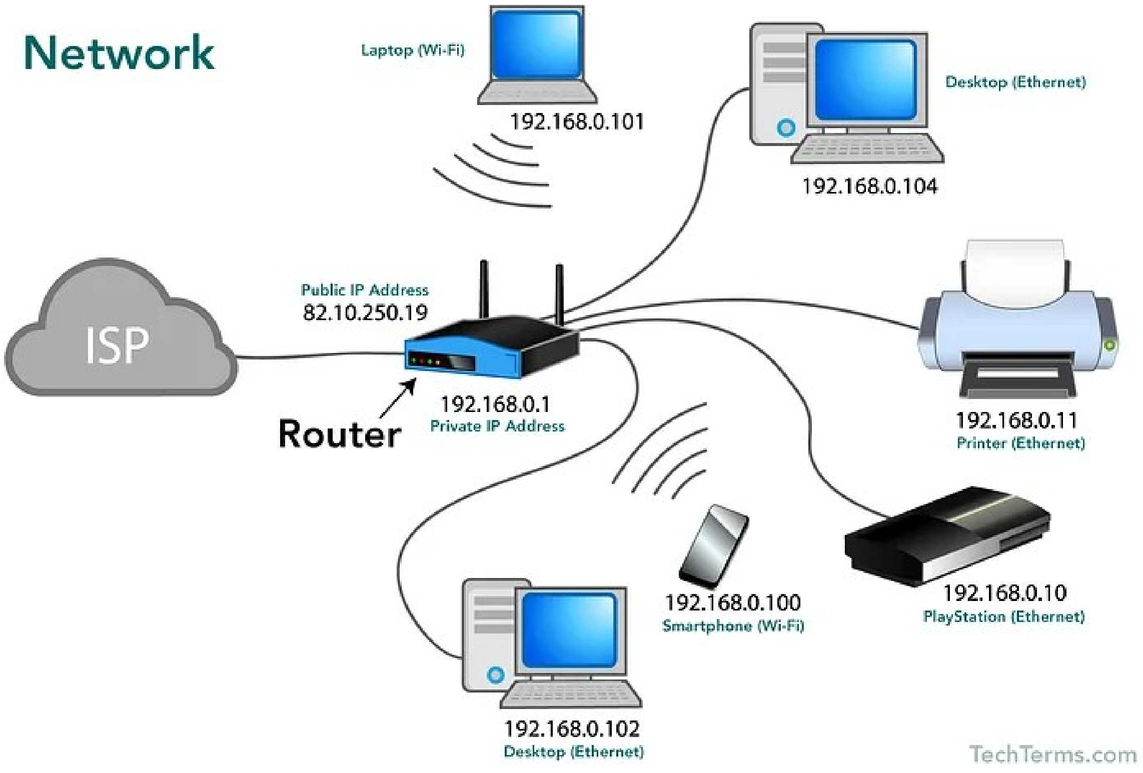
NAT, UDP hole punching, WebRTC

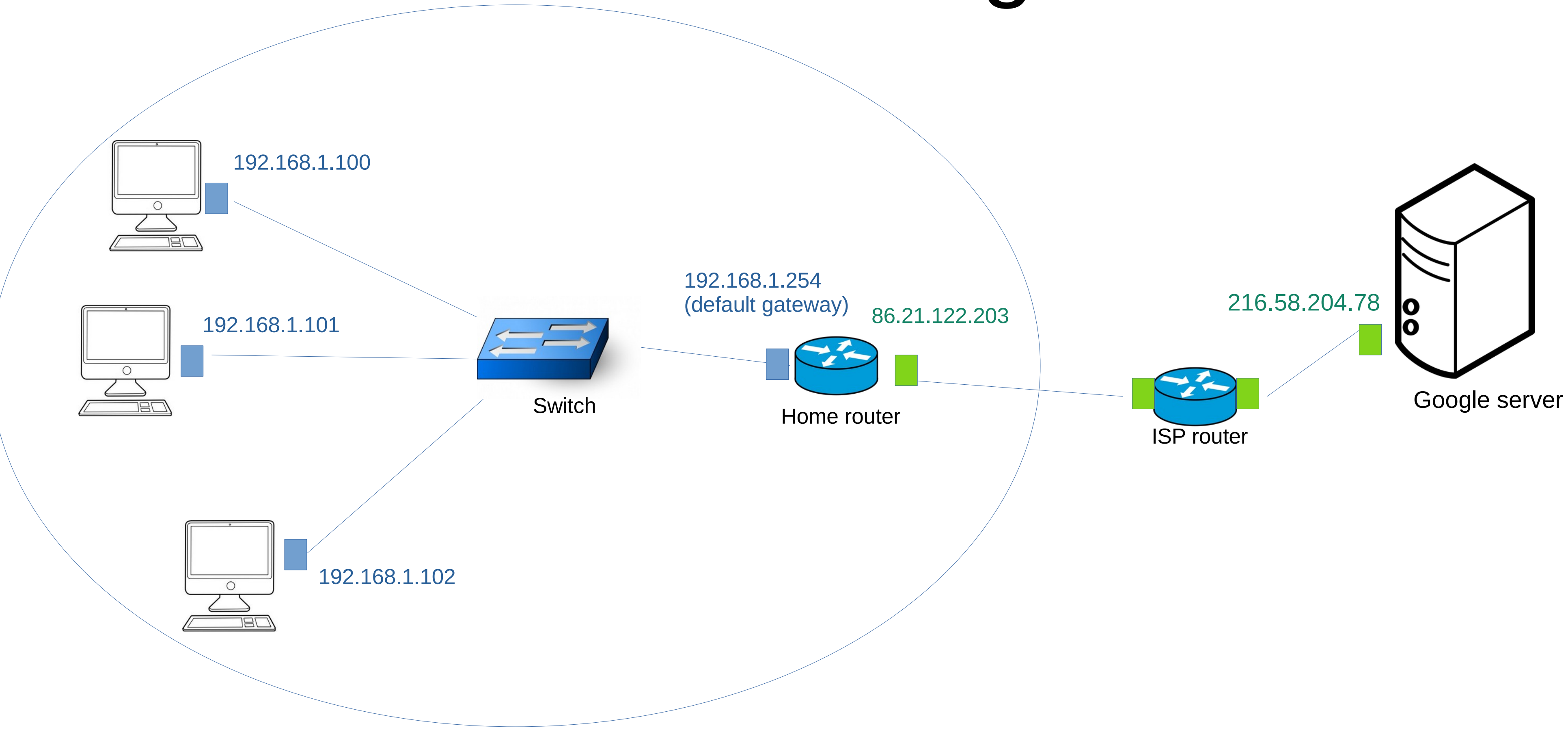
Network Address Translation (NAT)

- Only ~4.3 billion IPv4 address
- IPv6 is dumb or something idk why people don't use it
- The solution:
  - All devices on a local network have the same public IP address
  - Router must forward traffic to devices on the network

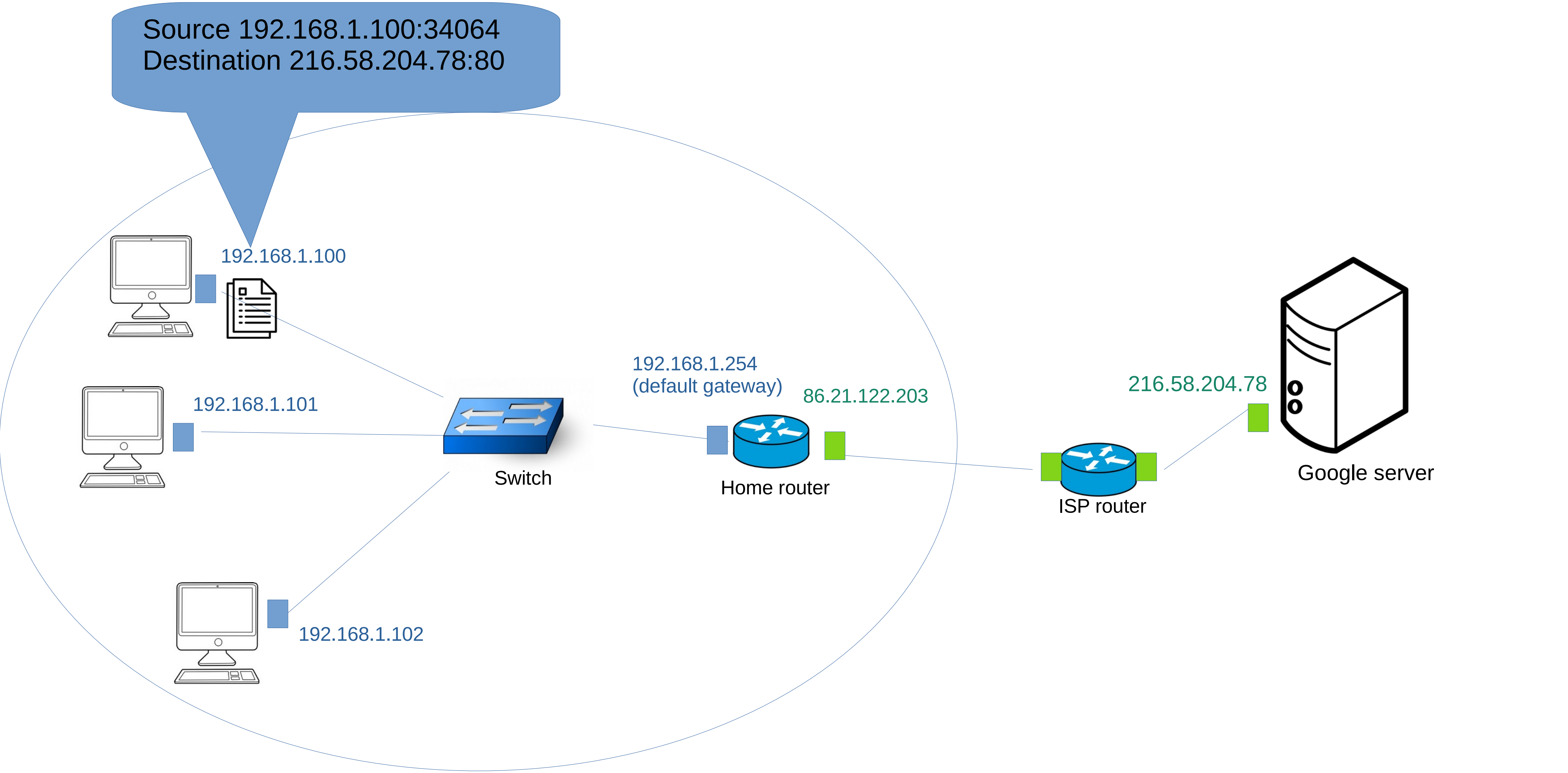


https://medium.com/yavar/nat-network-address-translation-2e5bd3c550f9

Network configuration

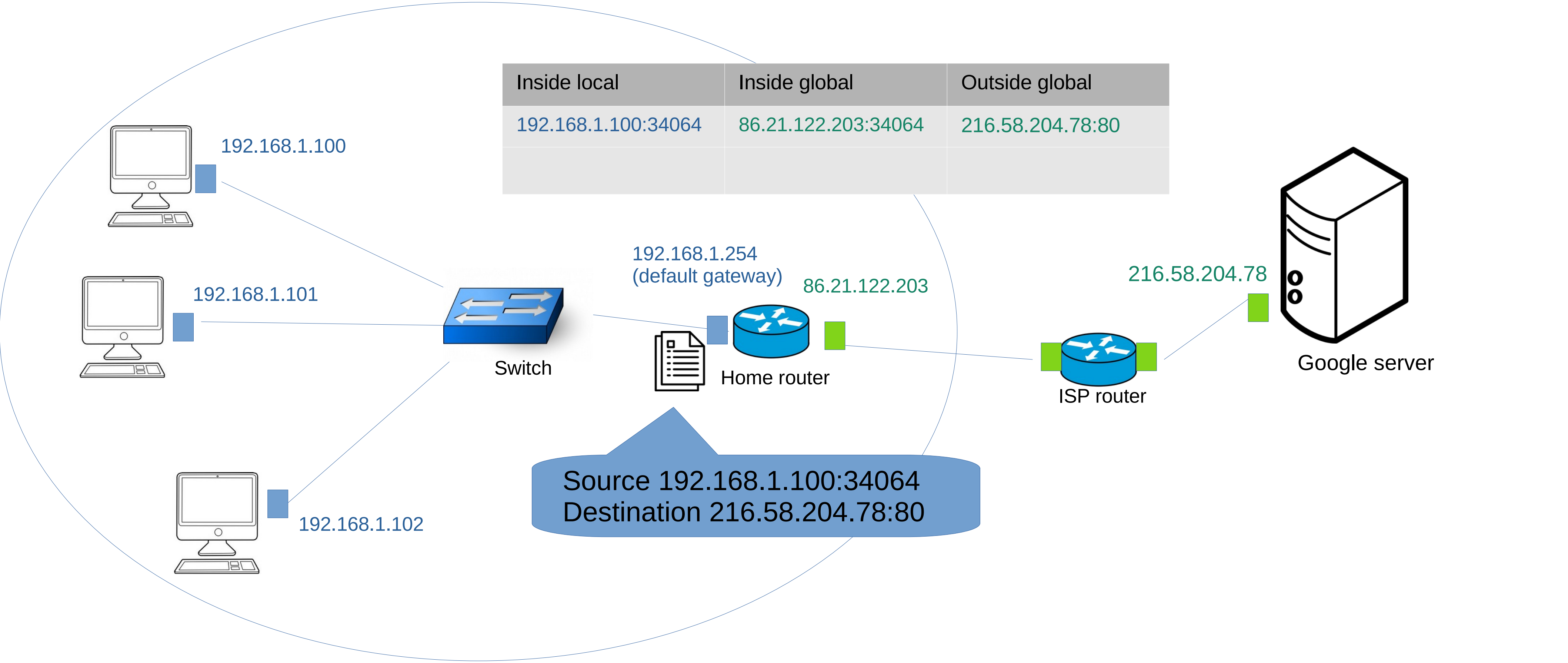


- Packets to 192.168.1.* go to the device directly
- Everything else is sent to the default gateway for forwarding

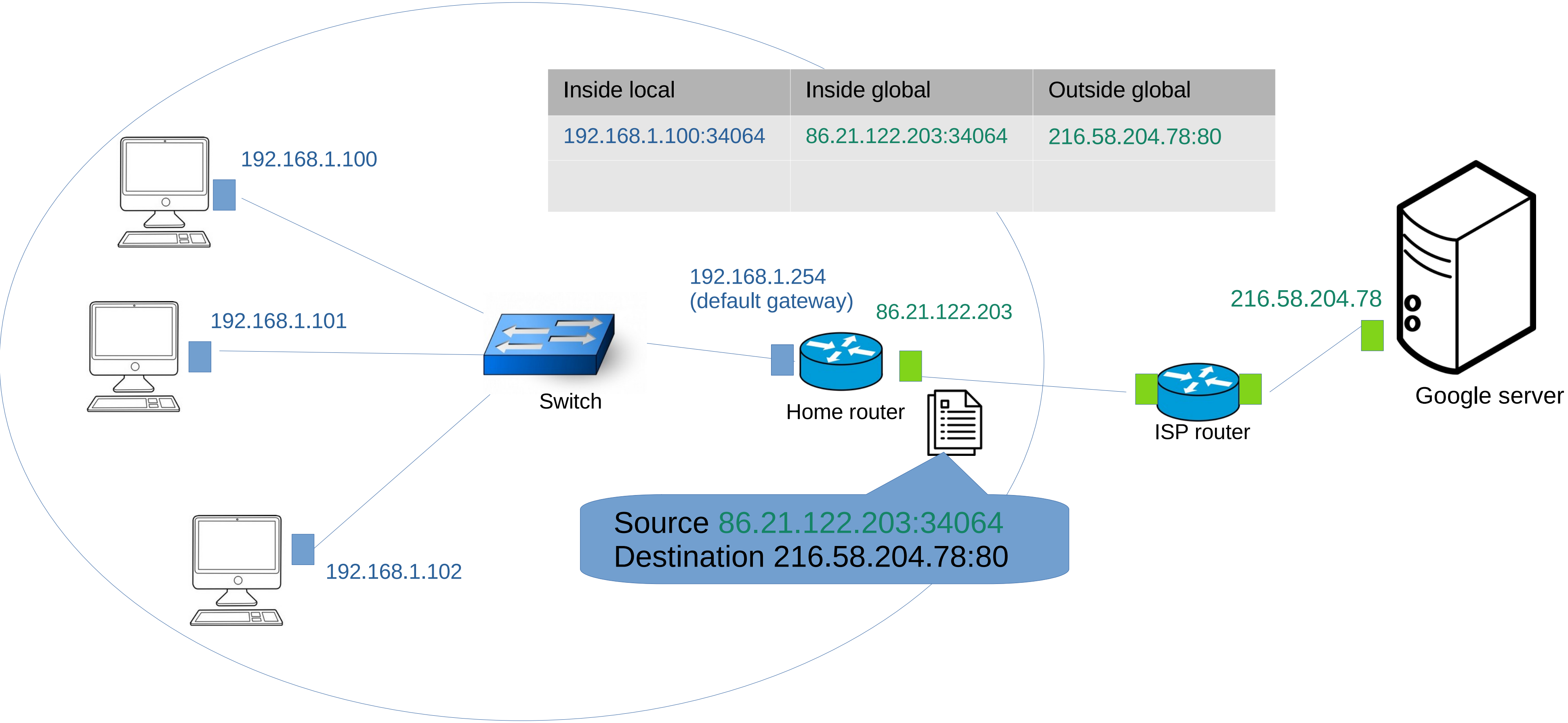


Source 192.168.1.100:34064
Destination 216.58.204.78:80

Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

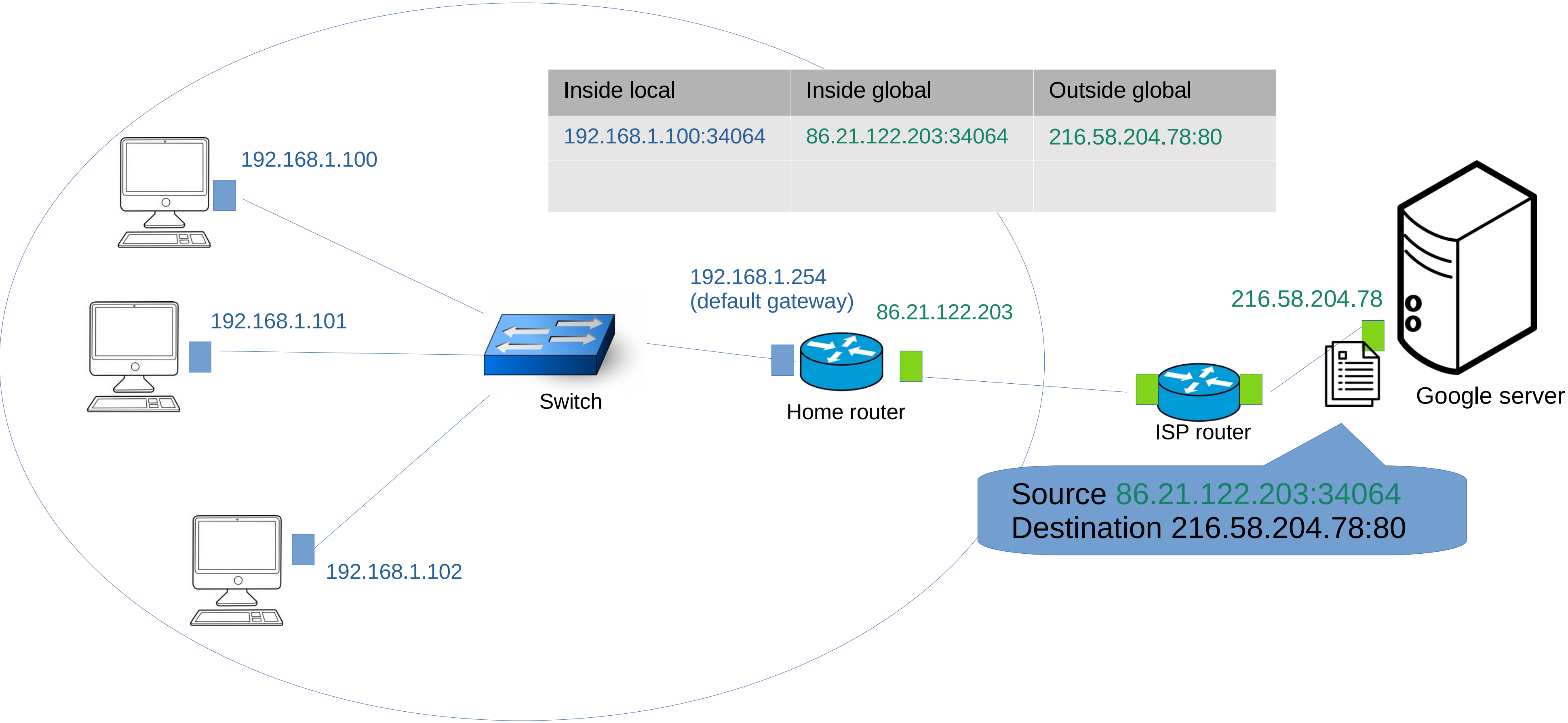


Source 192.168.1.100:34064
Destination 216.58.204.78:80



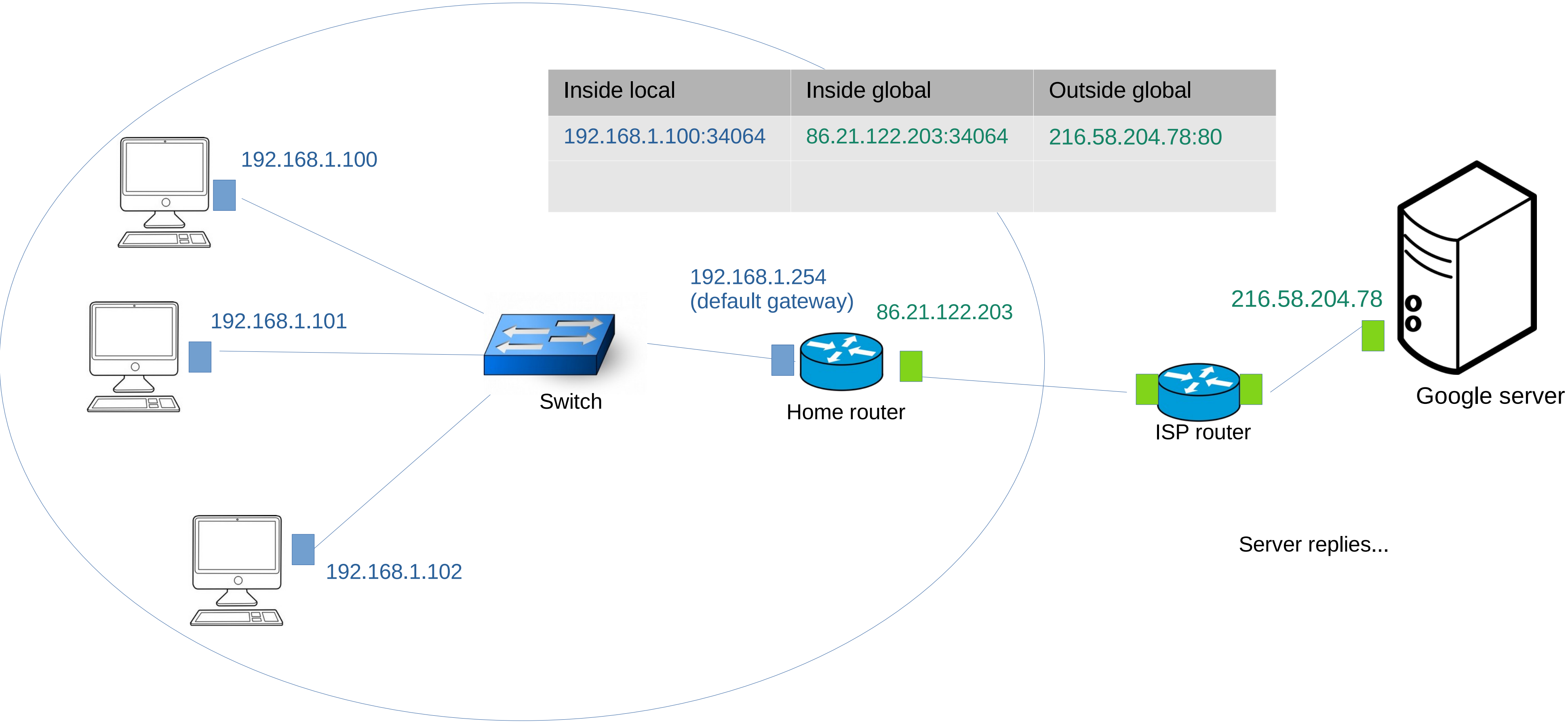
Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

Source 86.21.122.203:34064
Destination 216.58.204.78:80



Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

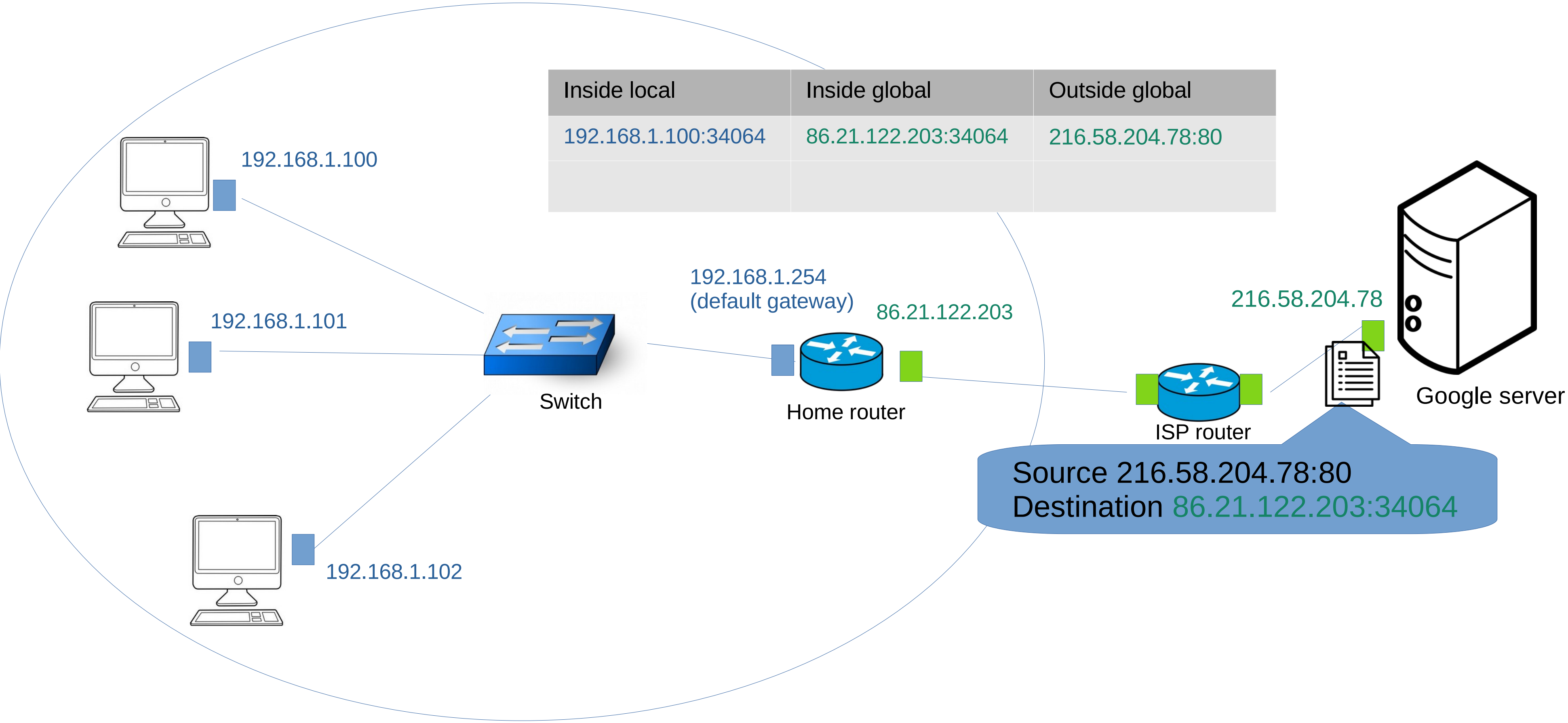
Source 86.21.122.203:34064
Destination 216.58.204.78:80



Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

Server replies...

Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80



192.168.1.100

192.168.1.101

192.168.1.102

Switch

192.168.1.254
(default gateway)

86.21.122.203

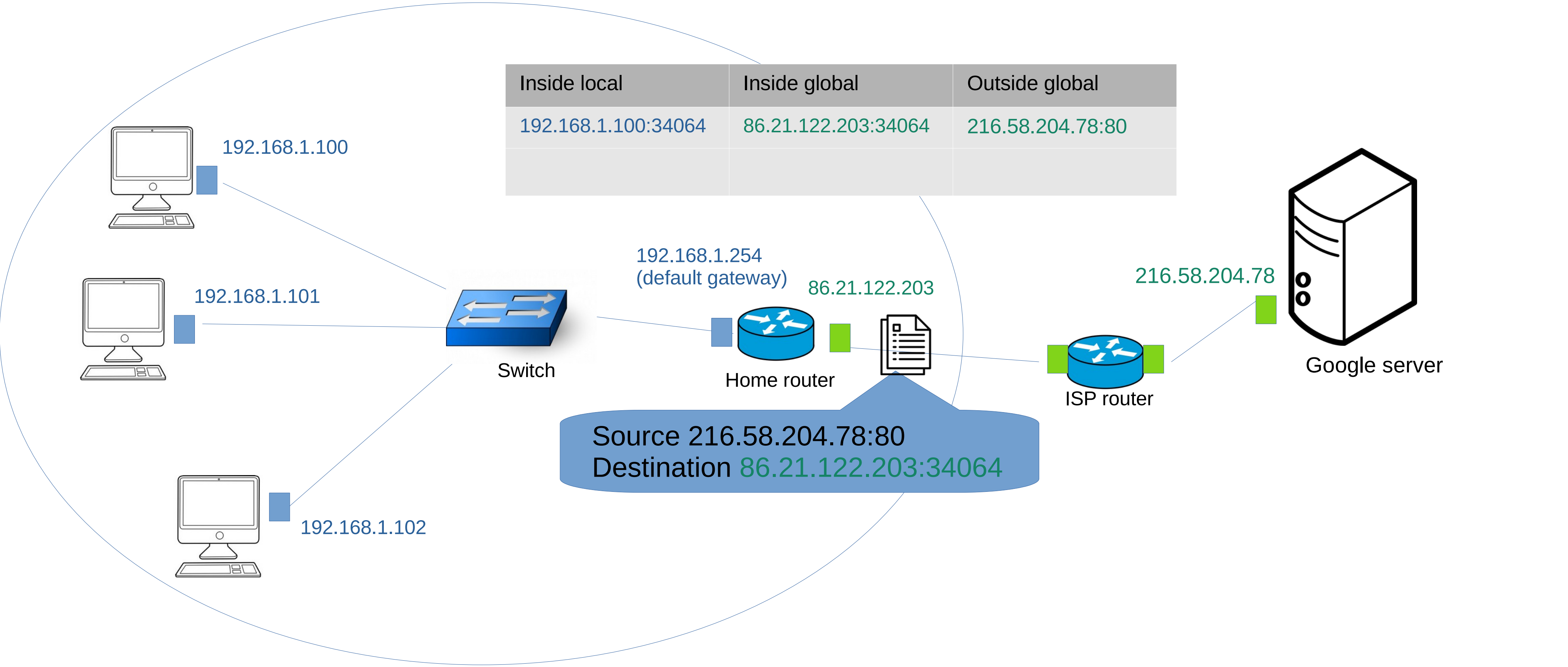
Home router

ISP router

216.58.204.78

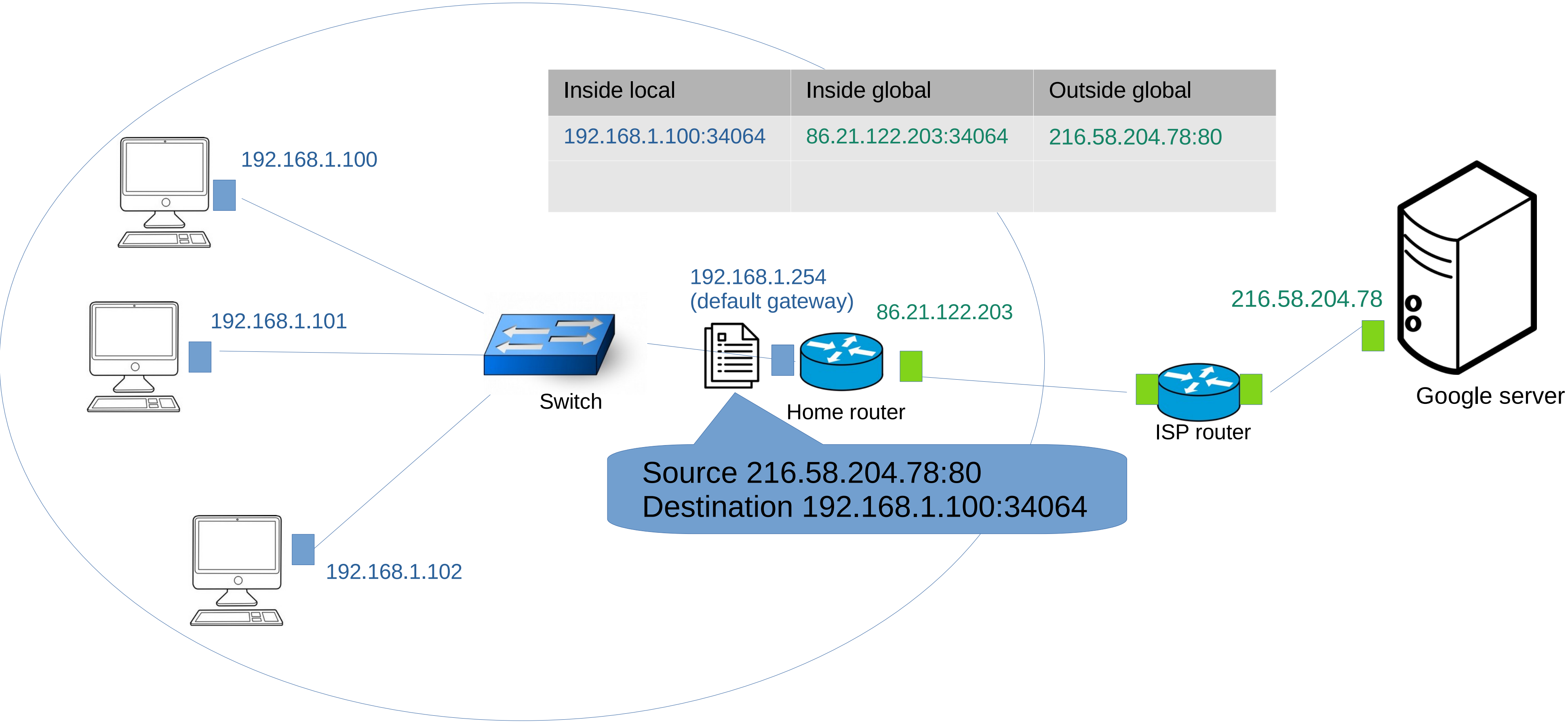
Google server

Source 216.58.204.78:80
Destination 86.21.122.203:34064



Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

Source 216.58.204.78:80
Destination 86.21.122.203:34064

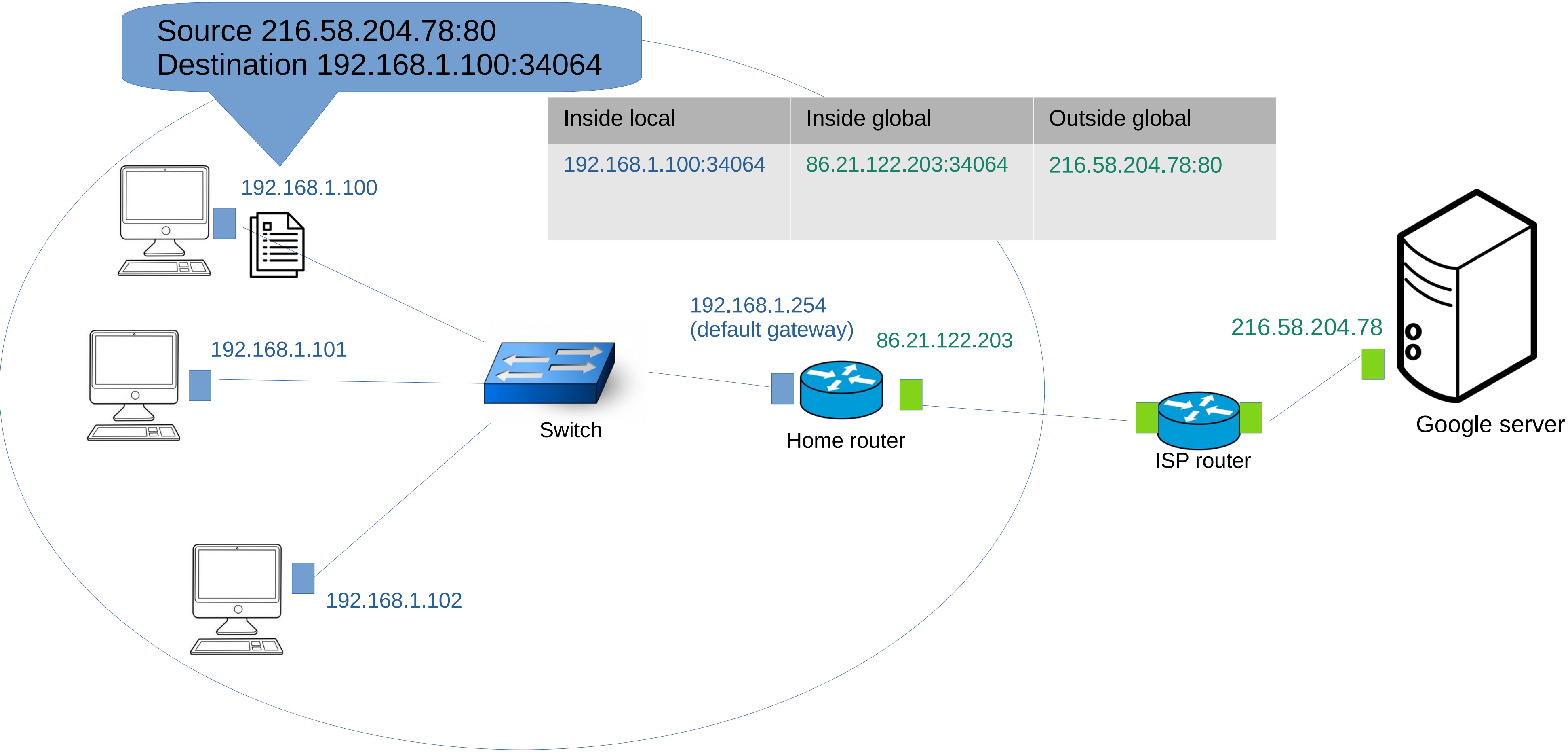


Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80

Source 216.58.204.78:80
Destination 192.168.1.100:34064

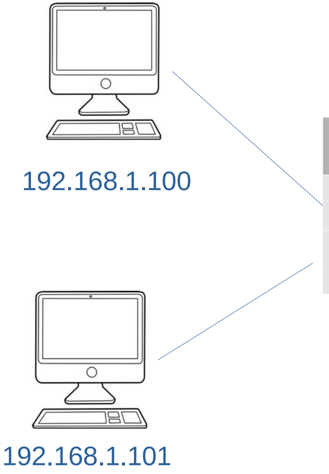
Source 216.58.204.78:80
Destination 192.168.1.100:34064

Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80



Network Address Translation (NAT)

- If two computers are using the same inside local port number, they will need to be mapped to different inside global ports

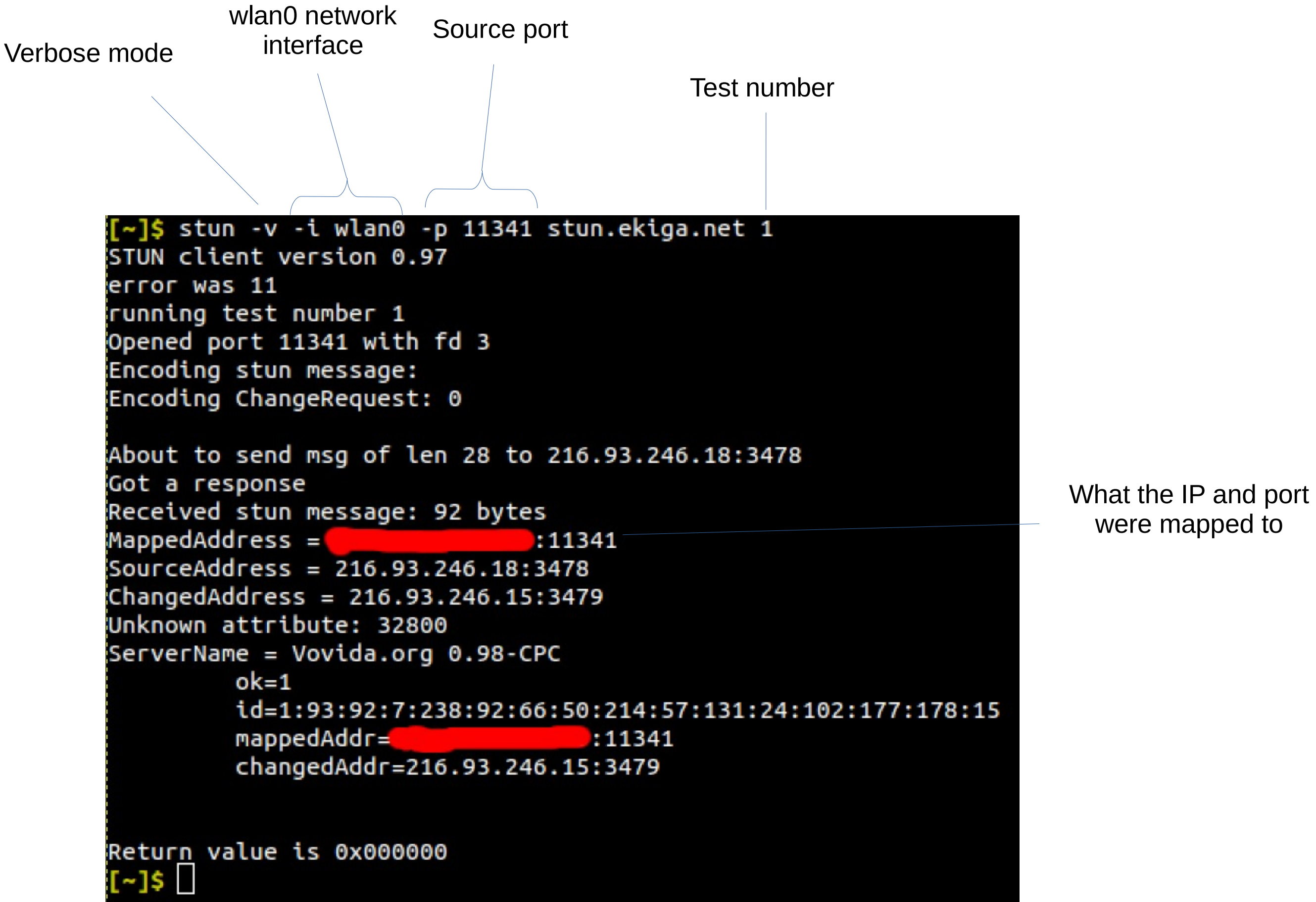


Inside local	Inside global	Outside global
192.168.1.100:34064	86.21.122.203:34064	216.58.204.78:80
192.168.1.101:34064	86.21.122.203:12453	216.58.204.78:80

- Some routers change ports by default
- In general completely up to the router how it wants to map ports

Session Traversal Utilities for NAT (STUN)

- A protocol to work out how the router is translating a source address
- Send a UDP packet to a STUN server (e.g. stun.ekiga.net)
- STUN server replies with what it saw for the source IP and port

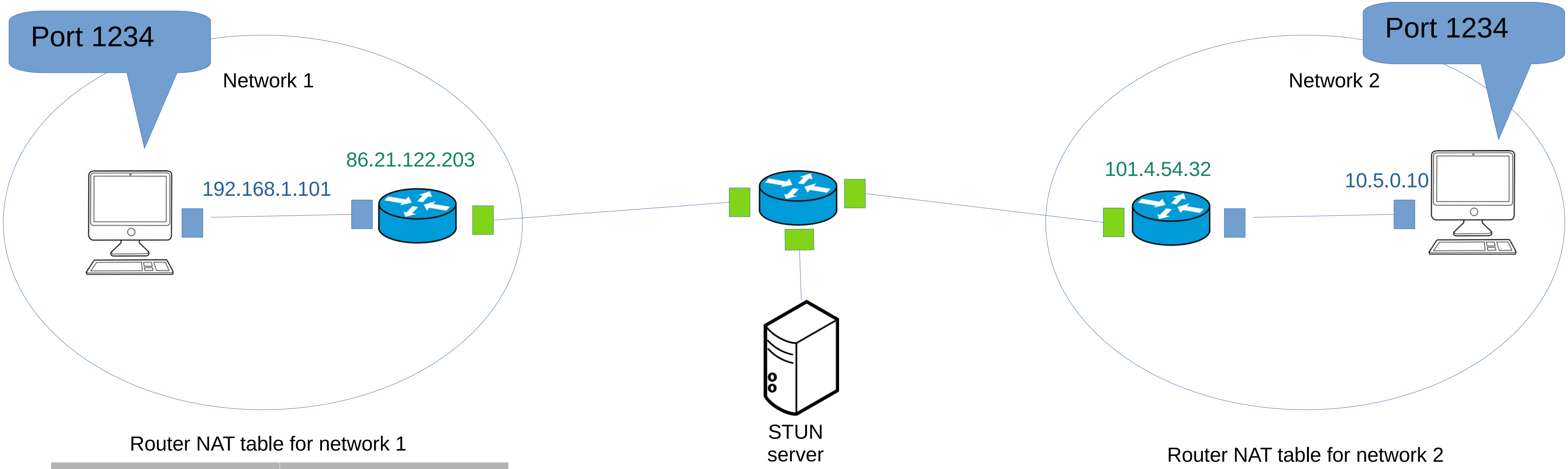


```
[~]$ stun -v -i wlan0 -p 11341 stun.ekiga.net 1
STUN client version 0.97
error was 11
running test number 1
Opened port 11341 with fd 3
Encoding stun message:
Encoding ChangeRequest: 0

About to send msg of len 28 to 216.93.246.18:3478
Got a response
Received stun message: 92 bytes
MappedAddress = [redacted]:11341
SourceAddress = 216.93.246.18:3478
ChangedAddress = 216.93.246.15:3479
Unknown attribute: 32800
ServerName = Vovida.org 0.98-CPC
         ok=1
         id=1:93:92:7:238:92:66:50:214:57:131:24:102:177:178:15
         mappedAddr=[redacted]:11341
         changedAddr=216.93.246.15:3479


Return value is 0x000000
[~]$
```

UDP Hole Punching

- How can we create a peer-to-peer connection with all this NAT stuff????
- A solution: **UDP hole punching**

1) Make a request to a STUN server to find the mapped address
2) Send mapped address to peer by any method
3) Both peers send UDP packets from their source port to the other’s mapped address



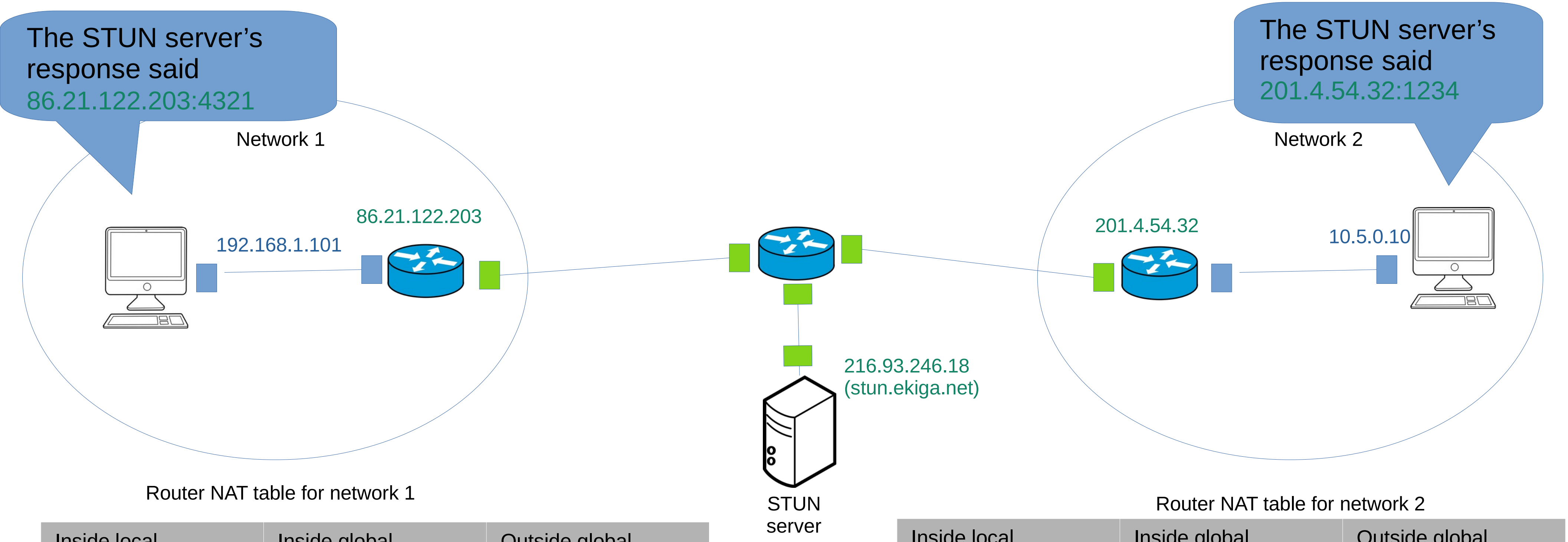
Router NAT table for network 1

Inside local	Inside global
192.168.1.100:34064	86.21.122.203:34064

Router NAT table for network 2

Inside local	Inside global
192.168.1.100:34064	86.21.122.203:34064

1) **Make a request to a STUN server to find the mapped address**
2) Send mapped address to peer
3) Both peers send UDP packets from their source port to the other’s mapped address



The STUN server’s response said 86.21.122.203:4321

The STUN server’s response said 201.4.54.32:1234

Router NAT table for network 1

Inside local	Inside global	Outside global
192.168.1.101:1234	86.21.122.203:4321	216.93.246.18:3478
		86.21.122.203:4321

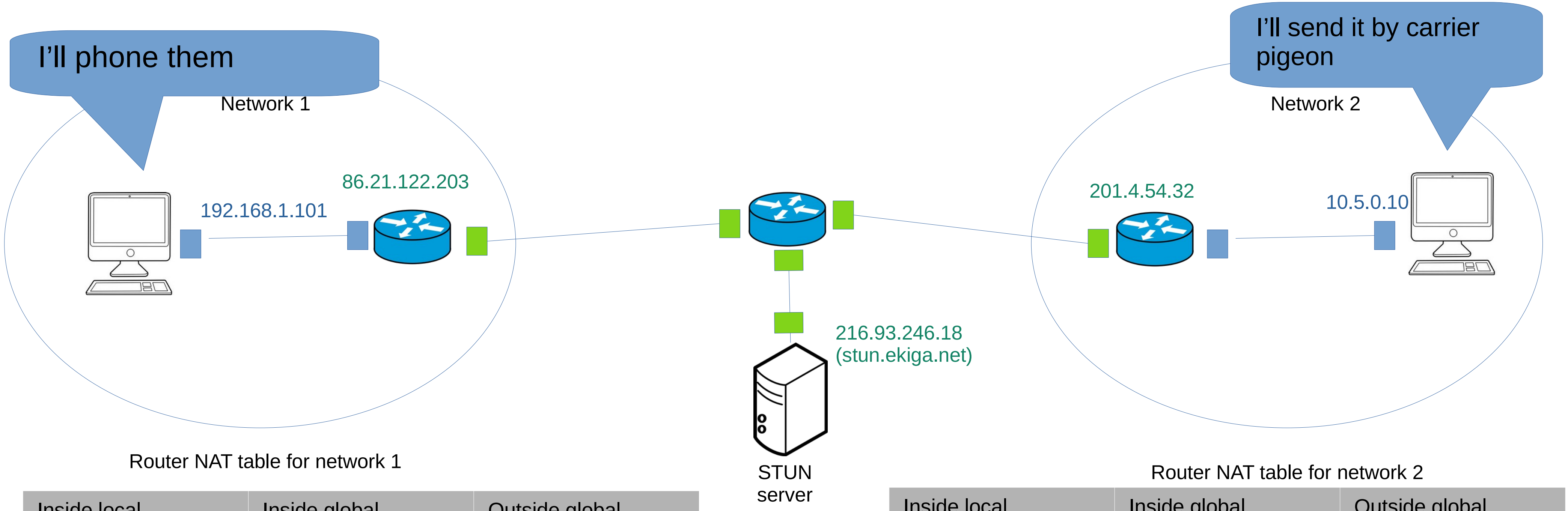
Router NAT table for network 2

Inside local	Inside global	Outside global
10.5.0.10:1234	201.4.54.32:1234	216.93.246.18:3478

1) Make a request to a STUN server to find the mapped address

2) Send mapped address to peer

3) Both peers send UDP packets from their source port to the other's mapped address



Router NAT table for network 1

Inside local	Inside global	Outside global
192.168.1.101:1234	86.21.122.203:4321	216.93.246.18:3478

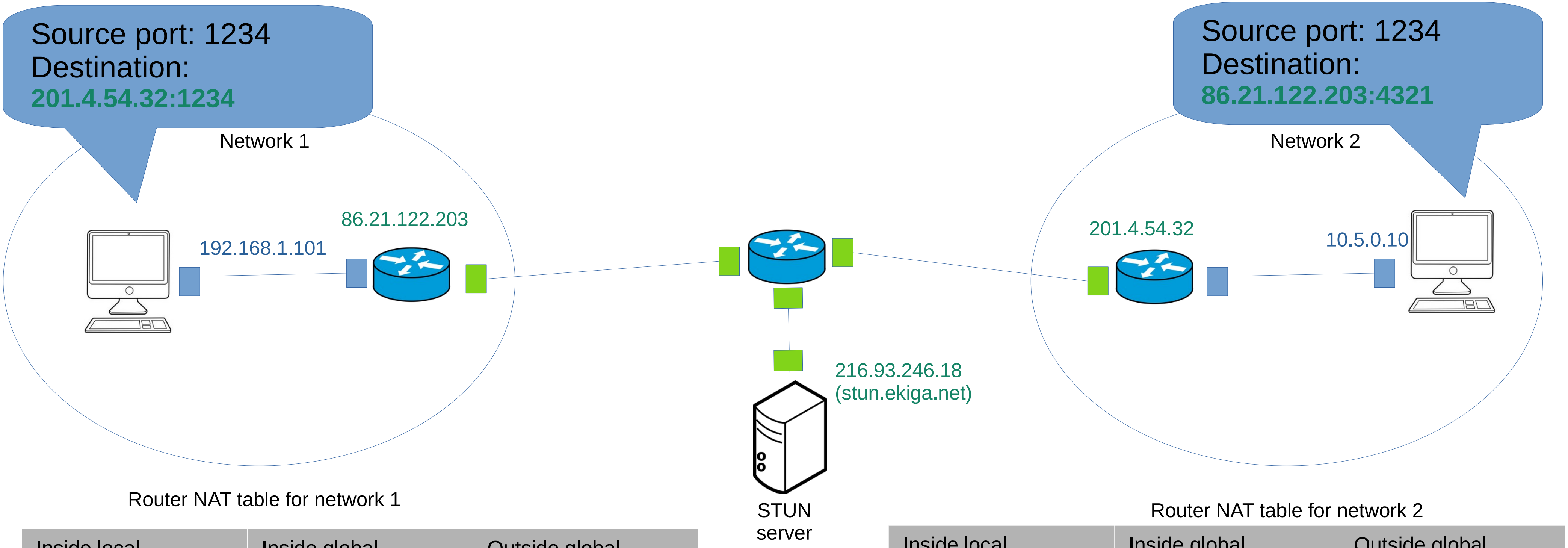
Router NAT table for network 2

Inside local	Inside global	Outside global
10.5.0.10:1234	201.4.54.32:1234	216.93.246.18:3478

1) Make a request to a STUN server to find the mapped address

2) Send mapped address to peer

3) Both peers send UDP packets from their source port to the other's mapped address



Source port: 1234
Destination:
201.4.54.32:1234

Source port: 1234
Destination:
86.21.122.203:4321

Router NAT table for network 1

Inside local	Inside global	Outside global
192.168.1.101:1234	86.21.122.203:4321	216.93.246.18:3478
192.168.1.101:1234	86.21.122.203:4321	201.4.54.32:1234

Hope that the mapped address is the same!

Router NAT table for network 2

Inside local	Inside global	Outside global
10.5.0.10:1234	201.4.54.32:1234	216.93.246.18:3478
10.5.0.10:1234	201.4.54.32:1234	86.21.122.203:4321

Hope that the mapped address is the same!

Problems

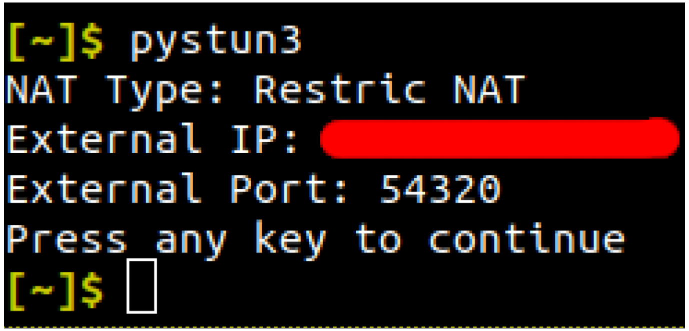
- Packets must be sent periodically so that the NAT table entry does not surpass its TTL
- Have to coordinate creating the connection somehow
- Uses UDP – fast but unreliable
- Doesn’t work with all NAT translation methods...

NAT translation methods

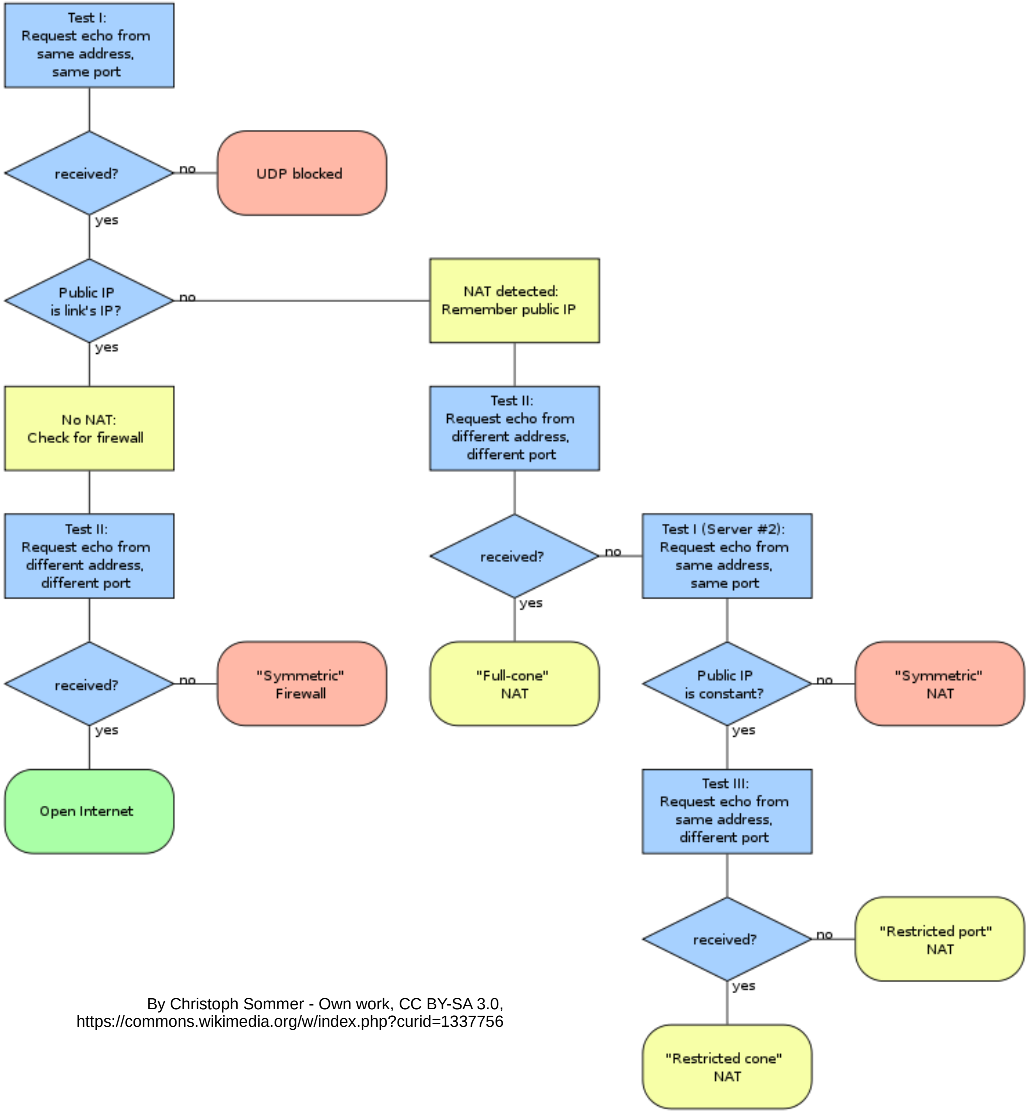
- Full cone
  - Addresses are mapped predictably
- (Address-)restricted cone
  - Addresses are mapped predictably
  - Only accept packets from IPs to which I have already sent a packet
- Port-restricted cone
  - Addresses are mapped predictably
  - Only accept packets from sockets (IP+port) to which I have already sent a packet
- Symmetric
  - Addresses are mapped **differently** depending on the destination
  - **Not possible** to do UDP hole punching on these networks.

NAT translation methods

- NAT translation method can be determined using this flowchart
- pystun3 (a pip package) can do this (https://github.com/talkiq/pystun3)



```
[~]$ pystun3
NAT Type: Restric NAT
External IP: [redacted]
External Port: 54320
Press any key to continue
[~]$ 
```


Test I:
Request echo from same address, same port
received?
no
UDP blocked
yes
Public IP is link's IP?
no
NAT detected: Remember public IP
yes
No NAT: Check for firewall
Test II: Request echo from different address, different port
received?
no
"Symmetric" Firewall
yes
Open Internet
Test II: Request echo from different address, different port
received?
no
yes
"Full-cone" NAT
Test I (Server #2): Request echo from same address, same port
Public IP is constant?
no
"Symmetric" NAT
yes
Test III: Request echo from same address, different port
received?
no
"Restricted port" NAT
yes
"Restricted cone" NAT

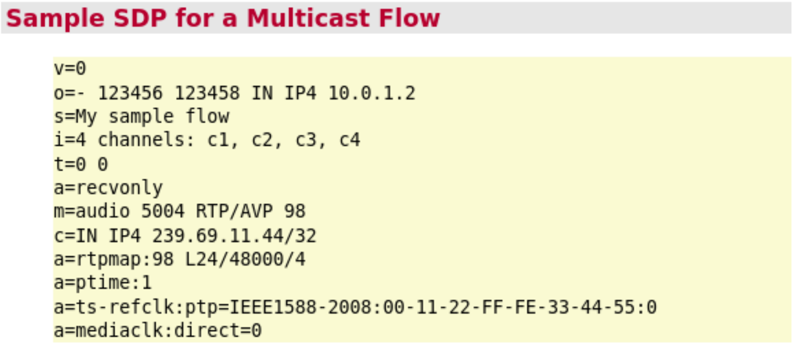
By Christoph Sommer - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1337756

WebRTC – a protocol that basically does all of this

- “Web Real-Time Communication”
- Built into all modern browsers
- Makes requests to STUN servers
- Does UDP hole punching
- Can create connections through relay servers instead if a peer connection is not possible
- Can make ordered UDP connections
- Multiple uses (in addition to creating a standard peer-to-peer socket)
  - Audio and video conferencing, screen sharing, file exchange...

WebRTC – how it works

- Peer A wants to initiate a peer connection to peer B
- Peer A creates a **Session Description Profile (SDP)**
  - Contains information about codecs used, etc
- This SDP is the **offer**
- Peer A sends the offer to peer B (via a third party)



Sample SDP for a Multicast Flow

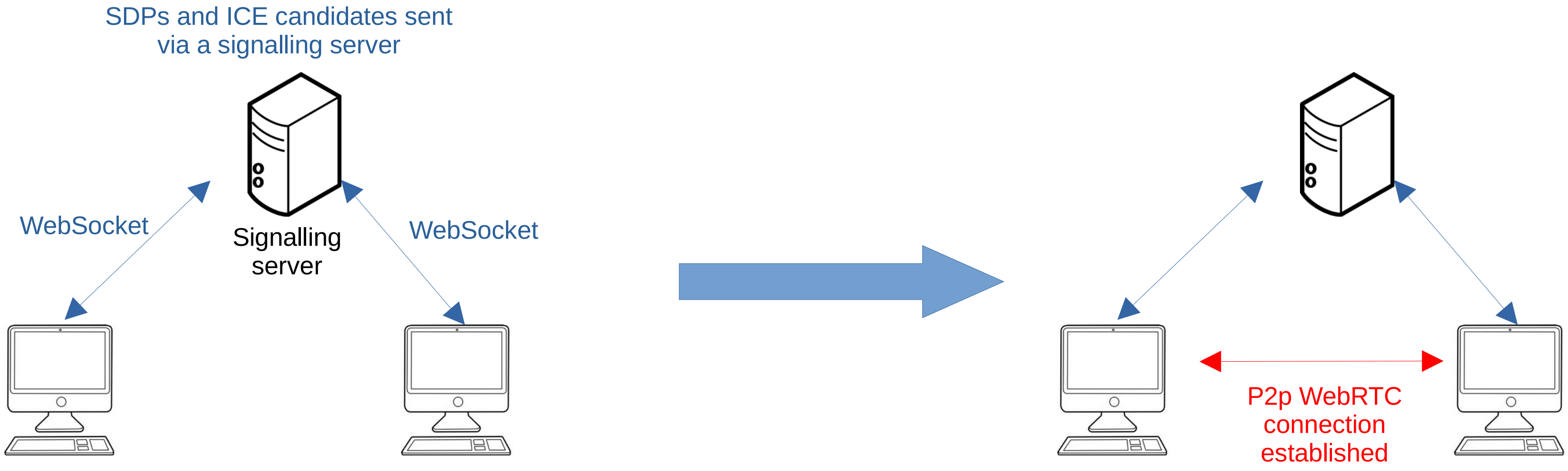
```
v=0
o=- 123456 123458 IN IP4 10.0.1.2
s=My sample flow
i=4 channels: c1, c2, c3, c4
t=0 0
a=recvonly
m=audio 5004 RTP/AVP 98
c=IN IP4 239.69.11.44/32
a=rtpmap:98 L24/48000/4
a=ptime:1
a=ts-refclk:ptp=IEEE1588-2008:00-11-22-FF-FE-33-44-55:0
a=mediaclk:direct=0
```

https://dev.audinate.com/GA/ddm/userguide/1.1/webhelp/content/appendix/sample_sdp_specification.htm

- Peer B creates an SDP **answer** and sends it back to peer A
- Peers A and B also exchange **Internet Connectivity Establishment** (**ICE) candidates**
  - Can refer to open ports on the local network, hole-punched UDP sockets, relay servers…
- The ICE candidates are tested in priority order until a connection is made
  - Local connections first, then UDP hole punching, then relays

Signalling

- Before the peer connection is created the SDPs and ICE candidates must be exchanged
- This process is called **signalling**
- Typically done using an external server to forward the data using WebSockets


SDPs and ICE candidates sent via a signalling server
WebSocket
Signalling server
WebSocket
P2p WebRTC connection established

- WebRTC does not define any specific protocol for signalling