

# UDP and Network Address Translation

Networked Systems (H)  
Lecture 8

# Lecture Outline

- The UDP protocol and datagram sockets
  - Guidelines for writing UDP-based applications
  - DNS as an example of a UDP-based protocol
- Network Address Translation (NAT)
  - Implications for transport protocols
  - TCP
  - UDP
  - NAT traversal

# UDP and Datagram Sockets

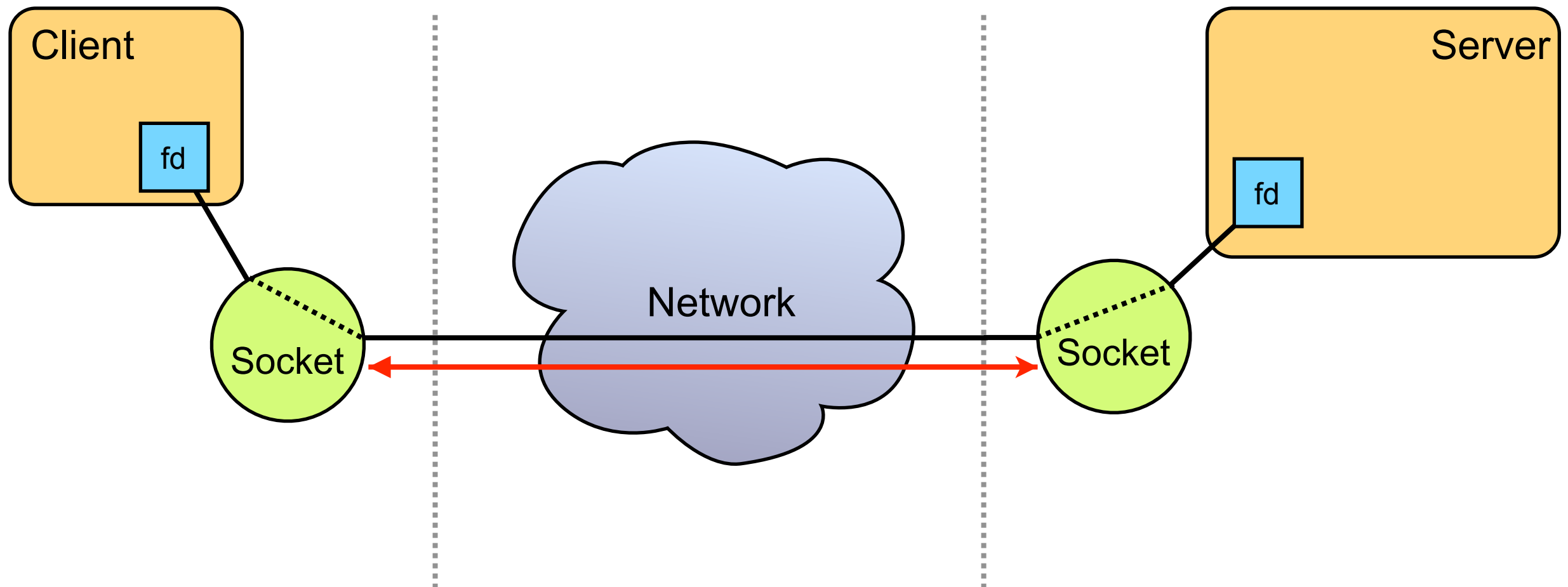
# Using UDP Datagrams

- UDP provides an unreliable datagram service, identifying applications via a 16 bit port number
- UDP ports are separate from TCP ports
- Often used peer-to-peer (e.g., for VoIP), so both peers must `bind()` to a known port
- Create via `socket()` as usual, but specify `SOCK_DGRAM` as the socket type:

```
int    fd;  
...  
fd = socket(AF_INET, SOCK_DGRAM, 0);
```

- No need to `connect()` or `accept()`, since no connections in UDP

# Using UDP Datagrams



```
int fd = socket(...)
```

```
bind(fd, ..., ...)
```

```
sendto(fd, data, datalen, addr, addrlen)
```

```
recvfrom(fd, buffer, buflen, flags, addr, addrlen)
```

```
close(fd)
```

# Sending UDP Datagrams

The `sendto( )` call sends a single datagram. Each call to `sendto( )` can send to a different address, even though they use the same socket.

```
int          fd;
char         buffer[...];
int          buflen = sizeof(buffer);
struct sockaddr_in  addr;
...
if (sendto(fd, buffer, buflen, (struct sockaddr *) addr, sizeof(addr)) < 0) {
    // Error...
}
```

Alternatively, `connect( )` to an address, then use `send( )` to send the data

There is no connection made by the UDP layer, a `connect( )` call only sets the destination address for future packets.

# Receiving UDP Datagrams

The `recv( )` call may be used to read a single datagram, but doesn't provide the source address of the datagram. Most code uses `recvfrom( )` instead – this fills in the source address of the received datagram:

```
int          fd;
char         buffer[...];
int          buflen = sizeof(buffer);
struct sockaddr addr;
socklen_t    addr_len = sizeof(addr);
int          rlen;
...
rlen = recvfrom(fd, buffer, buflen, 0, &addr, &addrlen);
if (rlen < 0) {
    // Error...
}
```

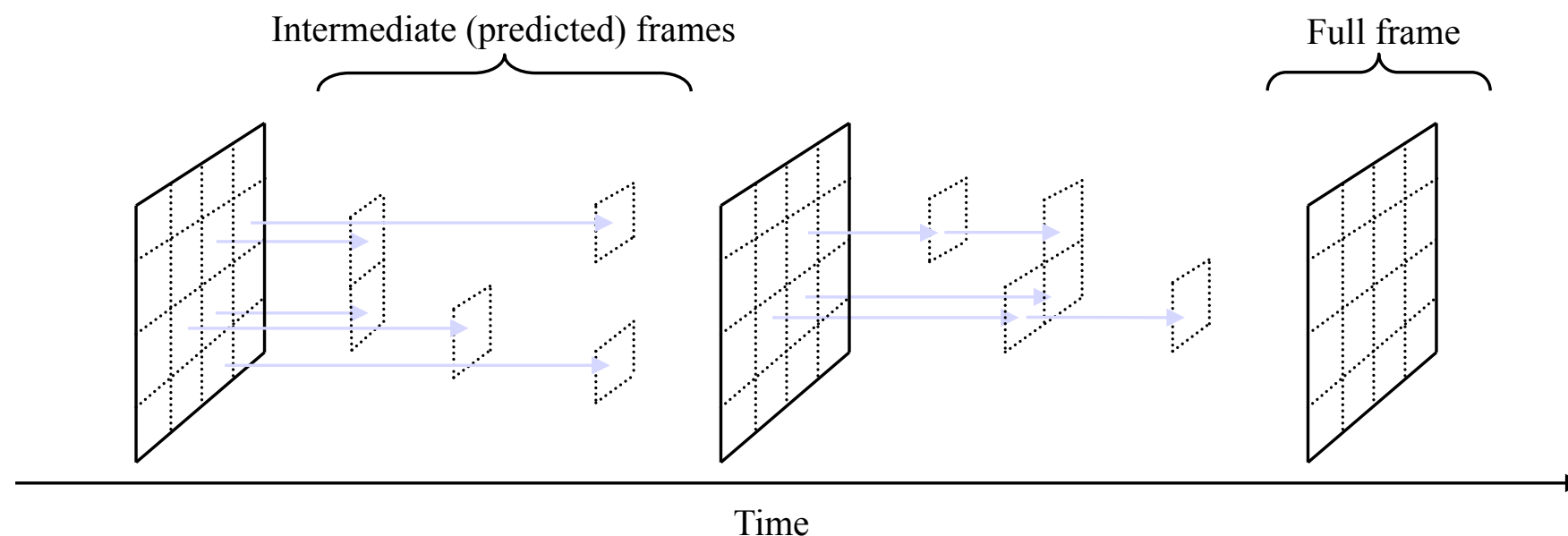
# UDP Framing and Reliability (1)

- Unlike TCP, each UDP datagram is sent as exactly one IP packet (which may be fragmented in IPv4)
  - Each `recvfrom()` corresponds to a single `sendto()`
- But, transmission is unreliable: packets may be lost, delayed, reordered, or duplicated in transit
  - The application is responsible for correcting the order, detecting duplicates, and repairing loss – if necessary
  - Generally requires the sender to include some form of sequence number in each packet sent



## UDP Framing and Reliability (2)

- UDP provides framing – data is delivered a packet at a time – but is unreliable
- Application must organise the data so it's useful if some packets lost
  - E.g. streaming video with I and P frames



D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. Proc SIGCOMM Conference, pages 200–208, Philadelphia, PA, USA, September 1990. ACM. <http://dx.doi.org/10.1145/99517.99553>

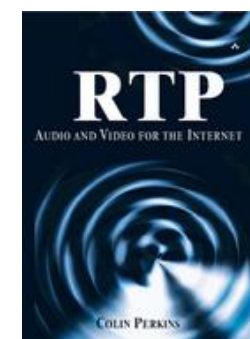
# Sequencing and Reliability

- Need to provide sequencing, reliability, and timing in applications
  - Sequence numbers and acknowledgements
  - Retransmission and/or forward error correction
  - Timing recovery
- Example: RTP data packet header

```

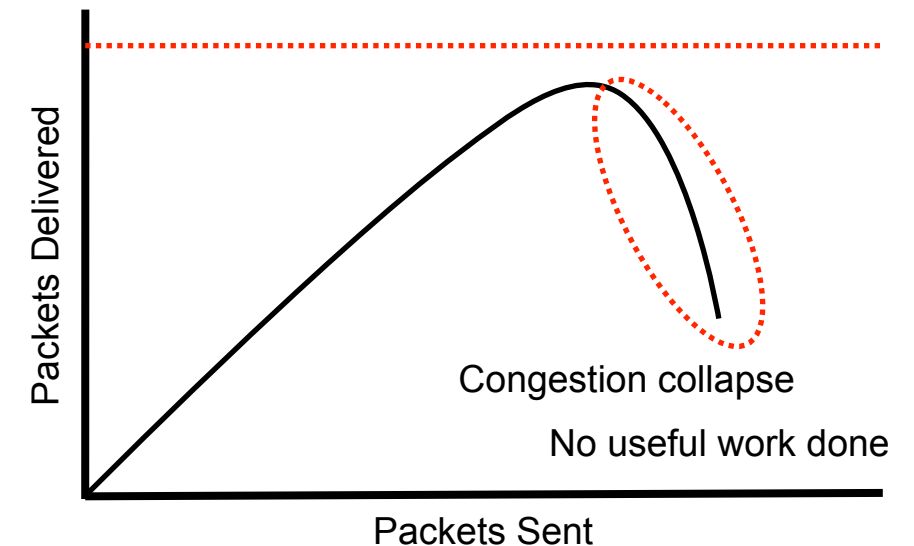
0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|V=2|P|X|  CC  |M|    PT    |          sequence number          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     timestamp                    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          synchronization source (SSRC) identifier              |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          contributing source (CSRC) identifiers                |
|                                     ....                          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

See RFC 3550 or



# Congestion Control

- Need to implement congestion control in applications
  - To avoid congestion collapse of the network
  - Should be approximately fair to TCP



- Difficult to do well – TCP congestion control covers many corner-cases that are easy to miss
  - RFC 3448 provides a detailed specification for a well-tested algorithm
  - IETF RMCAT working group developing standard congestion control algorithms for interactive video applications running over UDP  
<https://datatracker.ietf.org/wg/rmcat/charter/>
  - Google's QUIC protocol builds on UDP to give more sophisticated service

# Guidelines for writing UDP applications

- IETF provides guidelines for writing UDP-based applications
  - RFC 8085 – <https://tools.ietf.org/html/rfc8085>
- Read this before trying to write UDP-based code

# Example: Domain Name System

- The most widely used UDP-based application is the domain name system (DNS)
  - The network operates entirely on IP addresses, and has no concept of names for hosts
  - The DNS is an application that translates from user-visible names to IP address
    - `www.dcs.gla.ac.uk` → `130.209.240.1`
    - DNS is an application layer protocol, running over the network
    - Not necessary for the correct operation of the transport or network layers, or lower
  - Why run over UDP? Request-response protocol, where it was thought TCP connection setup and congestion control was too much overhead
    - Unclear if this design is appropriate

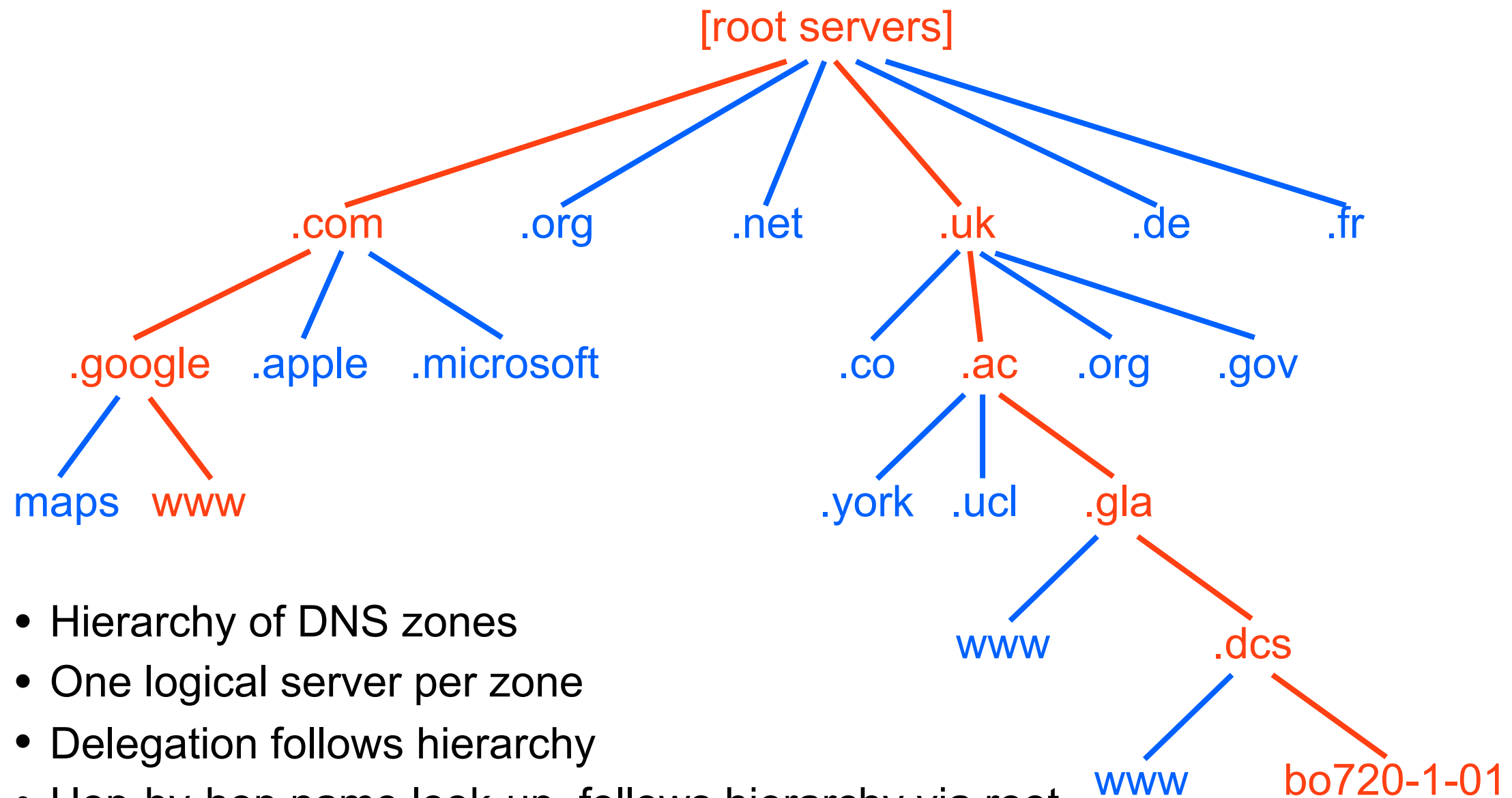
# History of the DNS

- Early Internet didn't use DNS
  - Flat file hosts.txt listing all host names and addresses
  - Maintained by central NIC; updated by email every few days; manually installed in hosts
- DNS proposed in 1983 as distributed database of host names
  - Solve scaling problems with hosts.txt



Paul Mockapetris

# Operation of the DNS



- Hierarchy of DNS zones
- One logical server per zone
- Delegation follows hierarchy
- Hop-by-hop name look-up, follows hierarchy via root
- Results have TTL, cached at intermediate servers
- getaddrinfo()

# Contents of a DNS Zone

```
$TTL 3600      ; 1 hour
example.org.   IN      SOA      ns1.example.org. admin.example.org. (
                                2006051501      ; Serial
                                10800           ; Refresh
                                3600            ; Retry
                                604800          ; Expire
                                86400           ; Minimum TTL
                                )

; DNS Servers
                                IN      NS      ns1.example.org.
                                IN      NS      ns2.example.org.

; MX Records
                                IN      MX 10    mx.example.org.
                                IN      MX 20    mail.example.org.

; Machine Names
ns1      IN      A      192.168.1.2
ns2      IN      A      192.168.1.3
mx       IN      A      192.168.1.4
mail     IN      A      192.168.1.5
mail     IN      AAAA    2001:200:1000:0:25f:23ff:fe80:1234
server1  IN      A      192.168.1.10
server2  IN      A      192.168.1.11

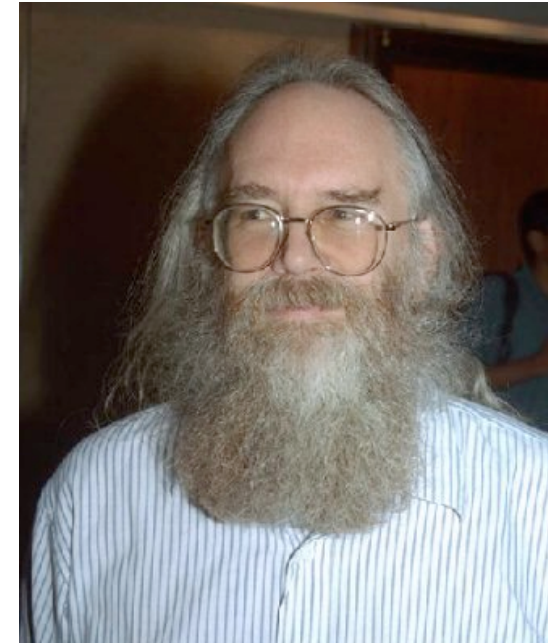
; Aliases
www      IN      CNAME    server1
```

Source: adapted from The FreeBSD Handbook



# DNS Politics

- The DNS was administered by IANA
  - Jon Postel was IANA from its creation until his death in 1998
  - <http://www.ietf.org/rfc/rfc2468.txt> – “I remember IANA”
- DNS now managed into ICANN
  - The US government asserts asserted ultimate control over ICANN, and hence the DNS
  - Significant attempts to move control of national domains to the UN, and hence to the countries concerned
  - Other attempts to set up alternate roots for the DNS, with different namespaces → significant technical problems



Jon Postel

# Summary

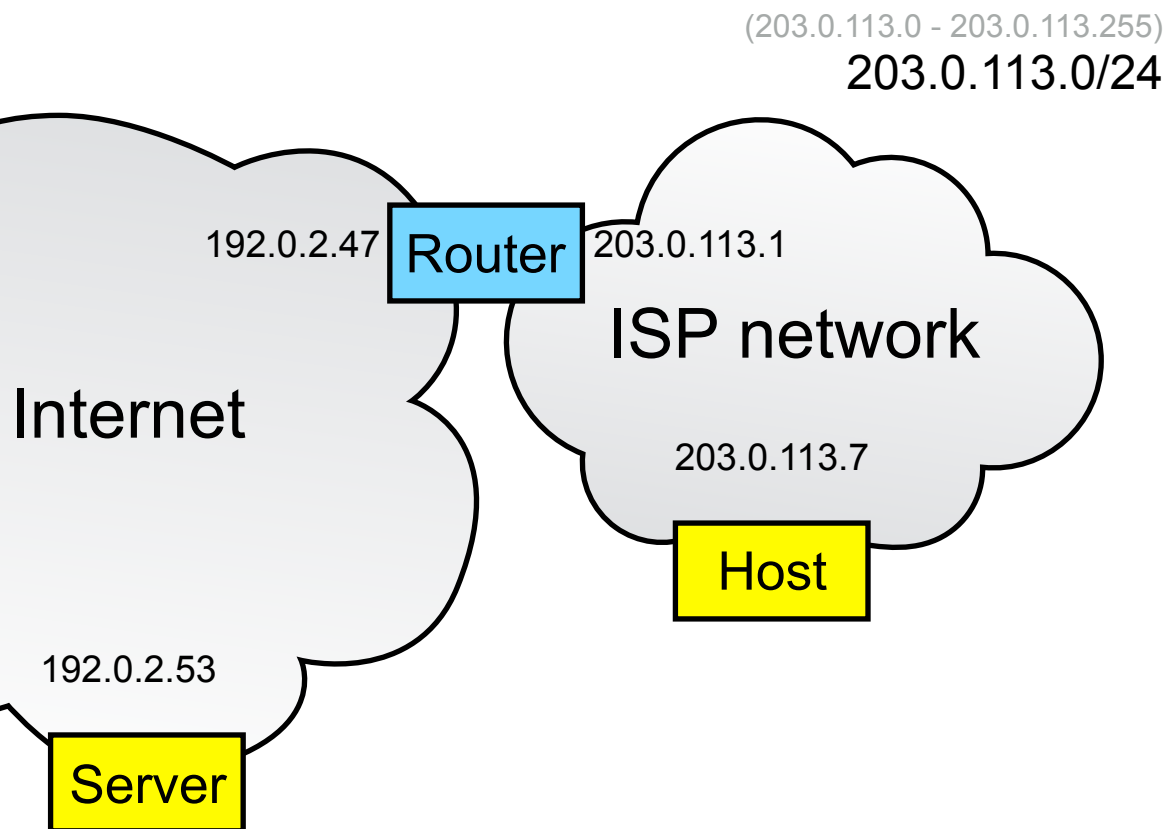
- UDP and datagram sockets
- Guidelines for using UDP
- A UDP-based application: DNS

# Network Address Translation

# Network Address Translation

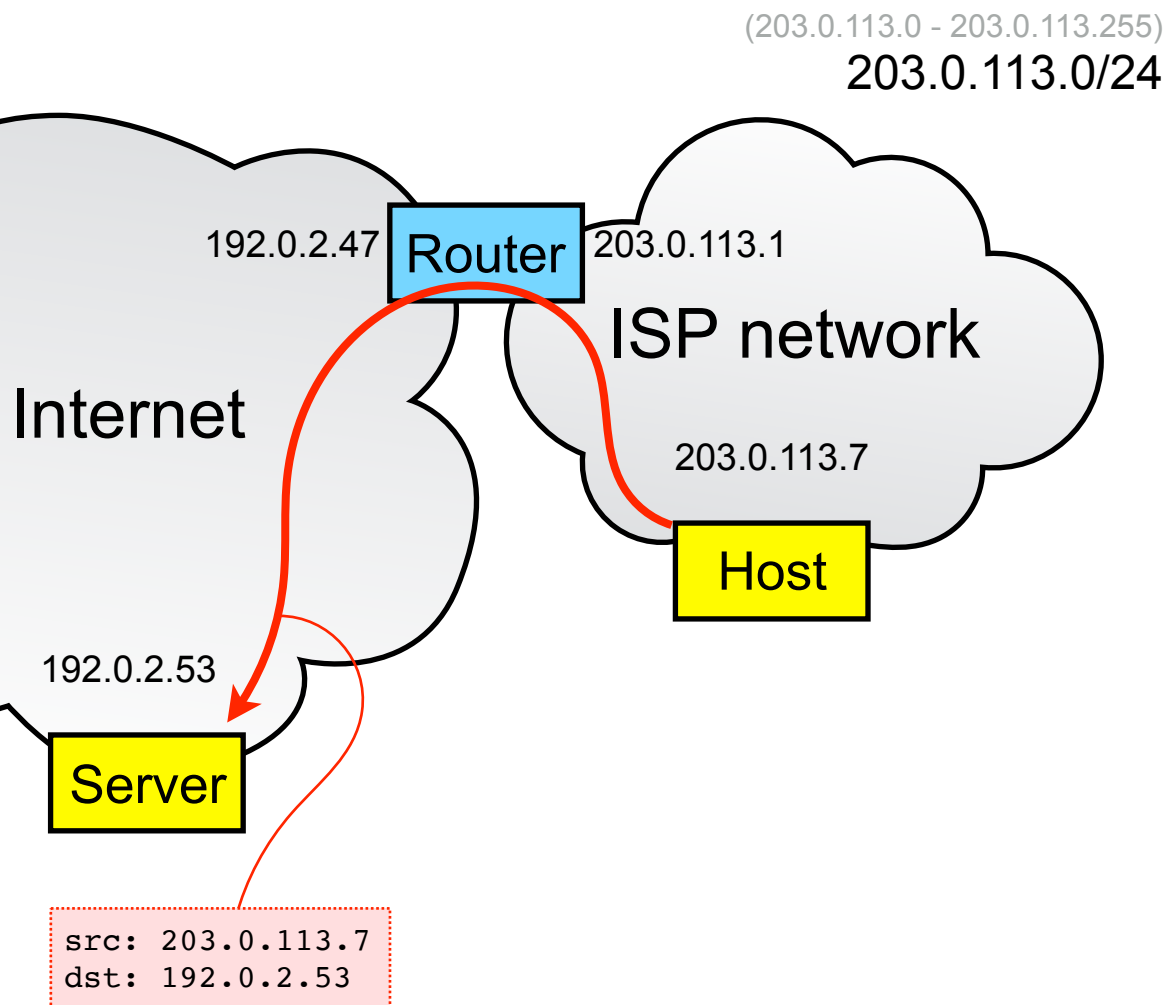
- IPv4 address space is exhausted
- IPv6 is the long-term solution
- There is a widely deployed work-around: NAT (network address translation)
- However, this has serious consequences for the transport layer

# Connecting a Single Host



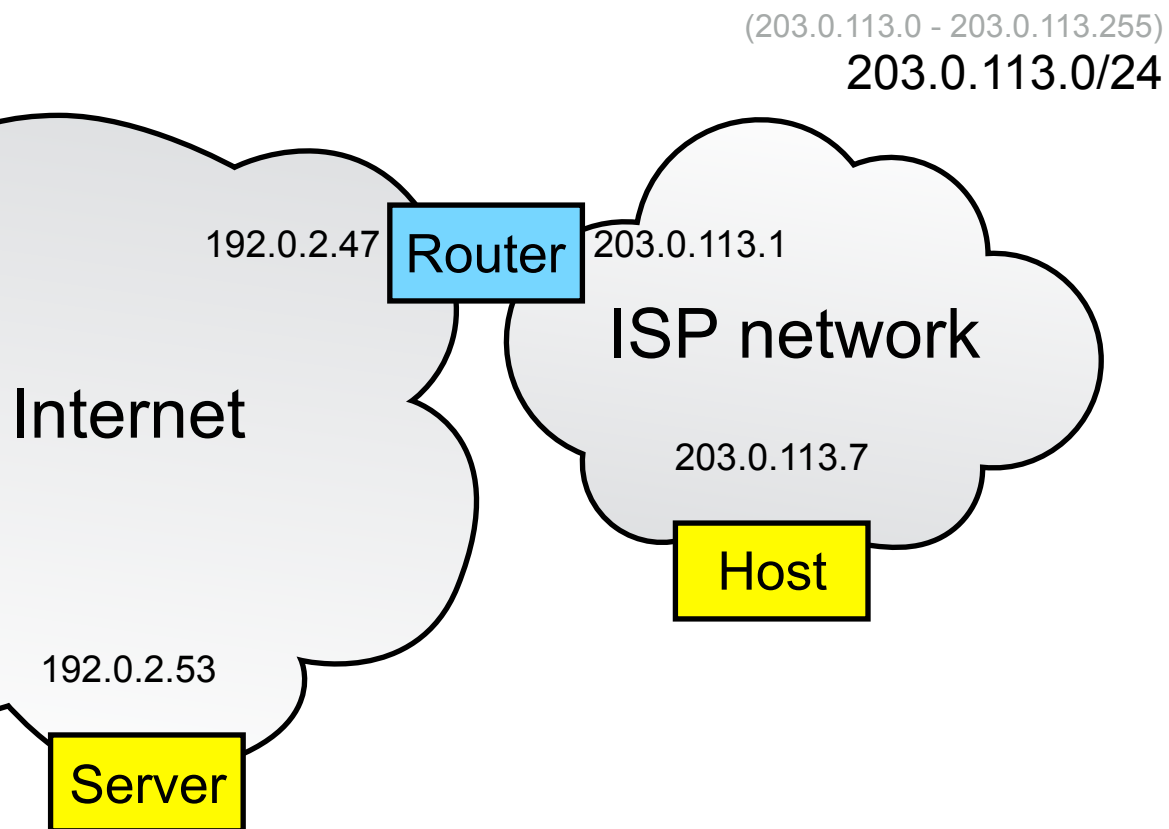
- An Internet service provider owns an IP address prefix
- They assign a customer a single address for a single host

# Connecting a Single Host



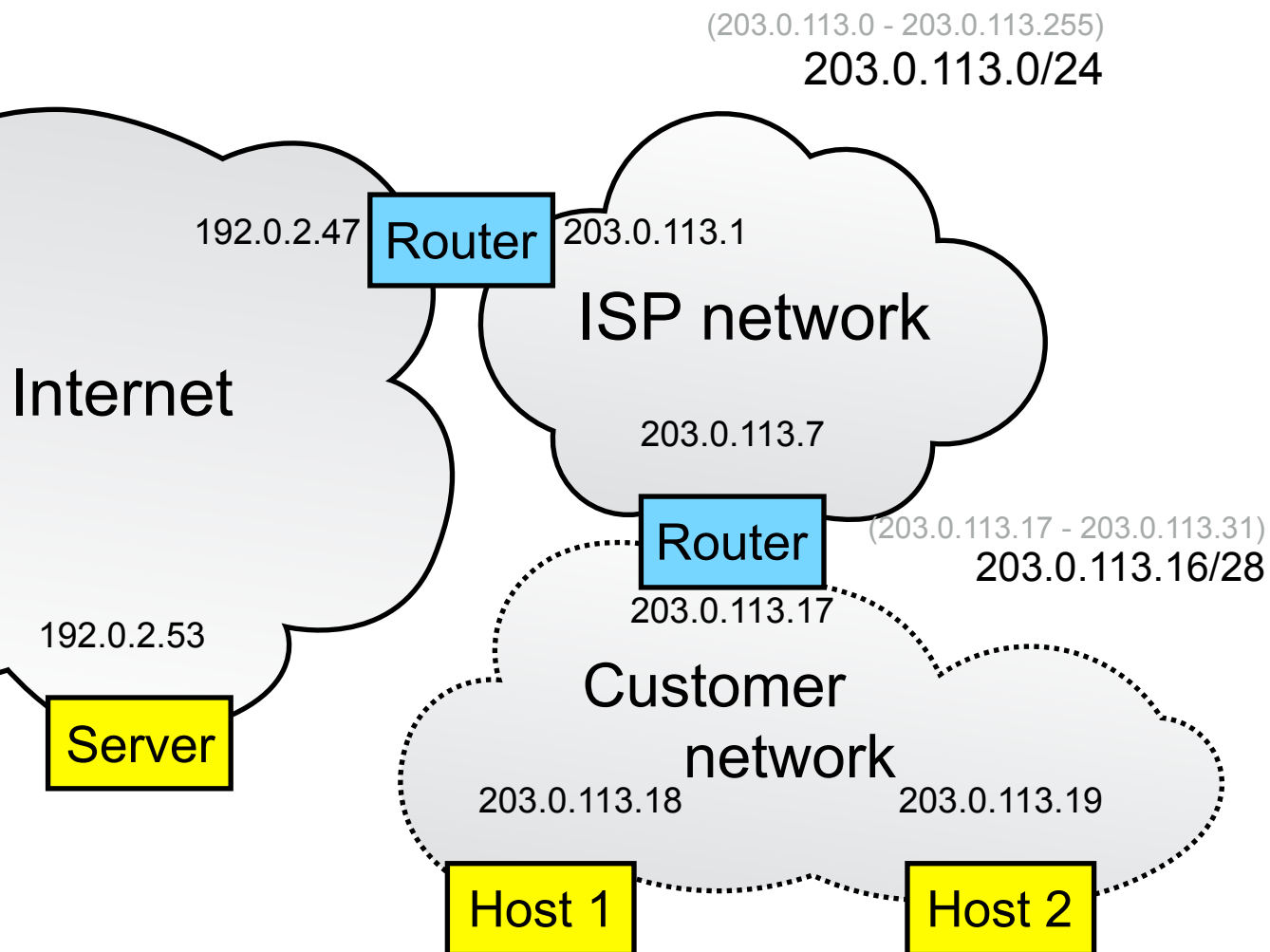
- An Internet service provider owns an IP address prefix
- They assign a customer a single address for a single host
- No address translation

# Connecting Multiple Hosts



- The customer buys another host
- How does it connect?

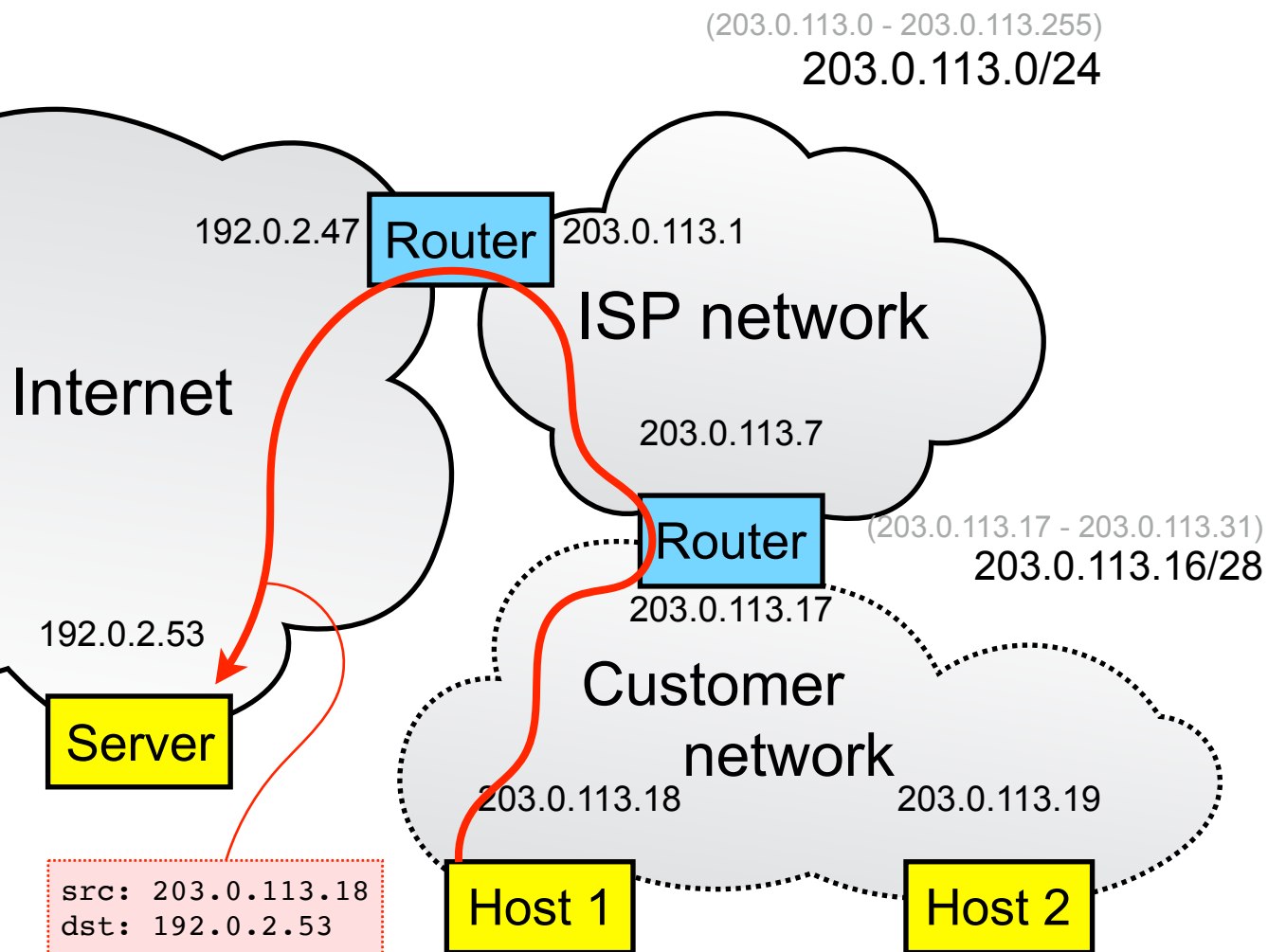
# Connecting Multiple Hosts



- The customer buys another host
- How does it connect?
- What's supposed to occur:
  - Customer acquires a router, which gets the customer's previous IP address
  - ISP assigns new range of IP addresses to customer (from the ISP's prefix)
  - Customer gives each host an address from that new range

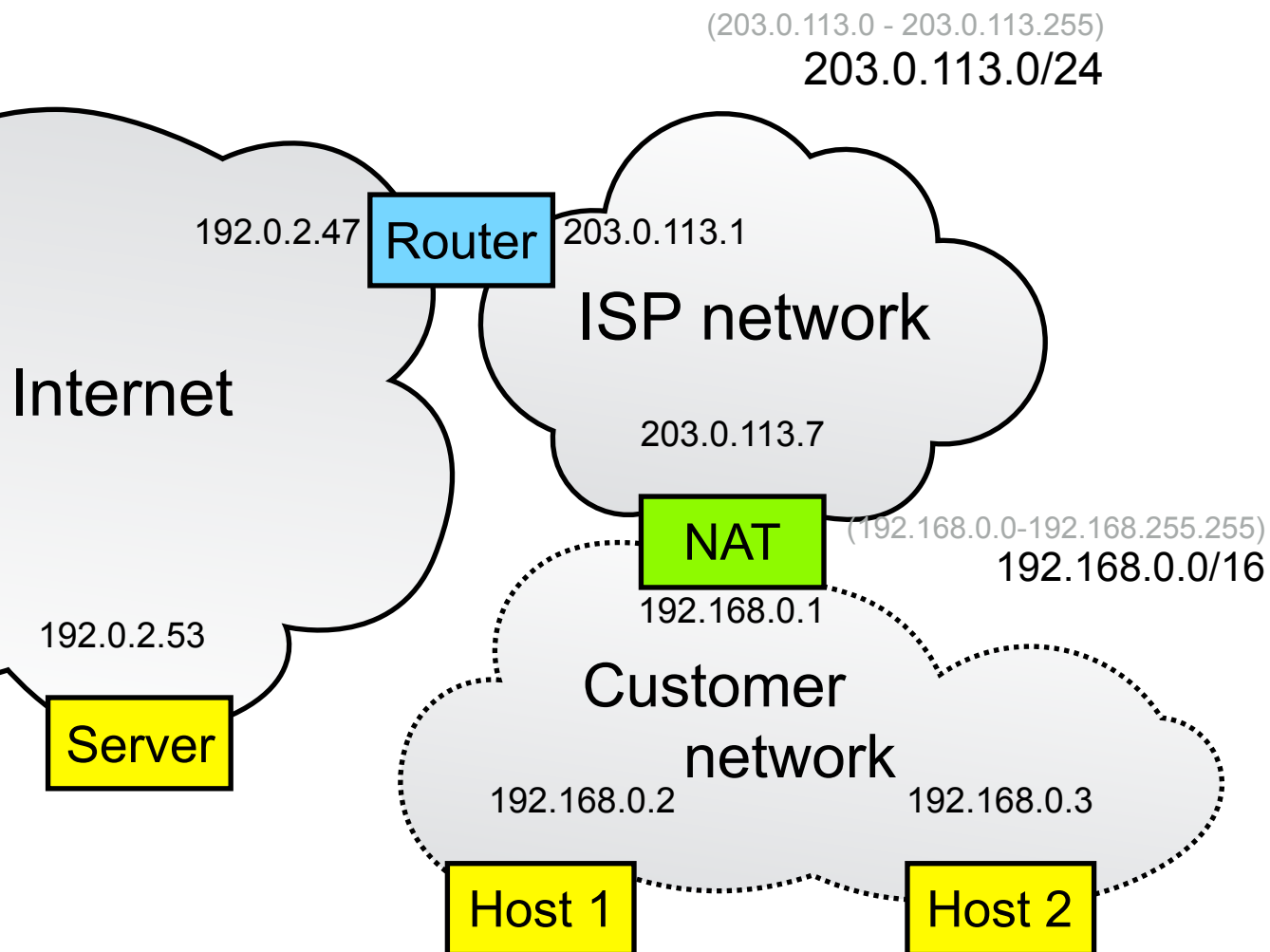


# Connecting Multiple Hosts



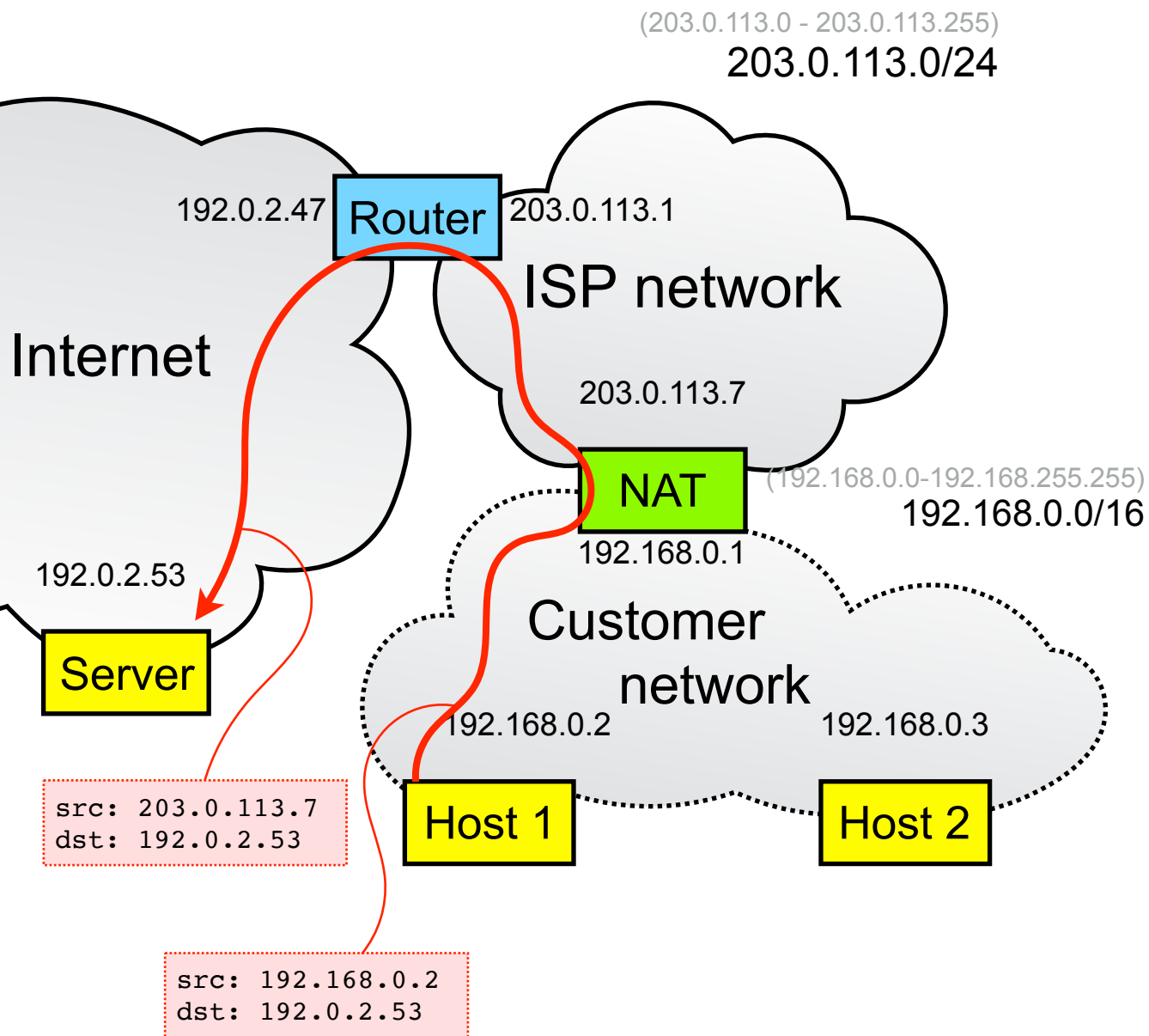
- The customer buys another host
- How does it connect?
- What's supposed to occur:
  - Customer acquires a router, which gets the customer's previous IP address
  - ISP assigns new range of IP addresses to customer (from the ISP's prefix)
  - Customer gives each host an address from that new range
  - No address translation

# Network Address Translation



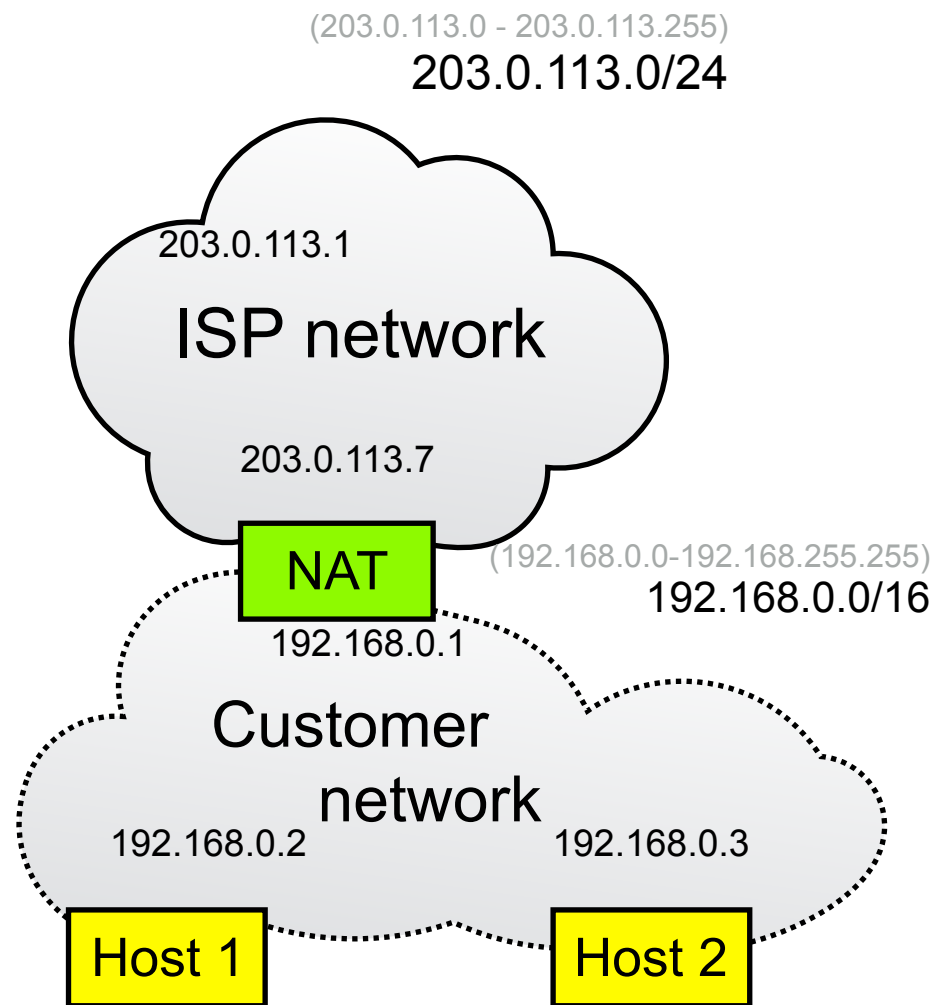
- The customer buys another host
- How does it connect?
- What actually happens:
  - Customer acquires a NAT, which gets the customer's previous IP address
  - Customer gives each host a private address

# Network Address Translation



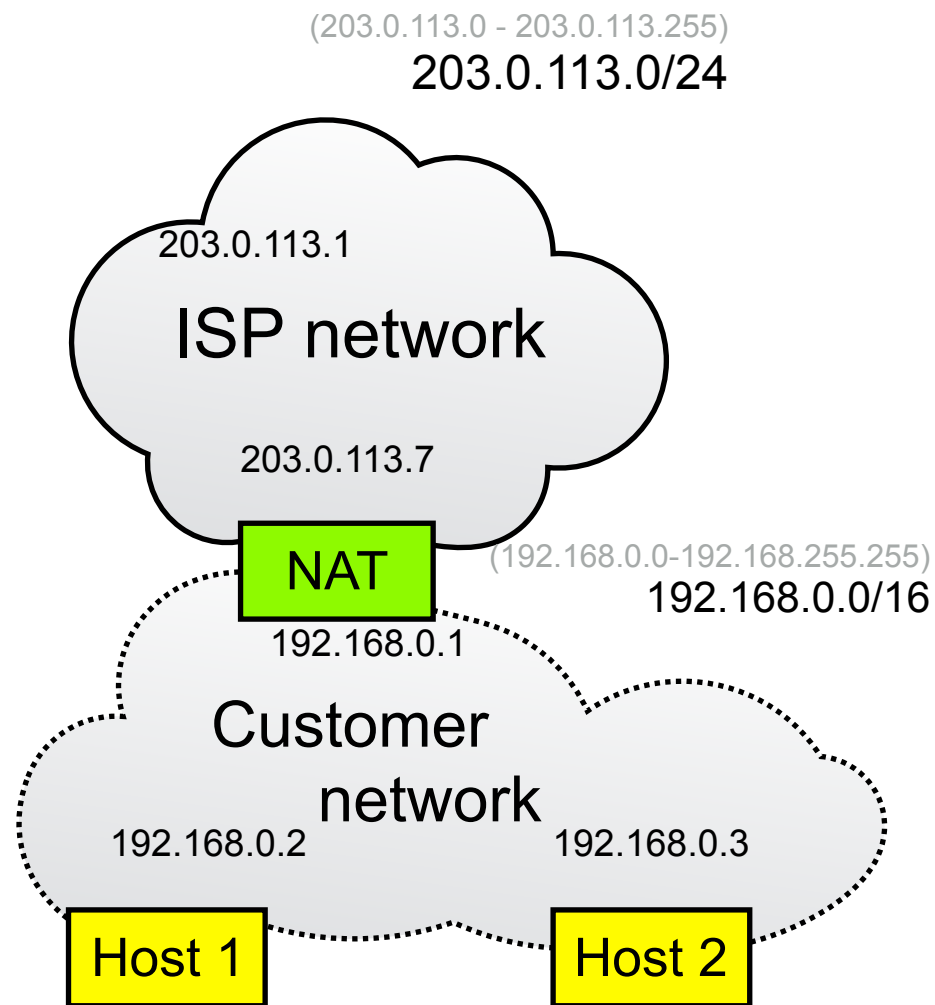
- The customer buys another host
- How does it connect?
- What actually happens:
  - Customer acquires a NAT, which gets the customer's previous IP address
  - Customer gives each host a private address
  - NAT performs address translation – rewrites packet headers to match its external IP address  
(likely also rewrites the TCP/UDP port number)

# NAT and Private Address Ranges



- The NAT hides a private network behind a single public IP address
- Private IP network addresses:
  - 10.0.0.0/8
  - 176.16.0.0/12
  - 192.168.0.0/16
- Tries to give the illusion of more address space

# Problems due to NAT



- Many applications fail with NAT:
  - Client-server applications with client behind NAT work without changes –web and email
  - Client-server applications with server behind NAT fail – need explicit port forwarding
  - Peer-to-peer applications fail – complex ICE algorithm needed to connect
- NAT provides no security benefit:
  - Most NATs also include a firewall to provide security, but NAT function gives no security or privacy benefits

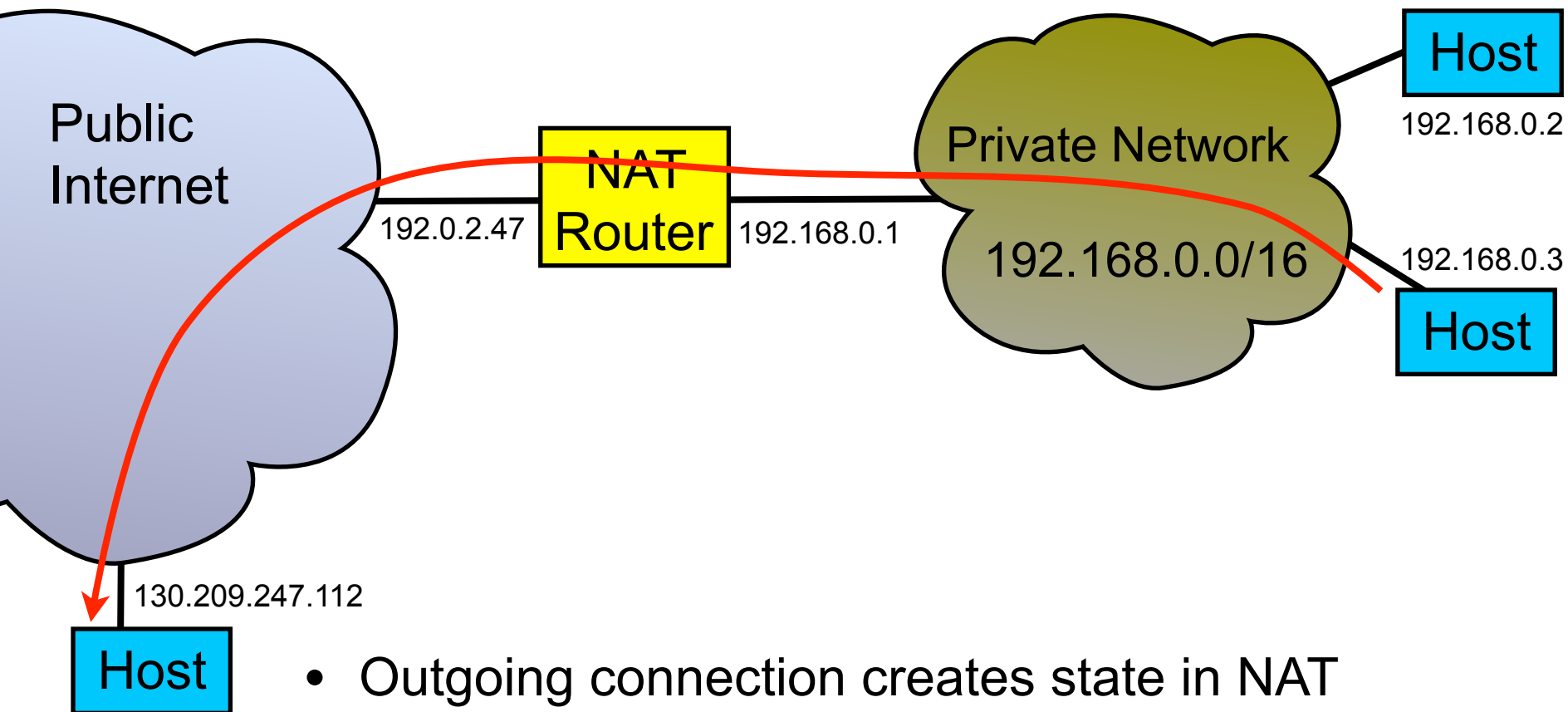
# Why use NAT? (1)

- NAT breaks many applications – so why use it?
  - Many ISPs have insufficient addresses to give customers their own prefix
  - Many customers don't want to pay their ISP more addresses
- Both problems due to limited IPv4 address space
  - IPv6 is designed to make addresses cheap and plentiful, to avoid these problems

## Why use NAT? (2)

- To avoid re-numbering a network when changing to a new ISP
  - Hard-coding IP addresses, rather than DNS names, in configuration files and application is a bad idea
  - Many people do it anyway – makes changing IP addresses difficult
- IPv6 tries to make renumbering networks easier, by providing better auto-configuration
  - Insufficient experience to know how well this works in practice
  - Some vendors also offer IPv6-to-IPv6 NAT [RFC 6296]

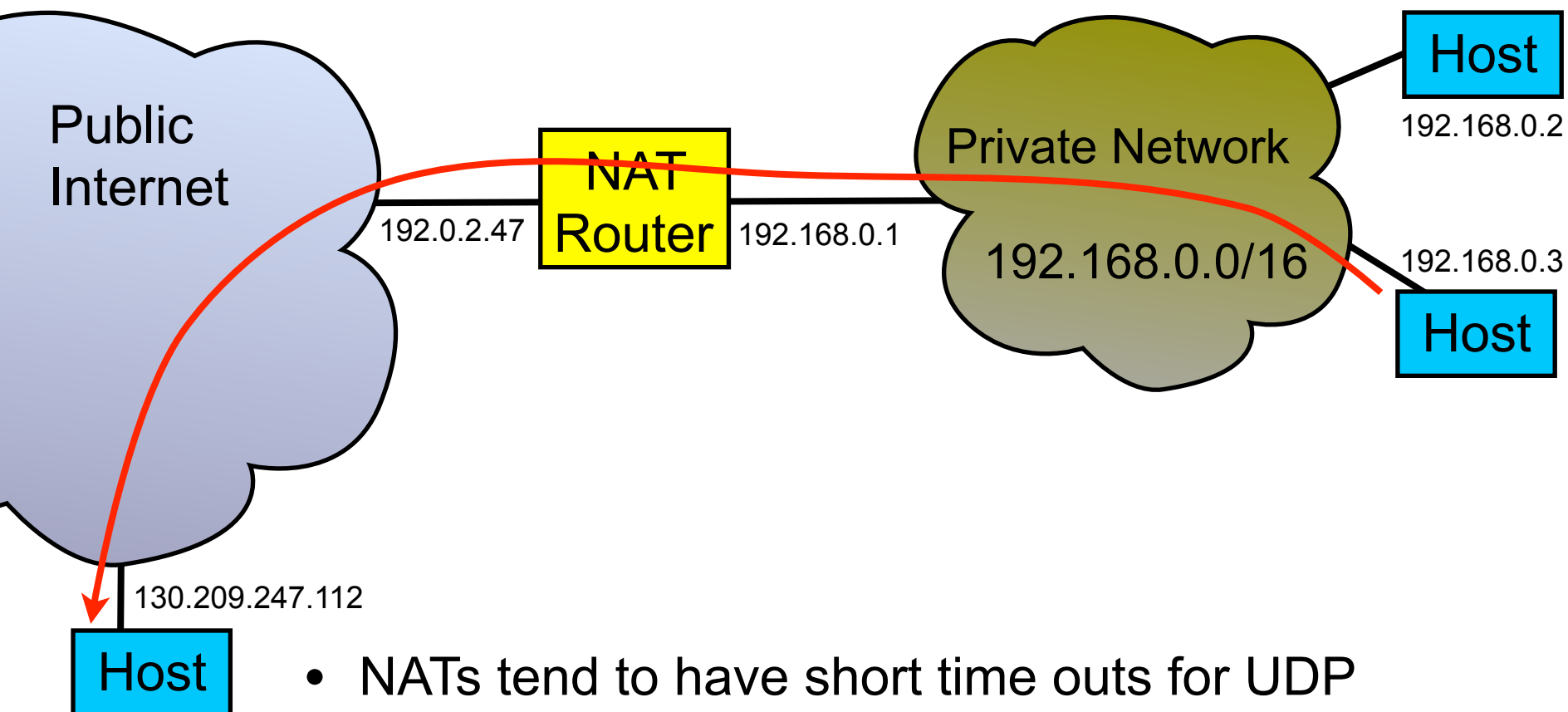
# Implications of NAT for TCP Connections



- Outgoing connection creates state in NAT
  - Need to send data periodically, else NAT state times out
  - Recommended time out interval is 2 hours, many NATs use shorter [RFC5382]
- No state for incoming connections
  - NAT can't know where to forward incoming connections, without manual configuration
  - Affects servers behind a NAT, or peer-to-peer applications

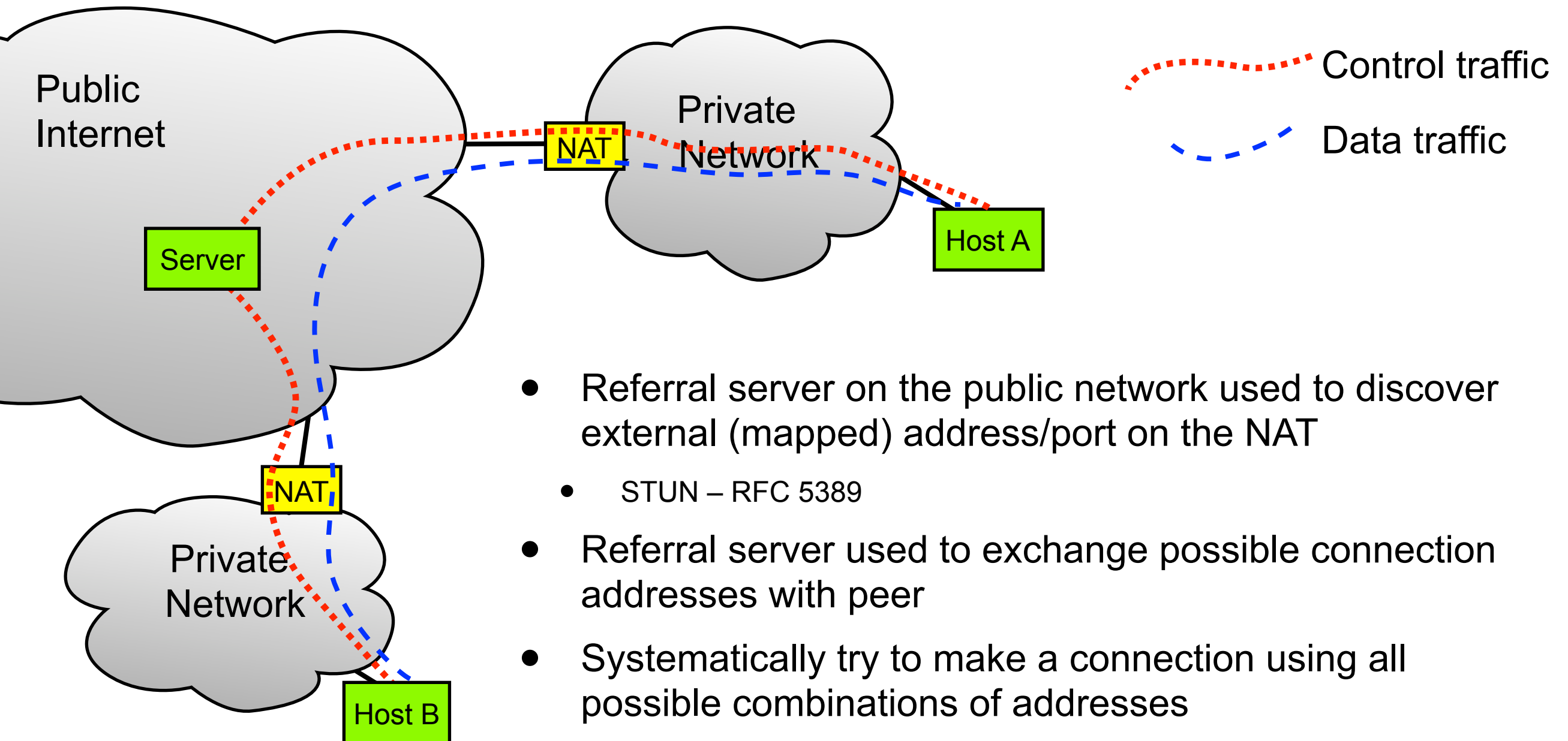


# Implications of NAT for UDP Flows



- NATs tend to have short time outs for UDP
  - Not connection-oriented, so they can't detect the end of flows
  - Recommended time out interval is not less than two minutes, but many NATs use shorter intervals – the VoIP NAT traversal standards suggest sending a keep alive message every 15 seconds [RFC4787]
- Peer-to-peer connections easier than TCP
  - UDP NATs often more permissive about allowing incoming packets than TCP NATs; many allow replies from anywhere to an open port

# NAT Traversal Concepts



- Referral server on the public network used to discover external (mapped) address/port on the NAT
  - STUN – RFC 5389
- Referral server used to exchange possible connection addresses with peer
- Systematically try to make a connection using all possible combinations of addresses
  - Every possible network interface and protocol, mapped and local
  - Complex and generates significant traffic overhead
  - The ICE algorithm – RFC 5245

# Summary

- Network address translation
  - Impact on transport protocols
  - NAT traversal concepts
- 
- NATs are widely deployed but greatly complicate applications, and hinder evolution of the network