Peer-to-Peer Systems and Applications

Lecture 10: Selected Topics II

TECHNISCHE UNIVERSITÄT DARMSTADT

* Original slides provided by Thomas Bocek (University of Zurich), David Hausheer (TU Darmstadt, Germany)

0. Lecture Overview

1. Kademlia
   1. Routing Concept
   2. Protocol
   3. Routing Table Construction
   4. Peer Selection Policy
   5. Enhancements
2. Network Coding
   1. Motivation
   2. Theory
   3. Example
   4. Application
3. WebRTC
   1. Introduction
   2. Architecture
   3. Example
   4. Outlook
4. NAT Traversal
   1. Background
   2. Relaying
   3. Hole Punching

1. Kademlia

Routing Concept, Protocol, Routing Table Construction,
Peer Selection Policy, Enhancements

1.0 Kademlia

TECHNISCHE UNIVERSITÄT DARMSTADT

- Use α parallel iterative lookup to locate data
  - Retrieve data faster
  - Overcome faulty nodes
  - Usually α = 3
- DHT-based overlay network using the XOR distance metric
  - Simple operation
  - Symmetrical routing paths (A → B == B → A)
    - due to $d_{XOR}(A,B) == d_{XOR}(B,A)$
- Store data with key X on k nodes closest to X according to XOR metric
  - “build in” replication ensuring data availability
  - Usually k = 20
- Use lookup messages to maintain the overlay network
  - Learn useful routing information from received lookup requests

XOR Distance Calculation:

ID Node A: 110101
ID Node B: 010001

$d_{XOR}(A,B) = d(110101,010001)$

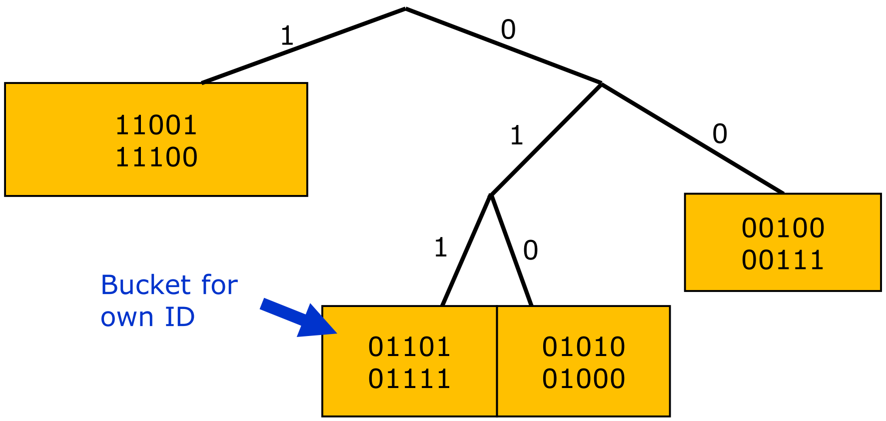
1 1 0 1 0 1
XOR
0 1 0 0 0 1
↓
1 0 0 1 0 0

$d_{XOR}(A,B) = 100100_2 = 36_{10}$

1.1. Concept of the Kademlia Routing Process

- Structure of routing table
  - Every node maintains a binary tree like routing table
  - Tree branches along the local node ID
  - Every leave is a bucket with k entries

Example routing table for node 01110 with bucket factor k = 2



Lookup procedure for a key X

Step 1: Traverse routing table tree and pick bucket with node IDs closest to key X

Step 2: Put node IDs in a node list

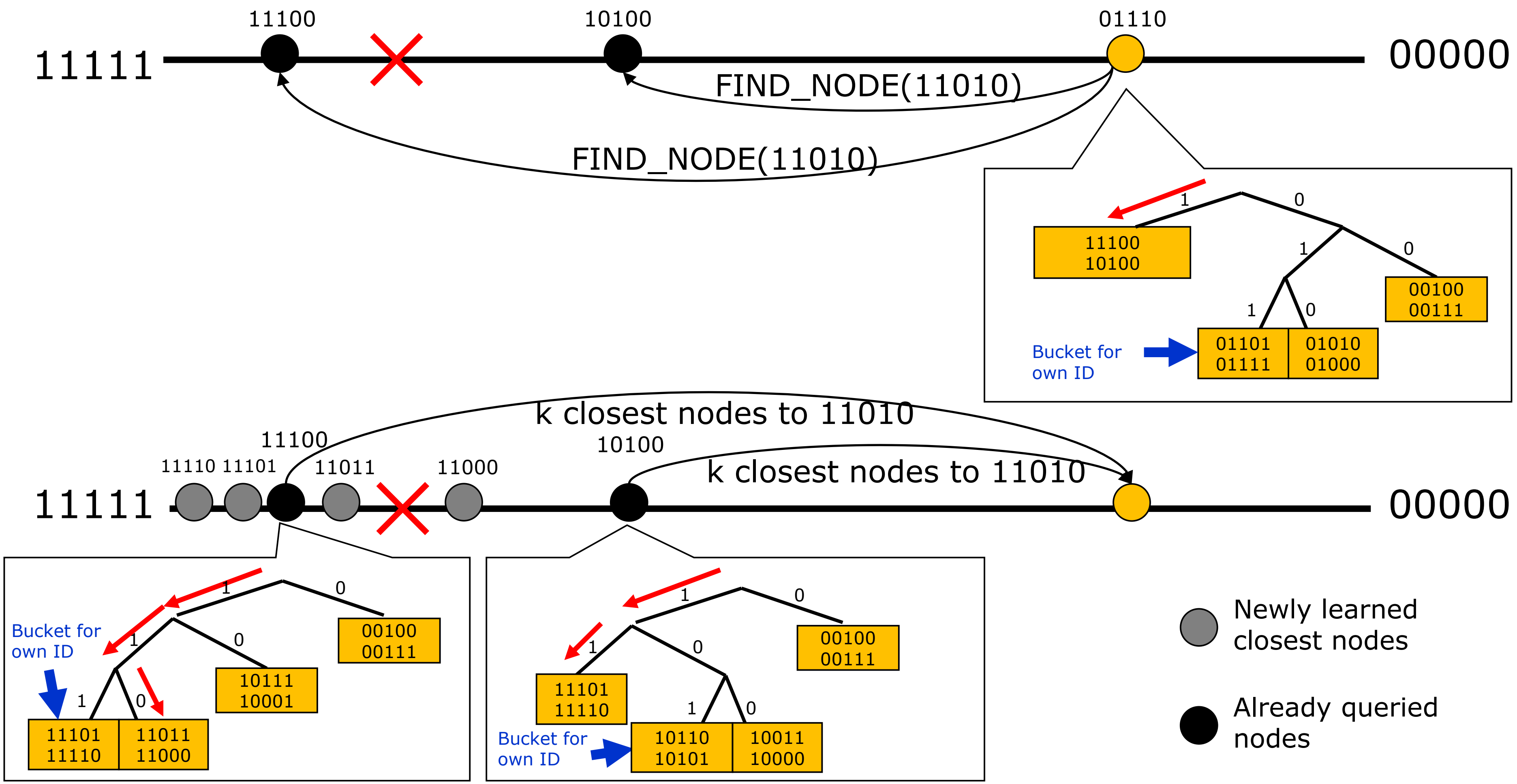
Step 3: Send a closest nodes request to nodes in the list closest to X

Step 4: Put received node IDs in node list

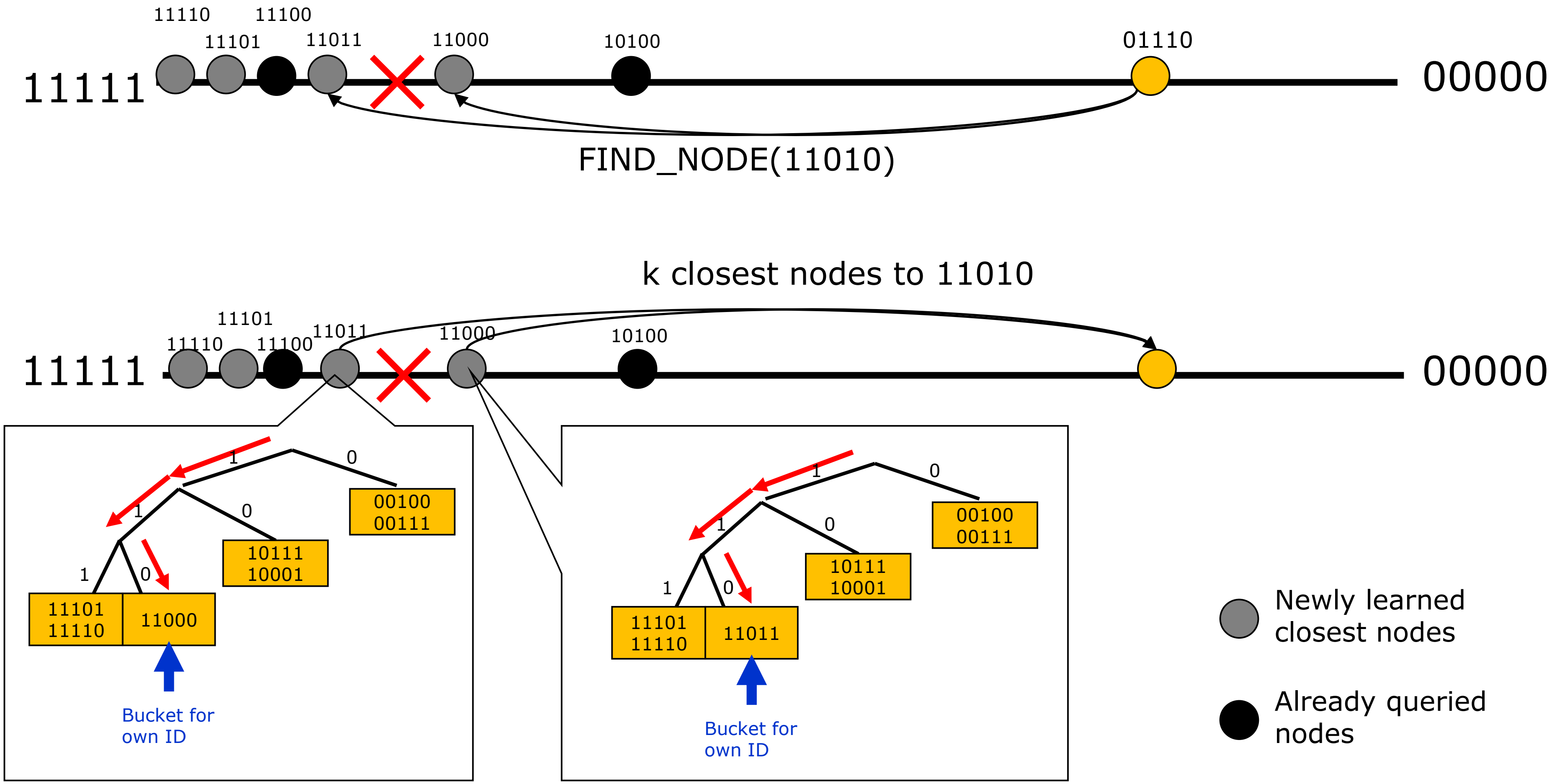
Step 5: Repeat until Step 3 and 4 until set of k closest node to not change anymore

Step 6: Pick k-closest nodes from node list and send store or get data request to them

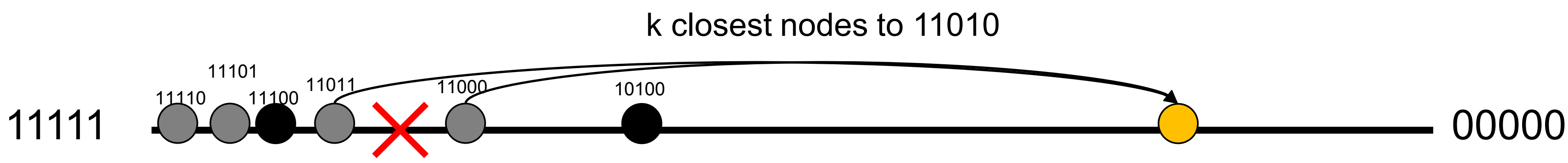
1.1. Routing from 01110 to 11010, α = 2, k = 2



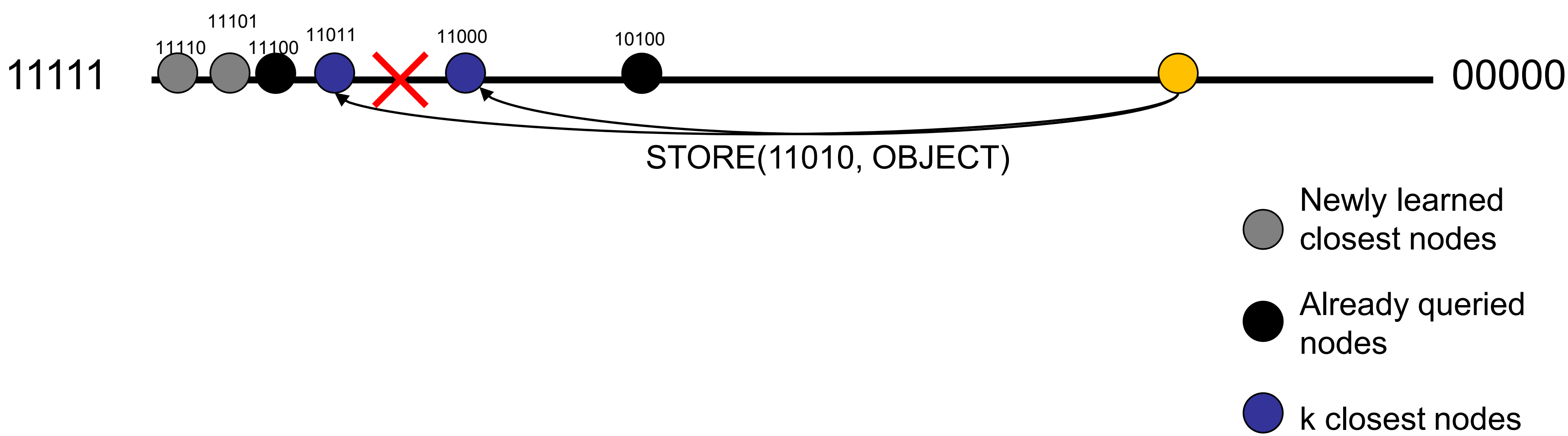
1.1. Routing from 01110 to 11010, α = 2, k = 2



1.1. Routing from 01110 to 11010, α = 2, k = 2



No new nodes discovered → stop lookup process



1.2. Kademlia Protocol

- The Kademlia protocol consists of 4 RPCs:
  - FIND_NODE(KEY):
    - Recipient returns <IP Address, UDP Port, Node ID> triples for k closest nodes he knows about
  - FIND_VALUE(KEY):
    - Like FIND_NODE
    - With exception:
      - If recipient already stores the value, the value is returned instead of k closest nodes
  - PING(IP ADDRESS)
    - Probes a node to see if it is online
  - STORE(KEY, VALUE)
    - Instructs a node to store a <key,value> pair

1.3. Construction of Routing Table

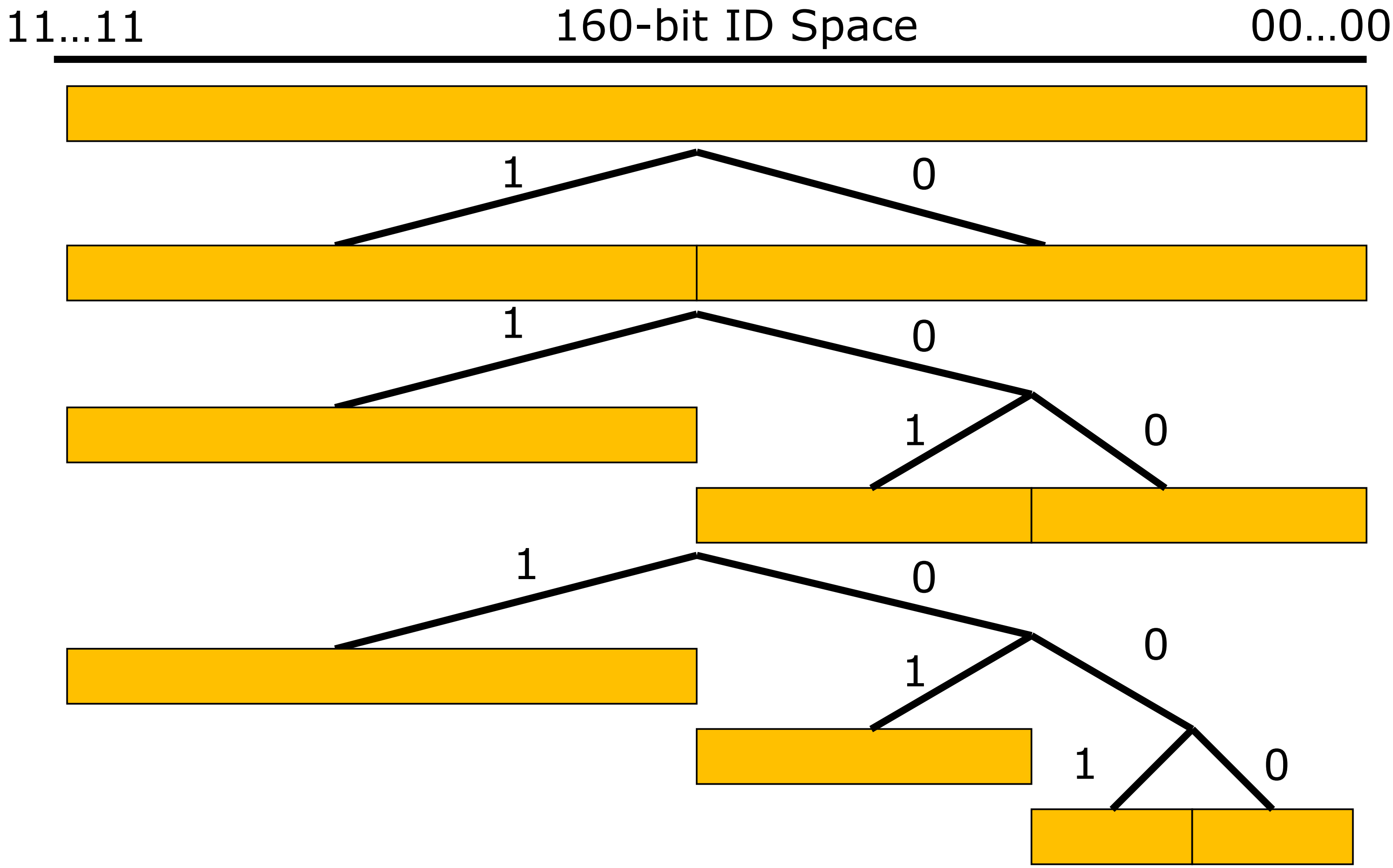
- Each node maintains routing table (k buckets)
  - Routing tables of different peers may be different

- For each 0 <= i < 160 every node
  - keeps a list of <IP Address, UDP Port, Node ID> triples
  - for k nodes within range [2 ^i ; 2 ^(i+1)[
  - in total k * 160 contacts

- Nodes learn from
  - messages they receive or
  - using the FIND_NODE method

- Preference towards old contacts
  - Study has shown that the longer a node has been up,
    the more likely it is to remain up another hour
  - Resistance against DoS attacks by flooding the network with new nodes

1.3. Evolution of the k Buckets



1.3. Example for Node ID 01110, k=2

11111 5-bit ID Space 00000

New node 11001:

11001

New node 01101:

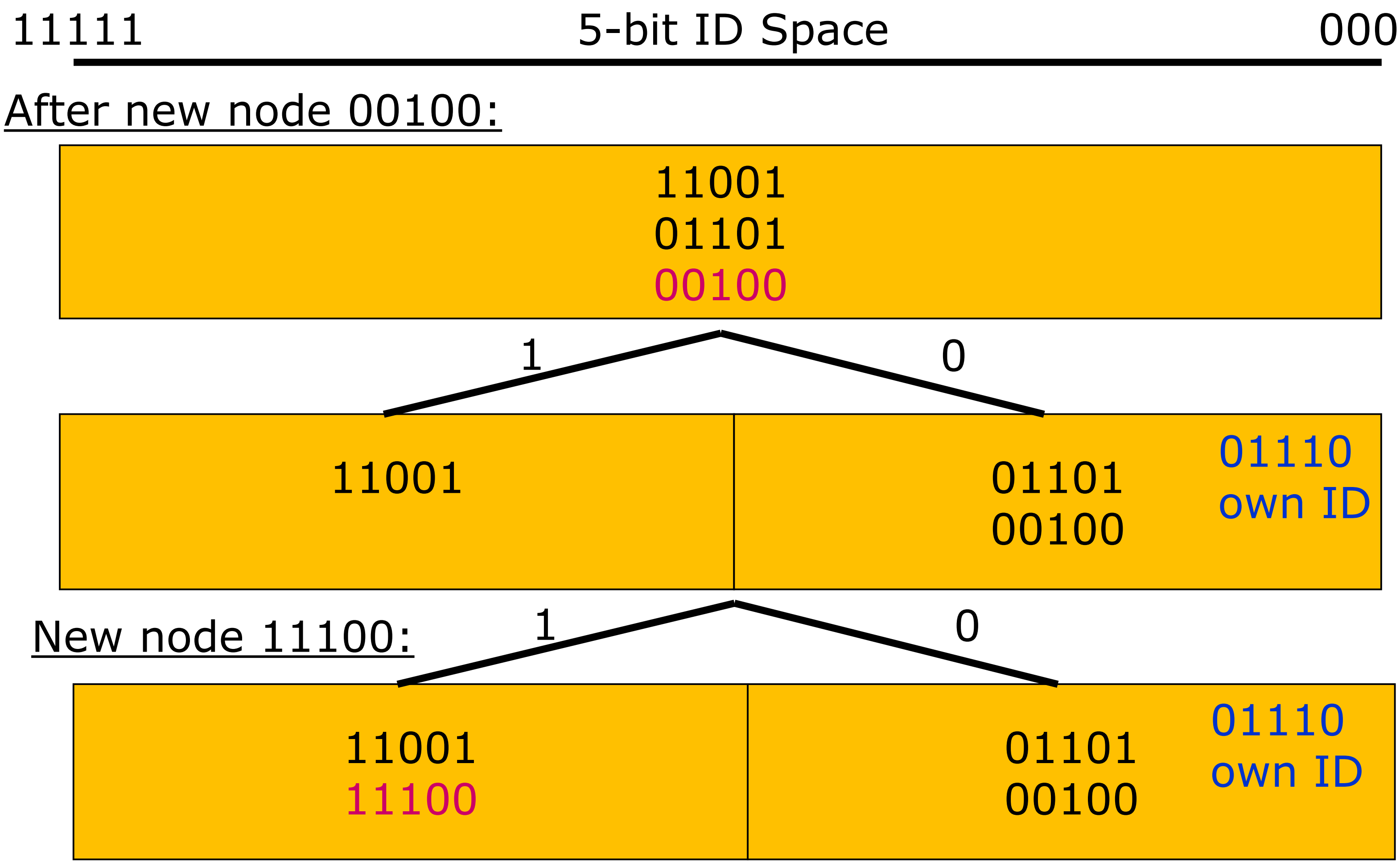
11001
01101

New node 00100:

11001
01101
00100 but … no because 3rd, and k=2 is max.

k bucket is full → split necessary

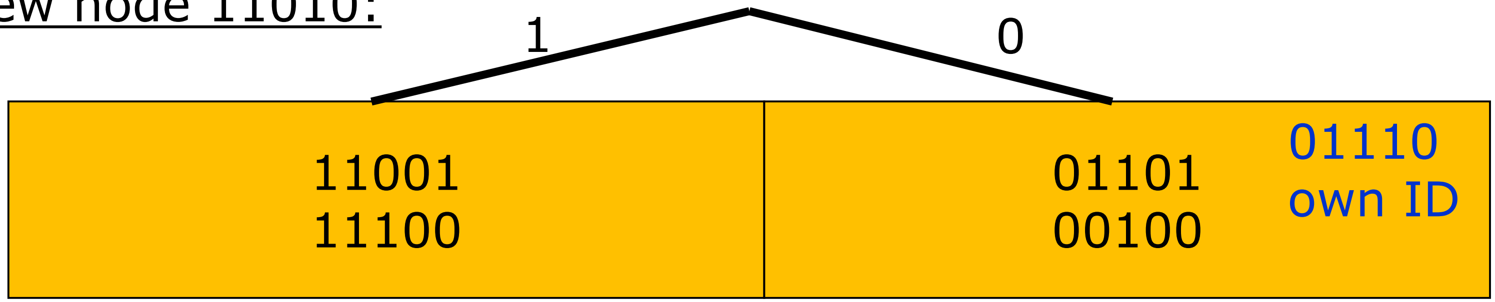
1.3. Example for Node ID 01110, k=2



1.3. Example for Node ID 01110, k=2

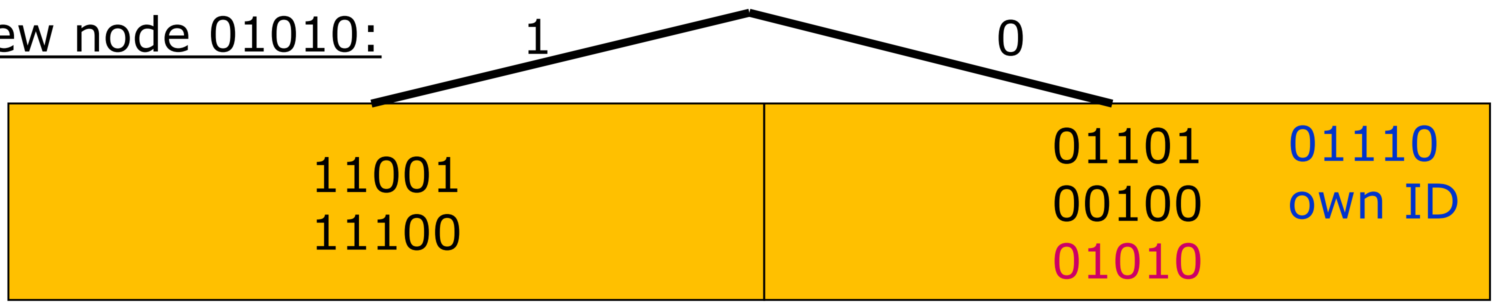
11111 5-bit ID Space 00000

New node 11010:



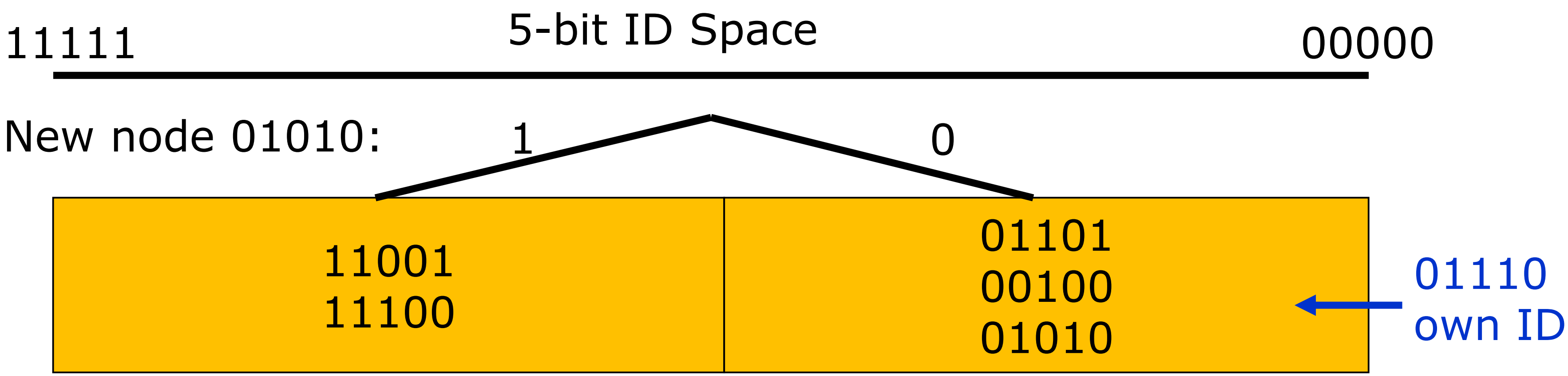
Left k bucket full and 11010 NOT in k-bucket-range of 01110 → node is dropped

New node 01010:

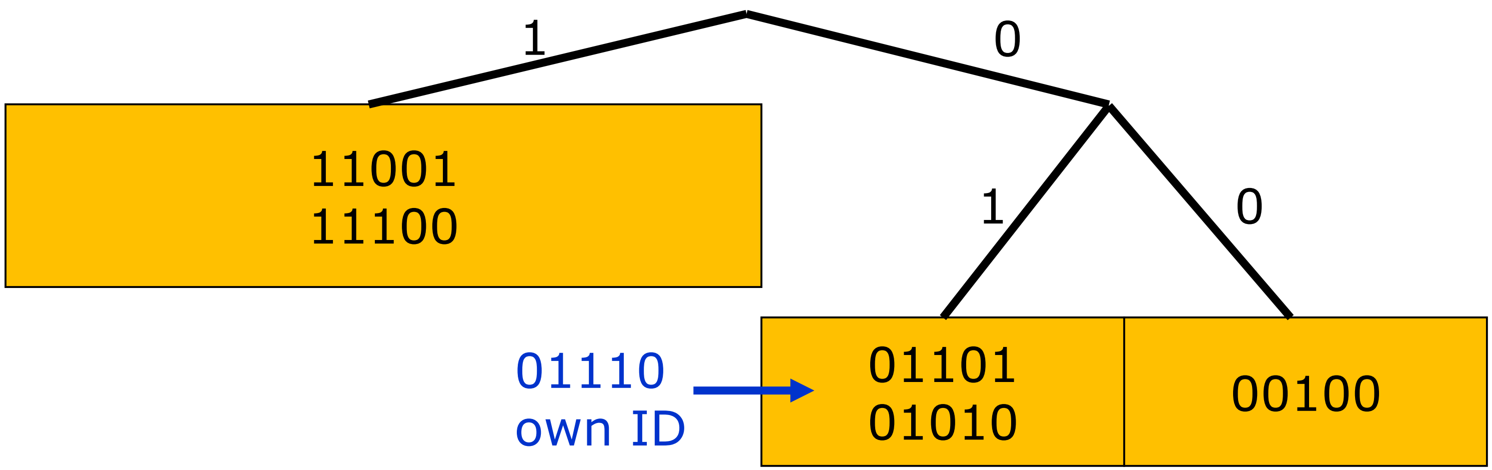


Right k bucket full and 01010 in k-bucket-range of 01110 → split necessary

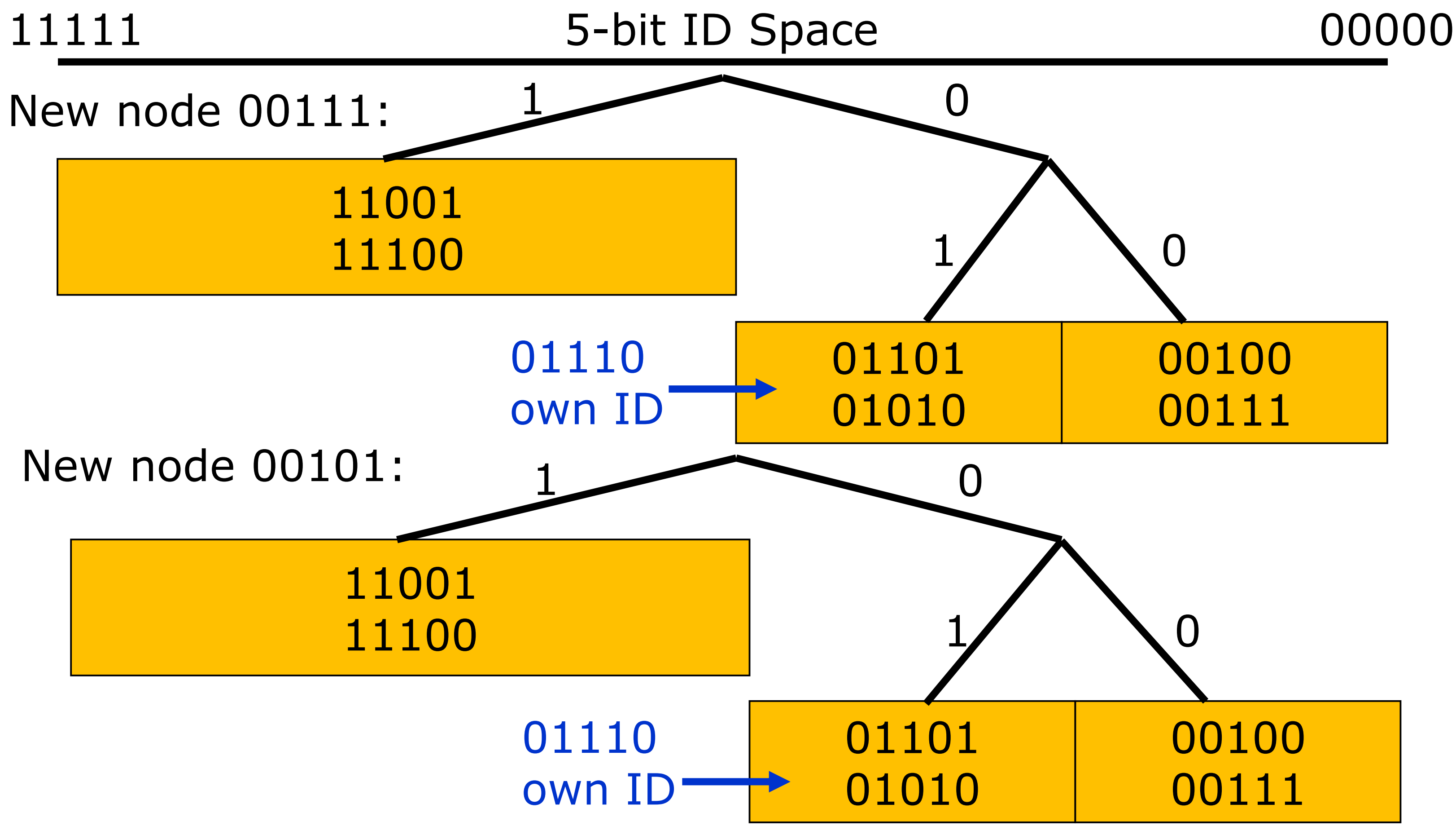
1.3. Example for Node ID 01110, k=2



Right K bucket full and 01010 in k-bucket-range of 00110 → split necessary



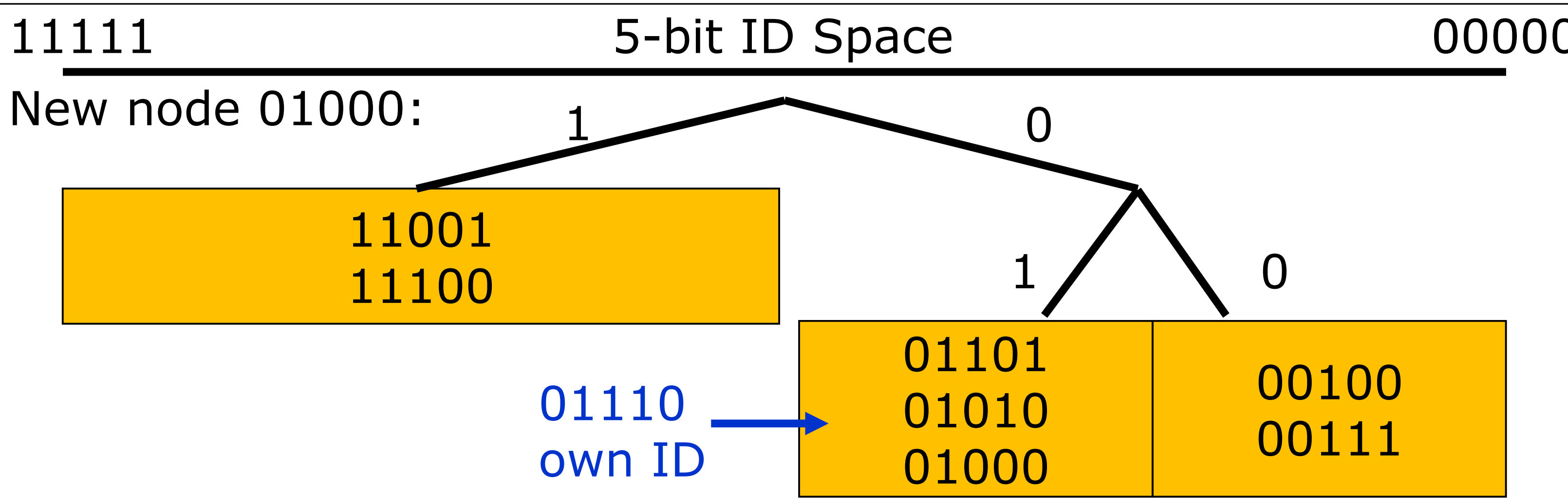
1.3. Example for Node ID 01110, k=2



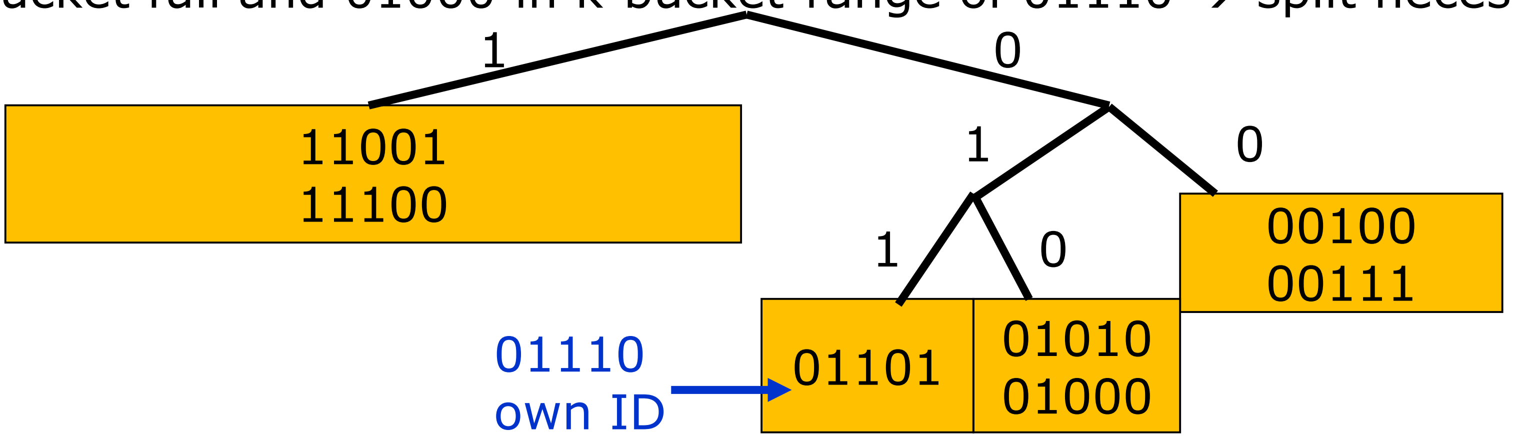
Left K bucket full and 00101 NOT in k-bucket-range of 01110 → node is dropped

1.3. Example for Node ID 01110, k=2

New node 01000:



K bucket full and 01000 in k-bucket-range of 01110 → split necessary



1.4. k Buckets: Peer Selection Policy

TECHNISCHE UNIVERSITÄT DARMSTADT

- If node u receives a message from node v then it adds node v to its k-bucket according to the following rules:

  - IF v is already in a k-bucket
    THEN move v to the tail of the bucket

  - IF v is not in the k-bucket and the bucket has fewer than k entries
    THEN insert recipient to the tail of list

  - IF the appropriate k bucket is full AND least recently seen node is alive
    THEN move least recently seen node to tail of bucket and discard node v

  - IF the appropriate k bucket is full AND least recently seen node is <u>not</u> alive
    THEN remove least recently seen node from bucket and add node v at the tail

- Note: approx. k = 20 in the real world

1.5. Kademlia: Enhancements

- Parallel queries
  - For one query, α (alpha) concurrent lookups are sent
  - More traffic load, but lower response times

- Network maintenance
  - In Chord: active fixing of fingers
  - In Kademlia: learning for bypassing queries
  - Check if peer IDs fit better in routing table

- Large routing tables
  - In Chord: 1 finger per distance 2^i to 2^(i+1)
  - In Kademlia: k contacts per distance 2^i to 2^(i+1)
  - Increased robustness

2. Network Coding

Motivation, Theory, Example, Application

2.1. Network Coding Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

- P2P problem area: distribute large data packets to many clients
  - Solution 1: exchange piece/chunk bitmap
    - Request contains the pieces we are looking for, recipient replies with piece or that pieces are not here
    - Peer may also collect piece/chunk bitmaps first
  - Solution 2: use network coding
    - Simple request, when there is nothing new, don't request.

- Random network coding: send linear combination of all received chunks

→ no need to exchange piece/chunk information

2.2. Network Coding Theory

TECHNISCHE UNIVERSITÄT DARMSTADT

- `M1, …, Mn` → original packets / `g1, …, gn` → random coefficients

- Encoding: $X = \sum_{i=1}^{n} g_i M^i$
  - Encoding vector `g`, information vector `X`
  - Send `X,g`

- Re-encoding, `h1, …, hm` → random coefficients:

$$X' = \sum_{j=1}^{m} h_j X^j$$

$$g_i' = \sum_{j=1}^{m} h_j g_i^j$$

  - Encoding vector `h` → `g'`
  - Information vector `X'`
  - Send `X',g'`

2.3. Network Coding Example

- Peer 1 chooses coefficients (encoding vector), calculates linear combination, sends two to Peer 2 and one to Peer 3

Coefficient (randomly chosen or try not to be linear dependant)

Peer 1

Peer 1	M1=17	1,3,2
	M2=8	2,1,3
	M3=9	2,3,1

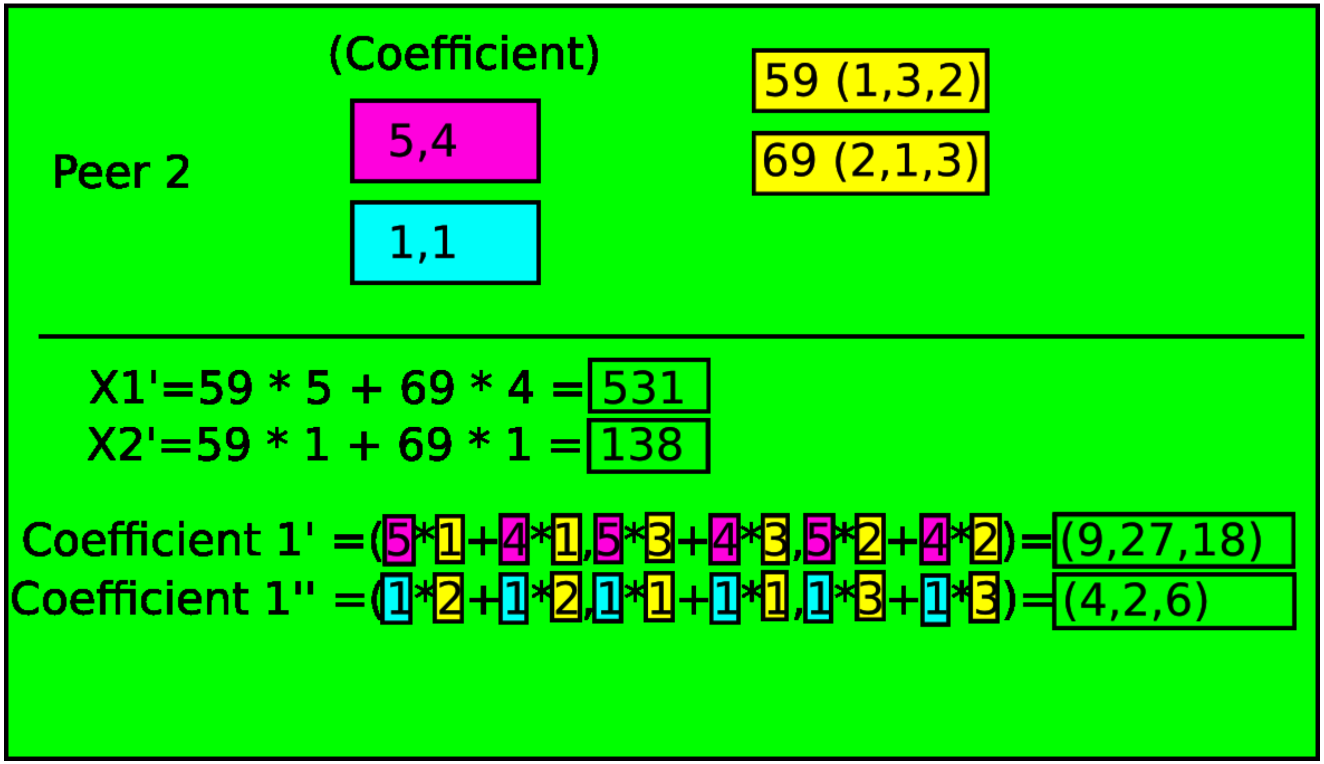
X1= 1 * 17 + 3 * 8 + 2 * 9 = 59 (1,3,2)

X2= 2 * 17 + 1 * 8 + 3 * 9 = 69 (2,1,3)

X3= 2 * 17 + 3 * 8 + 1 * 9 = 67 (2,3,1)

2.3. Network Coding Example

- Peer 2 also has random coefficients, gets two linear comb. from Peer 1, creates new linear comb. (adapt coeff.), send to Peer 3



2.3. Network Coding Example

TECHNISCHE UNIVERSITÄT DARMSTADT

- Peer 3 gets all 3 linear comb. with coding vector, does Gauss elimination (linear system with 3 equations & 3 unknowns) [1]

Peer 3

67 (2,3,1)
531 (9,27,18)
138 (4,2,6)

$$
\begin{aligned}
9M1 + 27M2 + 18M3 &= 531\\
4M1 + 2M2 + 6M3 &= 138\\
2M1 + 3M2 + 1M3 &= 67\\
\hline
0M1 - 10M2 - 2M3 &= -98\\
0M1 - 3M2 - 3M3 &= -51\\
\hline
0M1 + 0M2 + 1M3 &= 9
\end{aligned}
$$

M1=17

M2=8

M3=9

[1] http://www.gregthatcher.com/Mathematics/GaussJordan.aspx

2.3. Network Coding Example

- Peer 3 gets other linear comb. with coding vector, does Gauss elimination (linear system with 3 equations & 3 unknowns) [1]

X1 = 1 * 17 + 3 * 8 + 4 * 9 = 77 (1,3,4)

Peer 3

531 (9,27,18)

138 (4,2,6)

$$
\begin{aligned}
9M1 + 27M2 + 18M3 &= 531 \\
4M1 + 2M2 + 6M3 &= 138 \\
1M1 + 3M2 + 4M3 &= 77 \\
\hline
0M1 - 10M2 - 2M3 &= -98 \\
0M1 - 0M2 - 2M3 &= 18 \\
\hline
0M1 + 0M2 + 1M3 &= 9
\end{aligned}
$$

M1=17

M2=8

M3=9

[1] http://www.gregthatcher.com/Mathematics/GaussJordan.aspx

2.4. Network Coding Application

- Send packets as long as linear independant
  - No exchange of bitmap
- Random coefficients: low probability of collision
- Overhead of transmitting the encoding vectors is small, size of block is Kbytes
  - CPU overhead $O(n^3)$, (n = number of blocks)
- Avalanche: File Swarming with Network Coding [1]
  - Network Coding improves robustness and throughput

[1] http://research.microsoft.com/en-us/projects/avalanche/

3. WebRTC

Introduction, Architecture, Example, Outlook

3.1. WebRTC – Introduction (1)

- WebRTC for browser to browser communication
  - P2P, no server involved (~mostly)
- Google bought in 2010 GIPS and open sourced WebRTC
- Protocol standardized by IETF (codec requirements, media protocol), JavaScript API by W3C
- Supported by Chrome, Firefox (and others)
- Compatibility [1]
  - WebRTC support since Chrome 26+, Firefox 23+
  - SCTP: supported by Firefox, Chrome 31+
  - Binary data: supported by Firefox, Chrome 31+

[1] http://peerjs.com/status

3.1. WebRTC – Introduction (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Filling gap in the Web-Experience

  - Video Chat → Google Hangouts Plugin, Flash, Java
  - Multimedia / Confernces → Expensive, proprietary 3rd party apps
  - Customer Service → Chat only, 3rd party plugins/apps
  - Online Games → Flash
  - Real-time Feeds → Proprietary software
  - File Sharing → Requires Server / BitTorrent

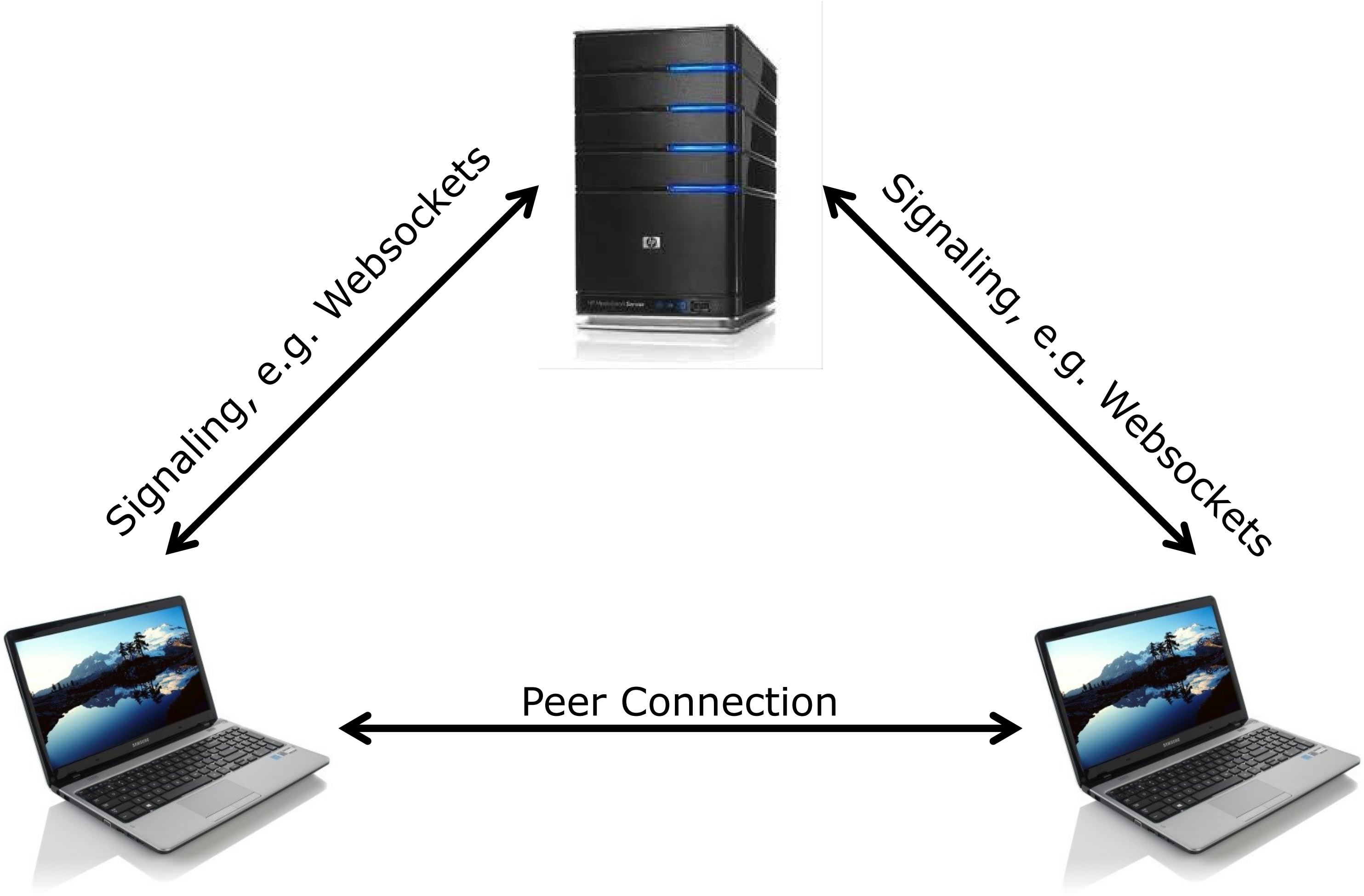
- WebRTC widely deployed, no client necessary!

3.1. WebRTC – Introduction (2)

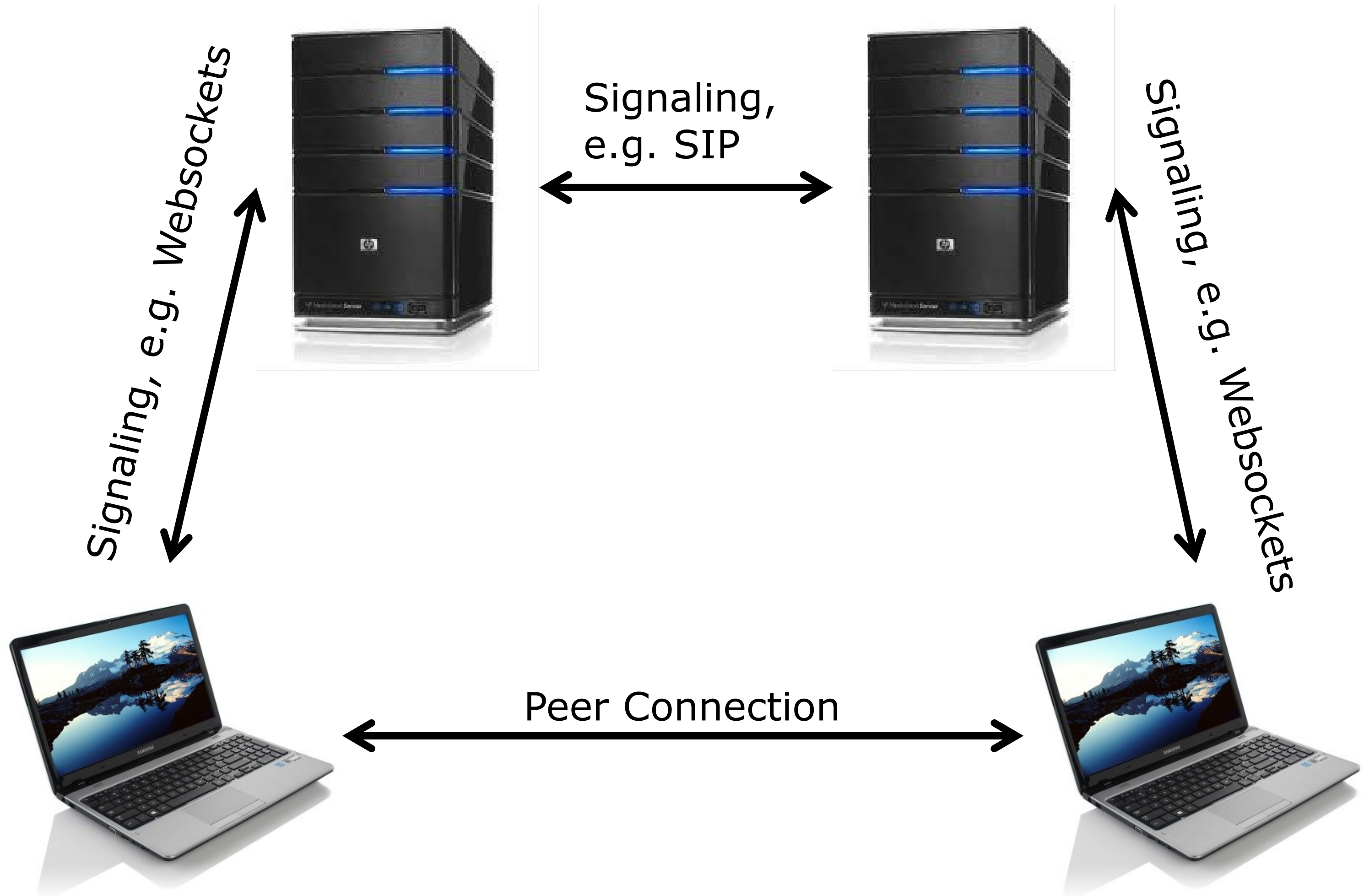
- Developer does not need to care about NAT
  - Abstraction, using STUN, ICE, TURN
  - STUN: session traversal utilities for NAT (detect which kind of NAT)
  - TURN: traversal using relays around NAT (relay)
  - ICE: interactive connectivity establishment , uses STUN and TURN
  - UPnP / NAT-PMP setup by the browser optional? [1]
    - Bugzilla@Mozilla – Bug 860045 [2]
- Once connection is established – easy API
  - `sendChannel.send("hallo")`
  - `sendChannel.onmessage = function …`
- Mandatory AES encryption
  - SRTP for Media, DTLS for Data, HTTPS for Signalling

[1] http://tools.ietf.org/html/draft-kaplan-rtcweb-api-reqs-00
[2] https://bugzilla.mozilla.org/show_bug.cgi?id=860045

3.2. WebRTC Architecture - Triangle

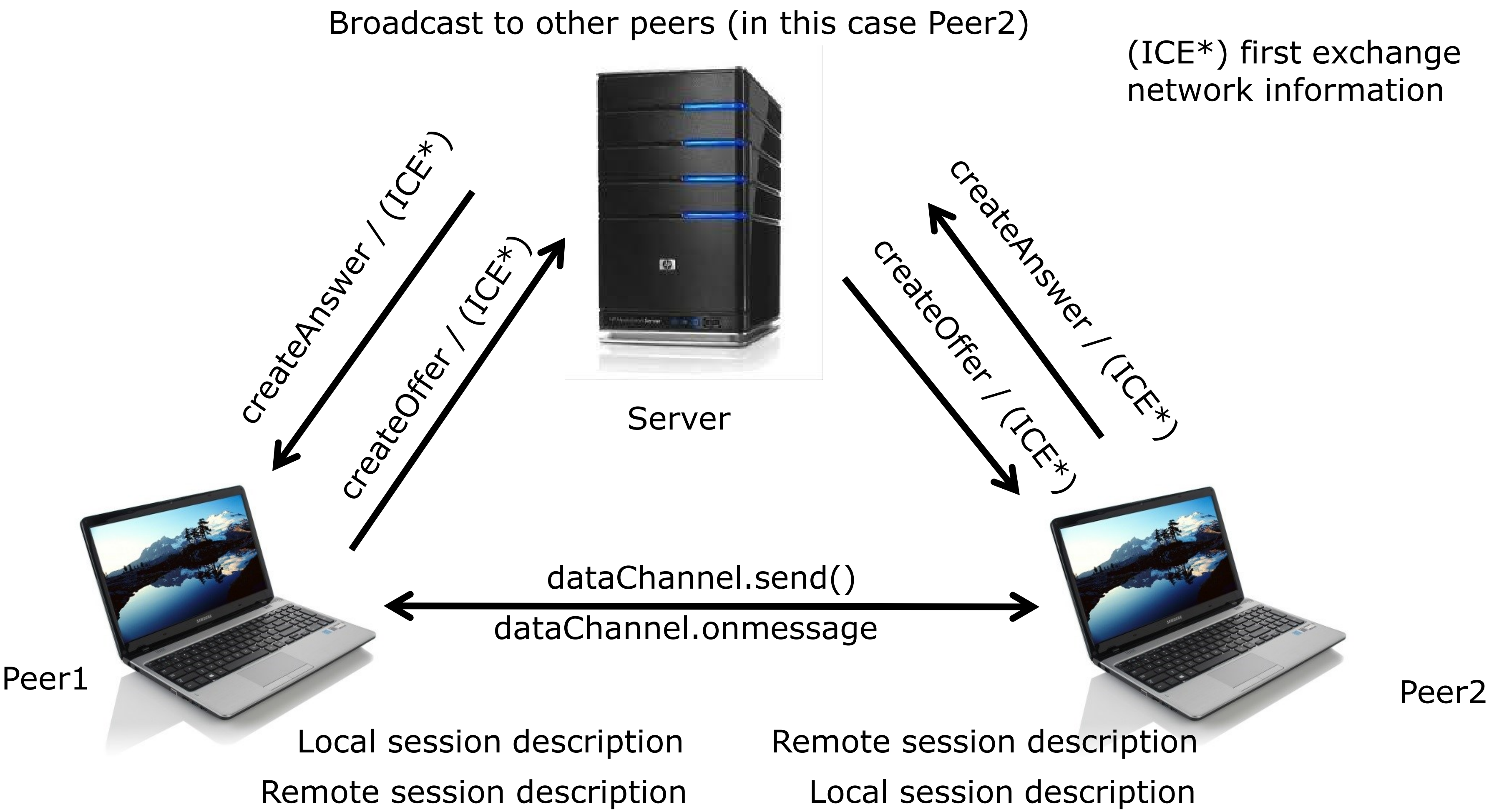


3.2. WebRTC Architecture - Trapezoid



3.3. WebRTC Architecture - Example

TECHNISCHE UNIVERSITÄT DARMSTADT



3.4. WebRTC - Outlook

- Examples:
  - WebRTC Plugin-free Screen Sharing [1], RTCPeerConnection Simple Demo [2]
- Strong focus on VoIP
  - Skype competitor?
  - Microsoft / IE and WebRTC?
  - SDP / signaling overhead if using with raw P2P data
- Fewer plugins (flash, java), fewer registrations
- Mandatory Codecs? [3]
  - Video codec in JavaScript [4]
- Web-based P2P frameworks
  - http://peerjs.com - make the API simpler
- New types of applications – Conferencing, gaming, P2P file sharing
  - PeerCDN [5], serve static content from browsers of other visitors

[1] https://www.webrtc-experiment.com/Pluginfree-Screen-Sharing/#CA134CZY-1FQD7VI
[2] https://www.webrtc-experiment.com/demos/client-side.html

[3] http://gigaom.com/2013/10/30/google-sticks-with-vp8-opposes-ciscos-push-for-h-264/
[4] https://brendaneich.com/2013/05/today-i-saw-the-future/

[5] http://peercdn.com/

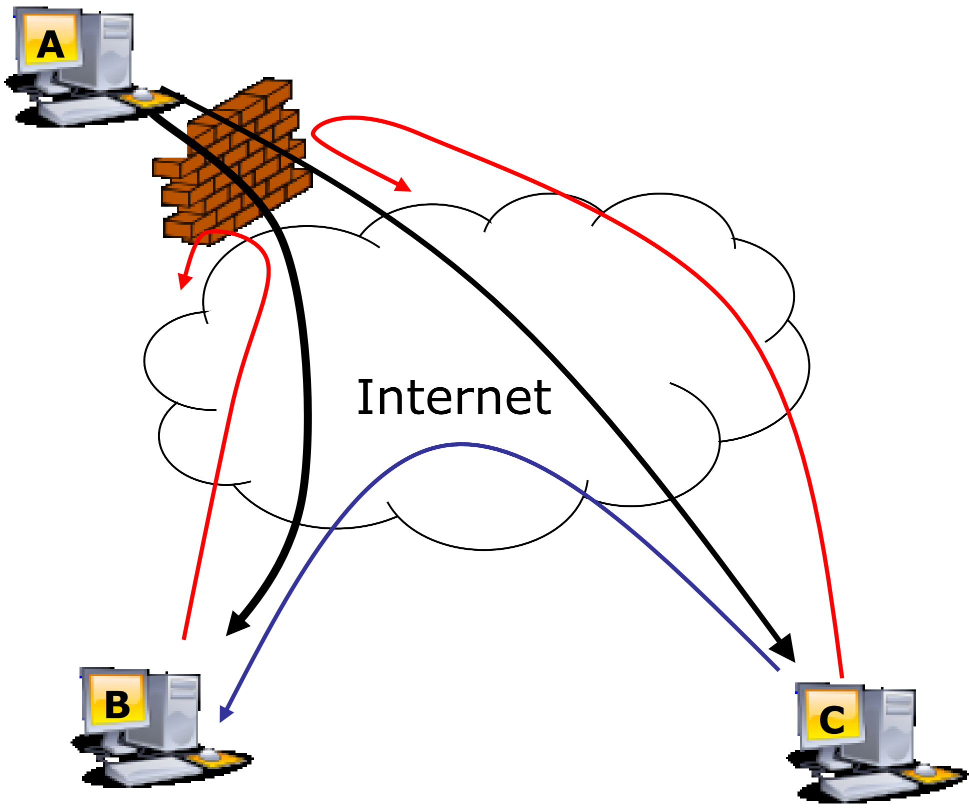
4. NAT Traversal

Background, Relaying, Hole Punching

4.0. NAT Traversal

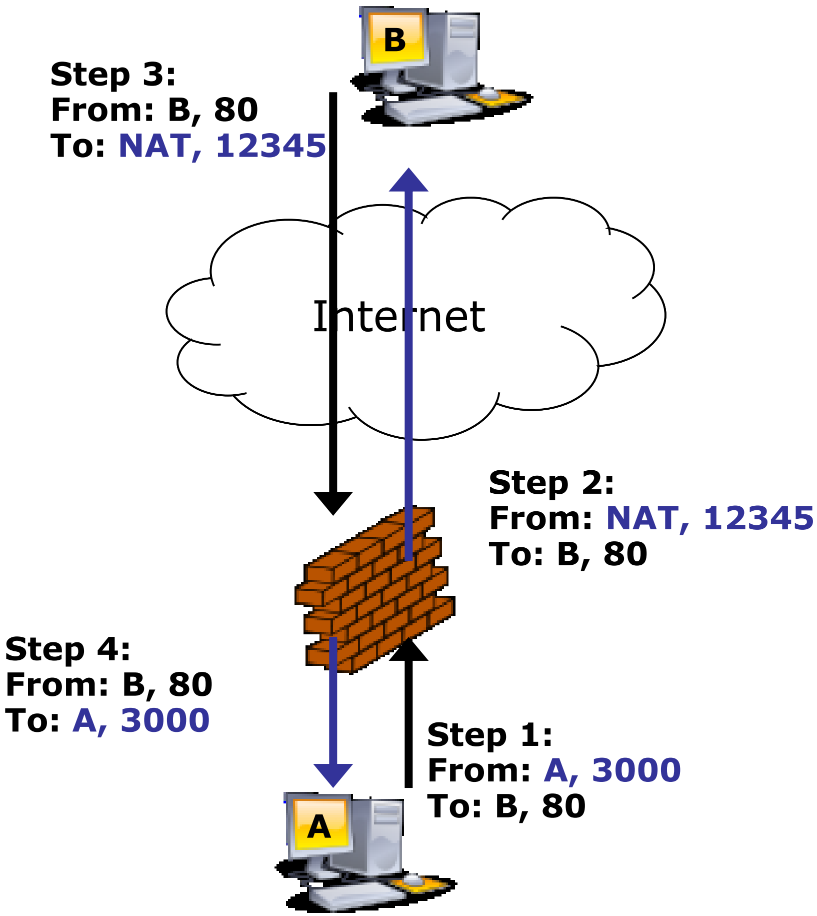
- Nodes must be able to contact each other
- Does not work for peers hidden behind NAT boxes and firewalls
  - Peers cannot be contacted arbitrary
- In real systems often 60-80% of peers are hidden
  - Less available resources
  - Smaller user base

→ P2P applications have to deal with NAT issues, but often this is done only later, as low priority, cf. Skype



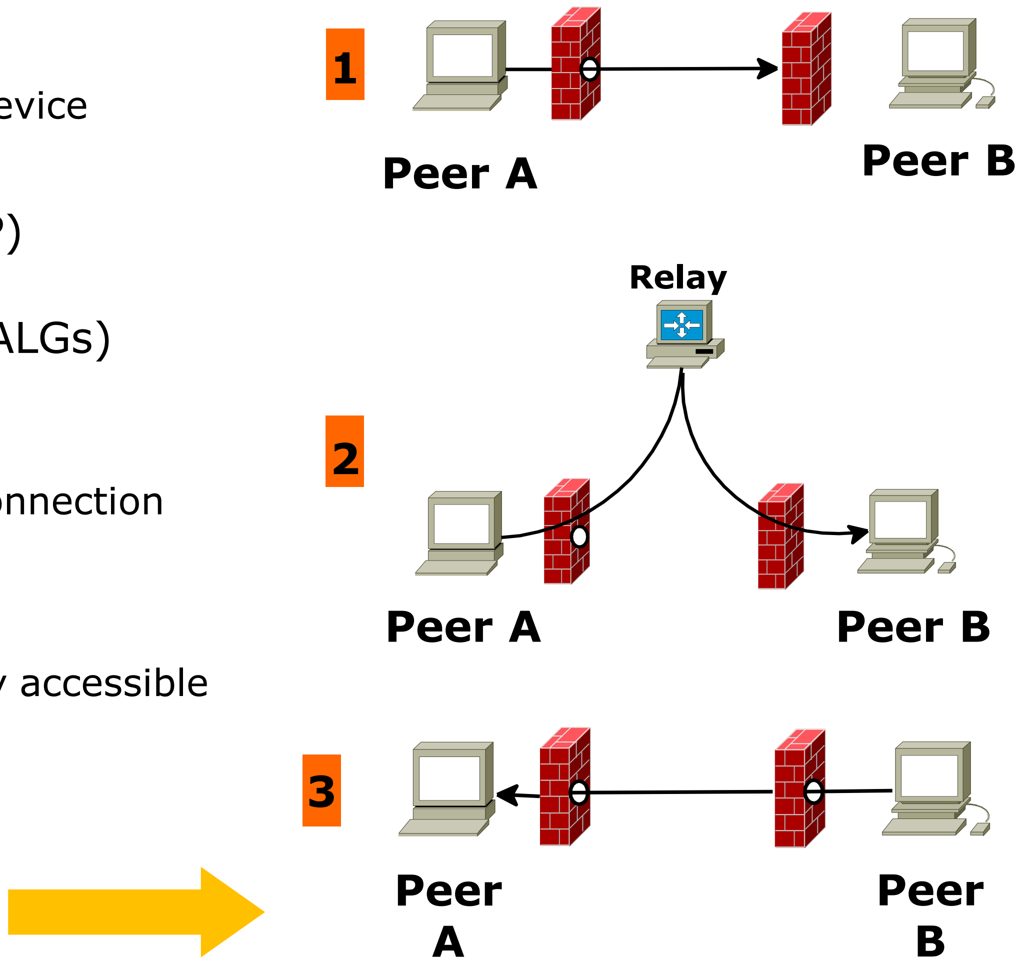
4.1. Background: NAT

- Reuse one IP address for many hosts
  - Solution for sparse IPv4 address space
  - Security features
- Functionality
  - Address rewriting in IP header
  - Works only if hidden host initiates the communication!
    - Applies for Client-Server
    - Not for P2P!

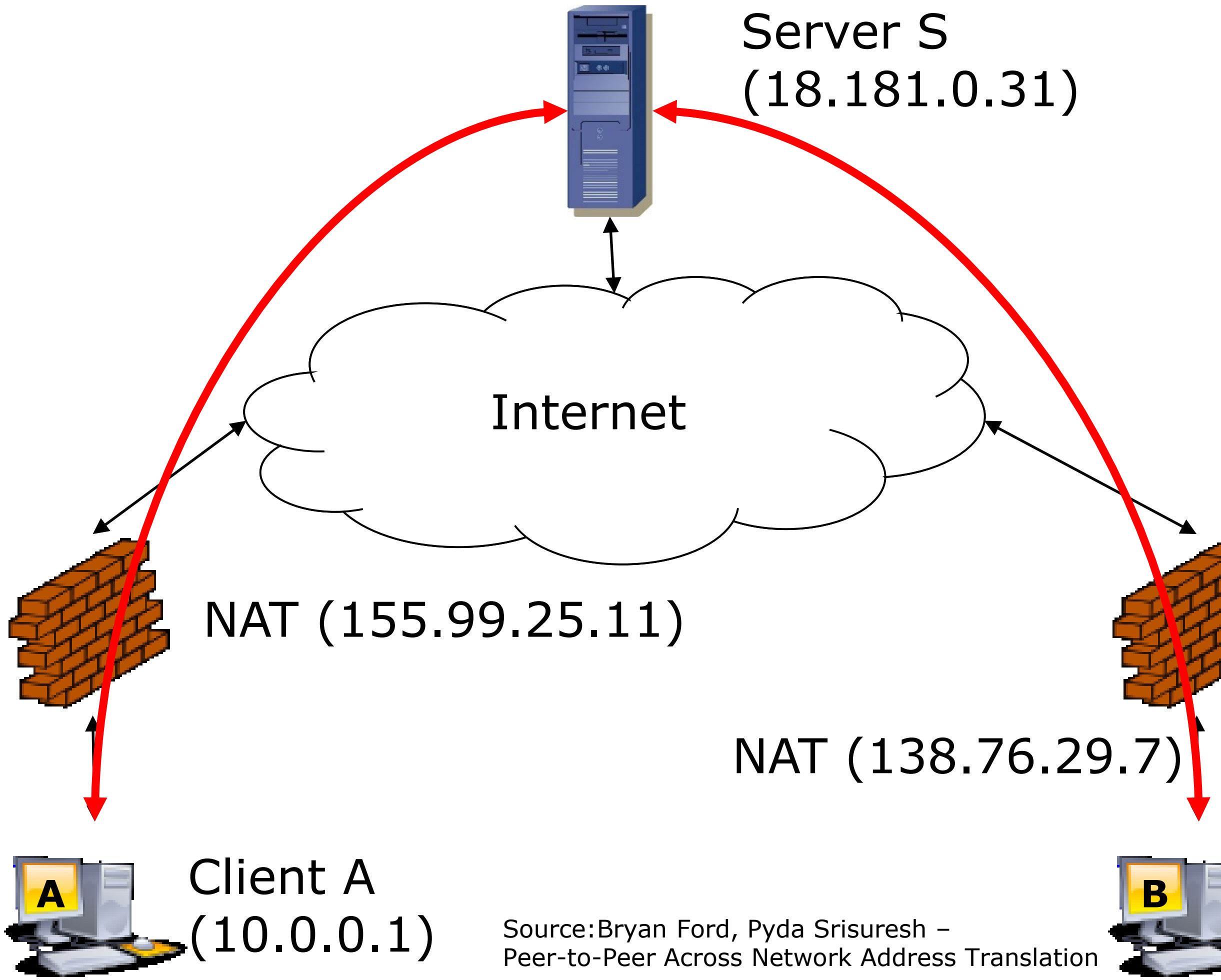


4.1. Background: NAT Traversal

- Port-Forwarding
  - Static mappings in the NAT device
- Universal Plug and Play (UPnP)
- Application Layer Gateways (ALGs)
- Connection Reversal
  - Let hidden host initiate the connection
- Relaying
  - Forward data through publicly accessible hosts
- Hole Punching
  - Exploit NAT functionality
  - Easy for UDP, hard for TCP …



4.2. Relaying



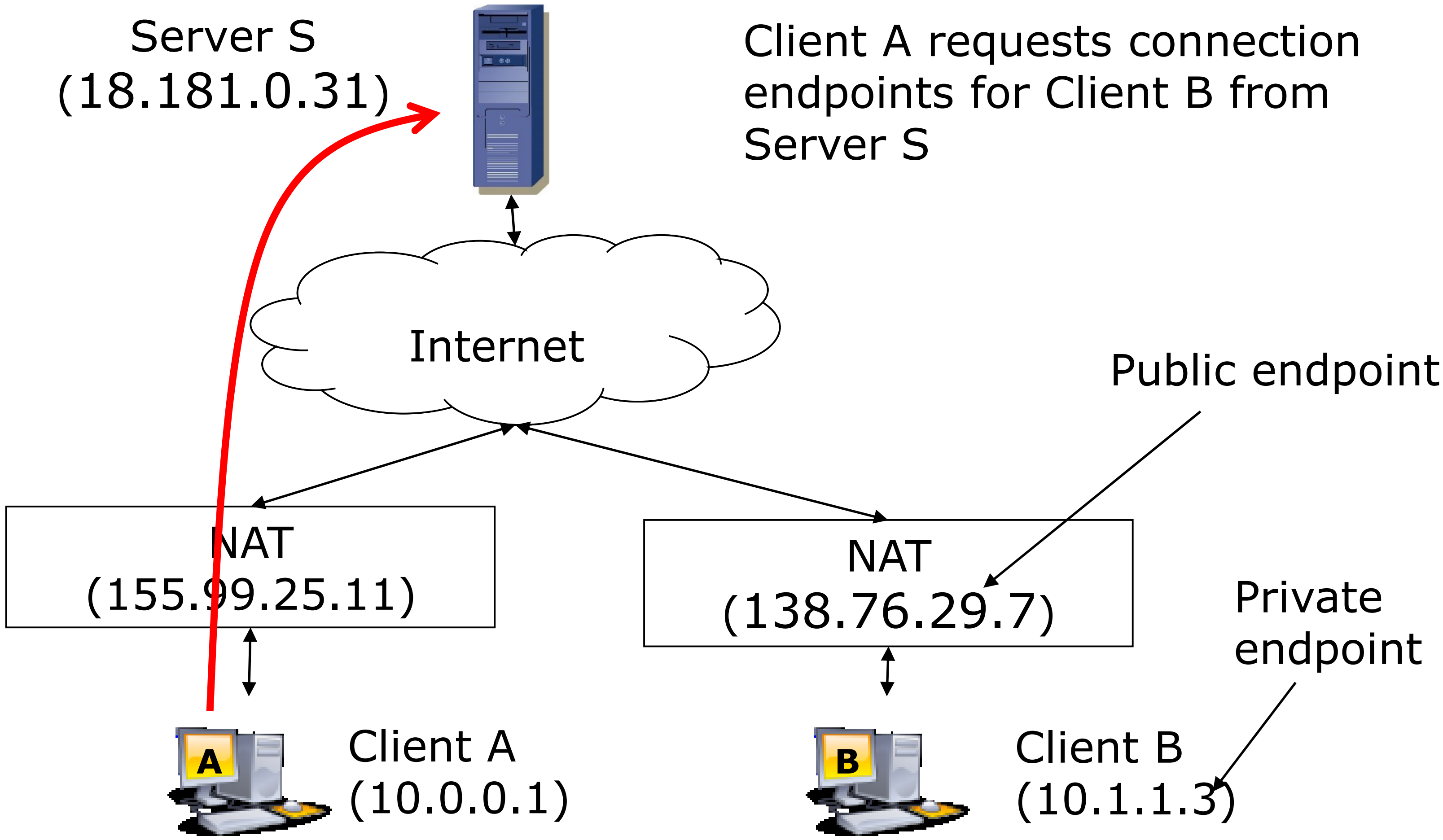
All data sent trough server

Reliable

Inefficient → processing power and bandwidth of server used

Source: Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

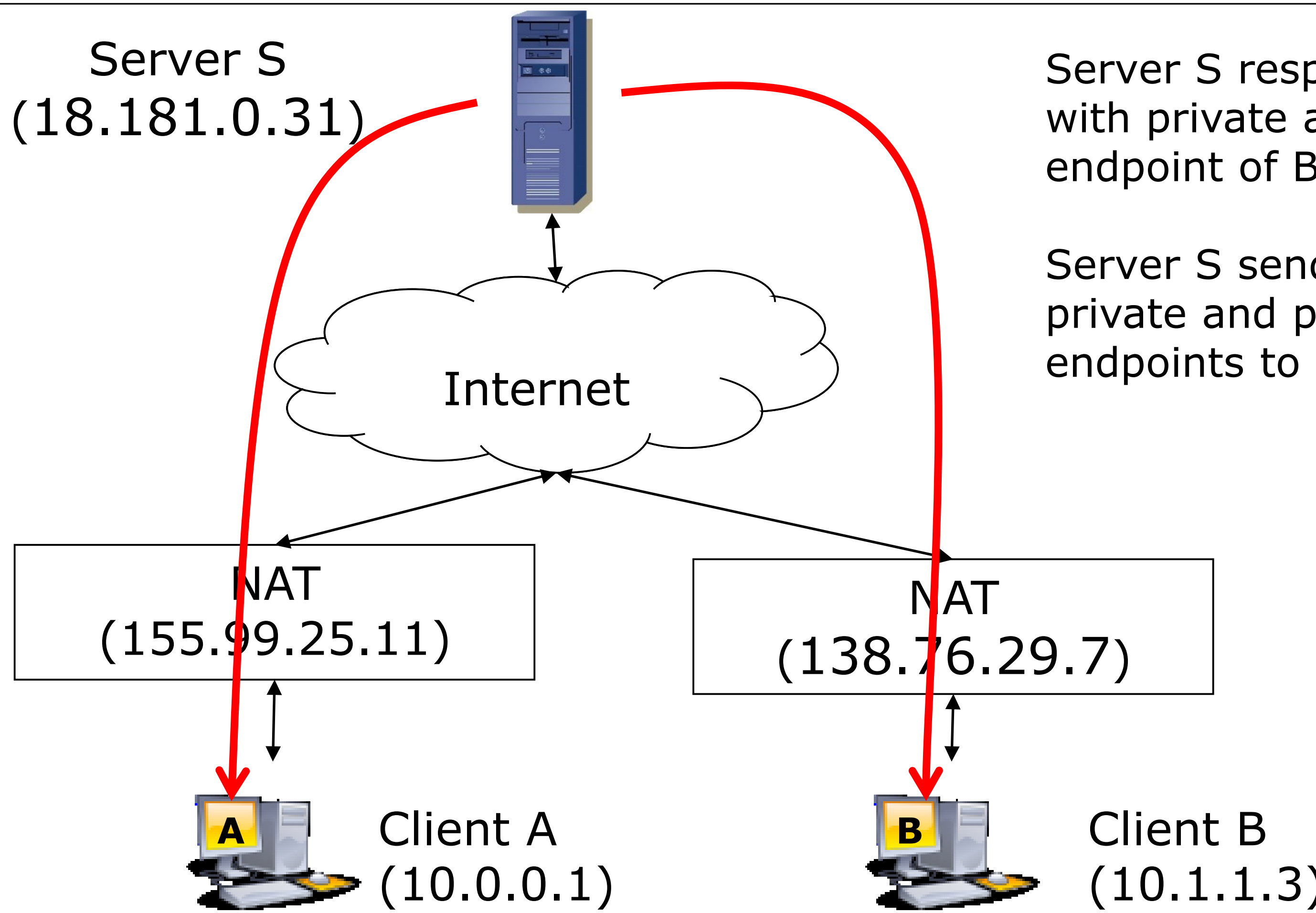
4.3. UDP Hole Punching – First Step



Client A requests connection endpoints for Client B from Server S

Source:Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

4.3. UDP Hole Punching – Second Step

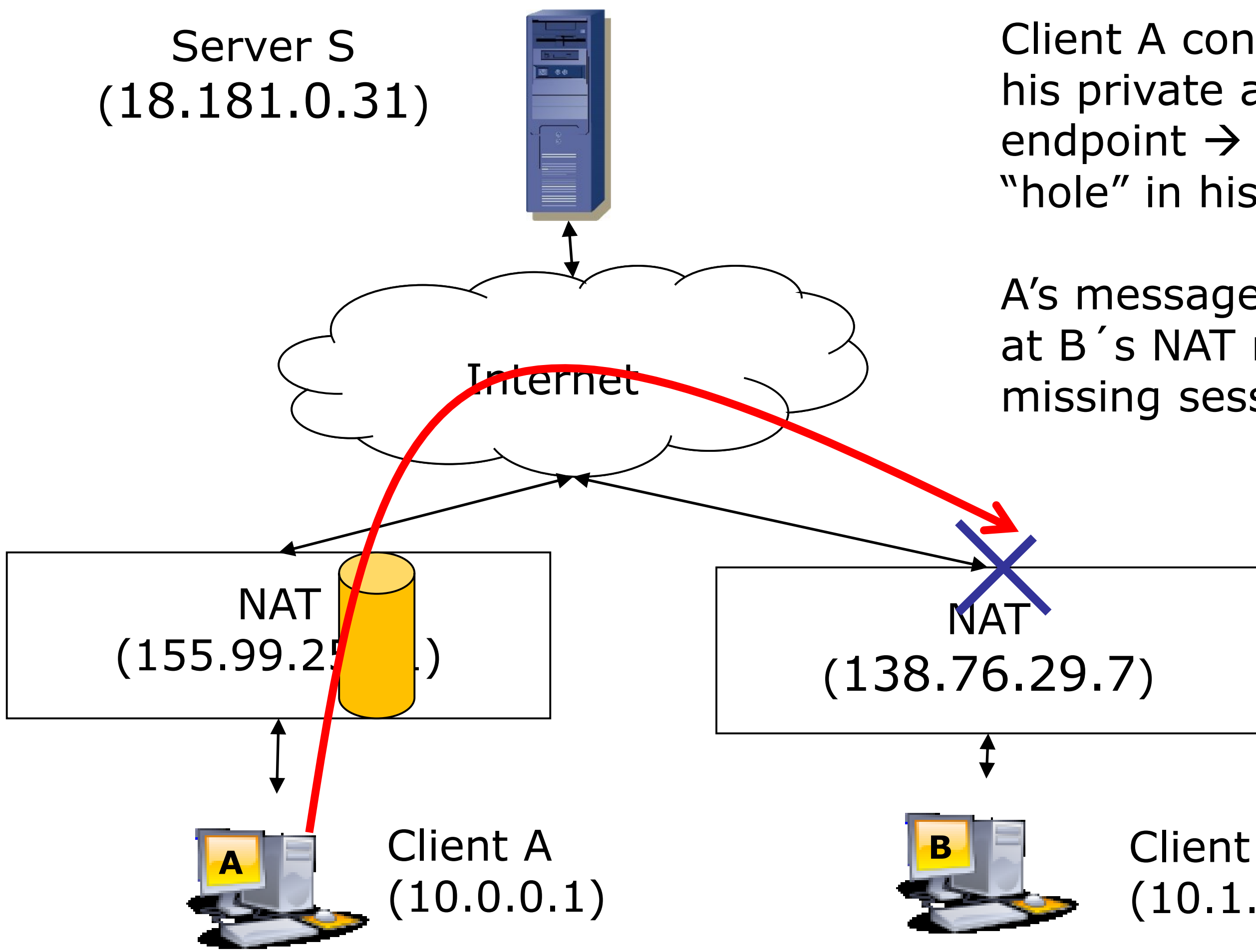


Server S responds to A with private and public endpoint of B and

Server S sends A's private and public endpoints to B

Source: Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

4.3. UDP Hole Punching – Third Step

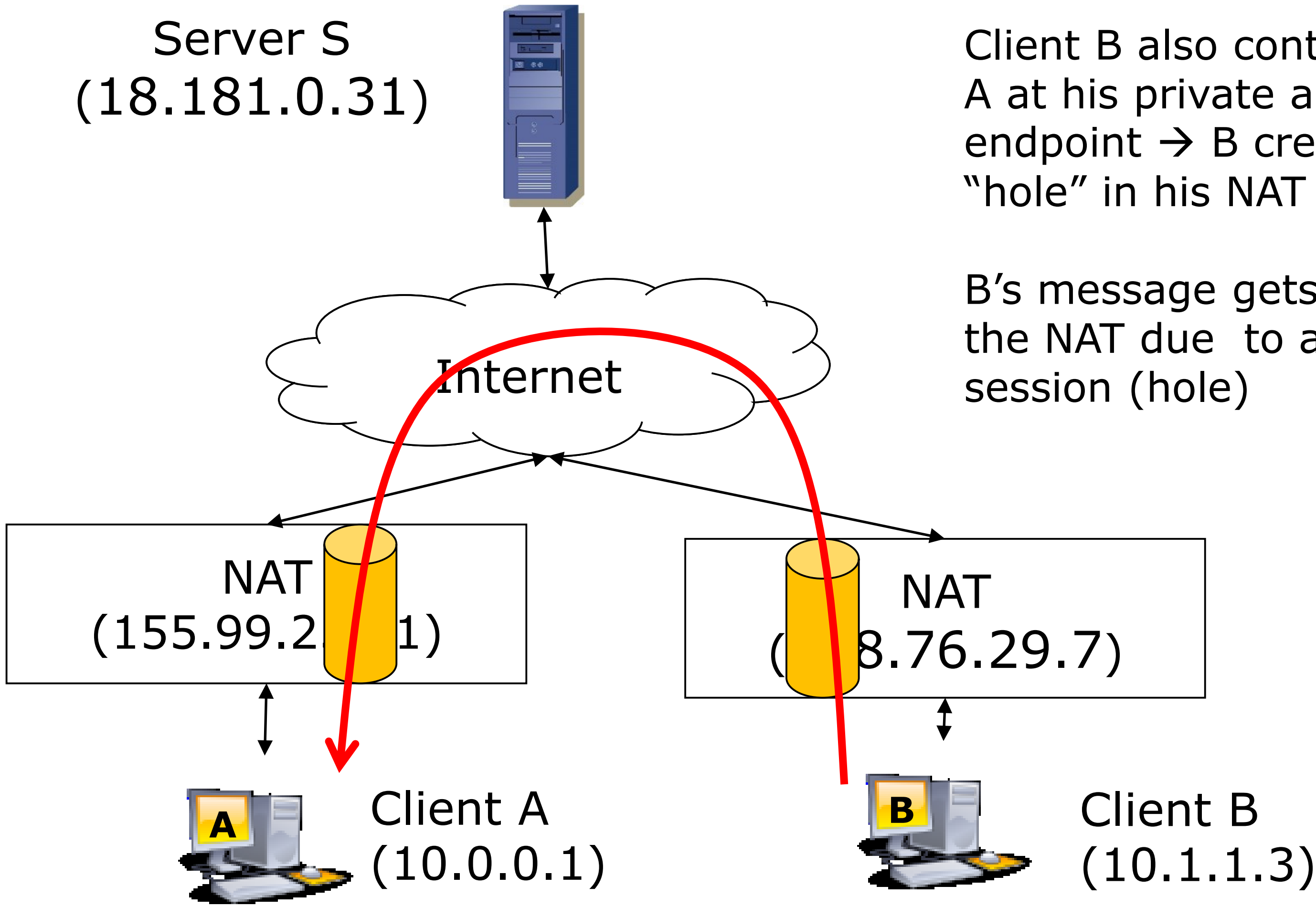


Client A contacts Client B at his private and public endpoint → A creates a "hole" in his NAT

A's message gets dropped at B´s NAT router due to missing session

Source:Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

4.3. UDP Hole Punching – Fourth Step

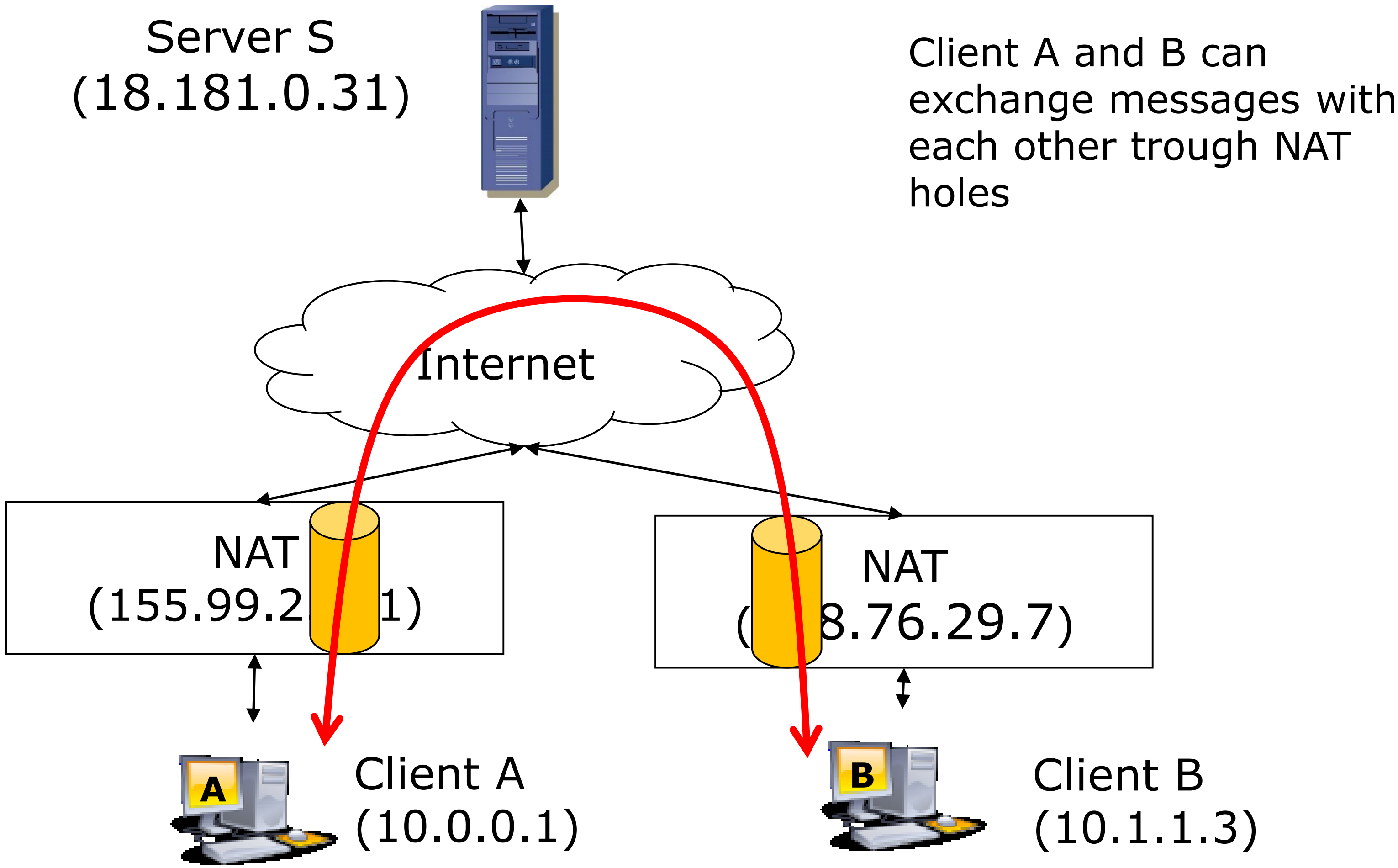


Client B also contacts Client A at his private and public endpoint → B creates a "hole" in his NAT

B's message gets trough the NAT due to active session (hole)

Source: Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

4.3. UDP Hole Punching – Fifth Step



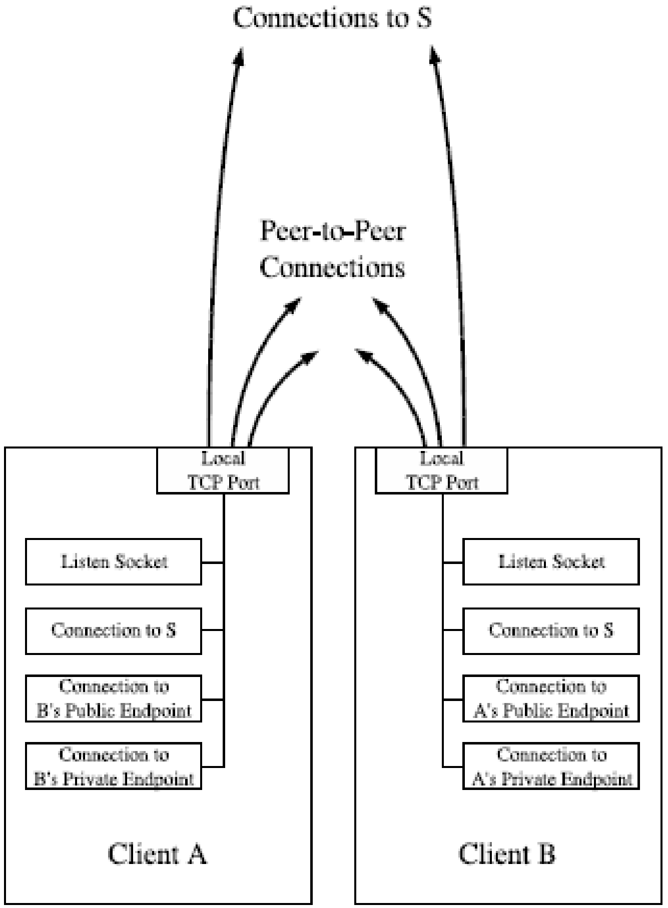
Client A and B can exchange messages with each other trough NAT holes

Source:Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation

4.3. Hole Punching

TECHNISCHE UNIVERSITÄT DARMSTADT

- UDP-NAT holes are closed after specific life time
  - Resending packets necessary
  - Re-opening hole on-demand
- Hole Punching also works for TCP
  - TCP socket option SO_REUSEADDR option necessary in order to bind multiple sockets to the same local port
- Success of hole punching cannot be guaranteed
  - Different NAT vendors
  - Symmetric NAT vs. cone NAT
  - Drop SYN packet vs. TCP RST packet



Source:Bryan Ford, Pyda Srisuresh – Peer-to-Peer Across Network Address Translation