the experiment involving these resources in the same SSH session**. Otherwise, if you stop and then resume the experiment from a new SSH session in a new location, you won't be able to access the clients anymore.

**CloudLab-specific instructions: Removing route via university gateway**

On CloudLab, once you remove the route via university gateway, you will only be able to log in to these client nodes using the shell in the CloudLab web portal. You won't be able to access them using the terminal application on your laptop or PC anymore.

When you're ready, on each of the two "client" nodes (and *only* on the client nodes), run

wget -qO- https://git.io/JTjLB | bash

to download and run a [script](https://gist.github.com/ffund/ebfa8f9eabbedd2bc4f26ee7f38ae2bd) that removes most of the network settings on the client that involve the university gateway.

(The output will look something like:

dhclient: no process found

127.0.0.1 localhost

sudo: unable to resolve host client-1

127.0.0.1 client-1

)

**GENI-specific instructions: If you lose access to resources**

**Note:** as mentioned above, if your own IP address changes, you may lose connectivity to your client nodes. Here's how to access your client nodes *if* that happens!

1. If you are on Windows, first enable and start the SSH agent: Open Services (Start Menu > Services), then select OpenSSH Authentication Agent, then set StartupType to Automatic.
2. If you are using Windows or Mac OS X: run ssh-add to start the SSH agent.
3. Use SSH to log in to the "gateway" node, but add the -A argument to the SSH command. Also specify the path to your private key with -i, if you normally do so when using SSH to log in to GENI hosts.
4. From the terminal on the "gateway" node, use the SSH command in the GENI Portal to log on to the "client" node. Do *not* use the -i argument and do not specify a path to a key, even if you normally do so when using SSH to log in to GENI hosts.

For example:

ffund@laptop:~$ ssh **-A** ffund01@pc1.instageni.metrodatacenter.com -p 25812

ffund01@gateway:~$ ssh ffund01@pc1.instageni.metrodatacenter.com -p 25810

ffund01@client-1:~$

**Set up gateway**

Now we'll set up the "gateway" node:

First, make sure packet forwarding is enabled, so that it can route traffic between the LAN and WAN. On the "gateway" node, run:

sudo sysctl -w net.ipv4.ip\_forward=1

Next, we will install dnsmasq, a lightweight DNS and DHCP server. On the "gateway" node, run:

echo "deb http://us.archive.ubuntu.com/ubuntu/ focal main restricted universe multiverse" | sudo tee -a /etc/apt/sources.list

sudo apt-get update

sudo apt-get -y install dnsmasq dnsmasq-base

Don't worry if you see an error message in the output that says:

failed to create listening socket for port 53: Address already in use

we'll fix that in a moment with a new configuration file!

We will now configure dnsmasq. On "gateway", run

sudo wget -O /etc/dnsmasq.conf https://git.io/JThhR

to download a [config file](https://gist.github.com/ffund/ae43447629c3c5b9da9eb54962f93adf) into "/etc/dnsmasq.conf". The contents of the config file are:

interface=eth1

dhcp-range=192.168.100.100,192.168.100.199,4h

dhcp-option=3,192.168.100.1

dhcp-option=6,192.168.100.1

bind-interfaces

This tells dnsmasq

* to accept incoming requests on the experiment interface, eth1,
* to give out IP addresses in the range 192.168.100.100-192.168.100.199, and
* to also notify clients to use it (192.168.100.1) as a default gateway (DHCP option 3),
* to also notify clients to use it (192.168.100.1) as the nameserver for DNS queries (DHCP option 6).
* to only bind to the interface on which it is listening for incoming connections (eth1)

Restart the dnsmasq service to load the new configuration:

sudo service dnsmasq restart

Now that both the gateway and the clients are set up, we will observe what happens when clients use some typical residential gateway services.

**Observe a DHCP request and response**

We will use tcpdump to capture traffic involving DHCP on the LAN. On "gateway", run

sudo tcpdump -i eth1 -n -e -v "udp port 67 or udp port 68"

where

* -i eth1 sets the interface to listen on
* -n says to show numerical addresses, rather than trying to resolve them to hostnames in the output,
* -e says to shoe MAC addresses,
* -v says to show verbose output (including details about the DHCP packets), and
* "udp port 67 or udp port 68" is a filter that says to only show traffic using the DHCP ports (UDP port 67 for the DHCP server, UDP port 68 for the DHCP client).

Now, from one of the client nodes, run:

sudo dhclient -d eth1

to initiate a request for an address from DHCP. (The -d argument is to specify debug mode; it keeps the dhclient process in the foreground, and lets us see its output.)

In this terminal you should see something like:

DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 3 (xid=0xf2a90c0)

DHCPREQUEST of 192.168.100.157 on eth1 to 255.255.255.255 port 67 (xid=0xf2a90c0)

DHCPOFFER of 192.168.100.157 from 192.168.100.1

DHCPACK of 192.168.100.157 from 192.168.100.1

bound to 192.168.100.157 -- renewal in 5679 seconds.

Meanwhile, in the tcpdump output, you should see each of those messages.

First, the client sends a "discover" request to try and find DHCP servers on the LAN. The client does not have an IP address yet, so this message has "0.0.0.0" (an invalid address) as the source IP address. Also, the client does not know the address of the DHCP server, so it uses the broadcast IP address "255.255.255.255" and the broadcast MAC address "ff:ff:ff:ff:ff:ff" in the destination fields of the Layer 2 and Layer 3 headers:

15:15:26.554336 02:eb:60:57:10:ad > ff:ff:ff:ff:ff:ff ethertype IPv4 (0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17), length 328)

0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 02:eb:60:57:10:ad, length 300, xid 0xc0902a0f, Flags [none]

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Discover**

Hostname Option 12, length 10: ""

Parameter-Request Option 55, length 13:

Subnet-Mask, BR, Time-Zone, Default-Gateway

Domain-Name, Domain-Name-Server, Option 119, Hostname

Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route

NTP

Next, a DHCP server that receives this request (in this case: our gateway at 192.168.100.1), responds with an offer. In the example below, it offers the IP address "192.168.100.157":

15:15:29.058621 02:e6:fb:b7:f3:a5 > 02:eb:60:57:10:ad, ethertype IPv4 (0x0800), length 342: (tos 0xc0, ttl 64, id 37575, offset 0, flags [none], proto UDP (17), length 328)

192.168.100.1.67 > 192.168.100.157.68: BOOTP/DHCP, Reply, length 300, xid 0xc0902a0f, Flags [none]

**Your-IP 192.168.100.157**

**Server-IP 192.168.100.1**

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Offer**

Server-ID Option 54, length 4: 192.168.100.1

Lease-Time Option 51, length 4: 14400

RN Option 58, length 4: 7200

RB Option 59, length 4: 12600

Subnet-Mask Option 1, length 4: 255.255.255.0

BR Option 28, length 4: 192.168.100.255

**Domain-Name-Server Option 6, length 4: 192.168.100.1**

**Default-Gateway Option 3, length 4: 192.168.100.1**

The server also includes other DHCP options in the message -in this case, it informs the client of the default gateway to use (option 3) and the name server to use (option 6).

The client responds with a request for the IP address that was just offered. In case multiple DHCP servers respond to the "Discover" message with an "Offer", the client will put in a "Request" for only one of them, from one DHCP server. The other DHCP servers will notice the request (since it is sent to the broadcast address, they will all receive it) and understand that the client does not accept their offer:

15:15:29.060208 02:eb:60:57:10:ad > ff:ff:ff:ff:ff:ff, ethertype IPv4 (0x0800), length 342: (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17), length 328)

0.0.0.0.68 > 255.255.255.255.67: BOOTP/DHCP, Request from 02:eb:60:57:10:ad, length 300, xid 0xc0902a0f, Flags [none]

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **Request**

**Server-ID Option 54, length 4: 192.168.100.1**

**Requested-IP Option 50, length 4: 192.168.100.157**

Hostname Option 12, length 10: ""

Parameter-Request Option 55, length 13:

Subnet-Mask, BR, Time-Zone, Default-Gateway

Domain-Name, Domain-Name-Server, Option 119, Hostname

Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route

NTP

Finally, the server acknowledges the request, completing the process:

15:15:29.090184 02:e6:fb:b7:f3:a5 > 02:eb:60:57:10:ad, ethertype IPv4 (0x0800), length 342: (tos 0xc0, ttl 64, id 37576, offset 0, flags [none], proto UDP (17), length 328)

192.168.100.1.67 > 192.168.100.157.68: BOOTP/DHCP, Reply, length 300, xid 0xc0902a0f, Flags [none]

Your-IP 192.168.100.157

Server-IP 192.168.100.1

Client-Ethernet-Address 02:eb:60:57:10:ad

Vendor-rfc1048 Extensions

Magic Cookie 0x63825363

DHCP-Message Option 53, length 1: **ACK**

Server-ID Option 54, length 4: 192.168.100.1

Lease-Time Option 51, length 4: 14400

RN Option 58, length 4: 7200

RB Option 59, length 4: 12600

Subnet-Mask Option 1, length 4: 255.255.255.0

BR Option 28, length 4: 192.168.100.255

Domain-Name-Server Option 6, length 4: 192.168.100.1

Default-Gateway Option 3, length 4: 192.168.100.1

If you run

ifconfig eth1

on the client (you can use Ctrl+C to stop the running dhclient instance), you should observe that it is now using the IP address assigned to it by DHCP, e.g.:

eth1 Link encap:Ethernet HWaddr 02:eb:60:57:10:ad

inet addr:**192.168.100.157** Bcast:192.168.100.255 Mask:255.255.255.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:67 errors:0 dropped:0 overruns:0 frame:0

TX packets:40 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:6314 (6.3 KB) TX bytes:3941 (3.9 KB)

**Cloudlab-specific instructions: Updating DNS configuration**

If you are using CloudLab, you should run

sudo /lib/systemd/systemd-resolved &

on the client after running dhclient, to allow the system's name resolution configuration to be updated by the DHCP client. Hit Enter a few times to return to your regular terminal prompt.

You can also see that the client has been configured to use the gateway as the name server, as specified in the DHCP options. On the client, run

cat /etc/resolv.conf

to see the address of the name server it uses. You should see:

nameserver 192.168.100.1

(You may also see some secondary nameservers from the host testbed site; that's OK!) Finally, we can see that the client uses the gateway as the default gateway for routing purposes. On the client, run

route -n

to see routing rules. We should see two rules associated with the interface connected to the LAN:

0.0.0.0 192.168.100.1 0.0.0.0 UG 0 0 0 eth1

192.168.100.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1

The first rule says to use "192.168.100.1" as the next hop for any traffic whose destination address does not meet any more specific rule, i.e. as the default rule. The second rule says to send traffic for the 192.168.100.0/24 subnet out of the eth1eth1 interface, which is connected to the LAN. On the second client node, use

sudo dhclient -d eth1

to get an IP address on this node as well.

**Cloudlab-specific instructions: Updating DNS configuration**

If you are using CloudLab, you should run

sudo /lib/systemd/systemd-resolved &

on the second client also, after running dhclient, to allow the system's name resolution configuration to be updated by the DHCP client. Hit Enter a few times to return to your regular terminal prompt.

**Observe a DNS query and response**

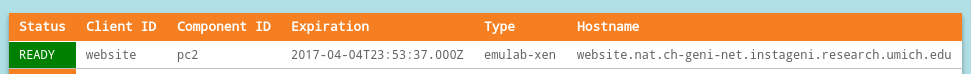
Next, we'll observe a DNS query and response. We can use a DNS lookup utility called [dig](https://linux.die.net/man/1/dig).

On the "gateway", run tcpdump to monitor traffic on UDP port 53, the DNS port:

sudo tcpdump -i eth1 -n -v "udp port 53"

On a client, we are going to look up the IP address associated with the "website" node in our topology.

**GENI-specific instructions: Find website hostname**

In the GENI Portal, click "Details" on your slice page and find the hostname associated with the "website" node. In my slice, it is "website.nat.ch-geni-net.instageni.research.umich.edu":

**Cloudlab-specific instructions: Find website hostname**

In the CloudLab Portal, click the "List View" tab on your experiment page and find the hostname associated with the "website" node. In my experiment, it is "aptvm068-1.apt.emulab.net":

Graphical user interface, text, application

Description automatically generated

Then, on a client node, run

dig **website.nat.ch-geni-net.instageni.research.umich.edu**

substituting *your* hostname in place of mine in the command above.

The output should look something like this:

; <<>> DiG 9.9.5-3ubuntu0.2-Ubuntu <<>> website.nat.ch-geni-net.instageni.research.umich.edu

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29901

;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 4096

**;; QUESTION SECTION:**

**;website.nat.ch-geni-net.instageni.research.umich.edu. IN A**

**;; ANSWER SECTION:**

**website.nat.ch-geni-net.instageni.research.umich.edu. 1 IN CNAME pcvm2-8.instageni.research.umich.edu.**

**pcvm2-8.instageni.research.umich.edu. 30 IN A 192.41.233.62**

**;; AUTHORITY SECTION:**

**instageni.research.umich.edu. 30 IN NS ns.emulab.net.**

**instageni.research.umich.edu. 30 IN NS ns.instageni.research.umich.edu.**

**;; ADDITIONAL SECTION:**

**ns.emulab.net. 7238 IN A 155.98.32.70**

**ns.instageni.research.umich.edu. 30 IN A 192.41.233.4**

;; Query time: 7 msec

;; SERVER: 192.168.100.1#53(192.168.100.1)

;; WHEN: Wed Mar 29 15:53:33 EDT 2017

;; MSG SIZE rcvd: 195

In particular, note:

* the QUESTION section describes the query: we are seeking an address ("A" [record type](https://en.wikipedia.org/wiki/List_of_DNS_record_types)) for the name website.nat.ch-geni-net.instageni.research.umich.edu..
* the ANSWER section gives the address of the name that was the subject of the query. In this instance, it informs us (through a CNAME [record type](https://en.wikipedia.org/wiki/List_of_DNS_record_types)) that the name website.nat.ch-geni-net.instageni.research.umich.edu. is an alias for the host with "canonical name" pcvm2-8.instageni.research.umich.edu., and then returns the address ("A" record) for pcvm2-8.instageni.research.umich.edu..
* the AUTHORITY section lists the DNS servers that are authoritative for answering queries about the name, and the ADDITIONAL section lists their addresses. Specifically, it lists "ns.emulab.net." at 155.98.32.70 and "ns.instageni.research.umich.edu." at 192.41.233.4 as authoritative name servers for any name ending in instageni.research.umich.edu.. (An authoritative nameserver is one that holds the actual records for a particular domain or address; if queried, it marks its response for that address as an authoritative response. Other non-authoritative nameservers may relay the information from a cache or from another nameserver.)

You should also see the DNS traffic in your tcpdump output on the gateway. First the query from the client to the gateway, seeking the address of "website.nat.ch-geni-net.instageni.research.umich.edu.":

15:56:51.703460 IP (tos 0x0, ttl 64, id 40389, offset 0, flags [none], proto UDP (17), length 109)

192.168.100.157.52370 > 192.168.100.1.53: 52641+ [1au] **A? website.nat.ch-geni-net.instageni.research.umich.edu**. (81)

and then the response from the gateway, that gives the address as "192.41.233.62" in my example:

15:56:51.704291 IP (tos 0x0, ttl 64, id 37703, offset 0, flags [DF], proto UDP (17), length 223)

192.168.100.1.53 > 192.168.100.157.52370: 52641\* 2/2/3 website.nat.ch-geni-net.instageni.research.umich.edu. CNAME pcvm2-8.instageni.research.umich.edu., pcvm2-8.instageni.research.umich.edu. **A 192.41.233.62** (195)

With the addition of the +trace option, dig will show you each successive hierarchical step that the query takes, until it reaches the name server that is authoritative for the name.

Let's try it! We can't run hierarchical query on a client node in our experiment, because at this point, the client nodes don't have access to the Internet to query DNS servers. (The client won't have Internet access until the next section, when we set up NAT.) Therefore, we will try this on the gateway node.

**On the "gateway"**, run:

dig +trace **website.nat.ch-geni-net.instageni.research.umich.edu**

again, substituting your own "website" node's hostname. This time, you'll see much more output.

First, the DNS server returns a list of root name servers:

. 77637 IN NS e.root-servers.net.

. 77637 IN NS a.root-servers.net.

. 77637 IN NS h.root-servers.net.

. 77637 IN NS j.root-servers.net.

. 77637 IN NS b.root-servers.net.

. 77637 IN NS m.root-servers.net.

**. 77637 IN NS c.root-servers.net.**

. 77637 IN NS i.root-servers.net.

. 77637 IN NS g.root-servers.net.

. 77637 IN NS l.root-servers.net.

. 77637 IN NS f.root-servers.net.

. 77637 IN NS d.root-servers.net.

. 77637 IN NS k.root-servers.net.

. 518071 IN RRSIG NS 8 0 518400 20170411170000 20170329160000 61045 . EhdIynzhYoqVrNs2csvZwCDUhQbDyUs8EujQ+XVKeQ02S2u0ZN39GY9h pBa7XpOyFAlNfYWNvWVLi1nkpJ7JKS+s4Gg/fU0Nw1MXc2hsbtdsduMs x9tNRjq5RduDfGBZLvNEikIoBndMU4esREkLtbSXrXp0TuvRpl3g5w5j +km7IImfv7KrfhZ16w6kKV8O1ye3yrQr9ow7/XBvUlvDnkcLXUphbfE3 M8vOF+5ABzb7KQp28S5WeI6mrC412jZQgGmdkdCVZArJtc+tKBOoLr64 0mwCMM+phVrLTkx9vdnZpnx9QBB7eqdSOqG9x09R7FiHVyaikueWedNs txIotQ==

;; Received 1097 bytes from 127.0.0.1#53(127.0.0.1) in 19 ms

Then it queries one of the root name servers ("c.root-servers.net" in this example), and *that* returns a list of .edu. servers:

edu. 172800 IN NS f.edu-servers.net.

**edu. 172800 IN NS c.edu-servers.net.**

edu. 172800 IN NS g.edu-servers.net.

edu. 172800 IN NS d.edu-servers.net.

edu. 172800 IN NS l.edu-servers.net.

edu. 172800 IN NS a.edu-servers.net.

edu. 86400 IN DS 28065 8 2 4172496CDE85534E51129040355BD04B1FCFEBAE996DFDDE652006F6 F8B2CE76

edu. 86400 IN RRSIG DS 8 1 86400 20170411170000 20170329160000 61045 . RrZnh2YskIX8cXI1YuiVY9mBh79izFT8HuEc0dxU9B8fKbC5ddQZNPrw d2z0Rn/Bhym6yAAbY0u+5j1+RPXNMLcW2qAcktyw4i3wp4UiGowIcQIa dd5PMODvNrFc8IJpahPhwWcCbK8okgyP1AL9dNtWGcgv7+2zVB1e5vWN n5bOVtthU4p8/X4JI5zRZTyP9Mwz7hPkNFTeeAumjm+hlHJkRAQ211Bz 92f5nRgaJyyIfTq1w9AXRJjE9gnbYRVm32Bh0uEPh1wUddmsN788dYmc xzbbNjZDnAksPoopkupQinhvgSBO9Zf9d21dtR3WQLt1w2YoX46likoO rUJnNQ==

;; Received 651 bytes from 192.33.4.12#53(**c.root-servers.net**) in 43 ms

Next, the "c.edu-servers.net." name server returns a list of name servers for the umich.edu. domain:

umich.edu. 172800 IN NS dns.cs.wisc.edu.

**umich.edu. 172800 IN NS dns2.itd.umich.edu.**

umich.edu. 172800 IN NS dns1.itd.umich.edu.

9DHS4EP5G85PF9NUFK06HEK0O48QGK77.edu. 86400 IN NSEC3 1 1 0 - 9N4FQMDDEN9N9I8F7VUF86JHJF574LHS NS SOA RRSIG DNSKEY NSEC3PARAM

9DHS4EP5G85PF9NUFK06HEK0O48QGK77.edu. 86400 IN RRSIG NSEC3 8 2 86400 20170405183649 20170329172649 43544 edu. unQy39EYi/666eEaXxmUQcsEgE/FtqxKCMVuODd4o8NhWp8gaTkukJNS OM1xfxo916xtQyVD2lXaoZ6pJCSENtYo8AUGja7Uvx2zGLpmH2a3gAWK gvZqHpkLchFQrlFcP3Qg6dqm6OJbjBBHDIcWyl8Q4ic5aYM1F15ewzA0 Y0Q=

NOS8G1ANJ5QIKT46L3QM4OLBQD9V72K5.edu. 86400 IN NSEC3 1 1 0 - O9PNA7OPKKO436OEO034N6ANUNH47GJ5 NS DS RRSIG

NOS8G1ANJ5QIKT46L3QM4OLBQD9V72K5.edu. 86400 IN RRSIG NSEC3 8 2 86400 20170405172140 20170329161140 43544 edu. ReOneHxFyyuuqcHaFrBftXfW5VCLGa0XCKWZAan1zW6vlQ3OtATJ0UHr +gR6UDb8nH8C+KAQCCdxanZM5feAOCsEDxl1X6D36nuMfVb86k3q8UZ/ jWy7OqDL2PMzRNjGI0Sdwl49BUx7VhGXjIt9ev87dFa1zHprJJHf8KCe mgU=

;; Received 682 bytes from 192.26.92.30#53(**c.edu-servers.net**) in 30 ms

And then "dns2.itd.umich.edu" returns name servers for the subdomain instageni.research.umich.edu.:

instageni.research.umich.edu. 300 IN NS ns.instageni.research.umich.edu.

**instageni.research.umich.edu. 300 IN NS ns.emulab.net.**

;; Received 141 bytes from 192.12.80.222#53(**dns2.itd.umich.edu**) in 43 ms

Finally, from "ns.emulab.net" we get the address of website.nat.ch-geni-net.instageni.research.umich.edu.:

**website.nat.ch-geni-net.instageni.research.umich.edu. 1 IN CNAME pcvm2-8.instageni.research.umich.edu.**

**pcvm2-8.instageni.research.umich.edu. 30 IN A 192.41.233.62**

instageni.research.umich.edu. 30 IN NS ns.emulab.net.

instageni.research.umich.edu. 30 IN NS ns.instageni.research.umich.edu.

;; Received 195 bytes from 155.98.32.70#53(**ns.emulab.net**) in 39 ms

**Use NAT**

Next, we will set up our gateway to use NAT. On the gateway, run

sudo iptables -A FORWARD -o eth0 -i eth1 -s 192.168.100.0/24 -m conntrack --ctstate NEW -j ACCEPT

sudo iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

sudo iptables -t nat -F POSTROUTING

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

Here,

* The first rule tracks connections involving the 192.168.100.0/24 network, and makes sure that packets initiating a new connection are forwarded from the LAN to the WAN.
* The second rule allows forwarding of packets that are part of an established connection.
* The third and fourth rules actually do the network address translation. They will rewrite the source IP address in the Layer 3 header of packets forwarded out on the WAN interface. Also, when packets are received from the WAN, it identifies the connection that they belong to, rewrites the destination IP address in the Layer 3 headers, and forwards them on the LAN.

Find out the public IP address of the NAT node. On the "gateway" device, run

wget -qO- http://ipinfo.io/

and find out the public IP address associated with yours, e.g.

{

"ip": "**192.41.233.23**",

"hostname": "No Hostname",

"city": "Ann Arbor",

"region": "Michigan",

"country": "US",

"loc": "42.2734,-83.7133",

"org": "AS36375 University of Michigan",

"postal": "48104"

}

With NAT, to the rest of the Internet, all of the hosts in our "home" network will appear as if they are coming from that IP address. On the client nodes, run

wget -qO- http://ipinfo.io/

Note that this connection will go to the Internet via the gateway, using interface eth1 which has a private IP address in the range 192.168.100.0/24. Despite this, you should see that the client, too, appears to be coming from the IP address belonging to the NAT:

{

"ip": "**192.41.233.23**",

"hostname": "No Hostname",

"city": "Ann Arbor",

"region": "Michigan",

"country": "US",

"loc": "42.2734,-83.7133",

"org": "AS36375 University of Michigan",

"postal": "48104"

}

To further explore our NAT functionality, we'll configure a web server, then access it from one of our client hosts and observe what IP address the client appears to be connecting from.

SSH into the "website" node and install the [Apache](https://en.wikipedia.org/wiki/Apache_HTTP_Server) web server application on it:

sudo apt-get update

sudo apt-get -y install apache2

**GENI-specific instructions: Open website in a web browser**

On the client node, install a basic terminal-based web browser with:

sudo apt-get update

sudo apt-get -y install lynx

We'll monitor traffic to and from the web server (which operates on TCP port 80) using tcpdump. We will monitor it in two places, on the LAN and on the WAN.

On the "gateway" node, monitor traffic to and from the LAN:

sudo tcpdump -i **eth1** -n "tcp port 80"

and on the "website" node, monitor traffic to and from the WAN:

sudo tcpdump -i **eth0** -n "tcp port 80"

Then, on the client, run

lynx http://**website.nat.ch-geni-net.instageni.research.umich.edu**/

(substituting the hostname of *your* "website" node in the URL above). You should see the website load in the terminal:

Graphical user interface

Description automatically generated with low confidence

**CloudLab-specific instructions: Open website in a web browser**

On the client node, install a web browser with:

sudo apt-get update

sudo apt-get -y install firefox lynx

We'll monitor traffic to and from the web server (which operates on TCP port 80) using tcpdump. We will monitor it in two places, on the LAN and on the WAN.

On the "gateway" node, monitor traffic to and from the LAN:

sudo tcpdump -i **eth1** -n "tcp port 80"

and on the "website" node, monitor traffic to and from the WAN:

sudo tcpdump -i **eth0** -n "tcp port 80"

Firefox is a graphical web browser, so to use it, you will need a VNC session. Open a VNC session on the client, and run

firefox

Then, inside your Firefox session, put the URL http://**aptvm068-1.apt.emulab.net**/ in the address bar (substituting the hostname of \_your\_ "website" node in the URL) and hit Enter. You should see the website load in the browser.

As a fallback, in case your VNC session doesn't work out (it can be a little bit flaky), that's OK: we have also installed a terminal-based web browser that you can use inside your shell session. To try the terminal-based option, in a terminal session on the client run

lynx http://**aptvm068-1.apt.emulab.net**/

(substituting the hostname of *your* "website" node in the URL above). You should see the website load in the terminal.

Meanwhile, in the tcpdump output you can see how the IP addresses are rewritten in the packet headers. On the LAN (the tcpdump running on the "gateway" interface facing the LAN), the Layer 3 packet header shows the connection between 192.168.100.157 (port 57962) and 192.41.233.62 (port 80):

16:53:13.663719 IP **192.168.100.157**.57962 > 192.41.233.62.80: Flags [S], seq 3245904370, win 29200, options [mss 1460,sackOK,TS val 1688452 ecr 0,nop,wscale 7], length 0

16:53:14.231939 IP 192.41.233.62.80 > **192.168.100.157**.57962: Flags [S.], seq 3992044697, ack 3245904371, win 28960, options [mss 1460,sackOK,TS val 1691466 ecr 1688452,nop,wscale 7], length 0

16:53:14.232738 IP **192.168.100.157**.57962 > 192.41.233.62.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 1688595 ecr 1691466], length 0

However, for the *same* packets, the tcpdump running on the "website" node (on the WAN) shows the connection as being between 172.17.3.4 (port 57962) and 192.41.233.62 (port 80):

16:53:14.032389 IP **172.17.3.4**.57962 > 192.41.233.62.80: Flags [S], seq 3245904370, win 29200, options [mss 1460,sackOK,TS val 1688452 ecr 0,nop,wscale 7], length 0

16:53:14.032440 IP 192.41.233.62.80 > **172.17.3.4**.57962: Flags [S.], seq 3992044697, ack 3245904371, win 28960, options [mss 1460,sackOK,TS val 1691466 ecr 1688452,nop,wscale 7], length 0

16:53:14.299903 IP **172.17.3.4**.57962 > 192.41.233.62.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 1688595 ecr 1691466], length 0

confirming that the packet headers are rewritten in the NAT gateway.

**Note**: your "gateway" node may itself be located behind a NAT! So the WAN-facing IP address on the gateway may not be the same as the address you observe on the "website" node, and either or both of these addresses may not be the "public" IP address you identified previously.

**Notes**

Updated September 2022: Added CloudLab instructions.

Updated November 2020: Thanks to Devesh Yadav and Professor Violet Syrotiuk at Arizona State University for corrections.

**Exercise**

**DHCP**

* Show the "Discover" packet sent by the client to try and find DHCP servers on the LAN. What are the source and destination IP addresses in this request? Why are these addresses used?
* Show the "Offer" packet sent by the server. What IP address does the server offer in this example? What is the range of addresses that the server in our experiment may offer? (You can refer to the dnsmasq configuration file.)
* Show the "Request" packet sent by the client. What is the destination address in this request? Why?
* Show the DHCP ACK sent by the server to complete the process.
* What command would you run at the client to verify each of the following? Take screenshots showing the command *and* the output for each of these, and annotate your screenshots by drawing a circle or a box around the configuration suggested by the server in the DHCP Offer/ACK.
  + That the eth1 interface will use the newly acquired IP address, and the Subnet-Mask suggested by the server?
  + That the client uses the Domain-Name-Server suggested by the server?
  + That the client uses the Default-Gateway suggested by the server?

**DNS**

* For the basic DNS resolution (not the one with +trace!) show the dig command and its output. Also show the DNS query and response from the tcpdump output. Answer the following questions using the dig output. No explanation is required - just copy and paste the relevant word from the dig output for each answer.
  + What is the hostname that you tried to resolve?
  + What is the DNS record *type* that your query relates to? ([Here is a list of DNS record types](https://en.wikipedia.org/wiki/List_of_DNS_record_types).)
  + What is the *address* for the hostname you asked to resolve?
  + Give the name of the first "authoritative" server listed for this name, and the IP address of that "authoritative" server.
  + What is the IP address of the server that the DNS response is from?
* For the hierarchical DNS resolution with +trace, show the dig command and its output. Draw a diagram showing how the hostname was resolved recursively, starting from the implied . at the end and moving toward the beginning.
  + At the top, show the nameservers for the root domain. Highlight the one that you queried for the top-level domain (as shown in the dig +trace output).
  + At the next level, show the nameservers for the top-level domain. Highlight the one that you queried for the second-level domain.
  + At the next level, show the nameservers for the second-level domain. Highlight the one that you queried for the subdomain.
  + Repeat until you have shown how the complete hostname is resolved.

**NAT**

* Show the three-way TCP handshake for a connection between client and website as seen by tcpdump at the website, and as seen by tcpdump at the gateway (on the LAN). Make sure you can see the IP addresses and port numbers used in the connection!
* Draw a diagram showing how NAT is used between client and website, similar to [this diagram](https://witestlab.poly.edu/blog/content/images/2017/03/gateway-nat-2.svg) but with the IP addresses, hostnames, and ports from *your* experiment.