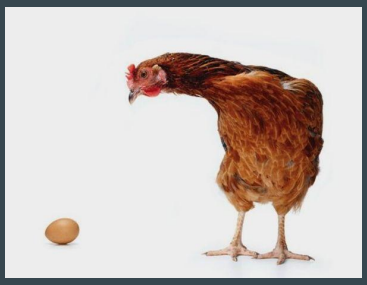
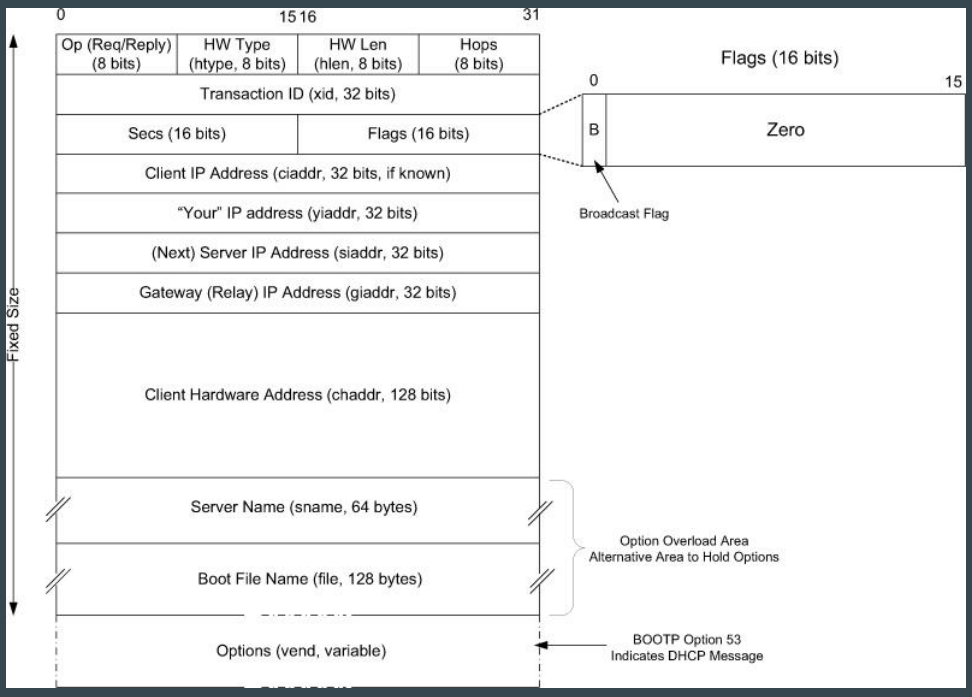
DHCP

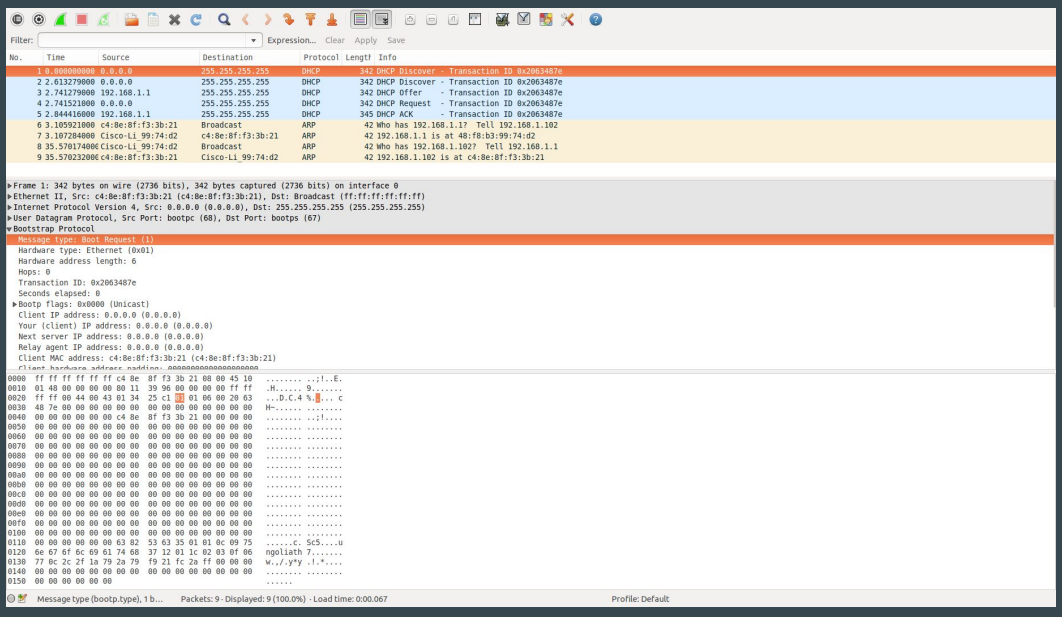
- Dynamic Host Configuration Protocol
- Evolved from BOOTP
- “Chicken and egg” problem: how do you speak IP and acquire:
  - an IP address
  - a “default” gateway - router to the Internet
  - a nameserver
  - ...



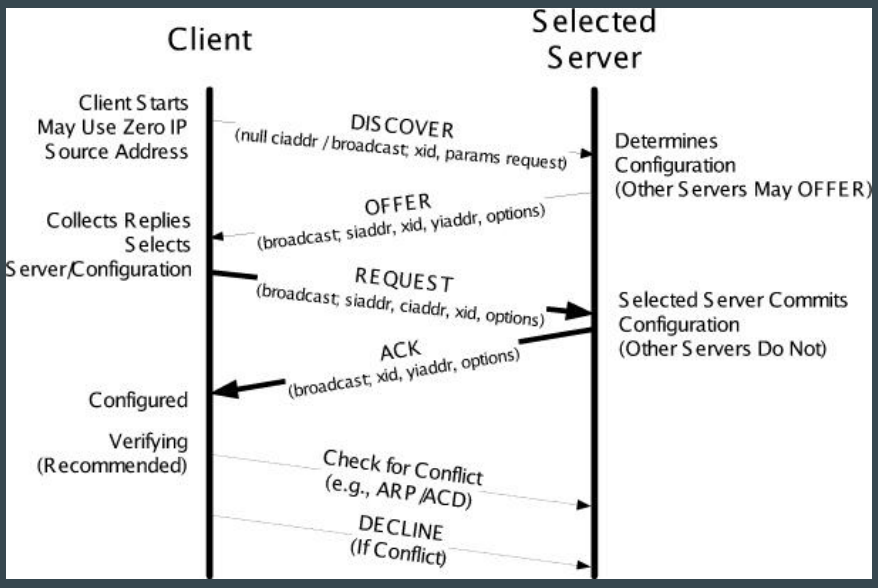

0
15 16
31
Op (Req/Reply) (8 bits)
HW Type (htype, 8 bits)
HW Len (hlen, 8 bits)
Hops (8 bits)
Transaction ID (xid, 32 bits)
Secs (16 bits)
Flags (16 bits)
Client IP Address (ciaddr, 32 bits, if known)
"Your" IP address (yiaddr, 32 bits)
(Next) Server IP Address (siaddr, 32 bits)
Gateway (Relay) IP Address (giaddr, 32 bits)
Client Hardware Address (chaddr, 128 bits)
Server Name (sname, 64 bytes)
Boot File Name (file, 128 bytes)
Options (vend, variable)
Fixed Size
Flags (16 bits)
0
15
B
Zero
Broadcast Flag
Option Overload Area
Alternative Area to Hold Options
BOOTP Option 53
Indicates DHCP Message

You can send out a packet with all 1s for source ip and mac (called DHCP discover) to get a respose from the lan telling you each dchp server's mac address. DHCP sits inside UDP. We then can send a request to this thing whose mac address we know and they will respond with an available ip address. From this we can know that an ipaddress is available and bind to it. Then, for some reason an ARP request is sent to the dhcp server we were talking to just now. We advertise our ip and mac addresses as a bind. Basically the ARP request is you checking that it is ok to attempt to bind that ip address to your mac address (checking for conflicts).

Wireshark capture



Typical exchange



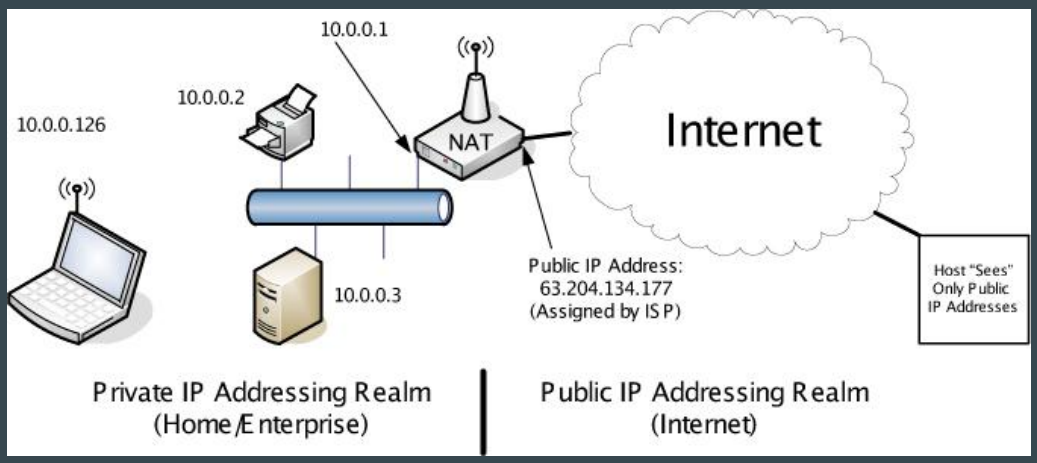
When a client attempts to bind to an ip they need to contact the server a bunch. The client sends out a discover packet telling the server what it needs. The server then responds with what the server has that might match those needs. From these options the client picks on and sends a request for that exact resource. The server then responds with an acknowledgement (could be positive or negative). The client then checks for conflicts using ARP and lets the server know the results (if we have bound to that or not).

This entire exchange, all packets have the same transaction id. The ARP exchanges are done to check that no one is configured with the selected IP address. We can shortcut this process because each offer is associeated with a lease time, renewal time, and rebinding time (lease time is greatest, renewal is least). When we already have an IP we just need to renew it which is much shorter.

Each packet has something called a magic cookie. This is used to determine what kind of UDP packet it is (look into stun packets if you are curious).

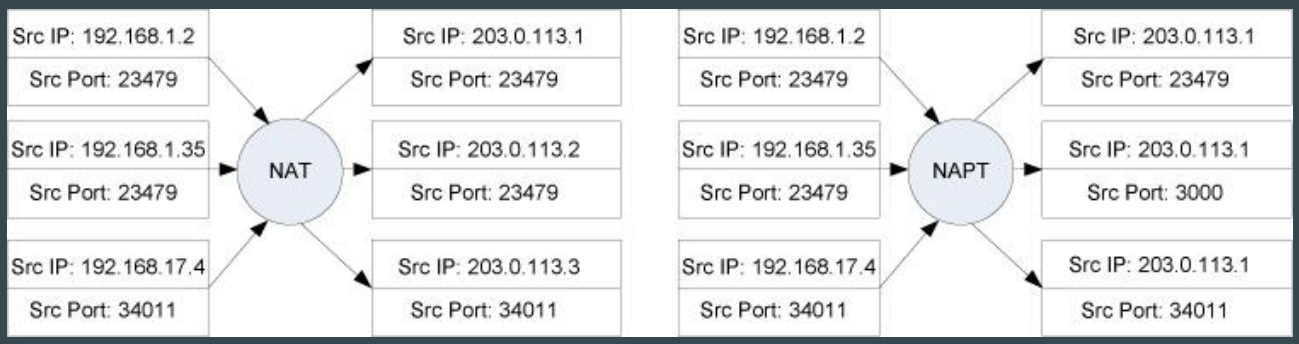
NAT

- Network Address Translation



The idea is simple the application sucks. You have your local network that is not visible to the outside world. Packets with 10.0/8 should be reserved for local addresses (there should be no public addresses with this prefix). When you transition between local and public ipaddresses you have to multiplex. This is because many local ips will correspond to the same public ipaddress. The reverse is very hard since we only have one public ip address that has to route to multiple local addresses.

Basic NAT & NAPT (NA Port T)



We want to have as many public ip addresses as sessions we want to have open to the internet. This way you have a public facing ip address that you can map each local address onto.

Basic NAT

- Rewrite IP addresses only
- Available # publicly routable addresses ≥ # internal hosts

NAPT

- Use IDs from higher-layer than IP to mux/demux
  - UDP/TCP port #s
  - ICMP query ID
  - ...
- Private IP address ranges: 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16

NAPT + TCP

- Will discuss later, once we introduce TCP
- Broadly, NAPT device has to detect:
  - Connection-setup and teardown
  - Every TCP segment that corresponds to connection

NAPT + UDP

- Potential challenges
  - No notion of connection in UDP
    - Adopt notion of a “session.” Guess when it starts, timer for when it ends.
  - Session identified by two-tuple only. Not 4-tuple as in TCP
    - *Port-preservation*:
      - Retain port # chosen by internal host
      - Use that as index to table to determine internal host for packets received from outside
      - Discussed more in a couple of slides in the context of translation “behaviour.”

Basically we try to use addresses from multiple layers to assist in multiplexing. We'll talk about tcp later.

Say we receive a packet that we have never seen before. We create an internal mapping from the public address to internal address. This mapping is maintained in memory only for the life time of the session. The session is essentially initialized when the internal host sends the first packet. From there a timer starts which allows us to assume that the session will be alive for 2 minutes (its super imprecise and can be configured). This is essentially a hole punched in your NAT device where we have assigned something and dont need to check. Everytime you see this kind of packet (same addresses and ports) it refreshes the session.

We want to retain the port number chosen by the internal host. Basically any mapping between addresses will be the port number for the public address. If two hosts chose the same internal port shit breaks, which is why we do conflict checking.

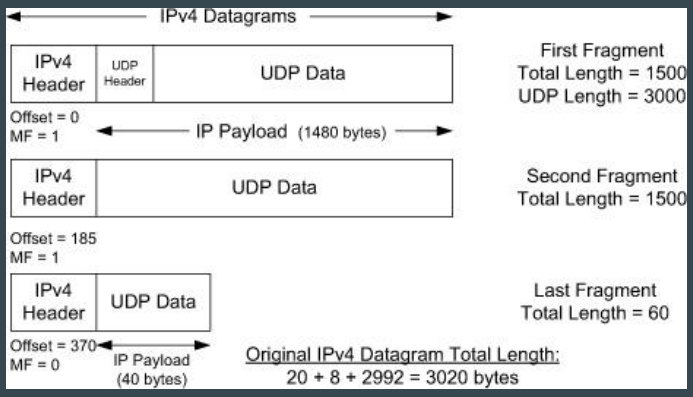
NAPT + UDP, mapping timer

- 1 timer per UDP session
- Expiry time: ≥ 2 min, recommended: 5 min.
- “Refresh behaviour” - when is timer reset
  - Required to be true for outgoing packets

For every device on the network you need to maintain a timer. When the internal host sends a packet corresponding to that session it should be refreshed. There is debate about whether this should happen when the external host sends a packet corresponding to the session. It could be a security flaw to do so, but not doing it might break applications with chatty external hosts and quite internal hosts.

NAPT + UDP + IP Fragmentation

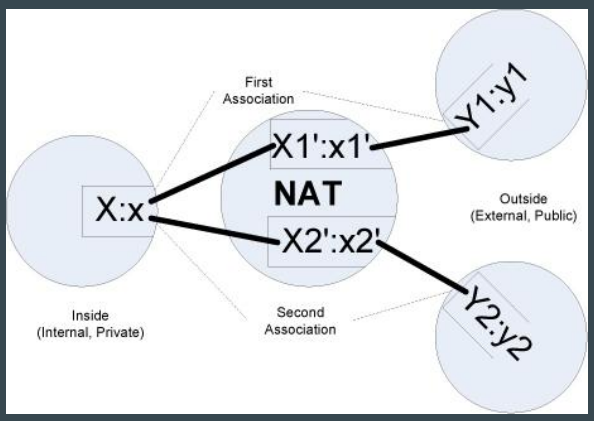
- Fragmentation cannot coexist with NAPT
  - UDP header not available for later fragments to be mapped



IP fragmentation fucks things up. Fragments might be missing the udp header which means that we don't have the ip addresses required for routing. We could make the NAT reassemble fragments but this is annoying and inefficient to do. What we ended up doing is that we don't allow them to exist at the same time.

Translation “behaviour”

What source-ip, source-port should be in packet that exits to Internet?



Translation behaviour, contd.

Behavior Name	Translation Behavior	Filtering Behavior
Endpoint-independent	*X1′:x1′* = *X2′:x2′* for all *Y2:y2* (required)	Allows any packets for *X1:x1* as long as any *X1′:x1′* exists (recommended for greatest transparency)
Address-dependent	*X1′:x1′* = *X2′:x2′* iff *Y1* = *Y2*	Allows packets for *X1:x1* from *Y1:y1* as long as *X1* has previously contacted *Y1* (recommended for more stringent filtering)
Address- and port-dependent	*X1′:x1′* = *X2′:x2′* iff *Y1:y1* = *Y2:y2*	Allows packets for *X1:x1* from *Y1:y1* as long as *X1* has previously contacted *Y1:y1*

- Required behaviour for TCP/UDP: endpoint-independent.

Translation behavior = how you rewrite the packet filetering behavior = how you determine which packets are allowed through

Endpoint independent behavior - For each thing you will have one mapping on the NAT device regardless of who the endpoint is. All packets destined for a address and port is allowed through if that device exists. This makes attacking easy because it doesn't even have to spoof its source ip address.

Address dependent behavior - There is a mapping for each thing, but things can share a mapping if they have the same endpoint ip (combine on ip address).

Address and port dependent behavior - There is a unique mapping for each endpoint value (combine on ip adress and port).

Other issues

- Pairing
  - Internal host A establishes several connections/sessions
  - Should all be associated with the same public IP address?
  - ‘Yes’ answer (i.e., “pairing”) is recommended
- Port-preservation
- Port-parity
  - Even numbered internal port mapped to even numbered externally-visible port

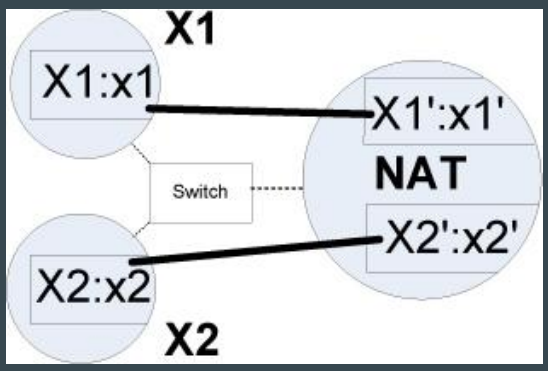
Pairing - when an internal host has several connections should be associated all of them with the same IP, we think yes

We run into problems when we need to preserve ports (conflicts and such).

Similarly port parity causes problems when people come up with crazy port mapping schemes that are hard to maintain.

More issues - hairpinning

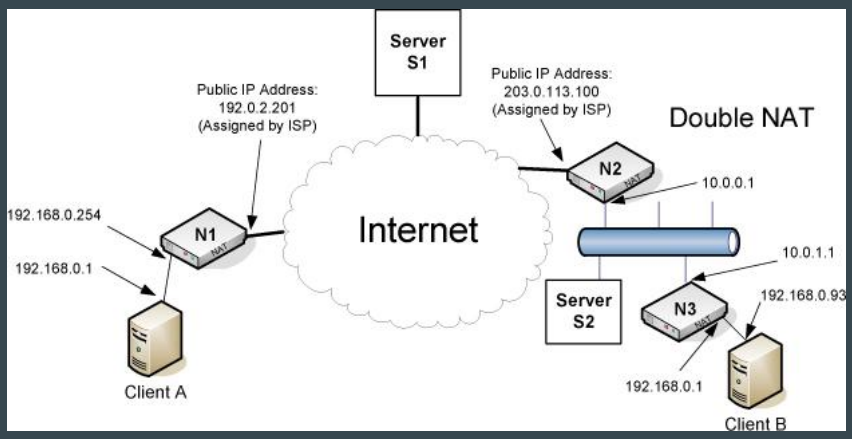
- When we run a server within NAT-ed network...
- In example below, should X2:x2 see as source, X1:x1 or X1':x1'?



When you're running a server behind a NAT device and something contacts it on a public port it should see public port of the device it connects to. Basically if you address me using my public port you should show me your public port. This is called hairpinning.

Hole-punching

- Two clients use server to discover and communicate directly with one another



There is a protocol called TURN that helps with running a server behind NAT.