Master Course
Computer Networks
IN2097

Prof. Dr.-Ing. Georg Carle
Florian Wohlfart

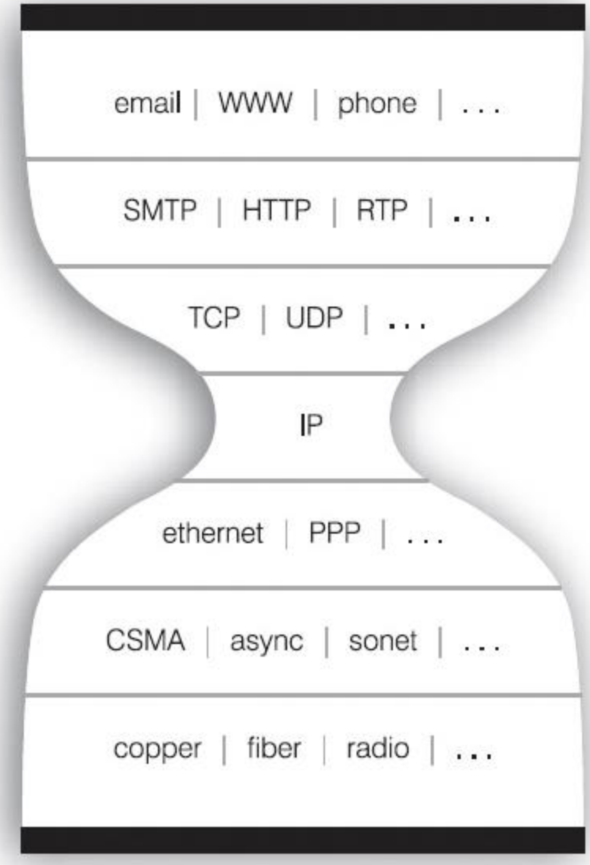
Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
http://www.net.in.tum.de

Outline: Network Layer Addresses

- DHCP
  - Automated Address Assignment
- NAT
  - Mode of Operation
  - NAT Behavior Types
  - NAT-Traversal
  - Large Scale NAT
- IPv6
  - Address Autoconfiguration

We are running out of IP addresses

- More and more devices connect to the Internet
  - PCs
  - Cell phones
  - Internet radios
  - TVs
  - Home appliances
  - Future: sensors, cars...



- IP addresses need to be globally unique
  - IPv4 provides a 32bit field
  - Many addresses not usable because of classful allocation

→ We are running out of IP addresses

DHCP

Dynamic Host Configuration Protocol

DHCP - Motivation

Motivation

- Manual network configuration of hosts not scalable
- Not all hosts in a network are online/switched on at the same time

  → Less IP addresses than hosts/customers needed

Goal: Automatic configuration of

- IP addresses
- Further information such as gateway, netmask, DNS server...

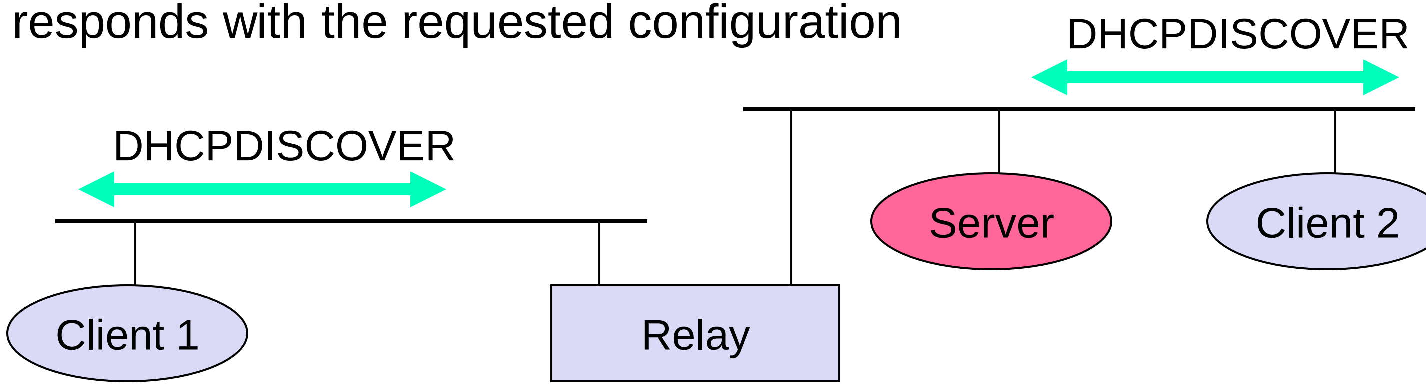
Design

- Support of several networking technologies
- Extensibility (future parameters)
- RFC 1541, current version: RFC 2131

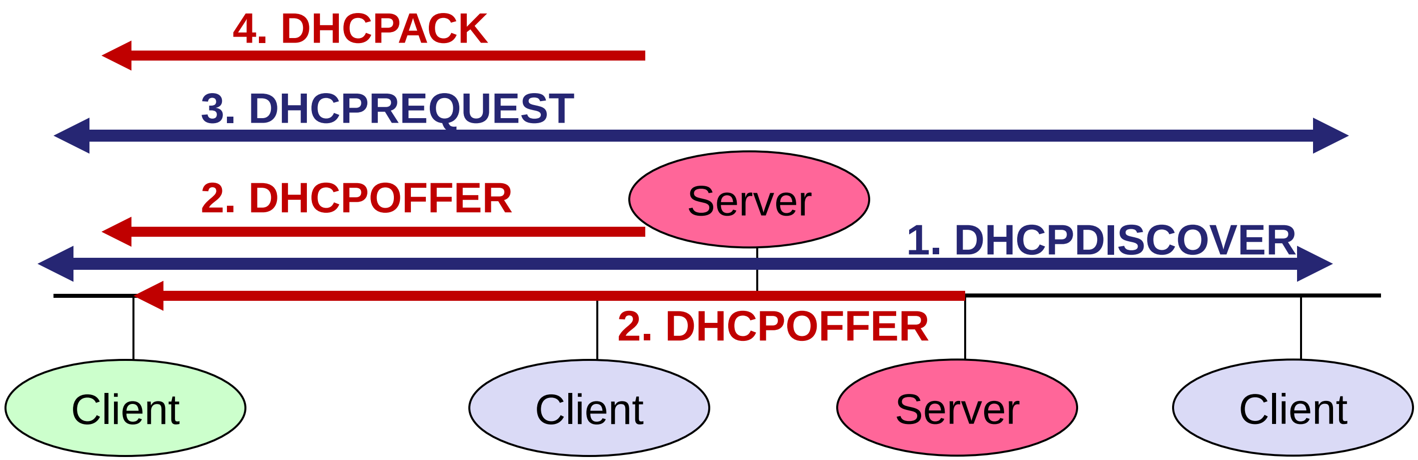
DHCP - Overview

- **Properties**
  - Simple installation and configuration of networked hosts
  - Delivers information about IP addresses, DNS server addresses, domain names, subnet masks, gateways …
  - Automatic integration of a host into the internet/intranet
  - Client leases an IP address for a specified period of time

- **Client/Server-Model**
  - Client requests a configuration via IP broadcast
  - Server responds with the requested configuration



DHCP – Messages (I)



- **DHCPDISCOVER**: Search for available servers
- **DHCPOFFER**: Server to client as a response for a DHCPDISCOVER. Includes a lease-offer.
- **DHCPREQUEST**: Client to server either for
  - (a) requesting the offered configuration file (at the same all offers from other servers are declined)
  - (b) checking if the currently used address is correct (e.g after a reboot)
  - (c) extending the lease of a currently used address
- **DHCPACK**: Server to client. Includes the configuration parameters and the mandatory network address for the client

DHCP – Messages (II)

Packet	Source MAC	Destination MAC	Source IP	Destination IP
DHCP Discover	Client	FF:FF:FF:FF:FF:FF	0.0.0.0	255.255.255.255
DHCP Offer	Server	Client	Server	Client
DHCP Request	Client	FF:FF:FF:FF:FF:FF	0.0.0.0	255.255.255.255
DHCP ACK	Server	Client	Server	Client

NAT

Network Address Translation

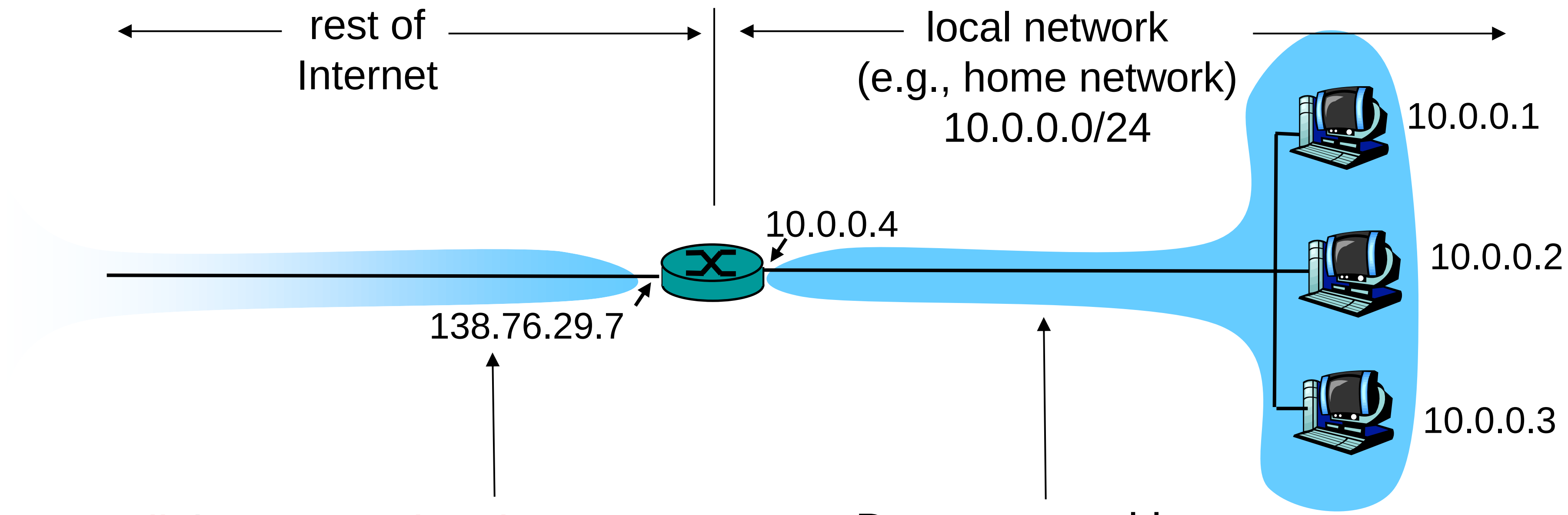
IPv4 Address Space

- IP addresses are assigned by the Internet Assigned Numbers Authority (IANA)

- RFC 1918 (published in in 1996) directs IANA to reserve the following IPv4 address ranges for private networks
  - 10.0.0.0/8 (10.0.0.0 – 10.255.255.255)
  - 172.16.0.0/12 (172.16.0.0 – 172.31.255.255)
  - 192.168.0.0/16 (192.168.0.0 – 192.168.255.255)

- The addresses may be used and reused by everyone
  - Not routed in the public internet
  - Therefore a mechanism for translating addresses is needed

NAT Scenario



All datagrams *leaving* local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0.0/24 address for source, destination as usual

NAT Mode of Operation (I)

Implementation: NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

  -> we have to maintain a state in the NAT

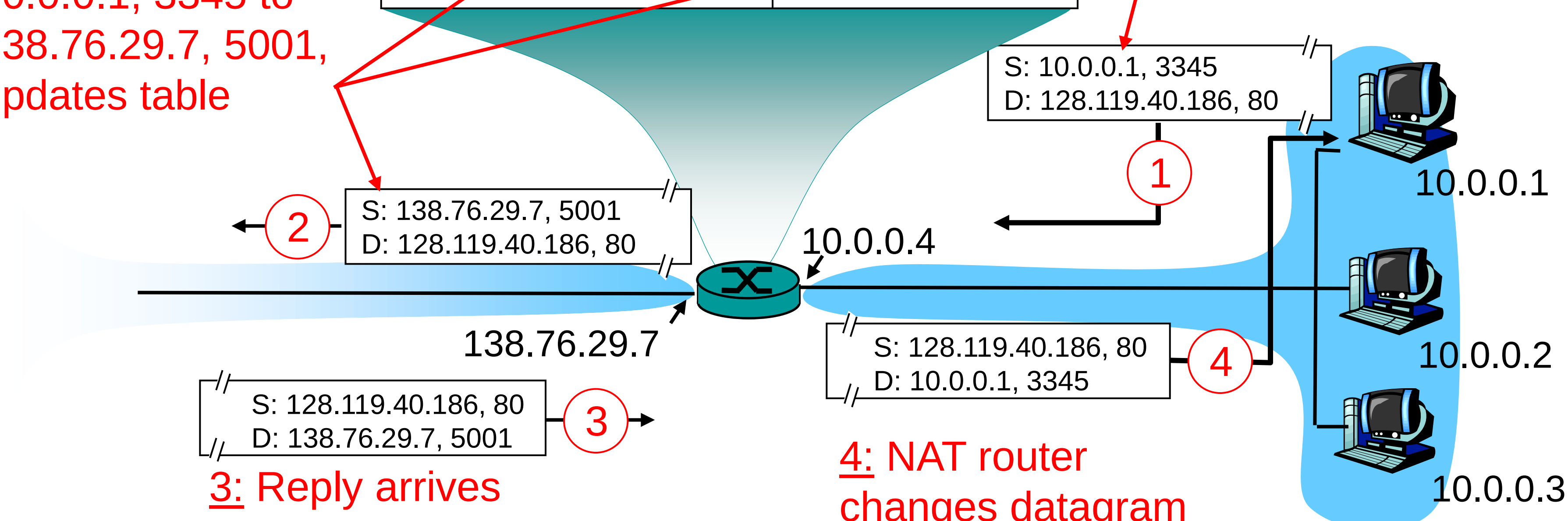
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT Mode of Operation (II)

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table



3: Reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

NAT Pros and Cons

- NAT advantages:
  - ~65000 simultaneous connections with a single LAN-side address!
  - Helps against the IP shortage
  - We can change addresses of devices in local network without notifying outside world
  - We can change ISP without changing local addresses
  - Devices inside local net not explicitly addressable/visible by the outside world (a security plus)

- NAT is controversial:
  - Ports should address applications, not hosts
  - Routers should only process up to layer 3

    → Violates end-to-end principle
  - Causes problems for certain applications

Modeling the Packet Processing of NAT

- Developed by Andreas Müller in his PhD Thesis
- Idea
  - NAT has 2 interfaces (internal and external)
  - 1st step **input**: receive packet on incoming interface
  - 2nd step **processing**: process packets internally (translation)
  - 3rd step **output**: forward packet to outgoing interface

- Simplified Notation

Path	Event	Processing
Input	Packet arrives	Look up state table
Processing	Known to state table	Translate + Forward
Output	Packet scheduled for forwarding	Forward packet

Modelling NAT: Notation

p	Packet p: (p.proto, p.sIP, p.sIP, p.sP, s.dP) p.proto: transport layer protocol p.sIP: source IP p.dIP: destionation IP p.sP: source port p.dP: destination port
receive (p, cond)	NAT receives packet p if cond is true
send (p, cond)	NAT sends packet p if cond is true
E(X)	Occurrence of event X
TE(X, options)	Triggers event X and passess options to it

NAT Modelling: Outgoing Packets

Path	Event	Processing
In	**E(A, int, p)**	**Receive(p) && TE(getDB, (p.sIP, p.sP))**
	Arrival event in internal interface, receive packet and lookup state table	
Proc.	**E(getDB)**	**TE(DB.(found(extPort) or false))**
	State table lookup returns external port (if state exists) or false	
	E(DB.found)	**TE(MASQ, extPort, ext)**
	If state was found, trigger masquerading event and pass external port to it	
	E(DB.false)	**TE(MASQ, allocMap()) && TE(setDB, p)**
	If state was not found, allocate new source port and trigger MASQ event	
	E(MASQ)	**(p.sIP = extIP, s.sP = extSP) && TE(FW, ext, p)**
	Masquerade packet and replace source IP address and source port	
Out	**E(FW, ext)**	**send(p)**
	Send packet p to external interface	

NAT Modelling: Incoming Packets

<table>
<tr><th>Path</th><th>Event</th><th>Processing</th></tr>
<tr><td rowspan="2">In</td><td>E(A, ext, p)</td><td>Receive(p) && TE(getDB, p.dP)</td></tr>
<tr><td colspan="2">Arrival event in external interface, receive packet and lookup state table</td></tr>
<tr><td rowspan="8">Proc.</td><td>E(getDB)</td><td>TE(DB.(found(intIP,intSP) or false))</td></tr>
<tr><td colspan="2">State table lookup returns internal port and IP (if state exists) or false</td></tr>
<tr><td>E(DB.found)</td><td>TE(MASQ, intSP, intIP)</td></tr>
<tr><td colspan="2">If state was found, trigger masquerading event and pass int. IP + port to it</td></tr>
<tr><td>E(DB.false)</td><td>TE(DROP, p)</td></tr>
<tr><td colspan="2">If state was not found, drop the packet</td></tr>
<tr><td>E(MASQ)</td><td>(p.dIP = intIP, p.dP = intSP) && TE(FW, int, p)</td></tr>
<tr><td colspan="2">Masquerade packet and replace source IP address and source port</td></tr>
<tr><td rowspan="2">Out</td><td>E(FW, int)</td><td>send(p)</td></tr>
<tr><td colspan="2">Send packet p to internal interface</td></tr>
</table>

NAT Behavior and Implementation

- Implementation is not standardized
  - Thought as a temporary solution
  - Implementation differs from vendor to vendor (and model to model)

- NAT behavior differs in:
  - Outgoing packets: Binding
    - Which external mapping is allocated?
    - **Port binding**
    - **NAT binding**
  - Incoming packets: Filtering
    - Who is allowed to access the mapping?
    - **Endpoint filtering**

NAT Behavior: Port Binding

- When creating a new state, the NAT has to assign a new source port and IP address to the connection

- **Port binding** describes the strategy a NAT uses for the assignment of a new external source port
  - Port Preservation (if possible)
  - Some algorithm (e.g. +1)
  - Random

NAT Behavior: Modelling Port Binding

Path	Event	Processing
Proc.	**E(allocMap)**	**newPort = s.sP**
	Port preservation	
	E(allocMap)	**newPort = lastPort + X**
	No port preservation (algorithm)	
	E(allocMap)	**newPort = rand(portRange)**
	No port preservation (random)	

NAT Behavior: NAT Binding

- **NAT binding** describes the behavior of the NAT regarding the reuse of an existing binding
  - two consecutive connections from the same transport address (combination of IP address and port)
  - 2 different bindings?
  - If the binding is the same → Port prediction possible

- Endpoint Independent
  - the external port is only dependent on the source transport address
  - both connections have the same IP address and port

- Endpoint Dependent
  - a new port is assigned for every connection
  - strategy could be random, but also something more predictable
  - Port prediction is hard

NAT Behavior: Endpoint filtering

- Filtering describes
  - how existing mappings can be used by external hosts
  - How a NAT handles incoming connections

- **Independent-Filtering:**
  - All inbound connections are allowed
  - Independent on source address
  - As long as a packet matches a state it is forwarded
  - No security

- **Address Restricted Filtering:**
  - packets coming from the same host (matching IP-Address) the initial packet was sent to are forwarded

- **Address and Port Restricted Filtering:**
  - IP address and port must match

NAT Behavior: Modelling Filtering Behavior

Path	Event	Processing
Proc.	**E(getDB)**	**TE(DB.(found(p.dP)))**
	Independent filtering	
	E(getDB)	**TE(DB.(found(p.dP, p.sIP)))**
	Address restricted filtering	
	E(getDB)	**TE(DB.(found(p.dP, p.sIP, p.sP)))**
	Address and port restricted filtering	

NAT Behavior: NAT Types

- With Binding and Filtering 4 NAT types can be defined (RFC 3489)

- Full Cone NAT
  - Endpoint independent
  - Independent filtering

- Address Restricted NAT
  - Endpoint independent binding
  - Address restricted filtering

- Port Address Restricted NAT
  - Endpoint independent binding
  - Port address restricted filtering

- Symmetric NAT
  - Endpoint dependent binding
  - Port address restricted filtering

NAT Behavior: NAT Types

- With Binding and Filtering 4 NAT types can be defined (RFC 3489)

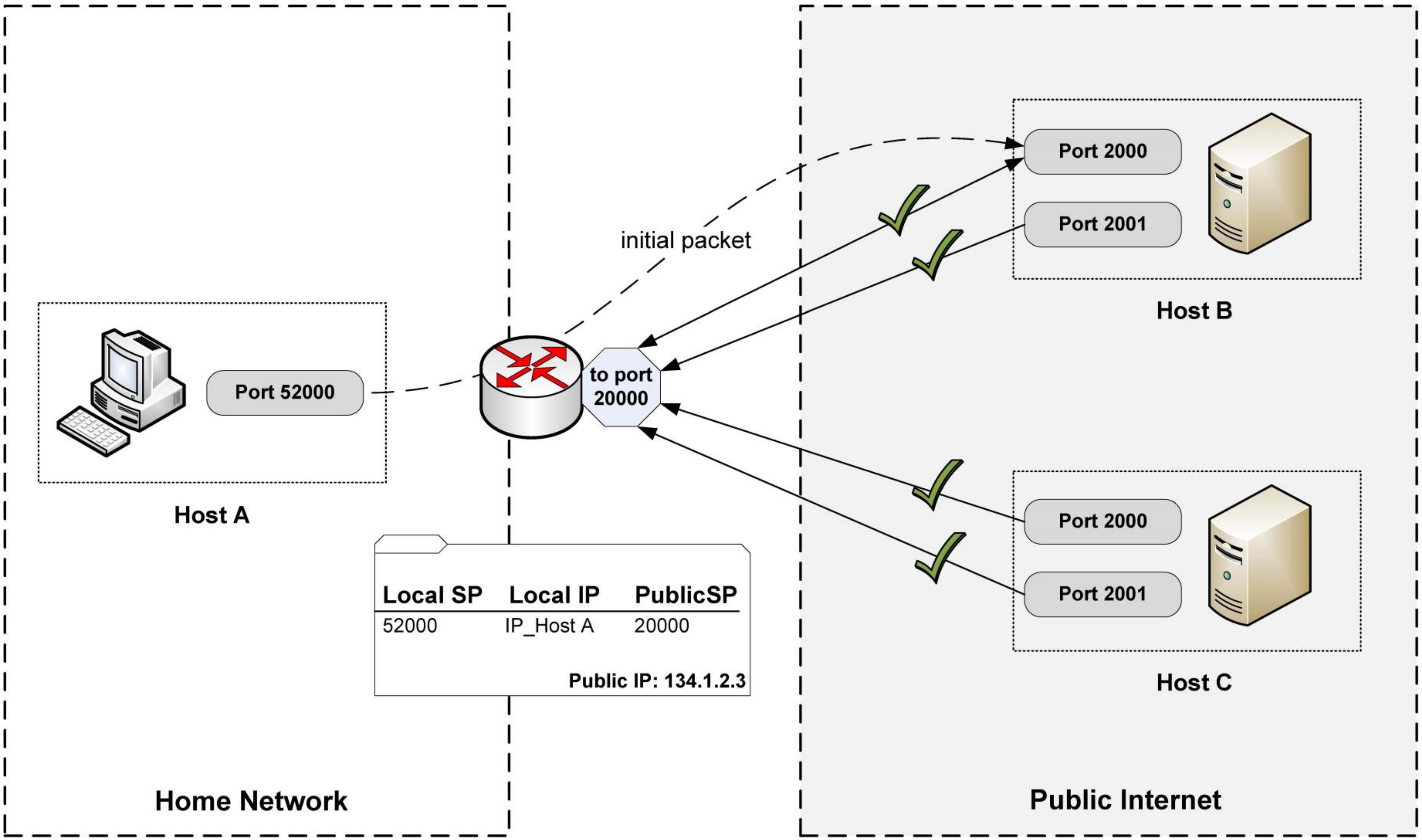
- **Full Cone NAT**
  - **Endpoint independent**
  - **Independent filtering**

- Address Restricted NAT
  - Endpoint independent binding
  - Address restricted filtering

- Port Address Restricted NAT
  - Endpoint independent binding
  - Port address restricted filtering

- Symmetric NAT
  - Endpoint dependent binding
  - Port address restricted filtering

Full Cone NAT



NAT Behavior: NAT Types

- With Binding and Filtering 4 NAT types can be defined (RFC 3489)

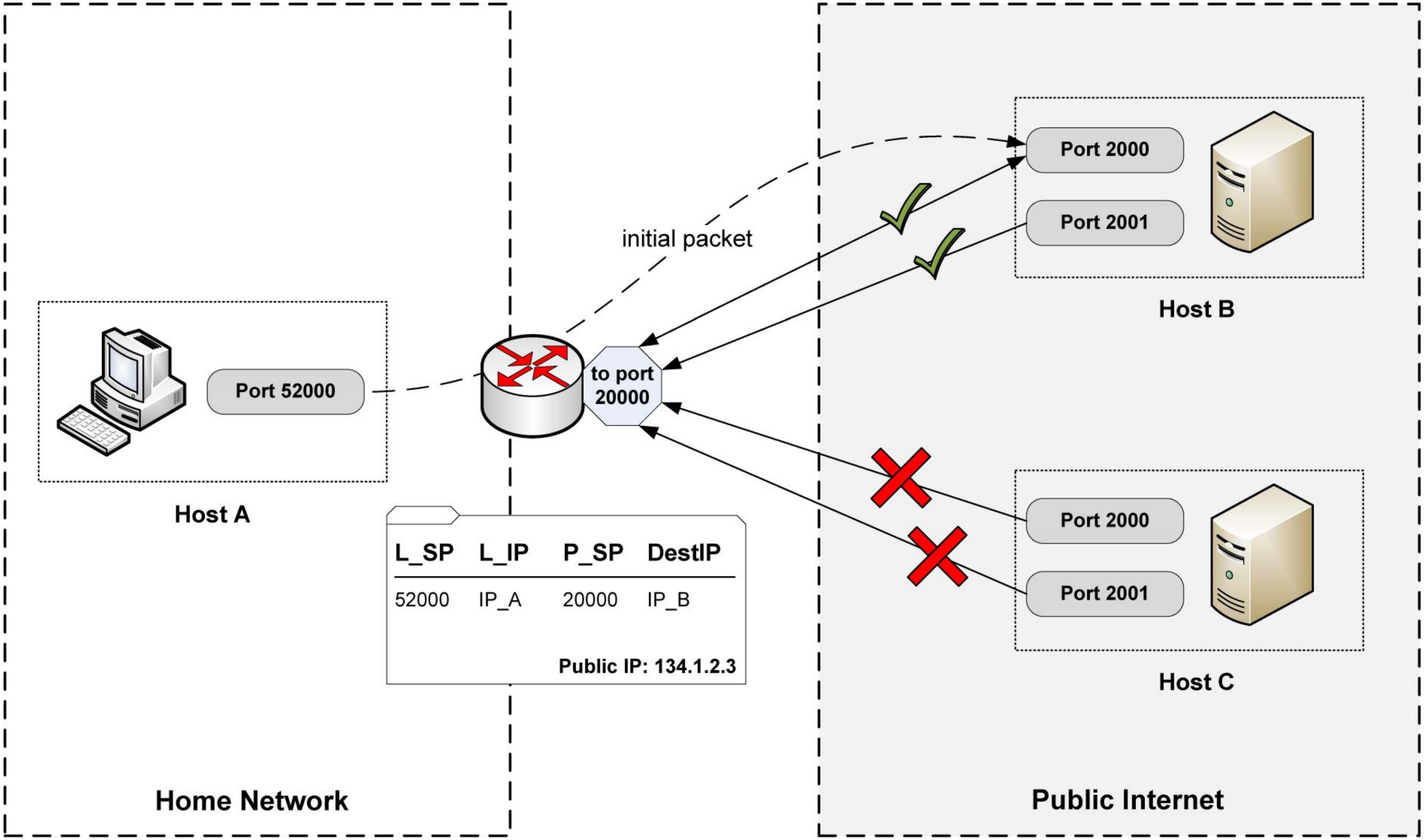
- Full Cone NAT
  - Endpoint independent
  - Independent filtering

- **Address Restricted NAT**
  - **Endpoint independent binding**
  - **Address restricted filtering**

- Port Address Restricted NAT
  - Endpoint independent binding
  - Port address restricted filtering

- Symmetric NAT
  - Endpoint dependent binding
  - Port address restricted filtering

Address Restricted Cone NAT



NAT Behavior: NAT Types

- With Binding and Filtering 4 NAT types can be defined (RFC 3489)

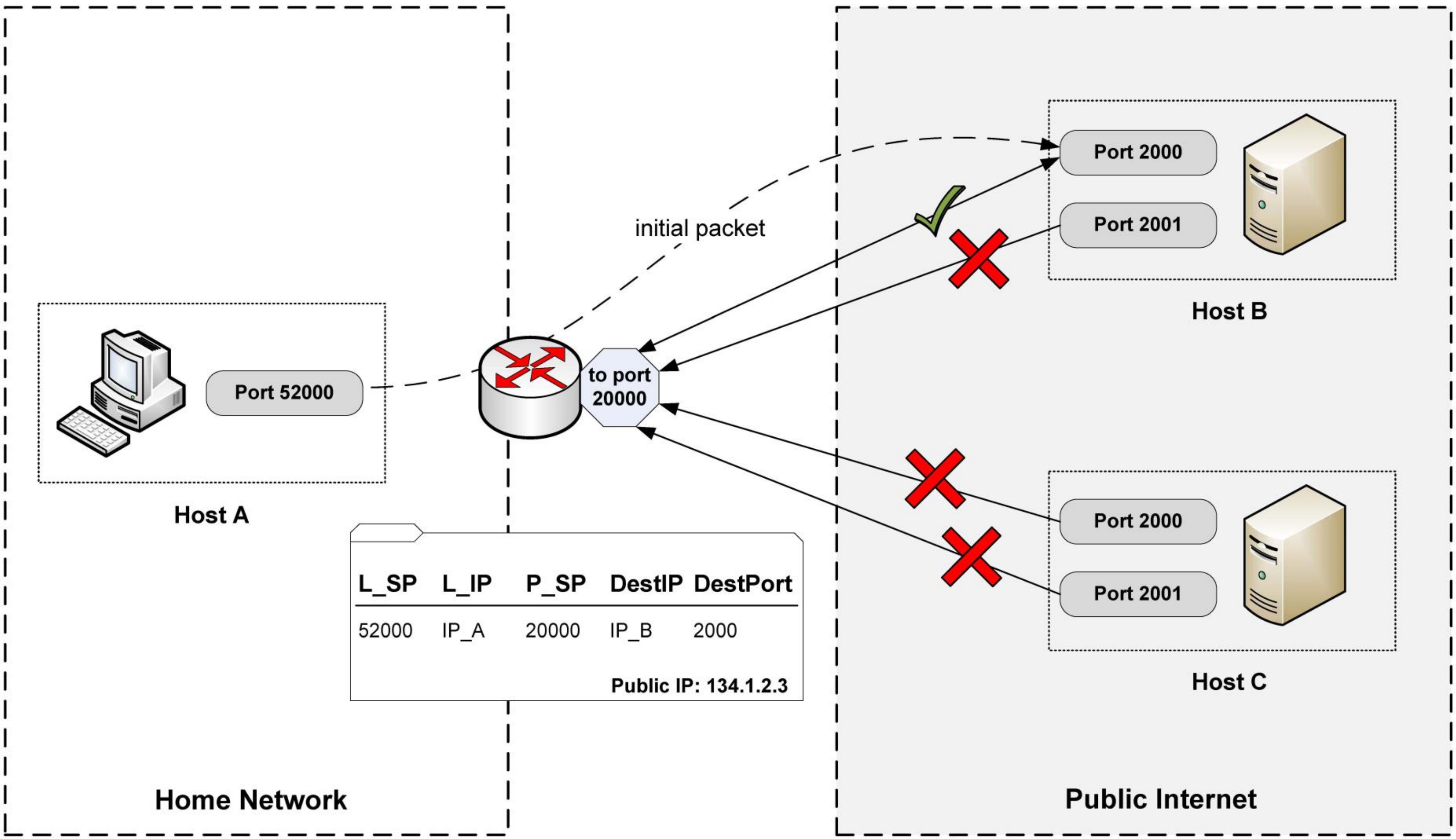
- Full Cone NAT
  - Endpoint independent
  - Independent filtering

- Address Restricted NAT
  - Endpoint independent binding
  - Address restricted filtering

- **Port Address Restricted NAT**
  - **Endpoint independent binding**
  - **Port address restricted filtering**

- Symmetric NAT
  - Endpoint dependent binding
  - Port address restricted filtering

Port Address Restricted Cone NAT



NAT Behavior: NAT Types

- With Binding and Filtering 4 NAT types can be defined (RFC 3489)

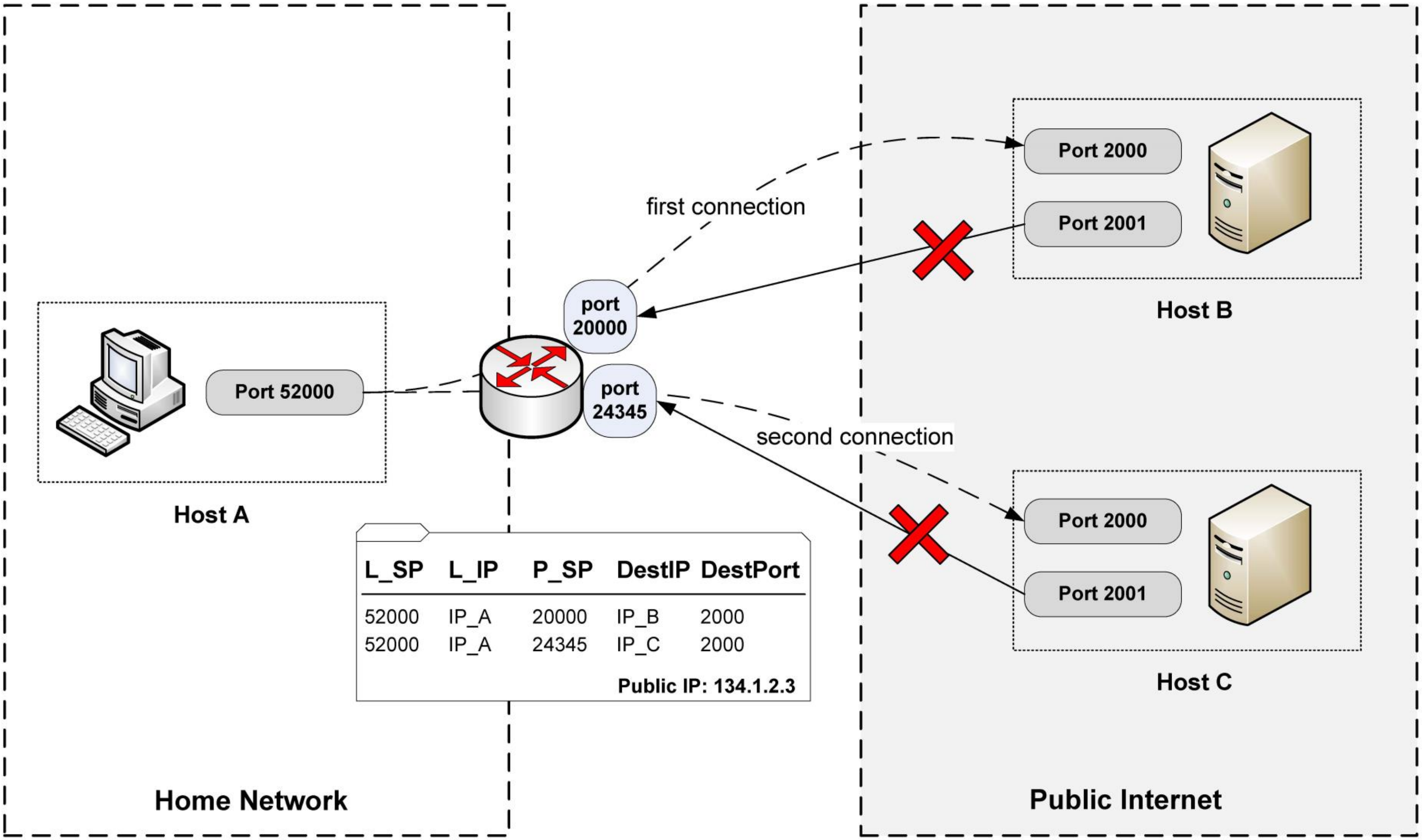
- Full Cone NAT
  - Endpoint independent
  - Independent filtering

- Address Restricted NAT
  - Endpoint independent binding
  - Address restricted filtering

- Port Address Restricted NAT
  - Endpoint independent binding
  - Port address restricted filtering

- **Symmetric NAT**
  - **Endpoint dependent binding**
  - **Port address restricted filtering**

Symmetric NAT



L_SP	L_IP	P_SP	DestIP	DestPort
52000	IP_A	20000	IP_B	2000
52000	IP_A	24345	IP_C	2000

Public IP: 134.1.2.3

And where is the problem?

- NAT was designed for the client-server paradigm

- Nowadays the internet consists of applications such as
  - P2P networks
  - Voice over IP
  - Multimedia Streams

- Protocols are getting more and more complex
  - Multiple layer 4 connections (data and control session)
  - Realm specific addresses in layer 7

- Connectivity requirements have changed
  - P2P is becoming more and more important
    - Especially for future home and services
  - Direct connections between hosts is necessary

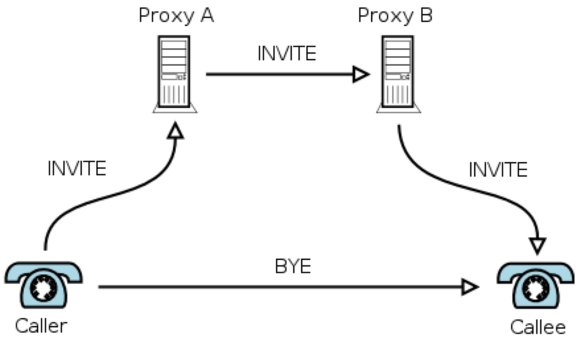
- NATs break the end-to-end connectivity model of the internet
  - Inbound packets can only be forwarded if an appropriate mapping exists
  - Mappings are only created on outbound packets

NAT-Traversal Problem

- Divided into four categories: (derived from IETF-RFC 3027)
  - **Realm-Specific IP-Addresses in the Payload**
    - *Session Initiation Protocol (SIP)*
  - **Peer-to-Peer Applications**
    - *Any service behind a NAT*
  - **Bundled Session Applications (Inband Signaling)**
    - *FTP*
    - *Real time streaming protocol (RTSP)*
    - *SIP together with SDP (Session Description Protocol)*
  - **Unsupported Protocols**
    - *SCTP (Stream Control Transmission Protocol)*
    - *IPSec*

Example: Session Initiation Protocol (SIP)

- Realm Specific IP addresses in the payload (SIP)
- Bundled Session Application (RTP)



Request/Respone Line — INVITE sip:Callee@200.3.4.5 SIP/2.0

Message-Header:

Via: SIP/2.0/UDP **192.168.1.5:5060**
From: < sip:Caller@**192.168.1.5** >
To: <sip:Callee@200.3.4.5>
CSeq: 1 INVITE
Contact: <sip:Caller@192.168.1.5:5060>
Content-Type: application/sdp

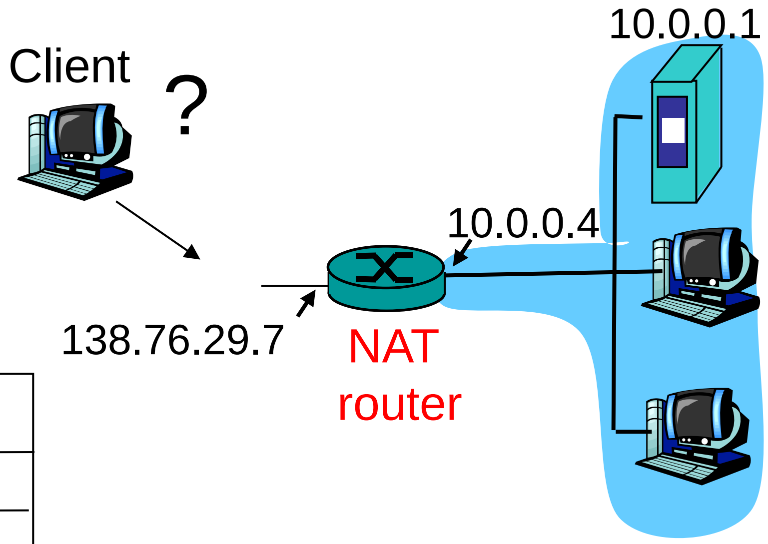
Message-Body (optional):

v=0
o=Alice 214365879 214365879 IN IP4 **192.168.1. 5**
c=IN IP4 **192.168.1.5**
t= 0 0
m=audio 5200 RTP/AVP 0 9 7 3
a=rtpmap:8 PCMU/8000
a=rtpmap:3 GSM/8000

(v= … t=: RTP-Session Specification (for 2nd channel); t= … a=: Media description for 2nd channel; whole body: SDP)

Example: P2P applications

- Client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
  - NAT does not have any idea where to forward packets to



NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 80	10.0.0.1, 80
……	……

Existing Solutions to the NAT-Traversal Problem

- Individual solutions
  - Explicit support by the NAT
    - Static port forwarding, ALG, UPnP, NAT-PMP
  - NAT-behavior based approaches
    - dependent on knowledge about the NAT
    - Hole Punching using STUN (IETF - RFC 3489)
  - External Data-Relay
    - TURN (IETF - Draft)

- Frameworks integrating several techniques
  - Framework selects a working technique
  - ICE as the most promising for VoIP (IETF - Draft)

Explicit support by the NAT (1)

- Application Layer Gateway (ALG)
  - Implemented on the NAT device and operates on layer 7
  - Supports Layer 7 protocols that carry realm specific addresses in their payload
    - SIP, FTP

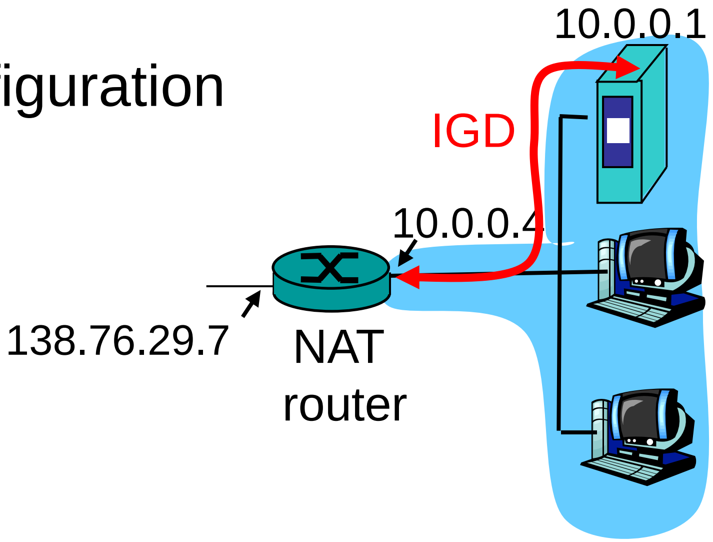
- Advantages
  - Transparent for the application
  - No configuration necessary

- Drawbacks
  - Protocol dependent (e.g. ALG for SIP, ALG for FTP...)
  - May or may not be available on the NAT device

Explicit support by the NAT (2)

- Universal Plug and Play (UPnP)
  - Automatic discovery of services (via Multicast)
  - Internet Gateway Device (IGD) for NAT-Traversal

- IGD allows NATed host to
  - Automate static NAT port map configuration
  - Learn public IP address (138.76.29.7)
  - Add/remove port mappings (with lease times)



- Drawbacks
  - No security, evil applications can establish port forwarding entries
  - Doesn‘t work with cascaded NATs

Behavior based (1): STUN

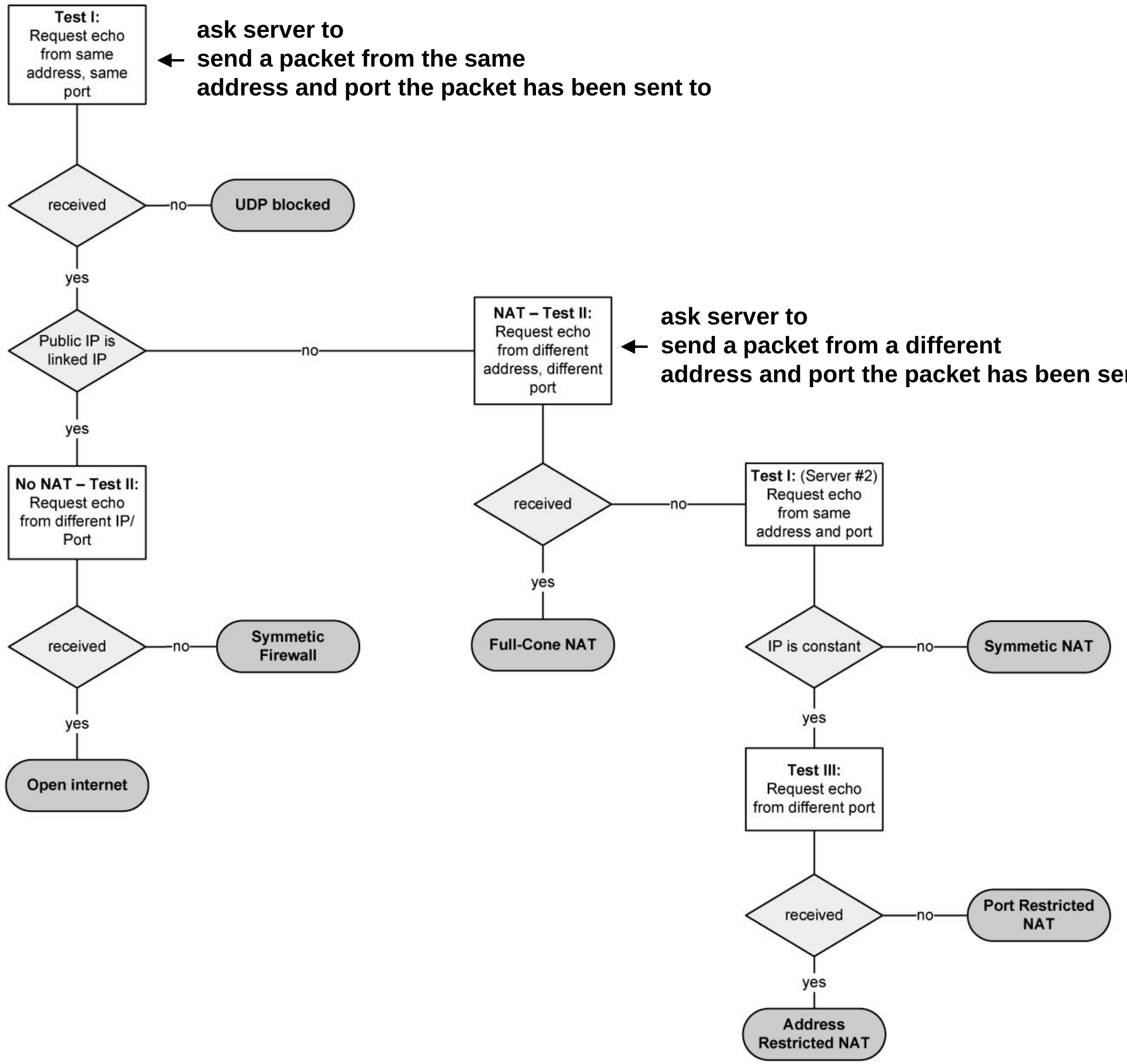
- Simple traversal of UDP through NAT (old) (RFC 3489)
  - Session Traversal Utilities for NAT (new) (RFC 5389)

- Lightweight client-server protocol
  - Queries and responses via UDP (optional TCP or TCP/TLS)

- Helps to determine the external transport address (IP address and port) of a client.
  - E.g. query from 192.168.1.1:5060 results in 131.1.2.3:20000

- Algorithm to discover NAT type
  - Server needs 2 public IP addresses

STUN Algorithm

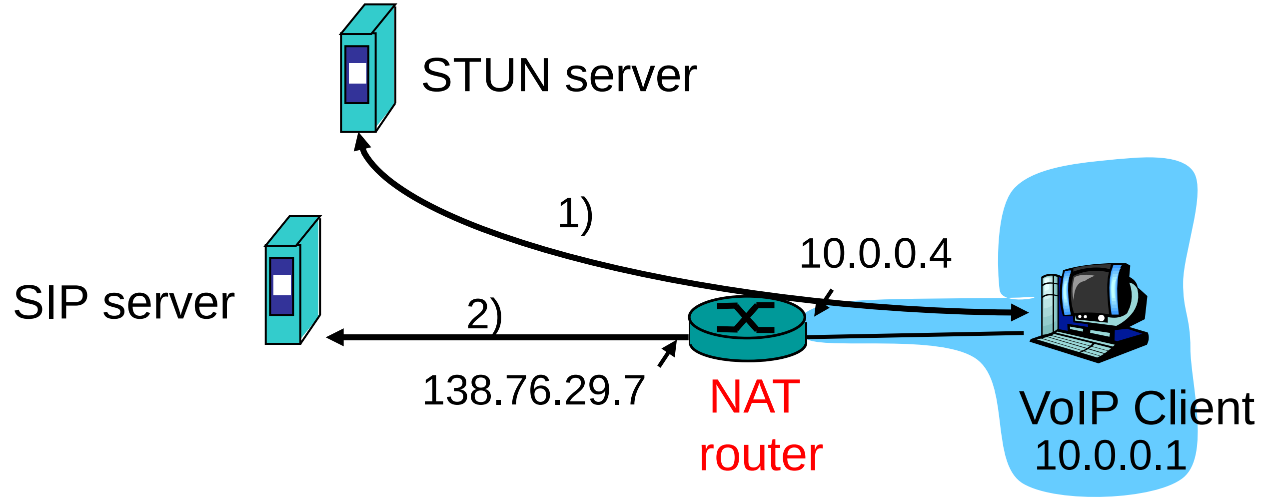


ask server to
send a packet from the same
address and port the packet has been sent to

ask server to
send a packet from a different
address and port the packet has been sent to

Example: STUN and SIP

- VoIP client queries STUN server
  - learns its public transport address
  - can be used in SIP packets



Request/Respone Line	INVITE sip:Callee@200.3.4.5 SIP/2.0
Message-Header	Via: SIP/2.0/UDP **138.76.29.7:5060** From: < sip:Caller@**138.76.29.7** > To: <sip:Callee@200.3.4.5> CSeq: 1 INVITE Contact: <sip:Caller@**138.76.29.7:5060**> Content-Type: application/sdp

Limitations of STUN

- STUN only works if
  - the NAT assigns the external port (and IP address) only based on the source transport address
  - Endpoint independent NAT binding
    - Full Cone NAT
    - Address Restricted Cone NAT
    - Port Address restricted cone NAT
  - Not with symmetric NAT!

- Why?
  - Since we first query the STUN server (different IP and port) and then the actual server
  - The external endpoint must only be dependent on the source transport address

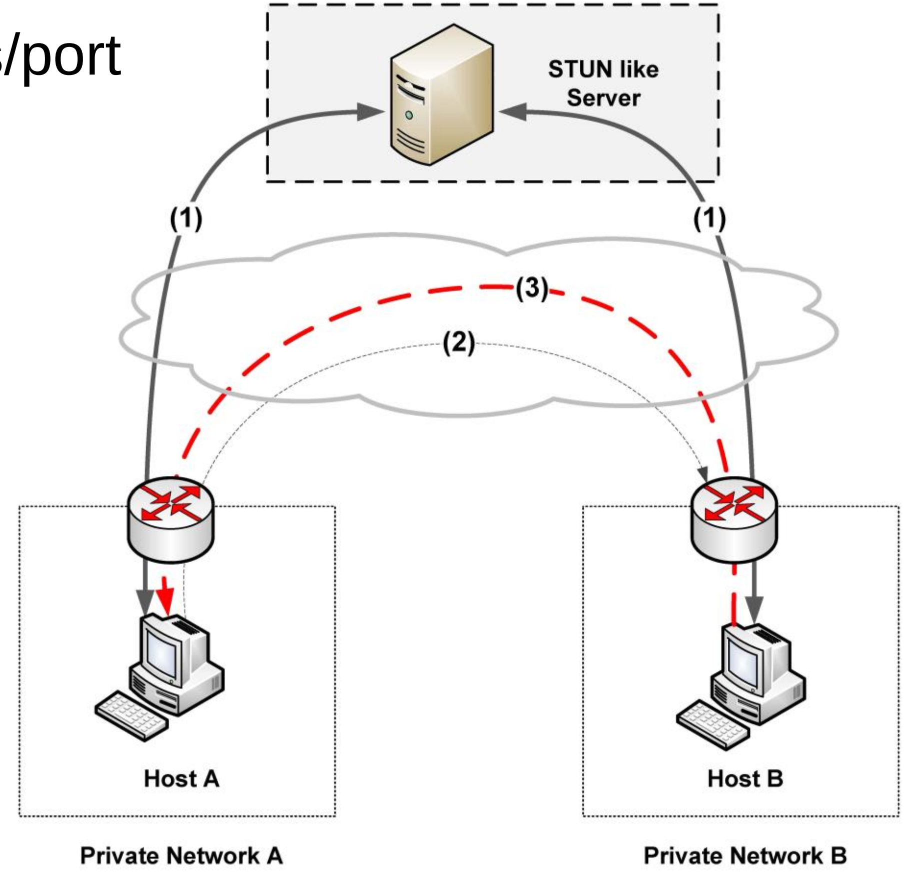
STUN and Hole Punching

- STUN not only helps if we need IP addresses in the payload
  - also for establishing a direct connection between two peers

1) determine external IP address/port and exchange it through Rendezvous Point

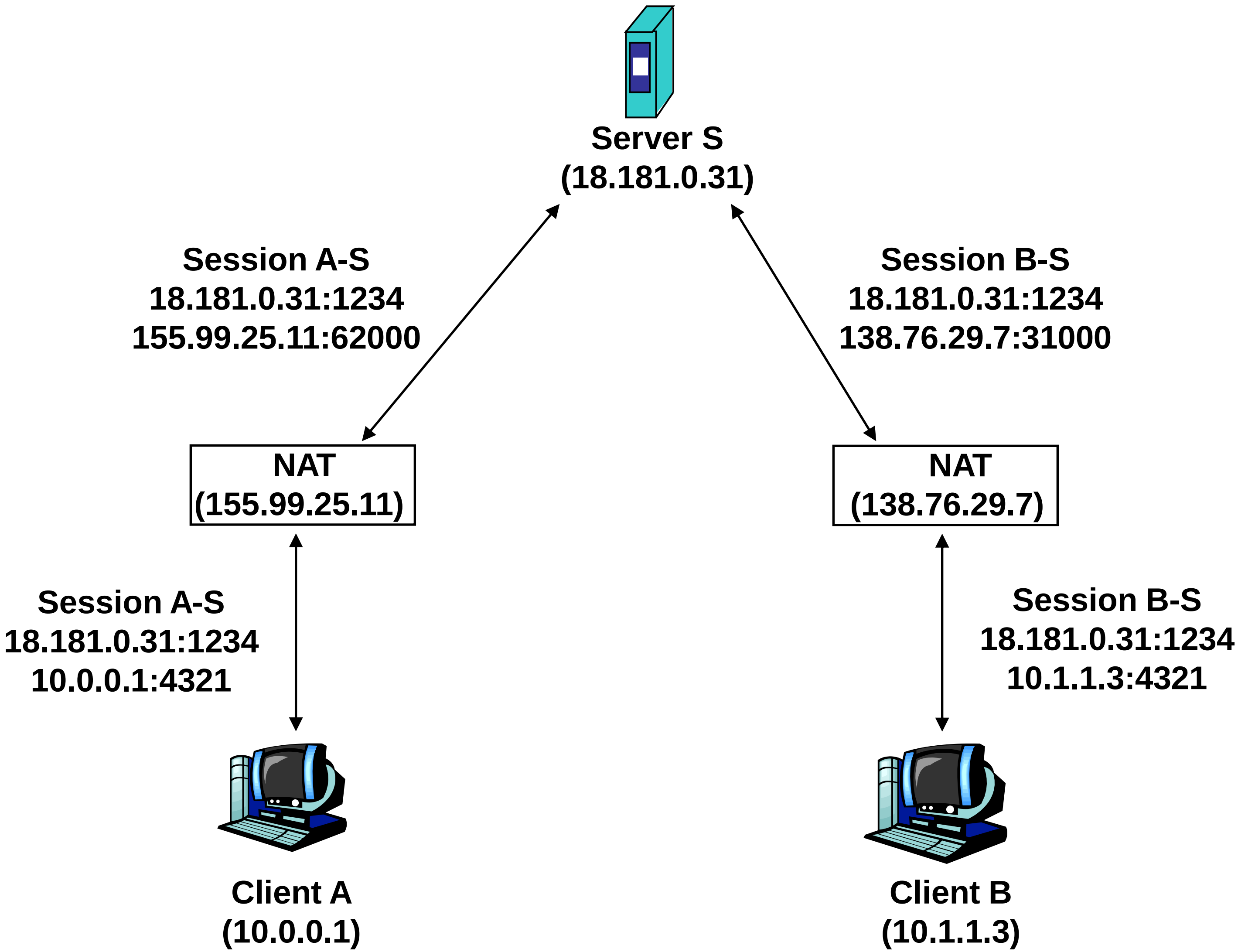
2) both hosts send packets towards the other host outgoing packet creates hole

3) establish connection. hole is created by first packet



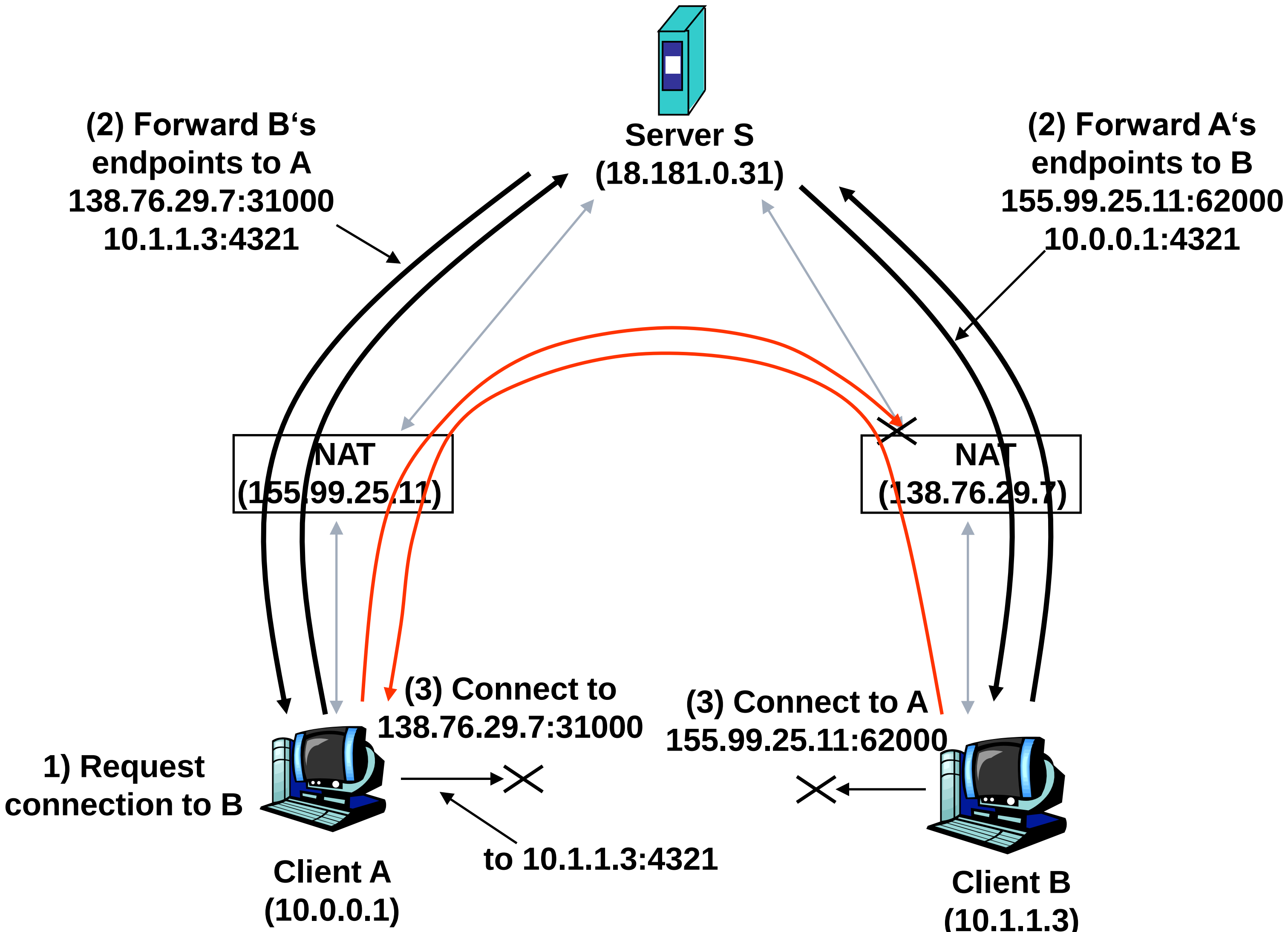
Hole Punching in detail

- Before hole punching



Hole Punching in detail

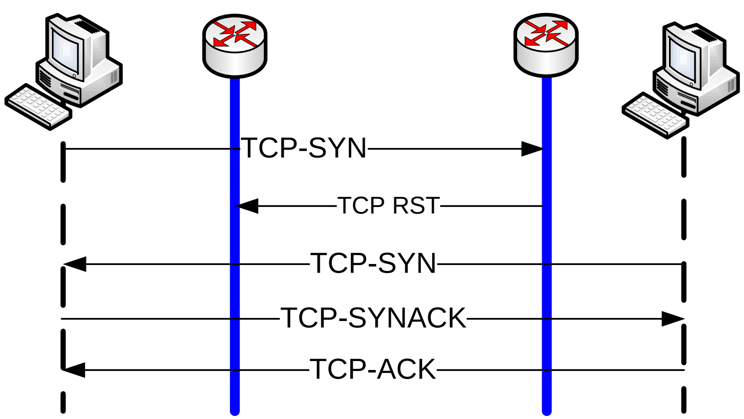
- Hole punching

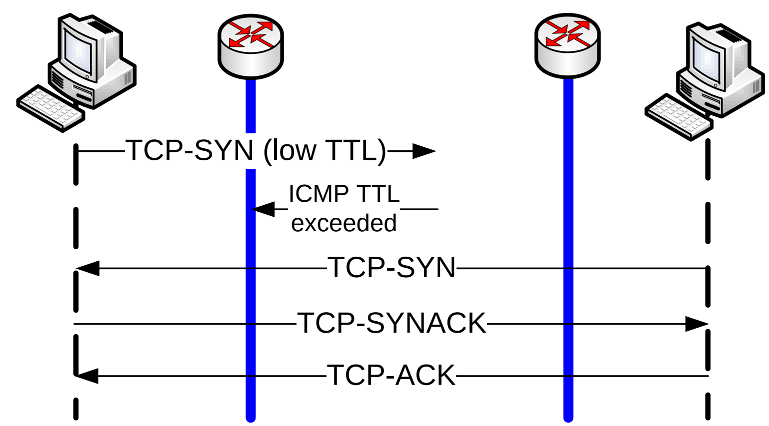


TCP Hole Punching

- Hole Punching not straight forward due to stateful design of TCP
  - 3-way handshake
  - Sequence numbers
  - ICMP packets may trigger RST packets

- Low/high TTL(Layer 3) of Hole-Punching packet
  - As implemented in STUNT (Cornell University)

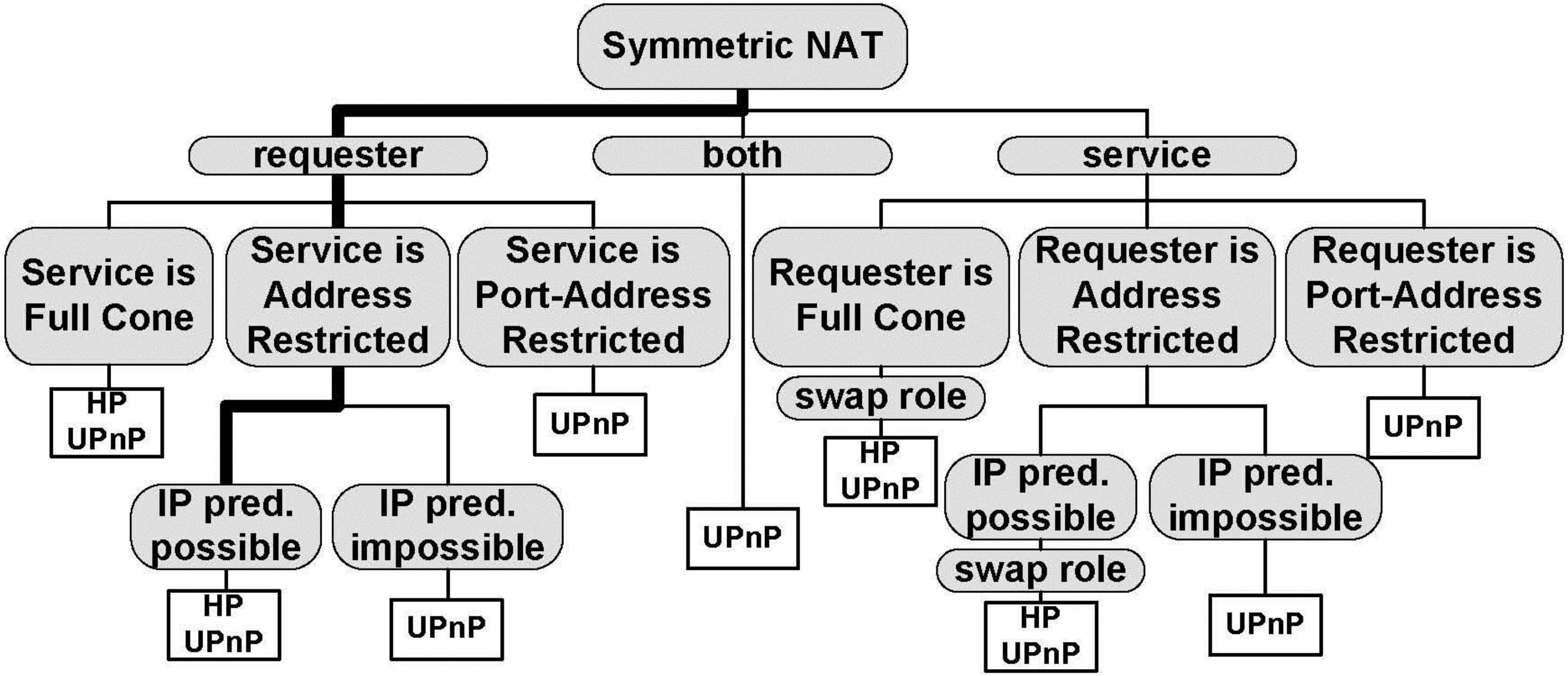




- Bottom line: NAT is not standardized

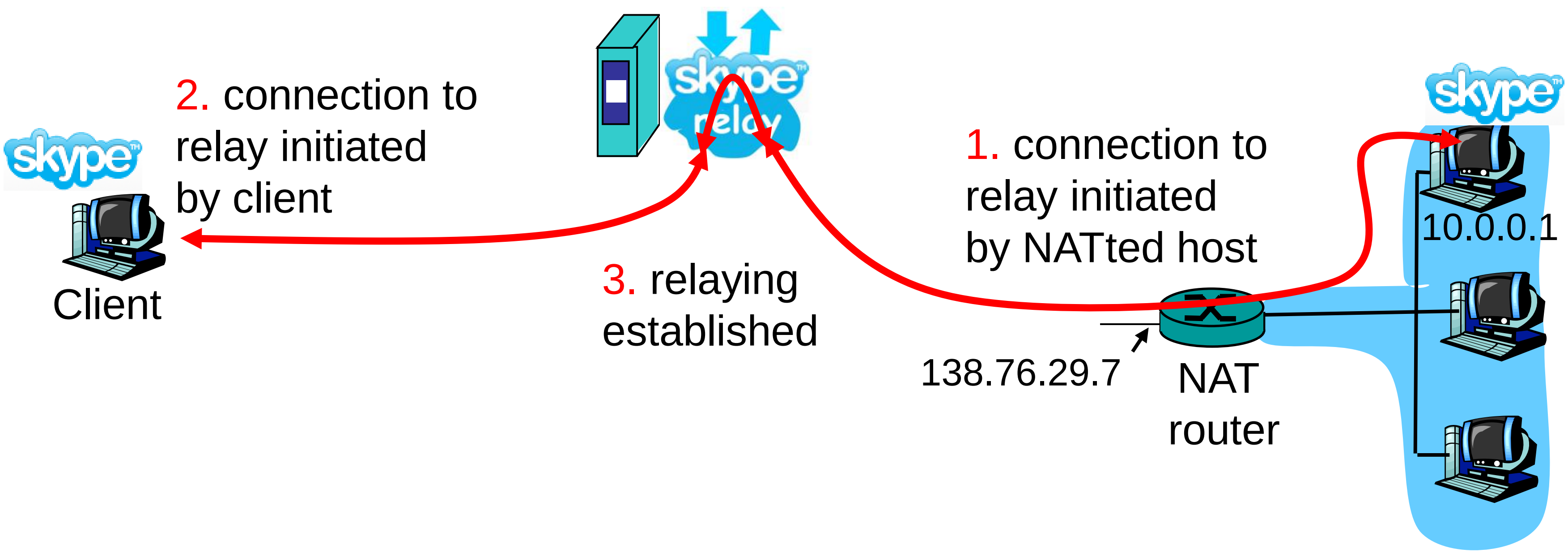
Symmetric NATs

- How can we traverse symmetric NATs
  - Endpoint dependent binding
    - hole punching in general only if port prediction is possible
  - Address and port restricted filtering



Data Relay

- relaying (used in Skype)
  - NATed client establishes connection to relay
  - External client connects to relay
  - relay bridges packets between to connections
  - Traversal using Relay NAT (TURN) as IETF draft



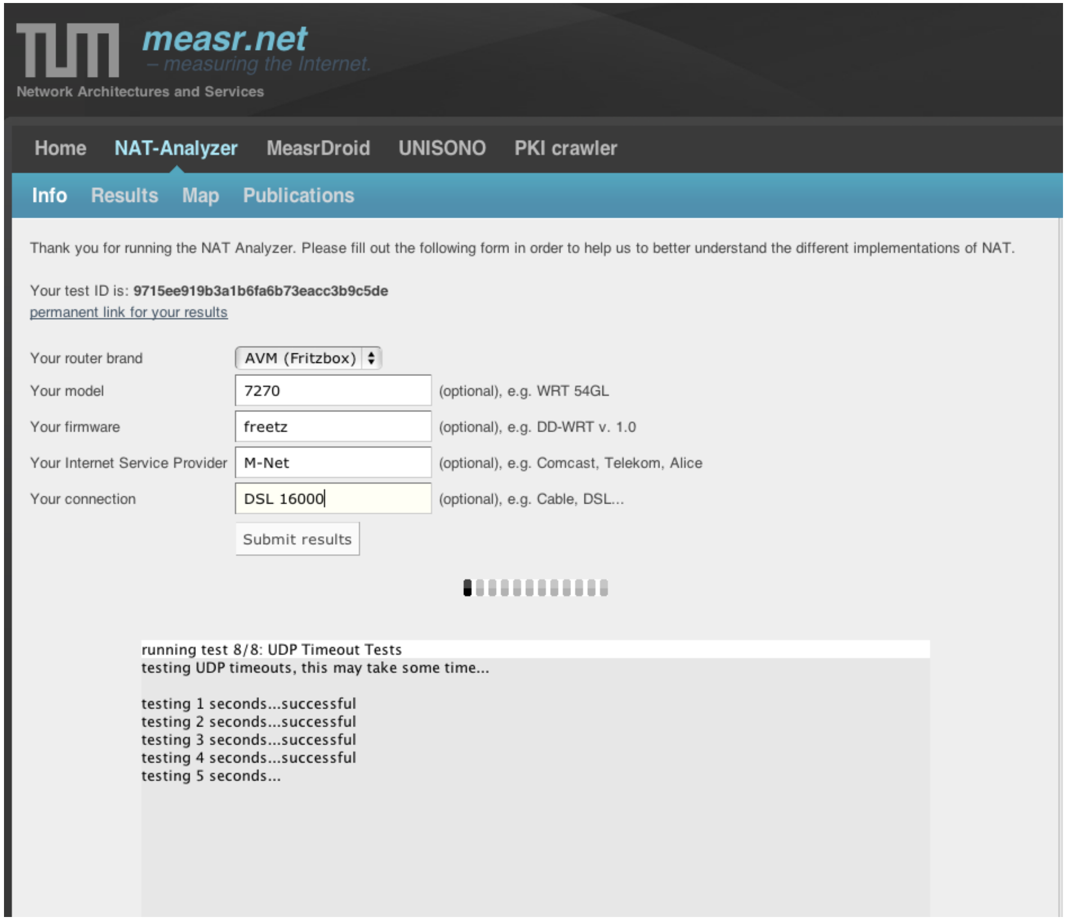
Frameworks

- Interactive Connectivity Establishment (ICE)
  - IETF draft
  - mainly developed for VoIP
  - signaling messages embedded in SIP/SDP
- All possible endpoints are collected and exchanged during call setup
  - local addresses
  - STUN determined
  - TURN determined
- All endpoints are „paired“ and tested (via STUN)
  - best one is determined and used for VoIP session
- Advantages
  - high sucess rate
  - integrated in application
- Drawbacks
  - overhead
  - latency dependent on number of endpoints (pairing)

NAT Analyzer

- Public field test with more than 1500 NATs
  - understand existing traversal techniques and NAT behavior

(http://nattest.net.in.tum.de)



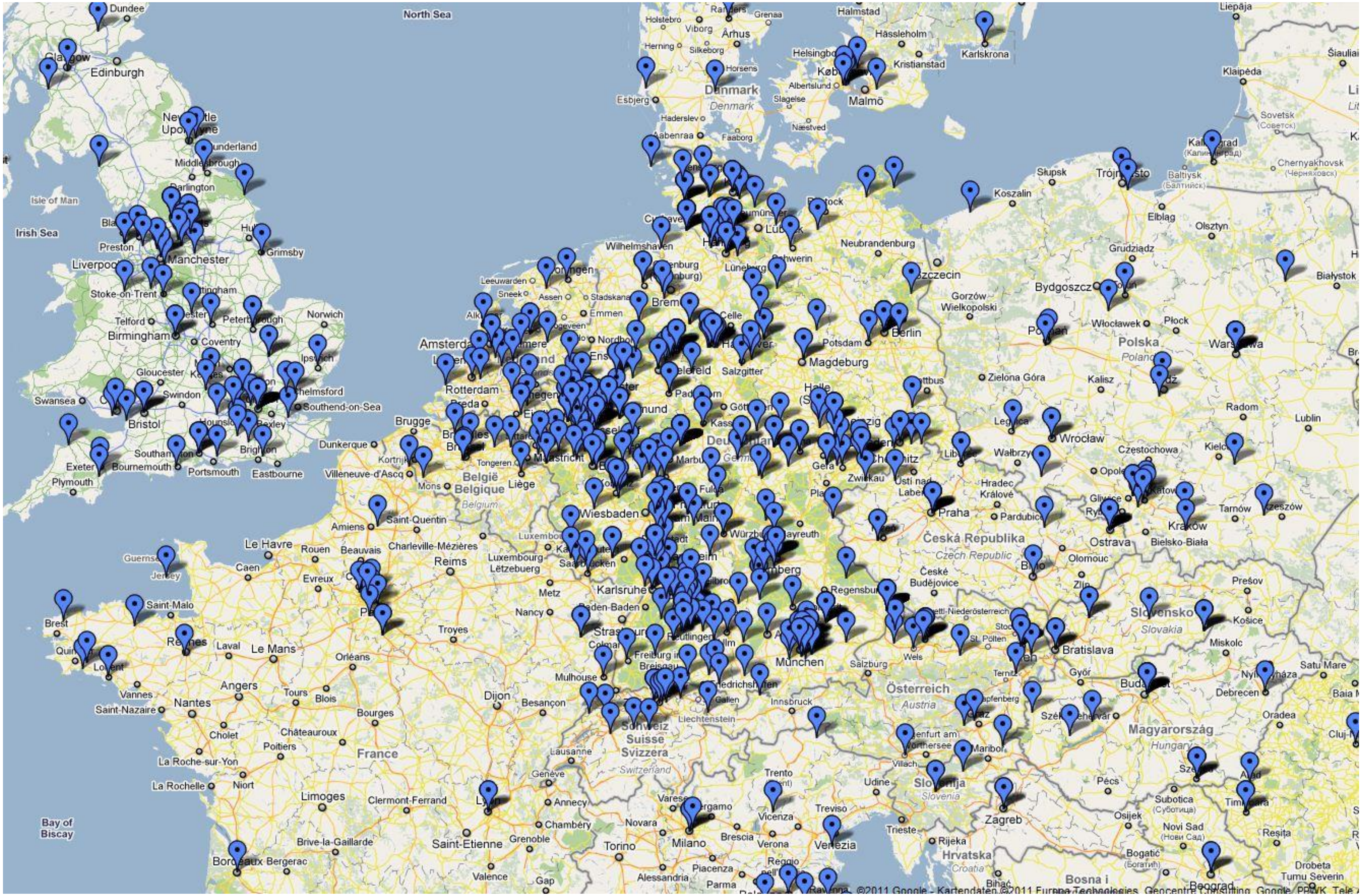
NAT Analyzer Tests

- Connectivity tests with a server at TUM
  - NAT Type
  - Mapping strategy
  - Binding Strategy
  - Hole Punching behavior using different techniques
  - Timeouts
  - ALGs

- Example Result

NAT Analyzer Participants (World)

NAT Tester Participants (Central Europe)



NAT Analyzer Results