Lecture 6

P2P with TomP2P

Advanced Topics

Universität Zürich[UZH]

*Original slides for this lecture provided by David Hausheer (TU Darmstadt, Germany), Thomas Bocek, Burkhard Stiller (University of Zürich, Department of Informatics, Communication Systems Group CSG, Switzerland, Jonas Wagner, Sebastian Golaszewski (Student UZH)

P2P in the news

- **4.4.2016 – Alpha testing of SmartCRS**
  - Student Project (Till Salinger)
  - WebRTC based classroom response system
- **26.3.2016 - Java Opus and H264 Wrapper**
  - Tested on OSX, Linux, Win?
  - Audio: Opus 1.1.2 (native, JNA), Video H264 (pure Java), Webcam grabber (native, OpenIMAJ , BridJ)
  - Run AudioVideoExample.java
- **4.4.2016 - OpenBazaar Team Releases First Version of Decentralized Marketplace**
  - Decentralized marketplace using Bitcoin
  - …Fully peer-to-peer marketplace where buyers and sellers engage in trade directly with each other…
    - Direct payment / moderated (escrow) payment

P2P in the news

- **30.3.2016 - The Trouble with Tor**
  - …Based on data across the CloudFlare network, 94% of requests that we see across the Tor network are per se malicious…
  - …A large percentage of the comment spam, vulnerability scanning, ad click fraud, content scraping, and login scanning comes via the Tor network…
- **31.3.2016 - The Trouble with CloudFlare**
  - ... CloudFlare has not described the nature of the IP reputation systems they use in any detail…
  - Akamai report:…Tor IP addresses clicking on ads and performing commercial activity was "virtually equal“ to that of non-Tor IP addresses)….

0. Lecture Overview

1. Advanced Topics in TomP2P

1. Mechanisms based on Hashing in DHTs
    1. And/Or Searches
    2. Similarity Searches
    3. Range Queries
2. Connectivity, Security, and Robustness
    1. NAT (UPNP/NAT-PMP/Hole punching)
    2. Security
    3. Replication
    4. Direct data connection / persistent connection
3. Consistency
    1. Paxos
    2. vDHT
4. Rsync

1. Mechanisms based on Hashing in DHTs

And / or searching

Similarity Search

Range queries

Mechanisms based on Hashing in DHTs

- **Search in DHT**
  - `DHT.get(h(„Communication Systems Group“))`
  - In order to find it: `DHT.put(h(„Communication Systems Group“), value)`
- **Keywords**
  - `DHT.get(h(„Communication“))`
  - Find it: `DHT.put(h(„Communication“), value)`, `DHT.put(h(„Systems“), value)`, `DHT.put(h(„Group“), value)`
  - value points to `h(„Communication Systems Group“)`
- **Keywords drawbacks**
  - Find good keywords → “the”, “a” are not good keywords
  - Exact matches only

Mechanisms based on Hashing in DHTs

- **Find "Communication" - OR Systems**
  - `DHT.get(h(„Communication"))` and `DHT.get(h(„Systems"))`, combine results
- **Find "Communication" - AND Systems**
  - 1. `DHT.get(h(„Communication"))` and `DHT.get(h(„Systems"))`, intersect results
    - Overhead – use Bloom Filters (sequential vs. parallel)
  - 2. `DHT.get(h(„Communication") xor h(„Systems"))`
    - In order to find it: `DHT.put(h(„Communication") xor h(„System"), value)`, `DHT.put(h(„Communication") xor h(„Group"), value)`, `DHT.put(h(„Group") xor h(„System"), value)`
    - Combination needs to be known in advance

Mechanisms based on Hashing in DHTs

- **Demo**
  - Keywords
  - Performance issue → consistent hashing (aggregation)

- **Performance issue: Aggregation not done in TomP2P**
  - Routing aggregation?

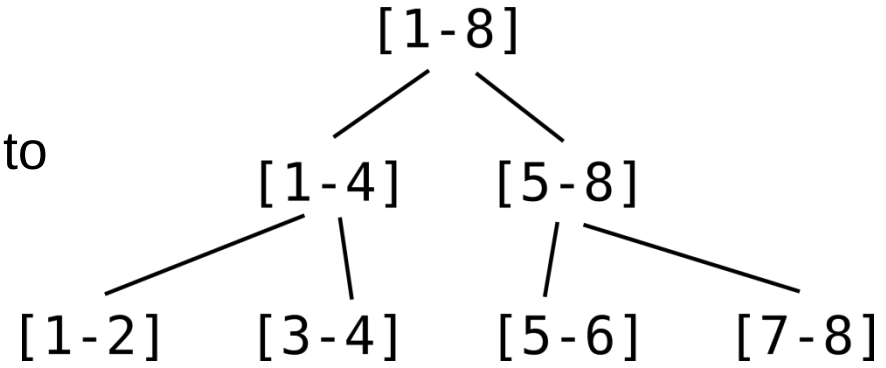
Mechanisms based on Hashing in DHTs

- **Range Queries**
  - Problem: random insert vs. sequence insert
  - Max. nr of items (n), nr of items per peer (m)
  - Sequence → [0..n] [n..2n] [2n..3n] […] → peer responsible for range, hash it, store it, done.
    - But random: worst case: 1 peers has 1 data item, range query for range [0..x] contacts x/n peers.
- **Over-DHT**
  - **PHT**: trie (prefix tree); **DST**: segment → tree on top of DHT
  - Main idea: hash of tree-node (resp. for range) → DHT
  - PHT: Peer stores n data items, if n reached, splits data (moves data across peers)
  - DST: stores data on each level (redundancy) up to a threshold
    - No data splitting

Mechanisms based on Hashing in DHTs

- **Example:**
  - Set n = 2, m=8
  - `1, "test"; 2, "hallo"; 3, "world"; 5, "sys"`

- **Tree: store value**
  - Translate `putDST(1, "test")` to
    - put(hash([1-8]),"test") → may be stored (only if threshold not reached)
    - put(hash([1-4]),"test") → may be stored
    - put(hash([1-2]),"test") → will be stored
    - Store `put(3, "world"),put(2, "hallo")` and `put(5, "sys")`



Mechanisms based on Hashing in DHTs

- **Query** `getDST(1..5)` **translates to**
  - `get(hash[5-6])` → returns “sys”
  - `get(hash[1-4])` → returns “test”, “world” and tells us that threshold has been reached
  - `get(hash[1-2])` → returns “hallo”, “test”
  - `get(hash[3-4])` → returns “world”
- **Range query as series of** `put()` **and** `get()`
- **Demo**
  - Storage modification

Mechanisms based on Hashing in DHTs

- **Similarity Search in DHT**
  - http://fastss.csg.uzh.ch

- **Project that brings similarity search to HT / DHT**
  - Problem: Search for “netwrk” fails for DHTs
- **Similarity: Edit distance / Levenshtein distance**
  - Min operations to transform one string into another, operations: insert, delete, replace
  - Calculated in matrix size O(m x n)

$$
\begin{aligned}
&d[i,0] = i,\ d[0,j] = j,\\
&d[i,j] = min\,(d[i-1,j]+1,\ d[i,j-1]+1,\\
&\quad d[i-1,j-1] + (if\ s1[i] = s2[j]\ then\ 0\ else\ 1))
\end{aligned}
$$

Mechanisms based on Hashing in DHTs

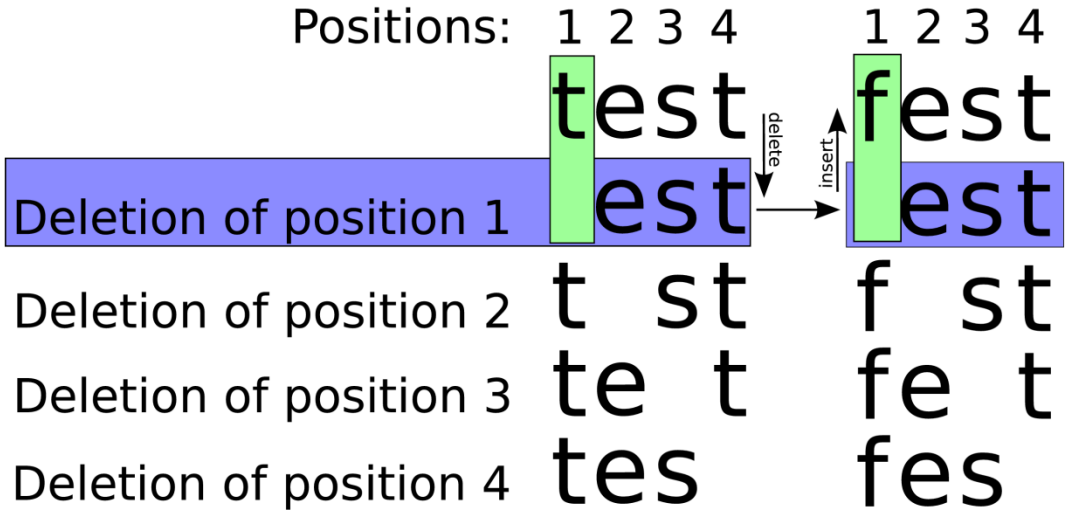
- **Example d(test,east) = 2 (remove a, insert t)**

		T	E	S	T
	0	1	2	3	4
E	1	1	1	2	3
A	2	2	2	2	3
S	3	3	3	2	3
T	4	3	4	3	2

- **Expensive operation if all words need testing**
- **Main idea: pre-calculate errors**
  - All possible errors? Neighbors for test with ed 2: test, testa, testaa, testab, ... , tea, teb, tec, ..., teaa, teab, ... → 23883 more of those!

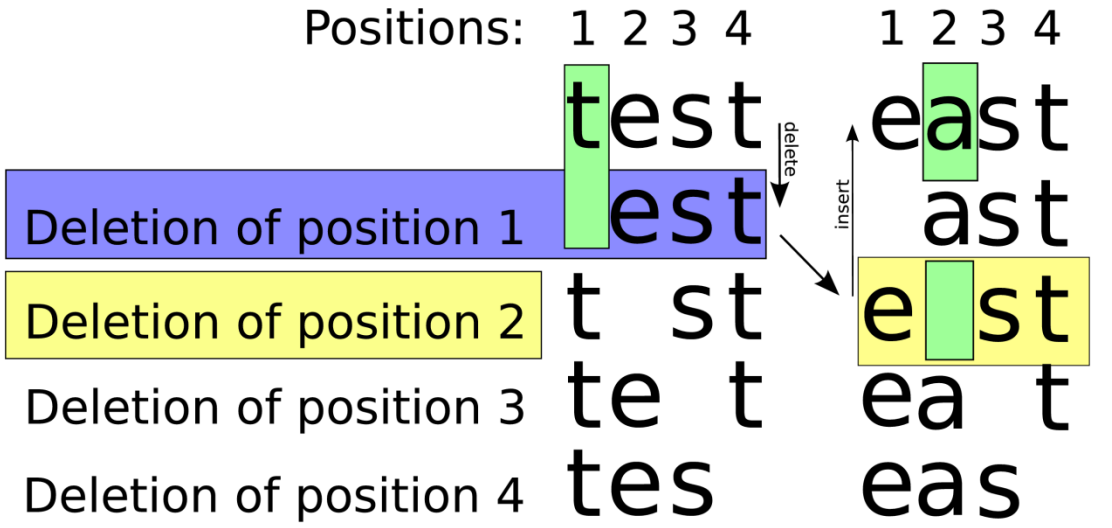
Mechanisms based on Hashing in DHTs

- **FastSS pre-calculates with deletions only**
  - Neighbors for *test* with ed 2: test, est, st, et, es, tst, tt, ts, tet, te, tes
  - Pre-calculation on query **and** index
  - 11 neighbors → 11 more queries, indexed enlarged by 11 entries
- **Example d(test,fest)=1** (query) (index)



Mechanisms based on Hashing in DHTs

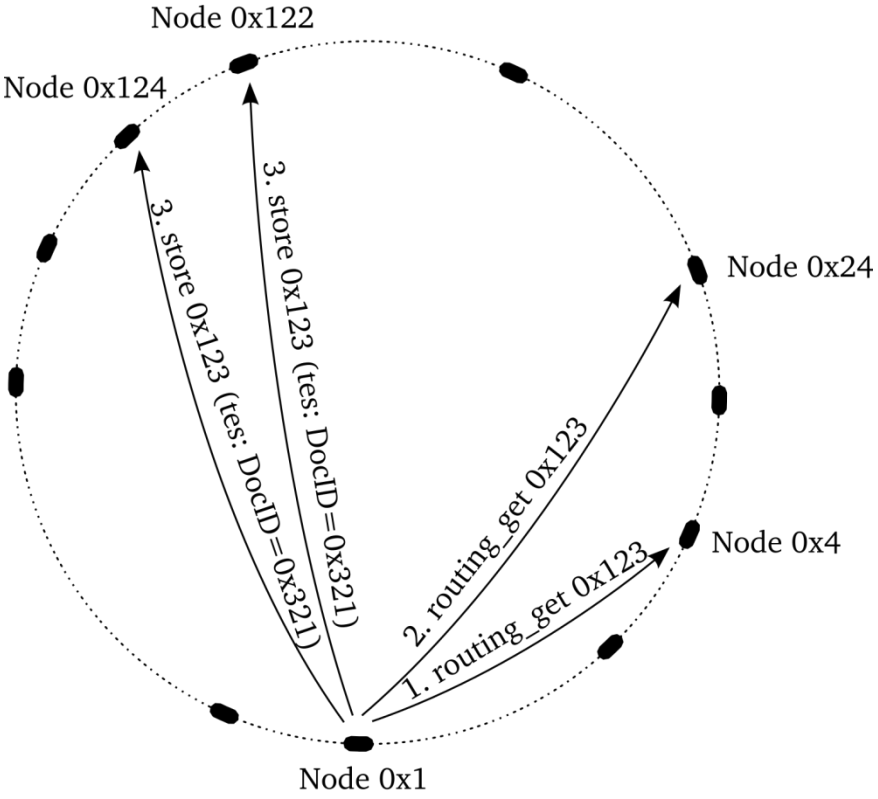
- Example d(test,east)=2 (query) (index)



- P2PFastSS implemented on top of TomP2P (early version) – tests with indexing Wikipedia abstracts

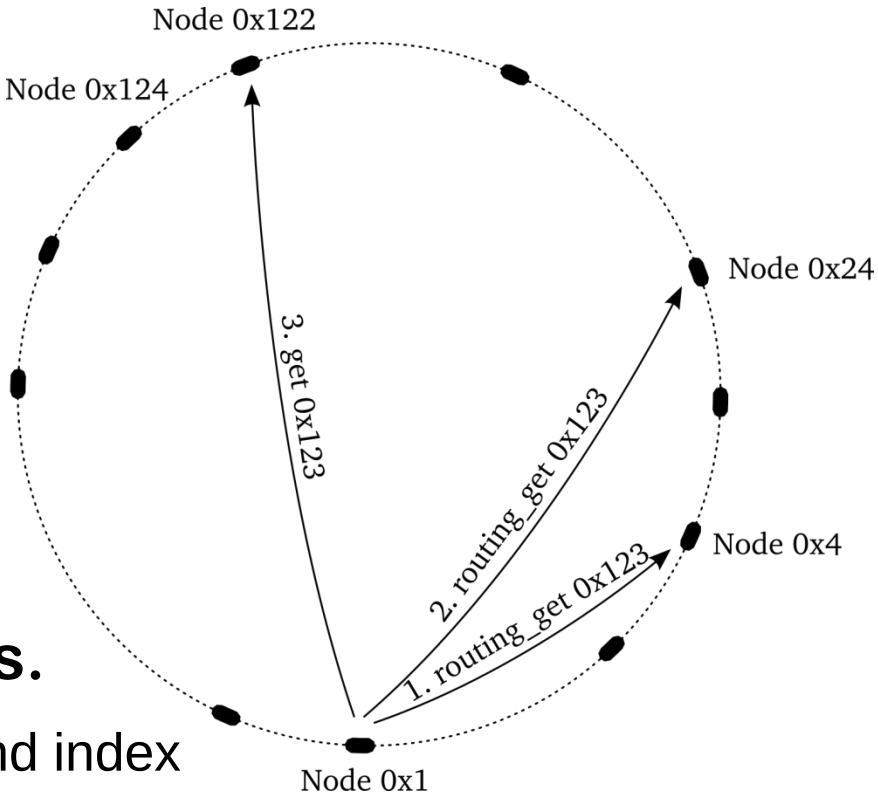
Mechanisms based on Hashing in DHTs

- **Index documents using**
  `put(hash(document), document)`
  - Document (0x321) contains word test
- **Index all neighbors (test, tes, tst, tet, est) using**
  `put(hash(neighbor), point to document)`
  - `hash("tes") = 0x123`



Mechanisms based on Hashing in DHTs

- **User searches for “tesx”**
- **Neighbors are generated (tesx, esx, tsx, tex, tes)**
  - get(hash(neighbor)) → 0x123
  - Find pointer to document (0x321)
  - document = get(0x321)
- **Tests with edit distance 1, partially 2, ignoring delete pos.**
  - Overhead (*n choose k*) for query and index
- **Similarity search as series of `put()` and `get()`**
- **Demo**



Mechanisms based on Hashing in DHTs

- **Direct data and persistent connections (data download)**
  - All connections in TomP2P are RPC and very short-lived
    - Open connection – request – reply – close connection
  - Direct `sendDirect(PeerAddress, …)` / with routing `send(key,…)`
  - Always use `setObjectDataReply()` or `setRawDataReply()`
    - Object serializes object to byte[] (easy)
    - Raw exposes (Netty) buffer to the user for your own protocol (more work)
  - Persistent connections set by the user
    - Only for direct send `sendDirect(PeerAddress, …)`

- **Demo with persistent connections (net.tomp2p.examples.ExamplePersistentConnection)**

2. Connectivity, Security, and Robustness

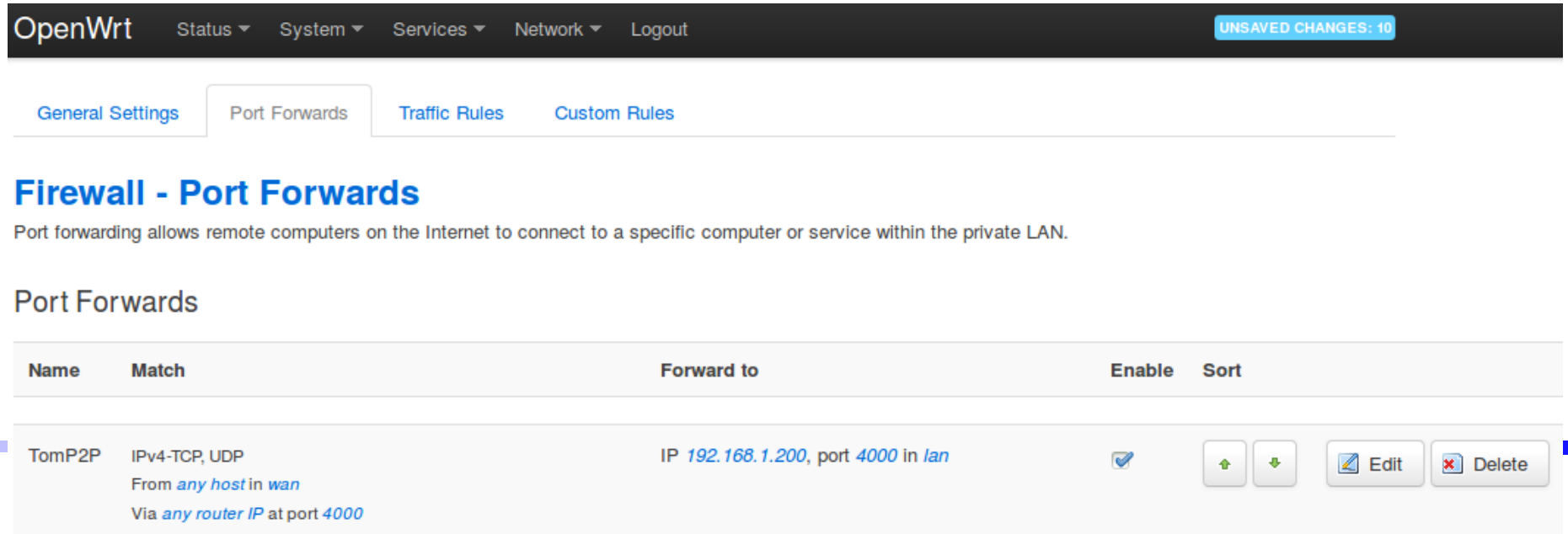
NAT (UPNP/NAT-PMP/Hole punching)

Security

Replication

Connectivity, Security, and Robustness

- **NAT**
  - Network Address Translation – breaks end-to-end
  - "If nothing else, [NAT] can serve to provide temporarily relief while other, more complex and far-reaching solutions are worked out" (RFC 1631 - The IP Network Address Translator (NAT))
- **Easy solution:**
  - Manual port forwarding: e.g., setup on your router



Connectivity, Security, and Robustness

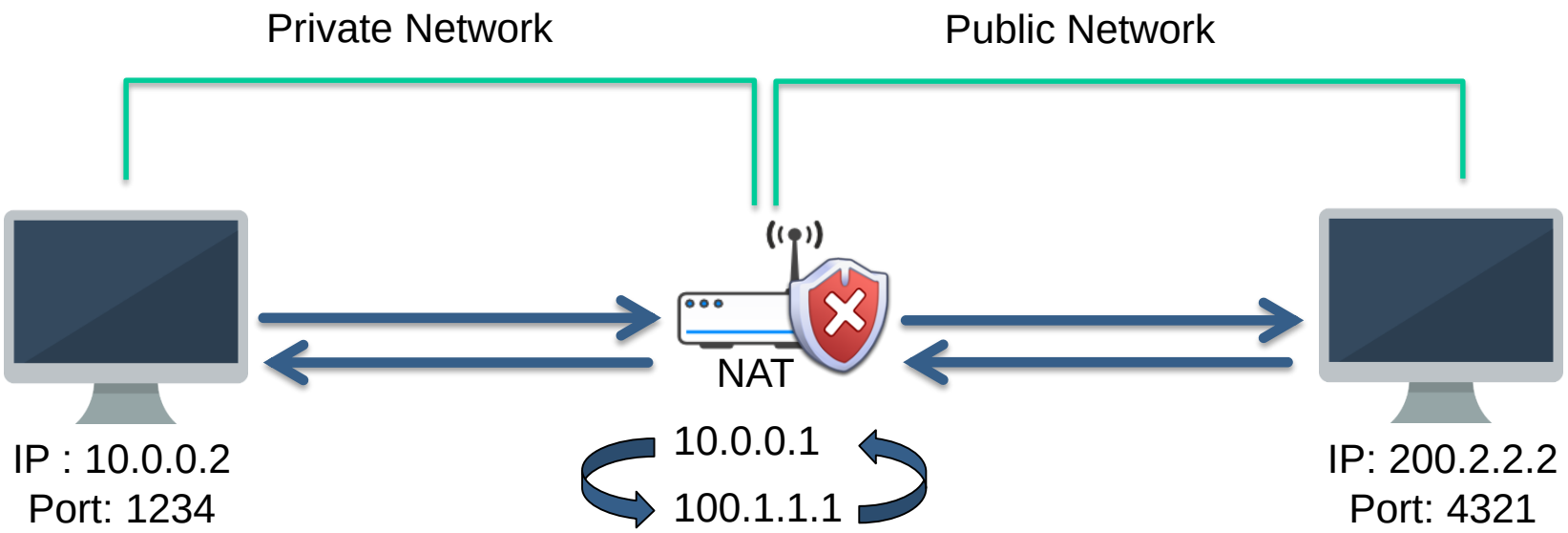
- **Easy solution: UPNP / NAT-PMP**
  - Both configure port forwarding, but UPNP is more: discover devices - uses broadcasting to find router (Simple Service Discovery Protocol)
  - UPNP: configure devices - uses HTTP and XML to configure port forwarding (Internet Gateway Device Protocol)
  - NAT-PMP: protocol made for configuring port-forwarding, but no discover (how to find router?)

Connectivity, Security, and Robustness

- **NAT example in TomP2P**
  - TomP2P supports NAT-PMP and UPNP, holepunching, and relaying
  - Before bootstrap: `peer.discover(PeerAddress);`
  - How it works: (1) send request how others peers sees our IP
    - If other peers sees the same IP as we see, we are fine
    - If not, we are most likely behind a NAT
  - (2) do UPNP, if it fails, do NAT-PMP, if it fails, mark it as firewalled, setup relays / rendez-vous
  - (3) If it works test connection, send request to other peer to contact us using the port we just set up.
  - (4) If we get contacted by this peer within 5 sec, port-forwarding works.
  - Manual setup possible using `Bindings.java`

Connectivity, Security, and Robustness

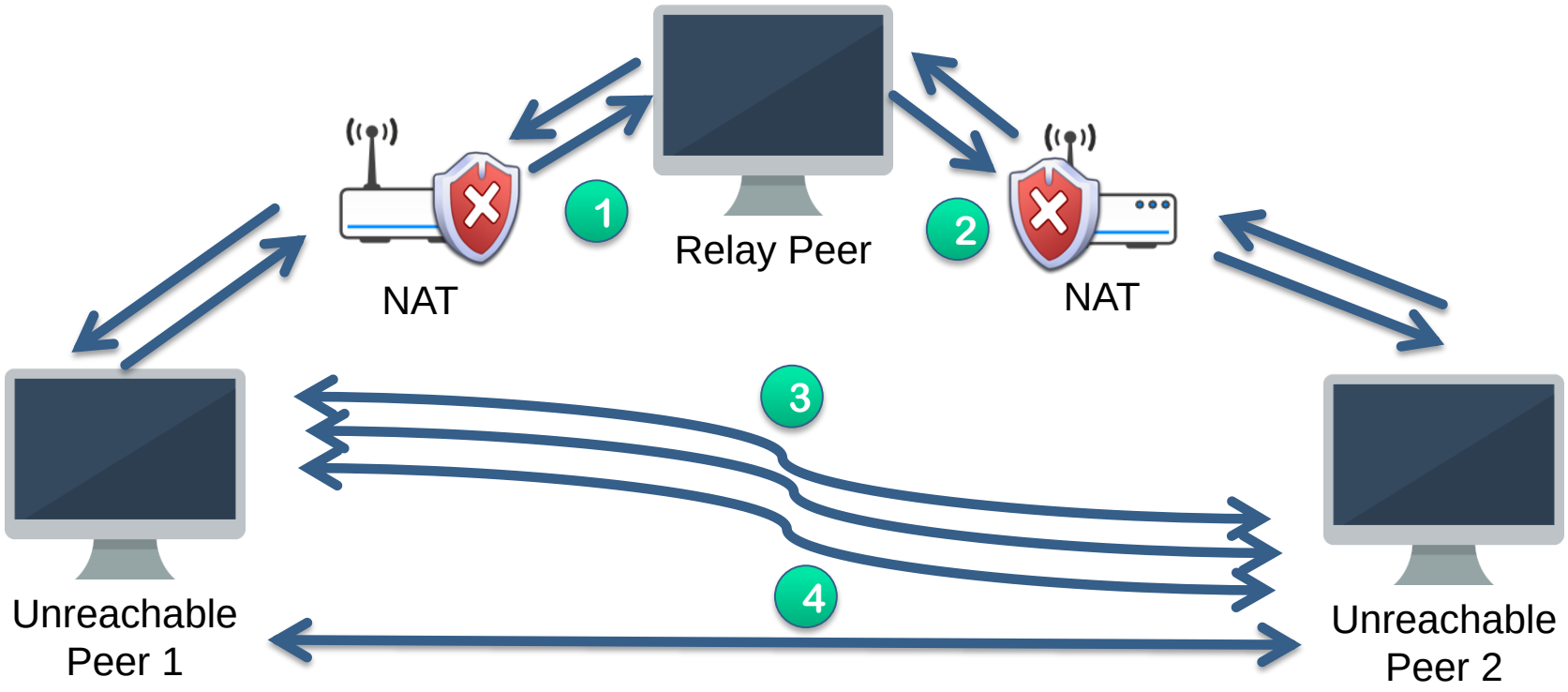
- **Difficult solution: hole punching**
  - rendezvous / relay peer which does “hole punching”, in worst case relay traffic.
- **NAT: translation table for private / public network**



NAT Table Entry: (10.0.0.2:1234, 200.2.2.2:4321; 200.2.2.2:4321, 100.1.1.1:3333)

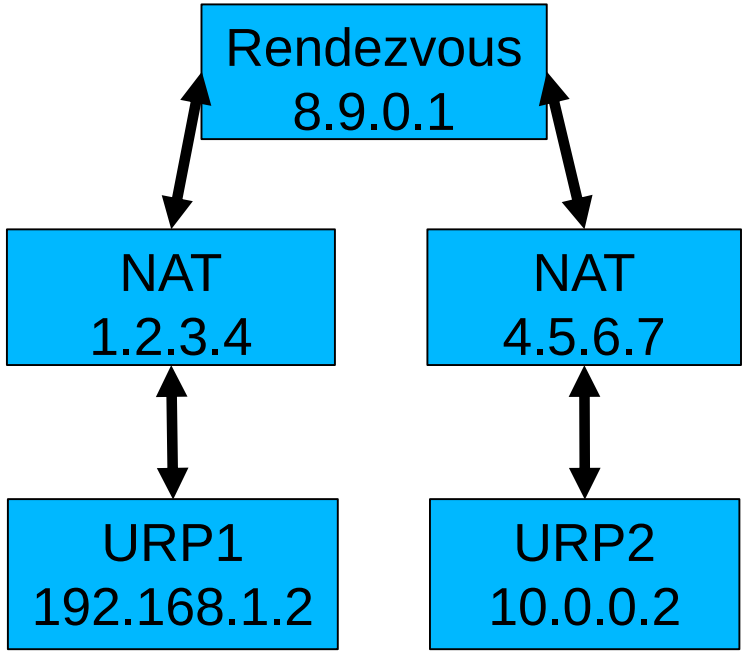
Connectivity, Security, and Robustness: Hole punching

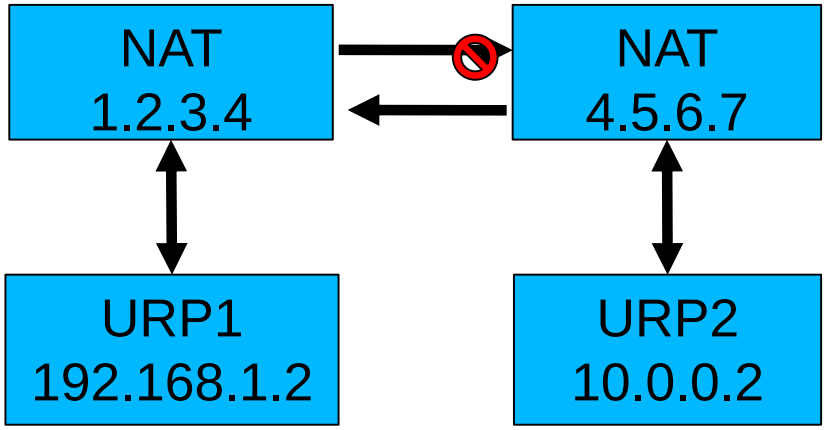
- 1) Peer1 initiates a new connection trial to peer2 via relay and signals its source ports and IP (relay/rendez-vous peer has connection to URP2)
- 2) Peer2 answers back with its source ports and IP
- 3) Both of the peers punch holes into their firewall/NAT
- 4) Established a connection



Connectivity, Security, and Robustness

- **Hole punching**
  - Unreachable peer 1 request to NAT 4.5.6.7, will fail – no mapping, however, unreachable peer 1 creates mapping with that request
  - Unreachable peer 2 sends request to unreachable peer 1 (1.2.3.4:Y) success!



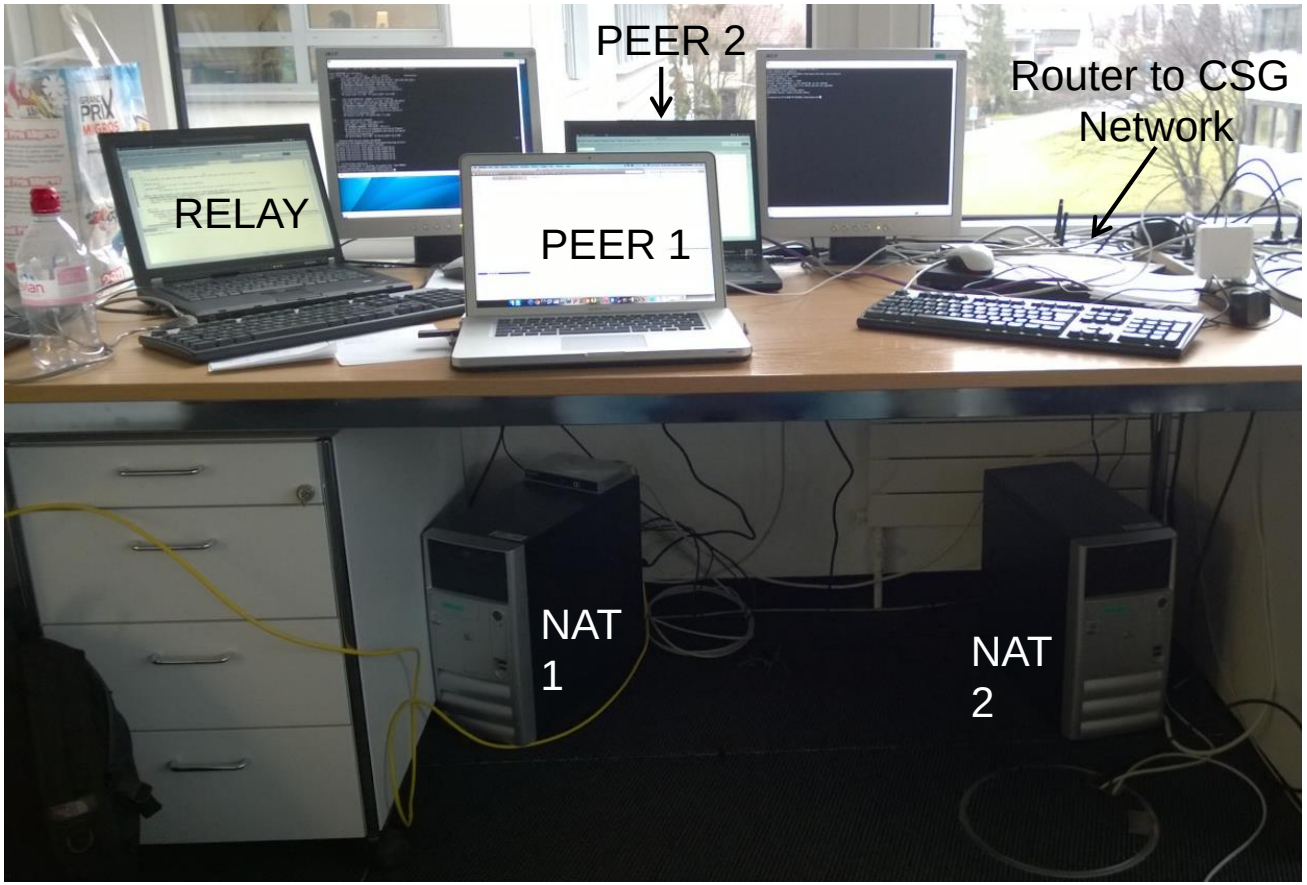


Mapping for NAT 1.2.3.4 (Unreachable peer 1)			
192.168.1.2:4000	…	1.2.3.4:Y	4.5.6.7:Z

Mapping for NAT 4.5.6.7 (Unreachable peer 2)			
10.0.0.2:5000	…	4.5.6.7:Z	1.2.3.4:Y

Connectivity, Security, and Robustness

- **Hole Punching (BA Jonas Wagner)**



- **Currently: network namespaces (since 2.6.24)**

Connectivity, Security, and Robustness

- **If everything fails, use relays**
  - Well connected / reachable peer
  - Forwards the data to and from the unreachable peer
- **Relay candidates are close neighbors**
  - Will be added to your PeerAddress
  - Other peers will see the relay from the peer address, contact them
  - Up to 5 relay peers
- **Relays keep TCP connection open**
  - UDP messages (ping / neighbor) handled by relays itself
  - Unreachable peer must update information for relays to be able to handle request

Connectivity, Security, and Robustness

- **Security in TomP2P (best-effort security)**
  - Signature-based, no data encryption
  - Messages are signed using SHA1 with DSA
  - Sybil attacks!
    - This attack creates large number of identities, may collude
- **How to prevent Data from being overwritten**
  - Domain and entry protection, requires cooperation
  - `StorageLayer.protectionDomainMode(…)`

For domains and entries		
protectionEnabled	ALL	NONE
protectionMode	NO_MASTER	MASTER_PUBLIC_KEY

Connectivity, Security, and Robustness

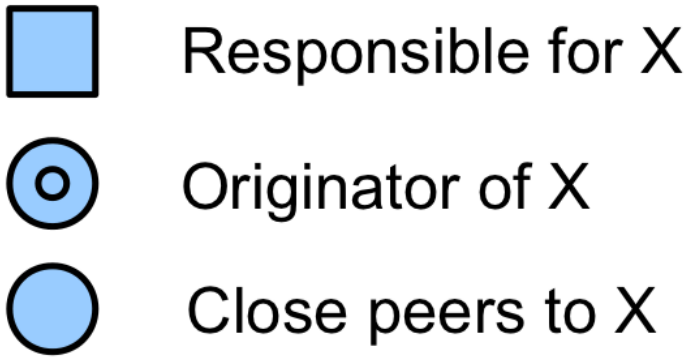
- **Domain protection**
  - Set publick key `new PeerMaker(PublicKey)`
    - Enable=ALL, Mode=NO_MASTER → every peer can protect domains, first come first served
    - Enable=NONE, Mode=NO_MASTER → no peer can protect domains
    - Enable=ALL, Mode=MASTER_PUBLIC_KEY → every peer can protect domains, the owner can claim domain
    - Enable=NONE, Mode=MASTER_PUBLIC_KEY → no peer can protect domains except the owner
  - Owner of domain 0x1234 is peer where 0x1234 == hash(public_key)
  - Same concept for entries
  - Tracker should have no domain protection and content protection set to Enable=NONE, Mode=MASTER_PUBLIC_KEY → WiP
- **Demo**

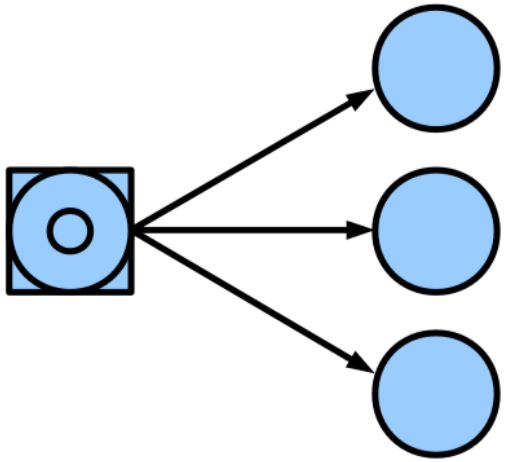
Connectivity, Security, and Robustness

- **Demo 1 (net.tomp2p.examples.ExampleDomainProtection):**
  - 3 peers, all with public keys
  - Setup for domains: Enable=ALL, Mode=MASTER_PUBLIC_KEY
  - (1) peer1 stores data in domain2 → success
  - (2) peer3 wants to store data in domain2 → fail
  - (3) peer2 wants to store data in domain2 → success
- **Demo 2 (net.tomp2p.examples.ExampleDomainProtection):**
  - 3 peers, all with public keys
  - Setup for domains: Enable=NONE, Mode=MASTER_PUBLIC_KEY
  - (1) peer1 stores data in domain2 → success
  - (2) peer3 wants to store data in domain2 → success
  - (3) peer2 wants to store data in domain2 → success
  - (4) peer3 wants to store data in domain2 → fail
- **TomP2P + Bitcoin Blockchain (former master project, not yet merged)**

Connectivity, Security, and Robustness

- **Replication**
  - Enough replicas
  - Direct replication
    - Originator peer is responsible
    - Periodically refresh replicas
    - Example: tracker that announces its data

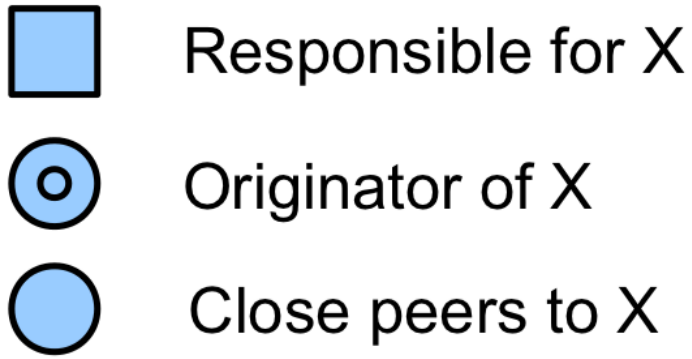


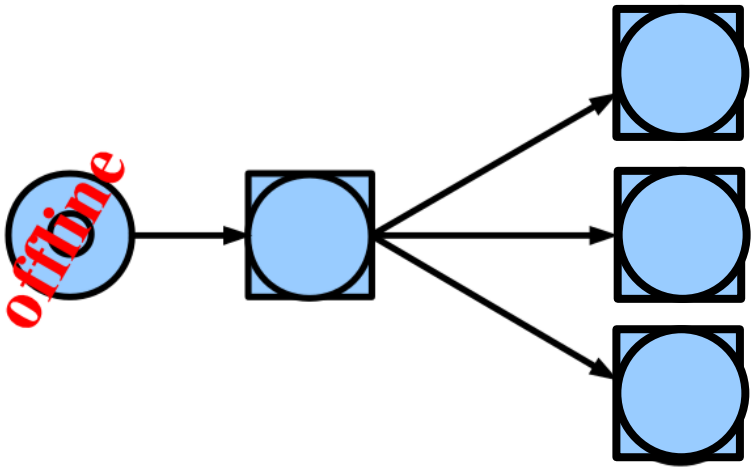


- **Problem**
  - Originator offline → replicas disappear. Content has TTL, e.g.
    `data.ttlSeconds(15)`

Connectivity, Security, and Robustness

- **Indirect Replication**
  - The closest peer is responsible, originator may go offline (0Root)
    - Periodically checks if enough replicas exist
    - Detects if responsibility changes





nRoot (default is 0Root)

- **Problem**
  - Requires cooperation between responsible peer and originator
  - Multiple peers may think they are responsible for different versions → eventually solved
  - **Replication Demo** (net.tomp2p .examples.ExampleDirectReplication)

3. Consistency

Paxos

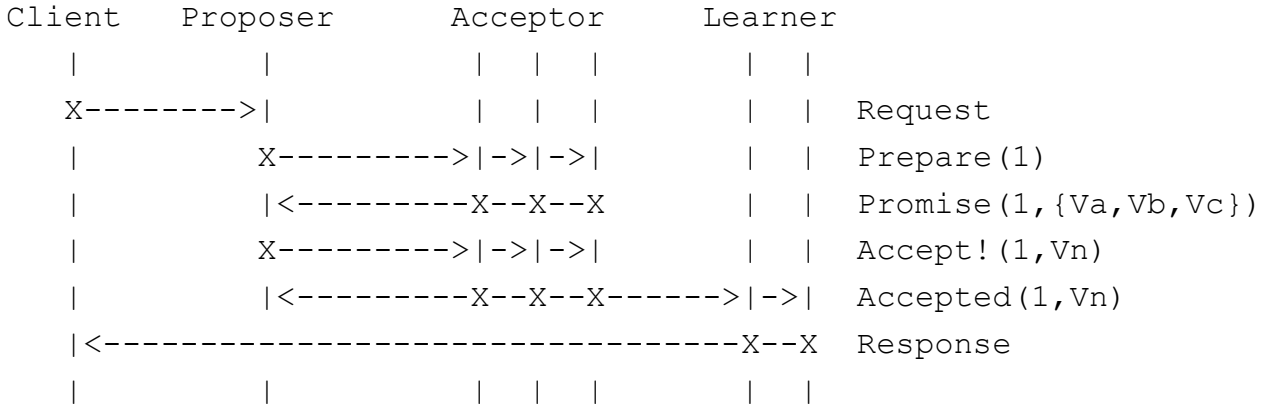
vDHT

Consistency

- **DHTs have weak consistency**
  - Peer A put X.1, Peer B gets X.1 modifies it puts B.2
  - Same time: Peer C gets X.1 modifies it puts C.2
    - Which one is stored B.2 of B or C.2 of C?
- **Consistency generic issue in distributed systems**
  - Coordinator required:
    - easy solution: centralized
    - Interesting solution: decentralized, in case failed peer, pick another peer
- **Coordinator needs to be defined**
  - Election, example Paxos

Consistency

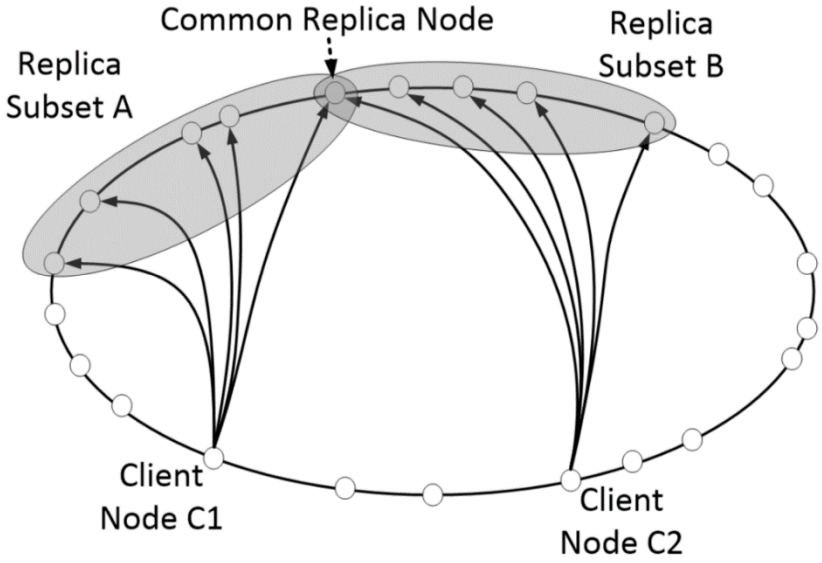
- **Paxos**
  - Protocol family for consensus (multi, cheap, fast, generalized, …)
  - Roles: Client/Proposer (requester), Acceptor (voter), Leader (coordinator), Learner (responder)
    - Client sends requests to a proposer
    - Proposer send proposal acceptor, send back promise
    - If majority promises, send value to acceptor, acceptor sent to learner
    - Learner sent result to client
- **2 Phases**
  - Phase 1: prepare / promise
  - Phase 2: accept / accepted

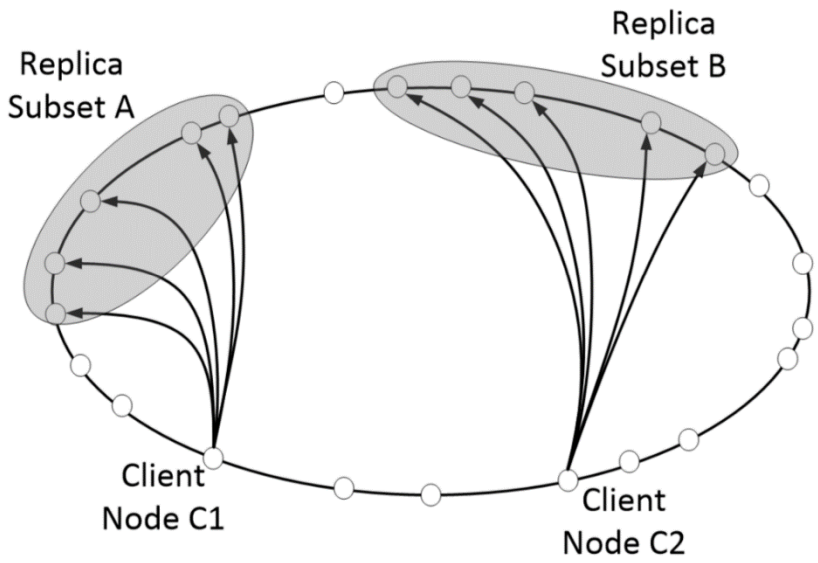


http://en.wikipedia.org/wiki/Paxos_%28computer_science%29

Consistency

- **Raft – Alternative to Paxos (easier), three roles: leader, follower, candidate**
  - Paxos and DHTs [1], [2]
- **Consistency in DHTs – vDHT**
  - CoW, versions, 2PC, replication, software transactional memory (STM) → for consistent updates. Works for light churn





Consistency

- **vDHT Basics**
  - No locking, no timestamps (replication time may have an influence)
  - Every update – new version
    - 1. get latest version, check if all replica peers have latest version, if not wait and try again
    - 2. put prepared with data and short TTL, if status is OK on all replica peers, go ahead, otherwise, remove the data and go to step 1.
    - 3. put confirmed, don't send the data, just remove the prepared flag
- **In case of heavy churn, API user needs to resolve**
- **Demo: net.tomp2p.examples. ExampleVDHT (new)**
  - Example: no consistency – traditional put strategy
  - Example: vDHT - pessimistic put strategy

4. Rsync

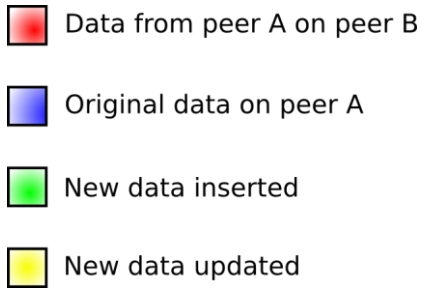
Introduction, Example, and Discussion

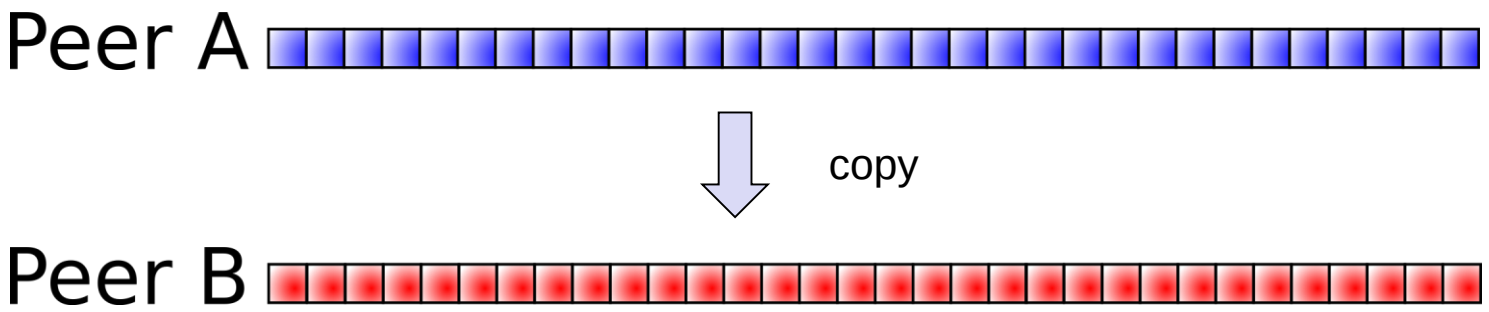
Rsync - Introduction

- **Rsync used to synchronize data over network**
  - Minimizing data transfer (delta)
- **Command line client (standard utility)**
  - E.g. `rsync -aP --link-dest=$HOME/Backups/current /path/to/important_files $HOME/Backups/back-$date`
  - Unchanged files are hard linked (`--link-dest`) → Can be used for incremental backups
- **Main idea**
  - Receiver compute two checksums (strong, weak) → sent to sender
  - Sender computes with weak checksum and checks for known blocks
  - Sender verifies with strong checksum → sends difference to receiver
- **Example with two peers:**

Rsync - Example

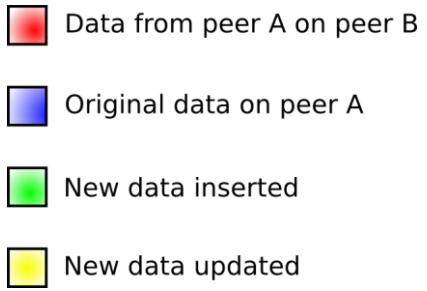
- **Peer B does not have the data → peer A copies it to peer B, no need for rsync**

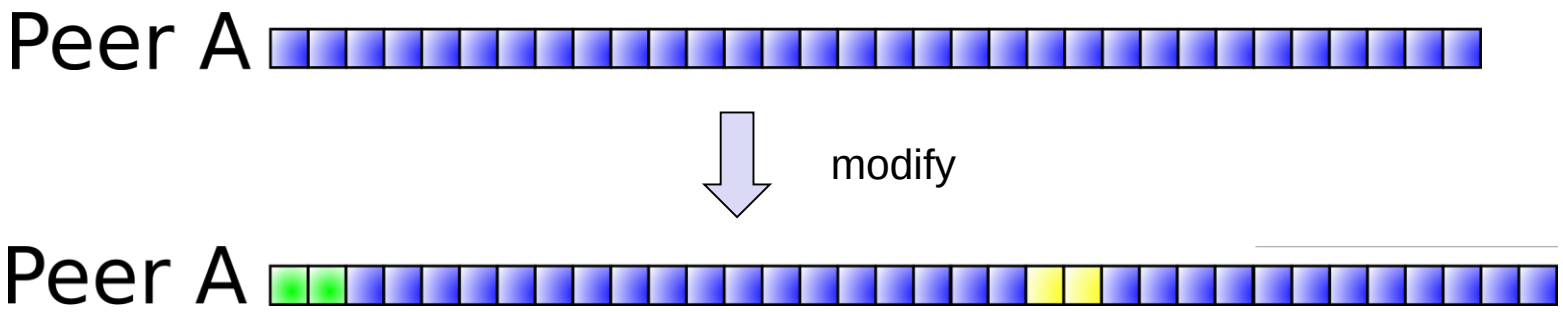




Rsync - Example

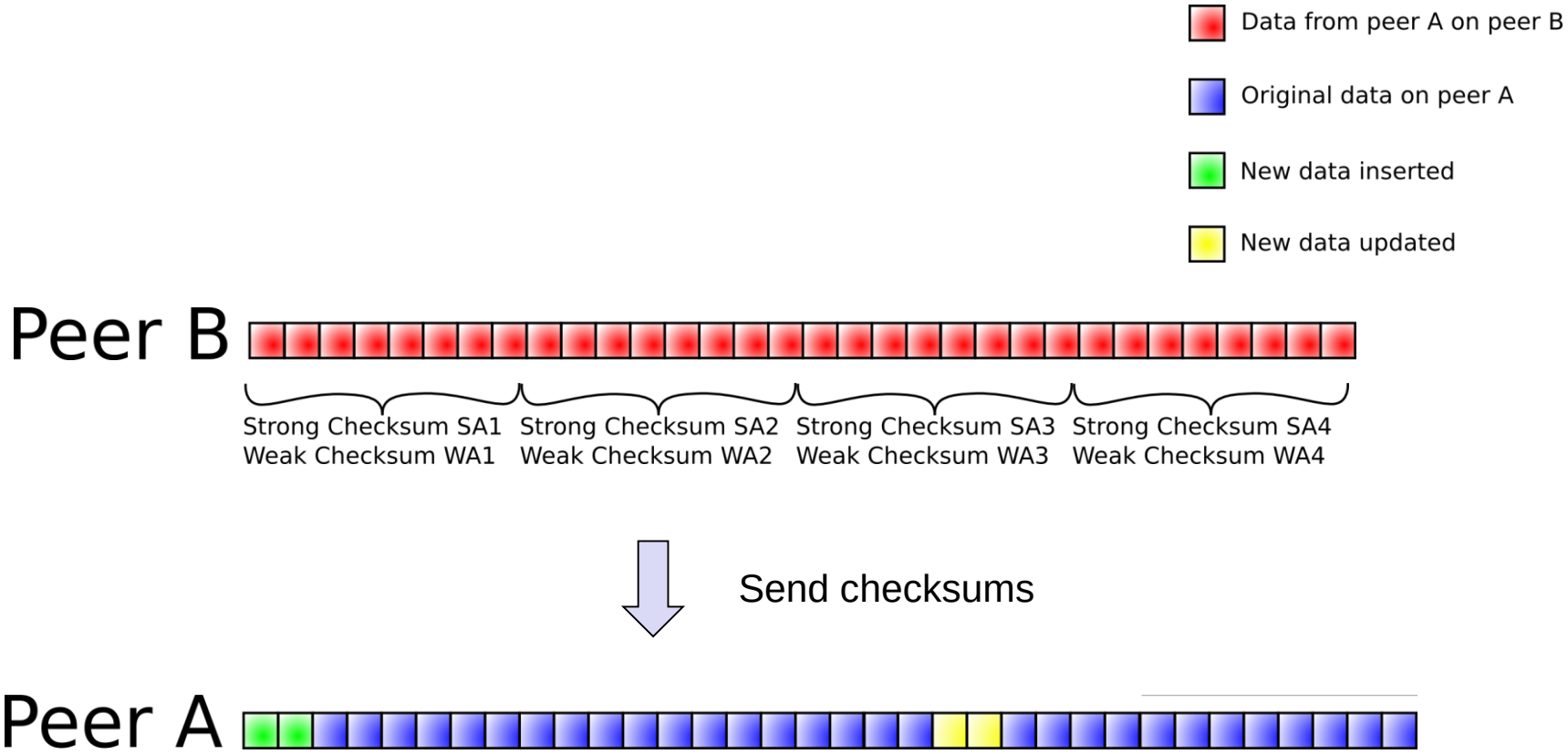
- **Peer A modifies data (insert, update)**
  - Wants to synchronize with peer B



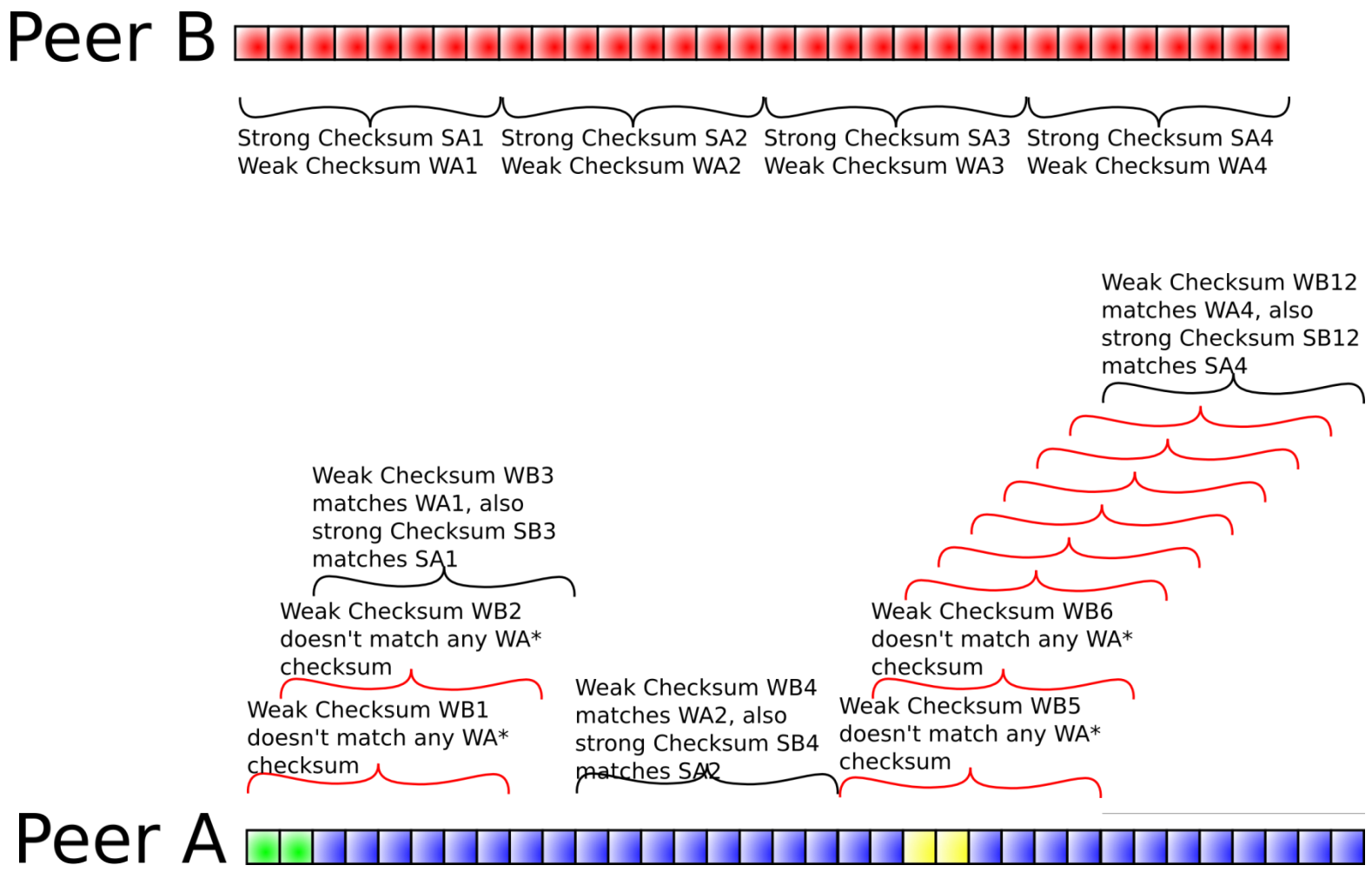


Rsync - Example

- **Peer A modifies data (insert, update)**
  - Wants to synchronize with peer B

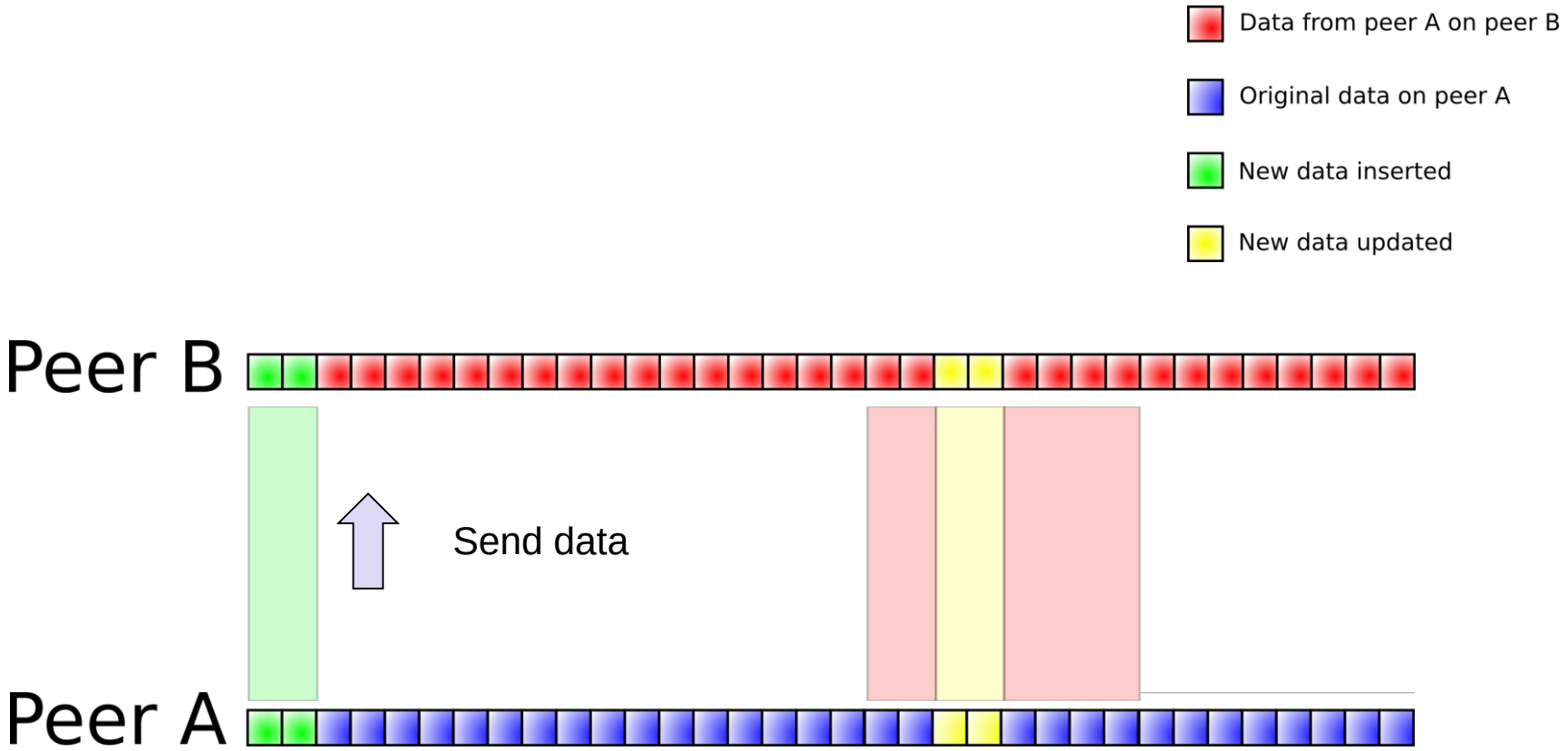


Rsync - Example



Rsync - Example

- **Peer A sends 2 + 8 blocks to peer B**
  - Peer A and peer B have same data



Rsync - Mechanism / Discussion

- **If data does not exist → copy**
  - Use-case: portion of data stays the same
  - Replication
- **Two checksums for performance (MD5 and Adler-32)**
  - Collisions possible, but unlikely 2^{-160}
- **Rsync in TomP2P (demo)**
  - If you use CoW, don't use Rsync!
  - net.tomp2p.examples.ExampleRsync (new)