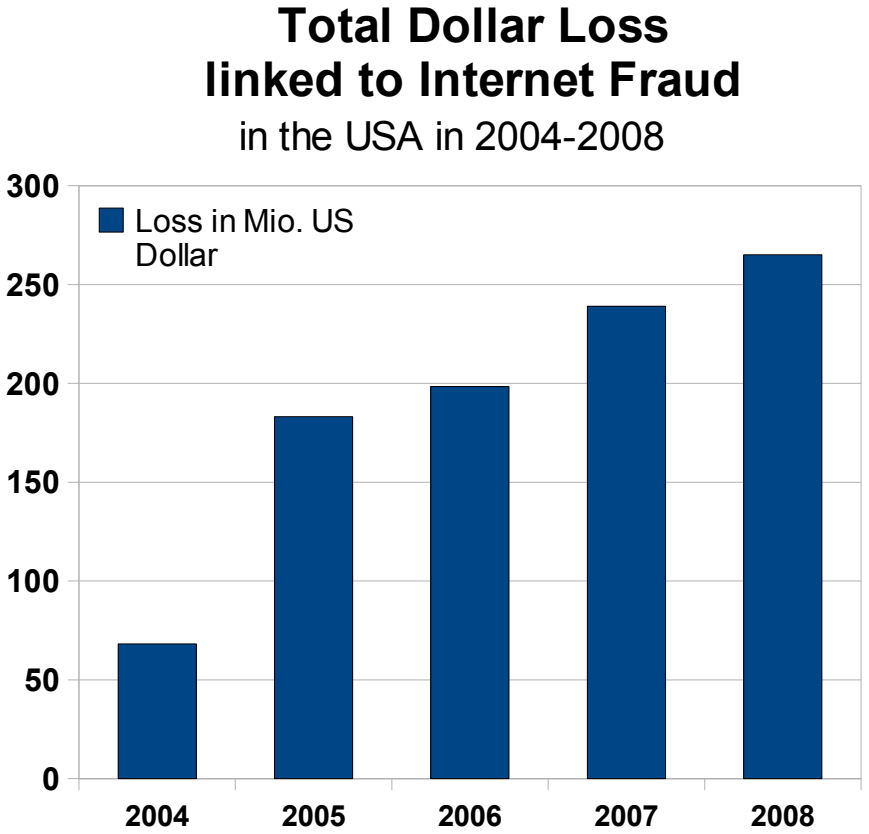
Blacklisting Malicious Web Sites using a Secure Version of the DHT Protocol Kademlia

Road Map

1. General Idea
2. Application Design
3. Implementation
4. Testing

Motivation
1. General Idea



Source:
Internet Crime Complaint Center (IC3) 2008 Annual Report

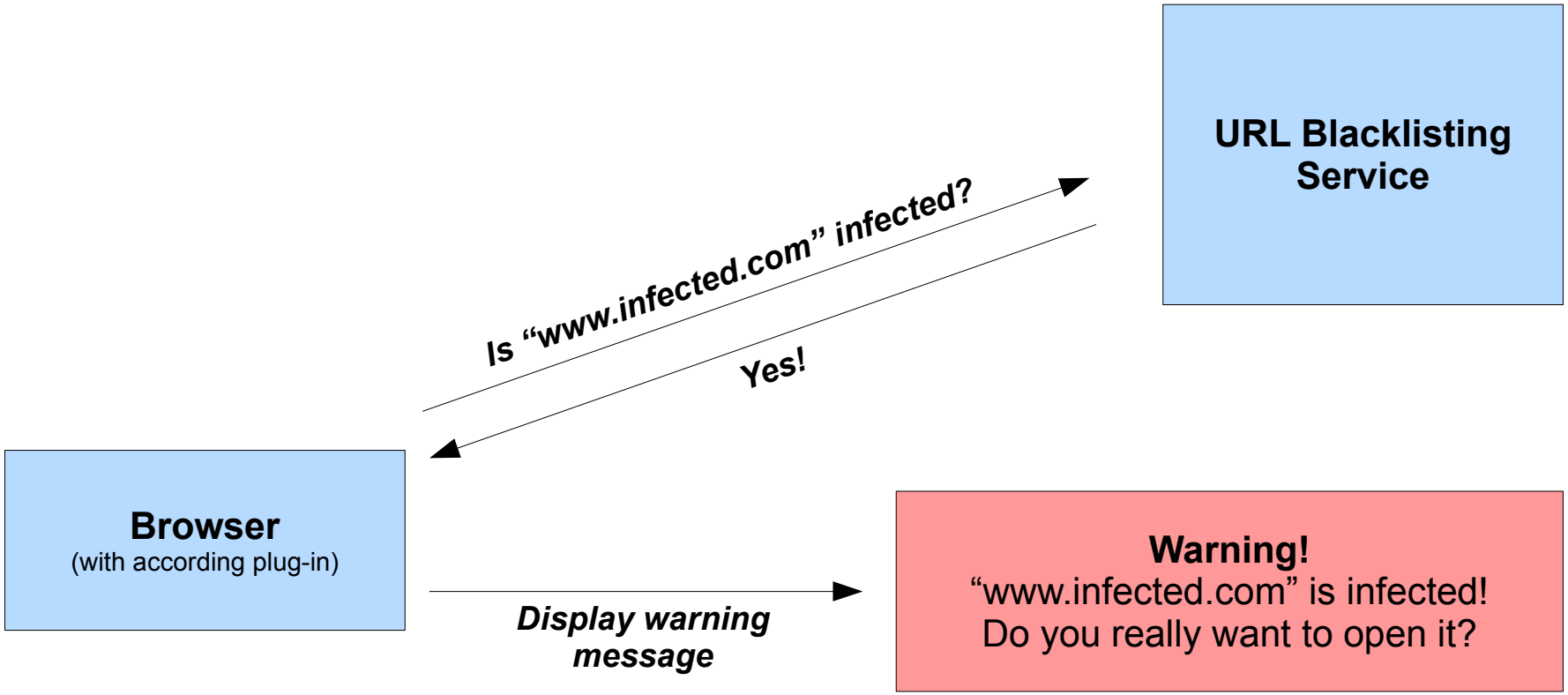
www.ic3.gov/media/2009/090331.aspx

- Rising Internet crime rates
- Less phishing, less virus-infected attachments
- More malicious Web sites → more *drive by*-downloads

Initial Situation / Problem Description

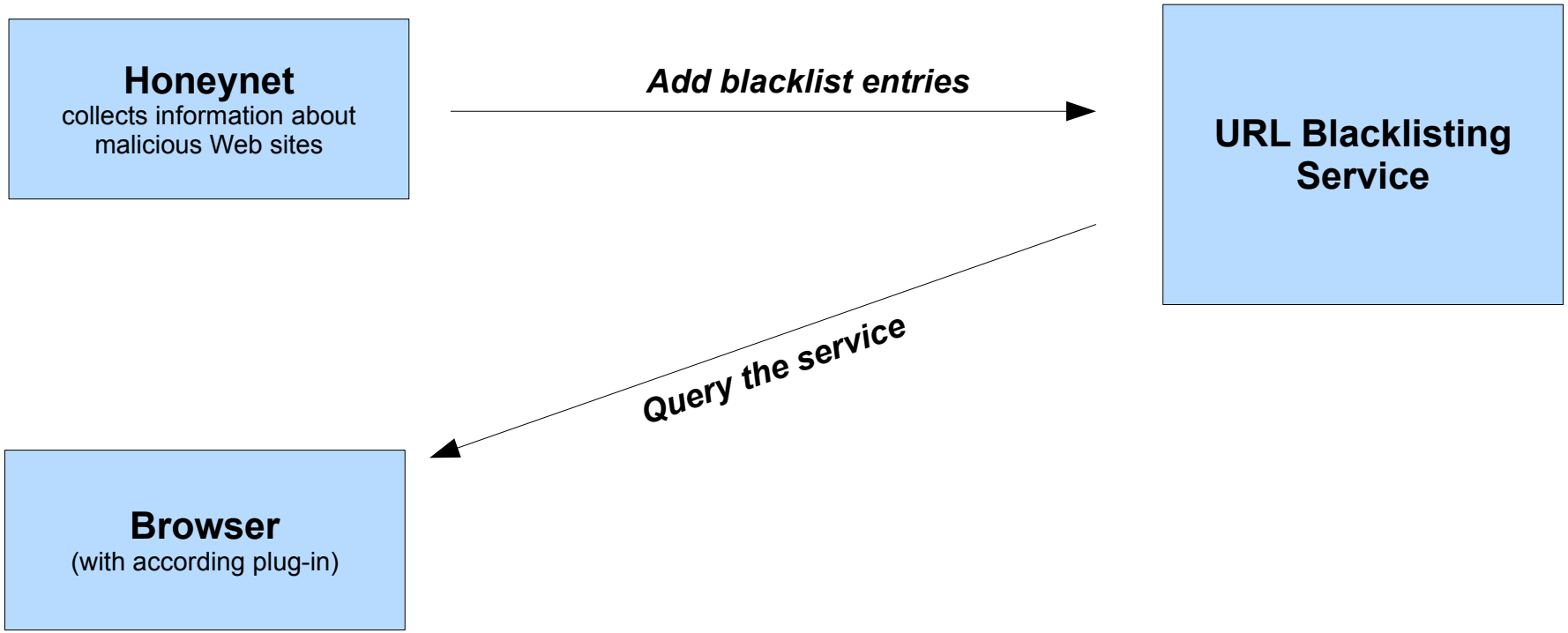
1. General Idea

Solution
1. General Idea



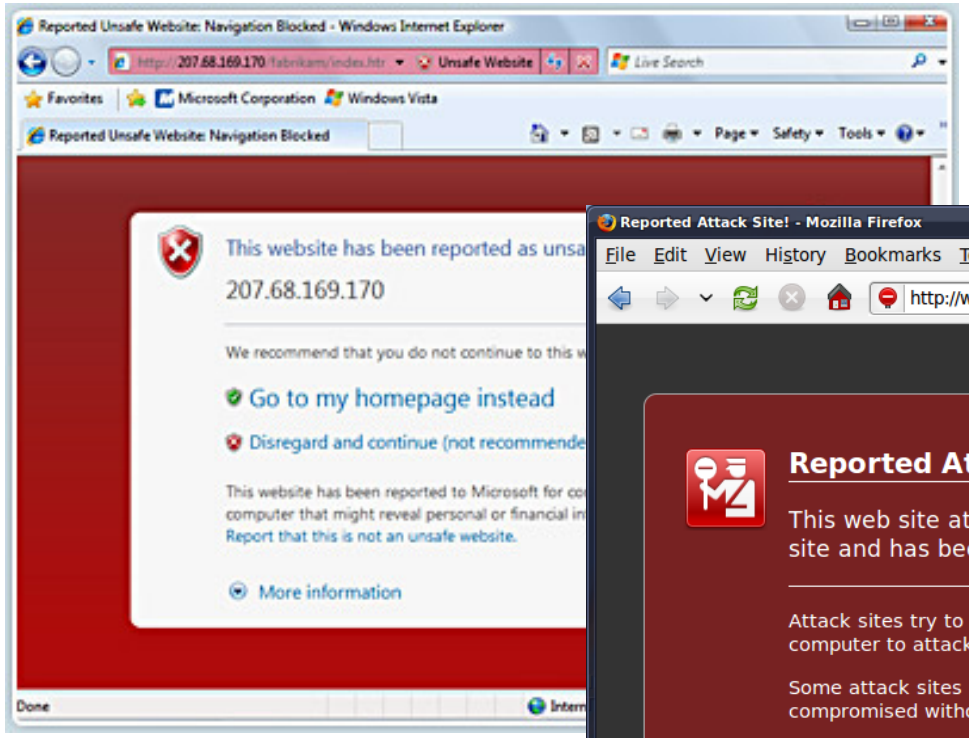
Blacklisting Service

1. General Idea

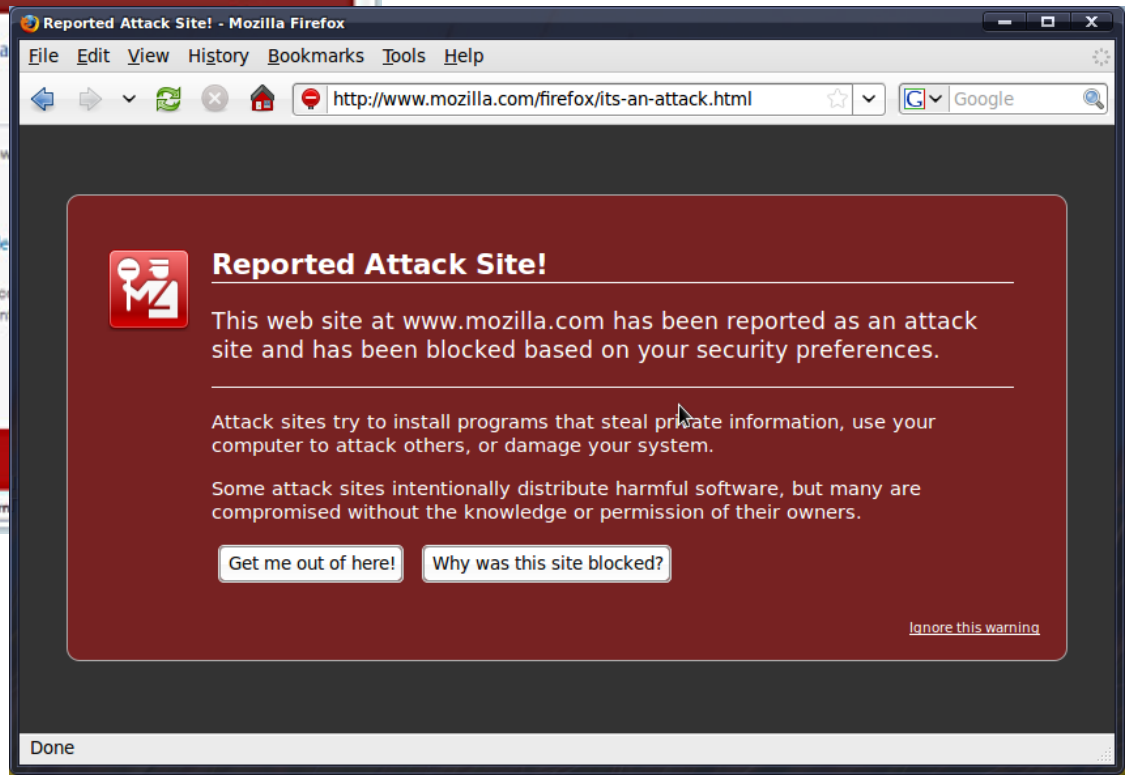


Related Work

1. General Idea



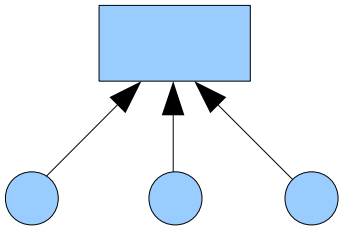
Microsoft SmartScreen (IE 8+)



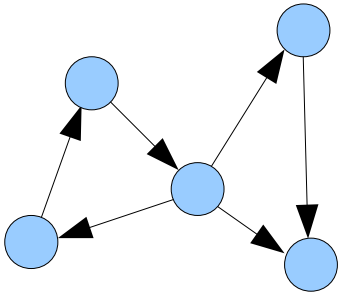
Google Safe Browsing (FF 3+ / Safari 3.2+)

Client-Server vs. Peer-to-Peer

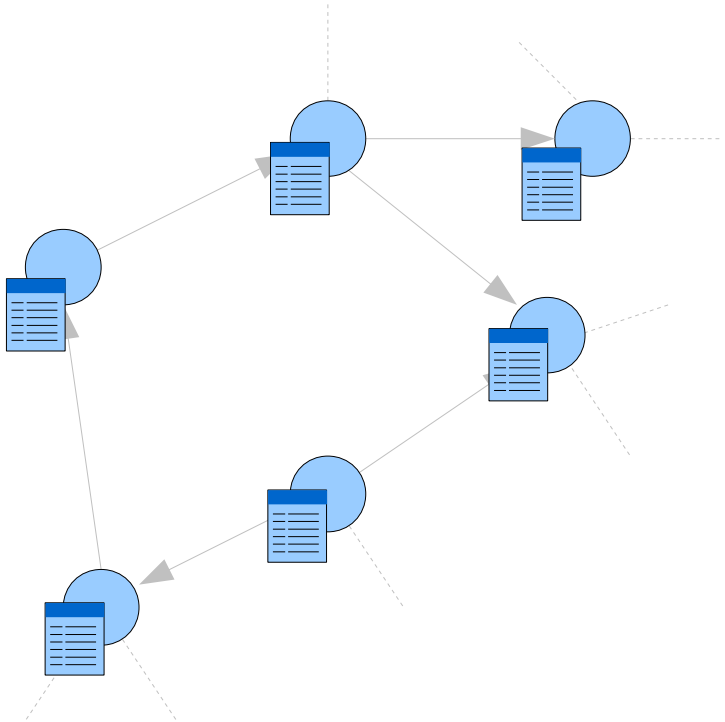
1. General Idea



- Problems in a client-server-based environment:
  - Limited scalability
  - Resource limitations (CPU time, RAM, bandwidth, ...)
  - Synchronization between servers
  - Data replication
  - Single point of failure/attack



- Peer-to-Peer (P2P):
  - Scalability *by design*
  - Resource flexibility
  - No single point of failure/attack, no replication, no synchronization



Distributed Hash Tables (DHT) ...

- are *structured* P2P networks
- define a logical position for each node and entry
- store content at specific positions in the network, redundantly
- are fault-tolerant and reliable

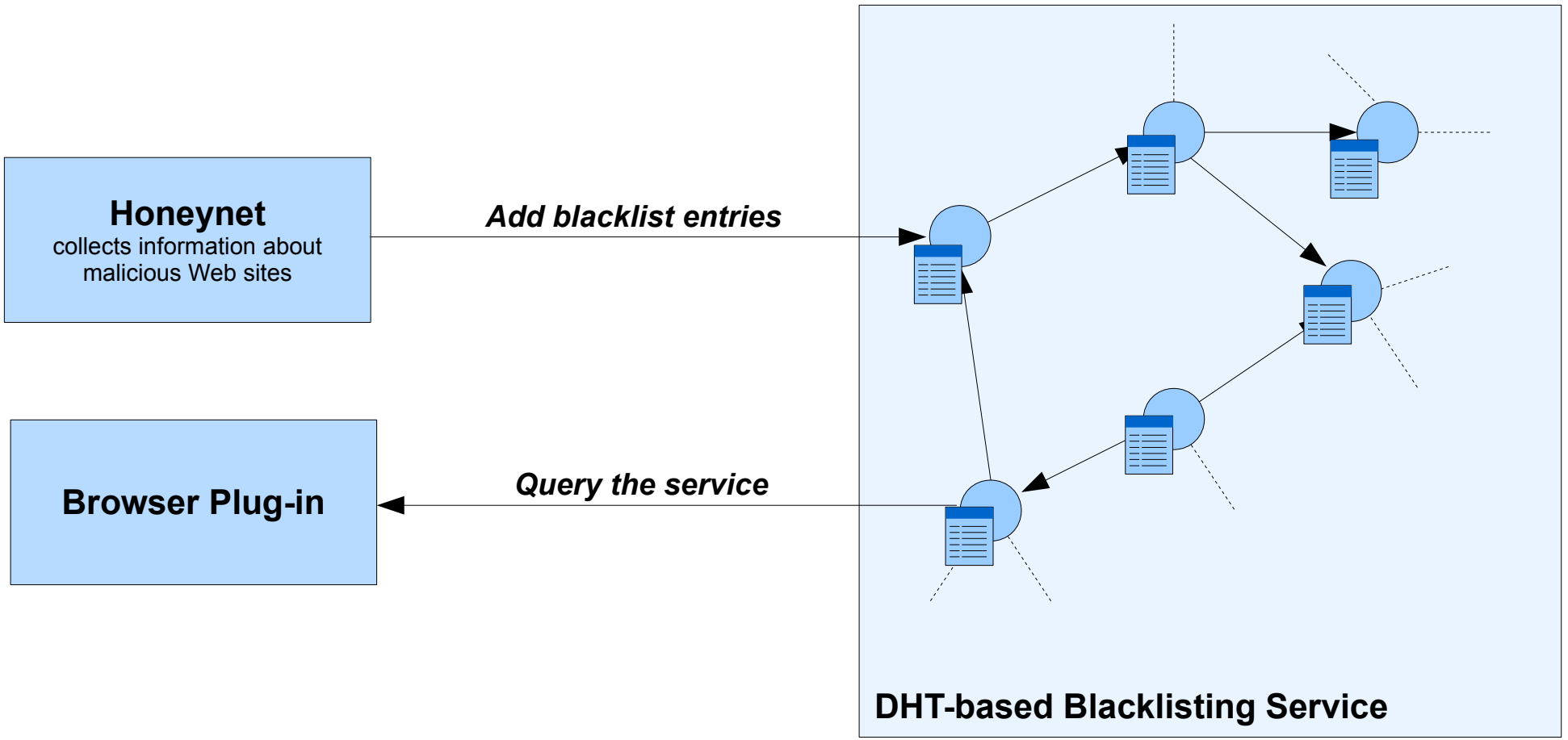
Example (Pseudo-Code):

```
nodes := locateResponsibleNodes("infected.com");

foreach nodes as node do
  node.store("infected.com", "infected");
```

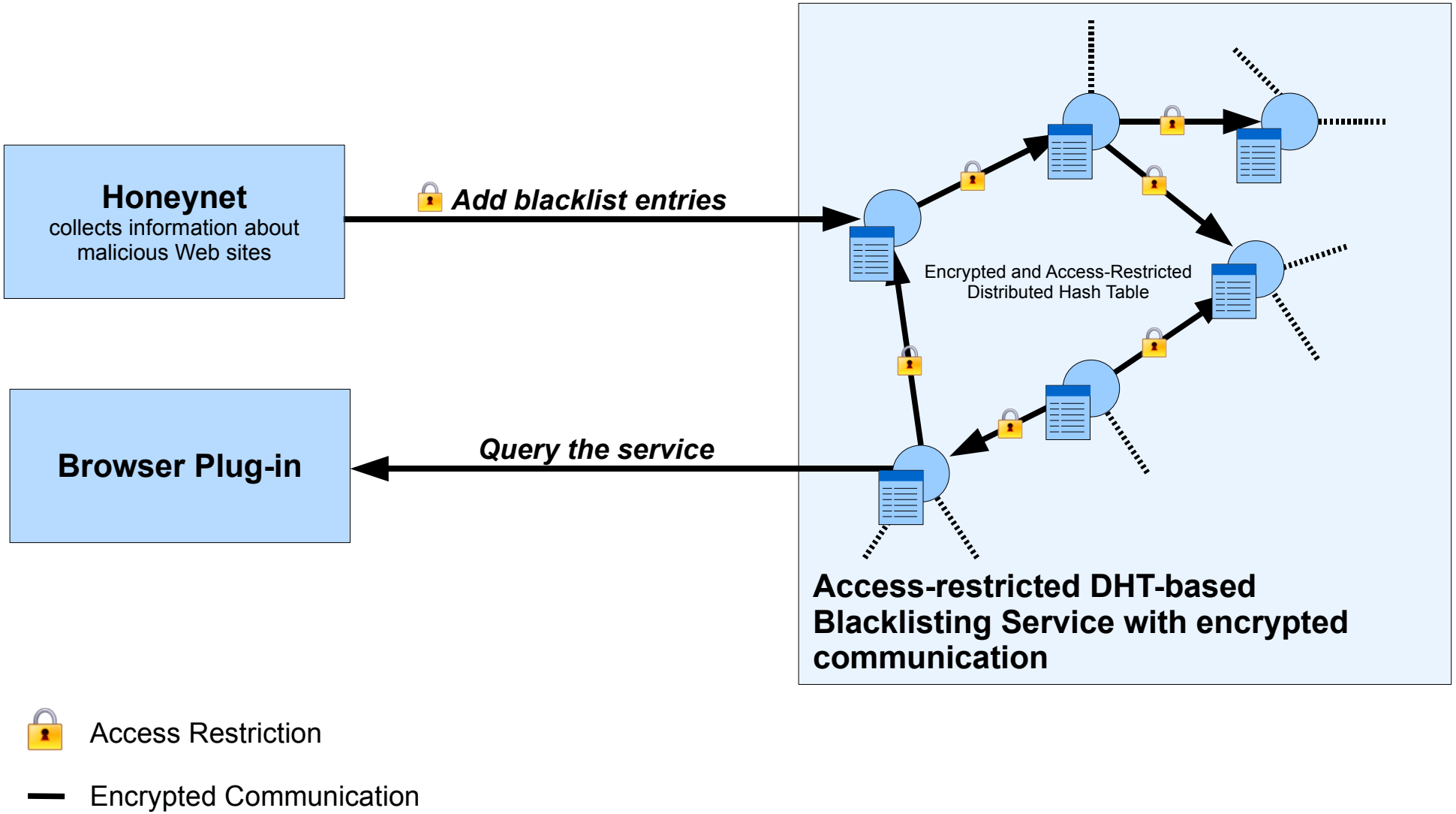
DHT-based Blacklisting Service

1. General Idea



DHT-based Blacklisting Service

1. General Idea

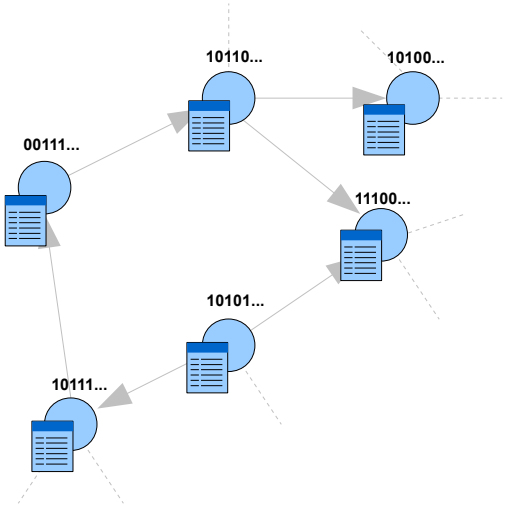


- *Use the honeynet-provided data to create a URL blacklisting service*
  - Handle large amounts of users (concurrency)
  - Fast queries, low round-trip time (RTT)
  - Scalable, extendable, reliable

- *Create a secure and trustworthy DHT protocol, called Kademlia Secure (KadS)*
  - Restricted access
  - Communication encryption

Underlying DHT Protocol: Kademlia

2. Application Design



Example (XOR metric):

id_1 := 10111...
id_2 := 10001...

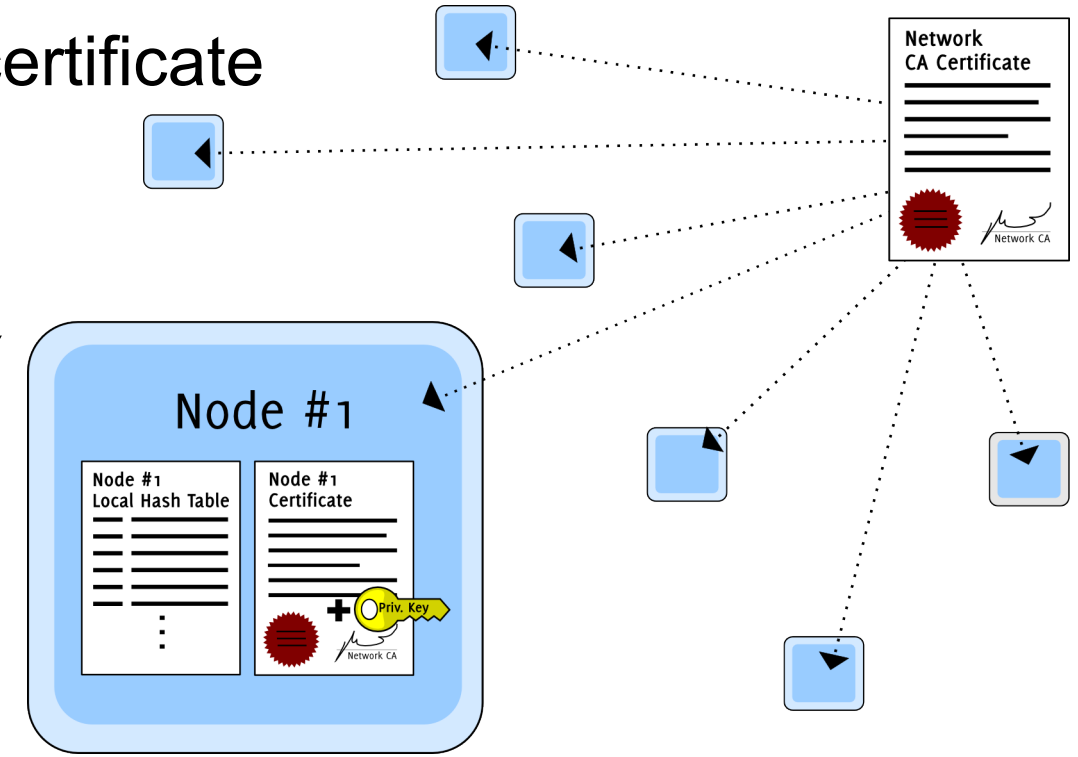
$d(id_1, id_2) = id_1 \oplus id_2$
= 00110...

- Fault-tolerant design, provable consistency, good performance
- Assigns a 160-bit ID to
  - each node (*nodeID*)
  - each DHT entry (*key*)
- Distance is based on the XOR metric
- Provides four RPCs:
  - PING(*nodeID*)
  - STORE(*key*, *value*)
  - FIND_NODE(*nodeID* or *key*)
  - FIND_VALUE(*key*)

Kademlia Secure (KadS):

- One *root CA* certificate per network
- Each node must possess
  - a copy of the network's CA
  - a CA-signed public key certificate
  - a matching private key

→ *Provable Node Identity*



Protocol Extensions:

- KadS handshake (TCP)
  - Exchange public key certificates
  - Verify each other's identity
  - Exchange a random session key

- Encrypted Messaging (UDP)
  - Exchange messages using the previously negotiated session key

Blacklist Data Structure:

- Domain
- Expiry Date
- Analysis Code
  - UNKNOWN
  - FAILED
  - CLEAN
  - PARTIALLY_INFECTED
  - INFECTED
- Path List

Example:

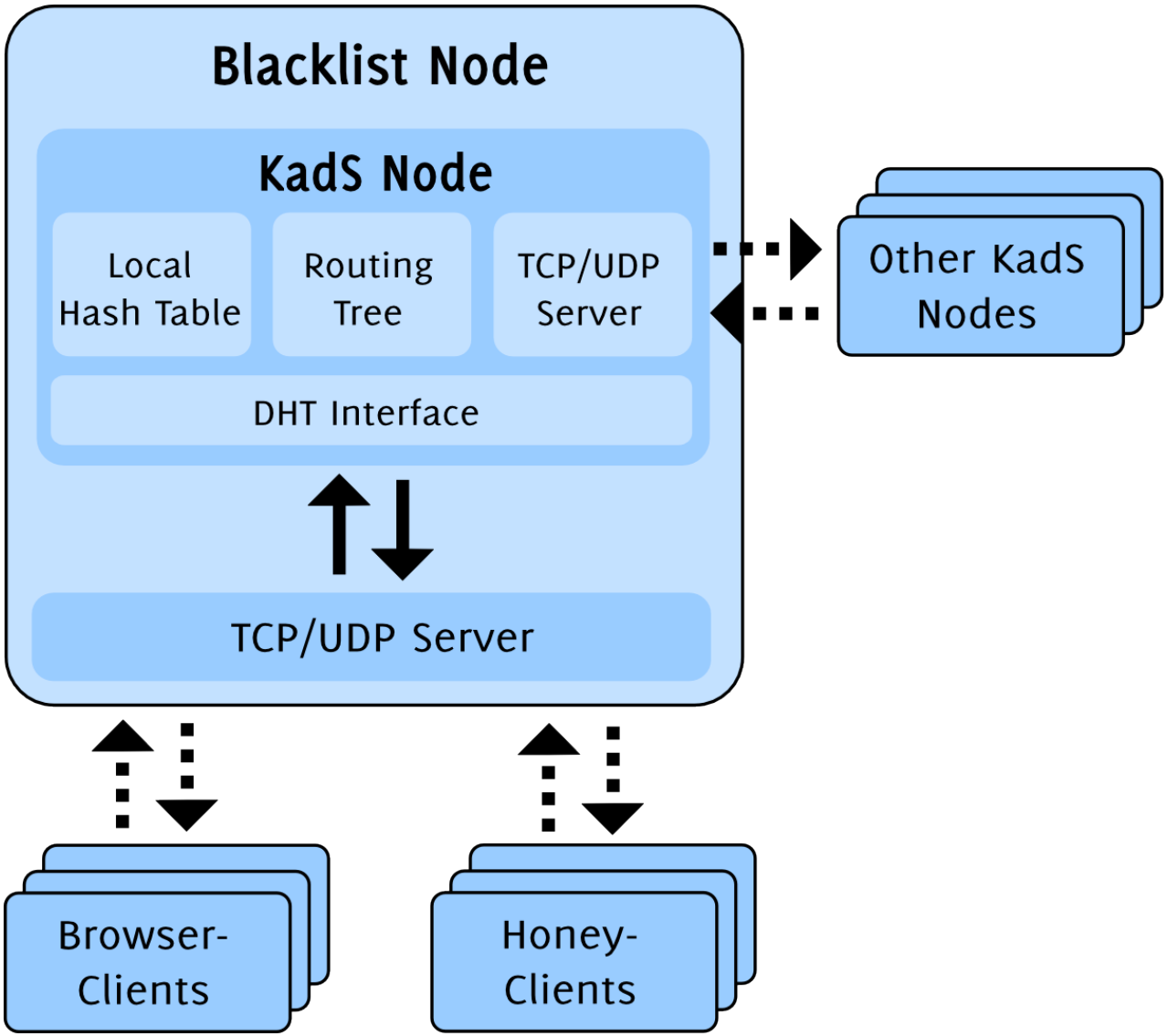
- Domain: infected.com
- Expiry Date: 01/01/2010
- Analysis Code:
  PARTIALLY_INFECTED
- Path List:
  </bad.html, /worse.html>

Blacklisting Service:

- KadS node (as distributed data storage)
- Client interfaces
  - for the honeyclients
  - for the browser plug-ins
- Business rules to enforce a specific data structure

Schematic Design of a Blacklist Node

2. Application Design



Interface `DistributedHashMap`

3. Implementation

```
public interface DistributedHashMap {
    public void connect(InetSocketAddress address);
    public boolean contains(Identifier key);
    public Serializable get(Identifier key);
    public void put(Identifier key, Serializable value);
    public void remove(Identifier key);
    public void close();
}

public class KadS implements DistributedHashMap {
    ...
}
```

Example: Using the KadS class

3. Implementation

```
Config config = new Config(new File("nodes/499/config.cfg"));
KadS kads = new KadS(config);

/* Connect to existing KadS network (KadS handshake, SK-Exchange) */
kads.connect(new InetSocketAddress("node1.kads.dyndns.org", 6852));

/* Add/Update a DHT entry */
kads.put("cities", new String[] { "Mannheim", "Heidelberg" });

/* Perform FIND_VALUE-Operation */
List<Country> countries =
    (List<Country>) kads.get("countries");

kads.close();
```

nodes/499/config.cfg

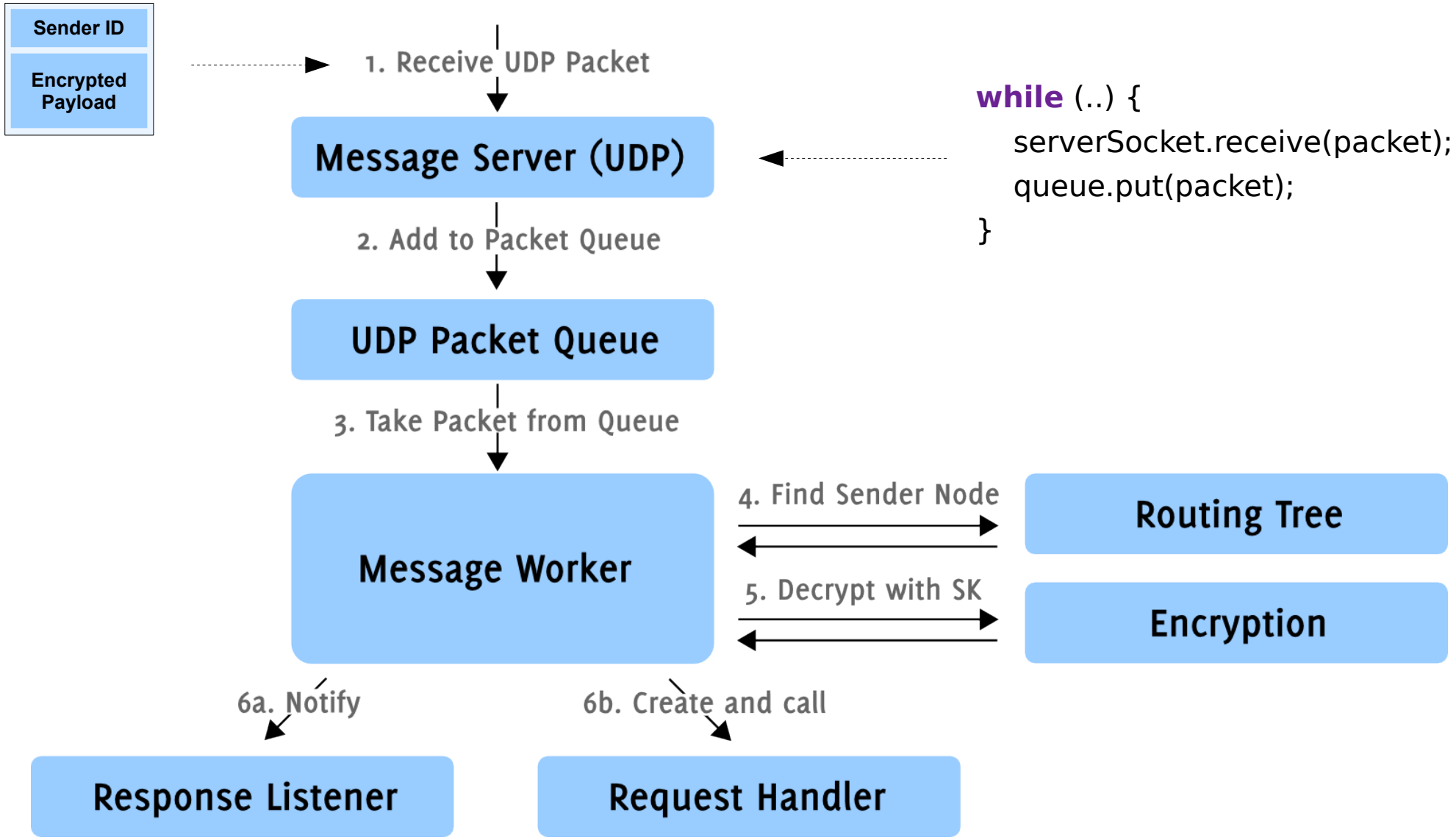
```
kads.ca = nodes/ca.cer
kads.node.cer = nodes/499/node.cer
kads.node.key = nodes/499/node.key

kads.port = 6852

kads.boot.sleep = 2000-10000
kads.boot.nodelist = nodes/bootstrap.list
kads.boot.tries = 5
```

Message Flow in a KadS Node

3. Implementation



Example: Using the `TrustworthyClient` class

3. Implementation

```
TrustworthyClient client =
    new TrustworthyClient("Honey1.cer", "Honey1.key", "BlacklistCA.cer");

/* Connect to a blacklist node in the network (SSL-Handshake) */
client.connect(new InetSocketAddress("node1.blacklist.dyndns.org", 7001));

if (!client.getAccessRights().hasWriteAccess())
    throw new Exception("No write access!");

/* Store a blacklist entry (via TCP/SSL socket) */
client.store(new Entry("uni-mannheim.de", Entry.CLEAN));
client.store(new Entry("uni-heidelberg.de", Entry.INFECTED));

client.store(new Entry("spiegel.de", Entry.PARTIALLY_INFECTED,
    Arrays.asList(new String[] {“/bad.html”, “/infected.html”})));

/* After all entries are stored, the TCP/SSL socket can be closed */
client.disconnect();
```

Example: Using the `UntrustworthyClient` class

3. Implementation

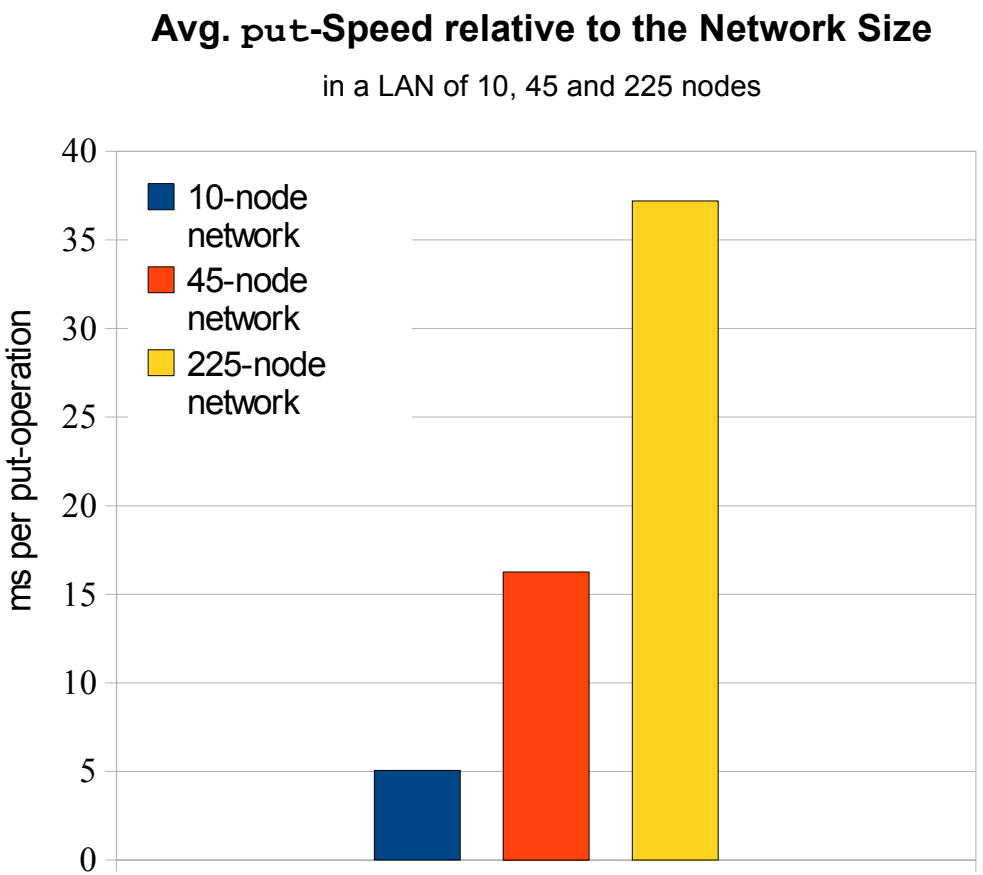
```
/* Called once when the browser starts */
public void onBrowserStart() {
    blacklistClient = new UntrustworthyClient("BlacklistCA.cer");

    /* Connect to a blacklist node in the network */
    blacklistClient.connect(
        new InetSocketAddress("node1.blacklist.dyndns.org", 7001) );
}

/* Called every time code is loaded from a so far foreign page */
public void onBeforeOpenWebsite(String domain, String path, ...) {
    Entry entry = blacklistClient.query(domain);

    if (entry.getAnalysisCode() == Entry.INFECTED)
        throw new MaliciousWebsiteException(...);
    ...
}
```

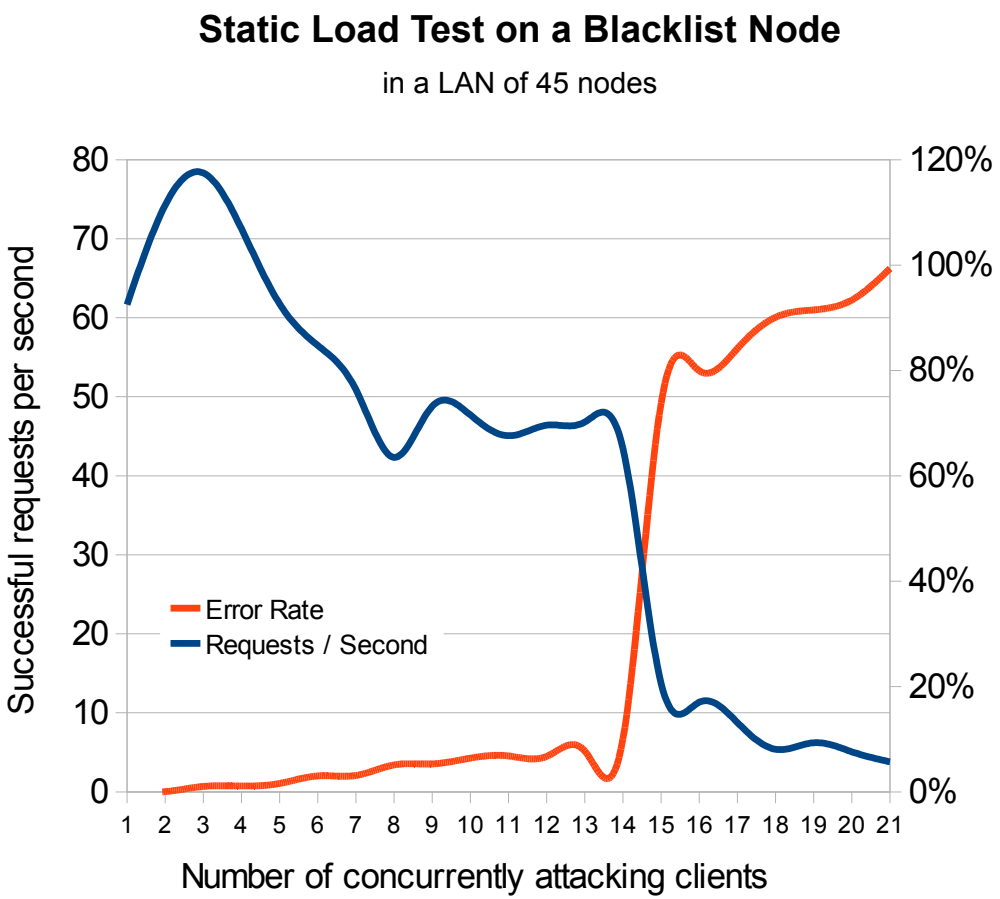
Average RTT of KadS' `put`-Operation

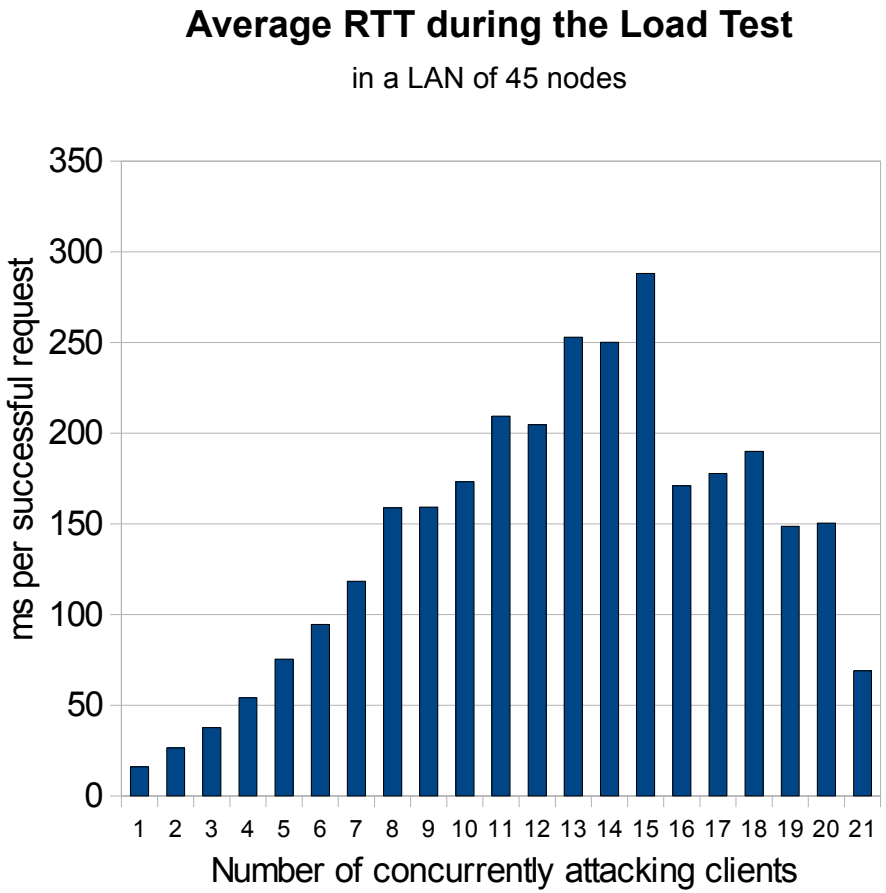


- How fast is the KadS network?
- Is the speed dependent of the network size?

Static Load Test of a Single Blacklist Node

4. Testing





- How many requests can a single node handle?

Conclusion

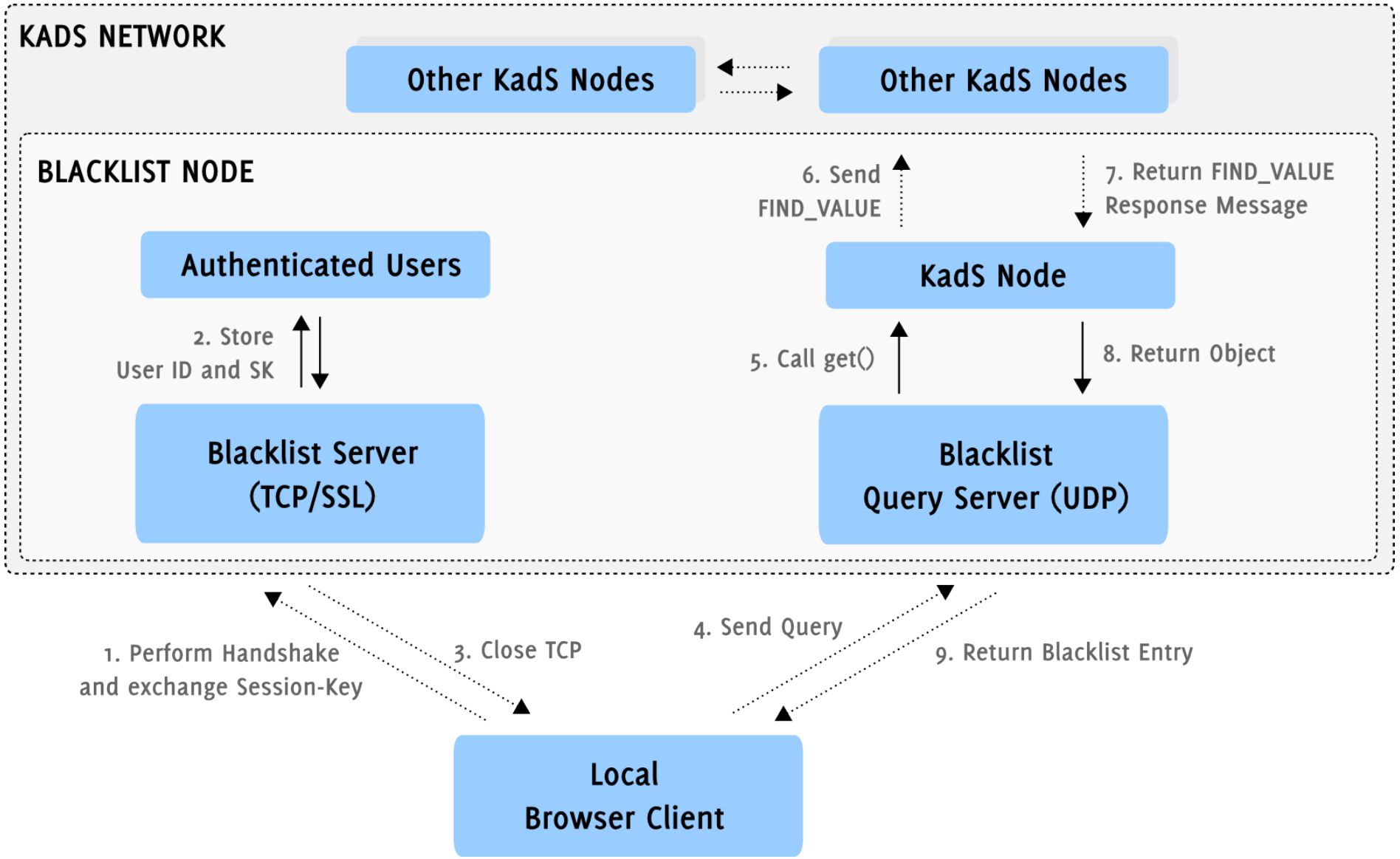
- Trustworthy, secure DHT Protocol *KadS*
  - Prototypical implementation
  - Satisfactory results
  - No logarithmic scalability
  - Enables possibility of further research

- KadS-based Blacklisting Service
  - Not yet tested with real data

EOF

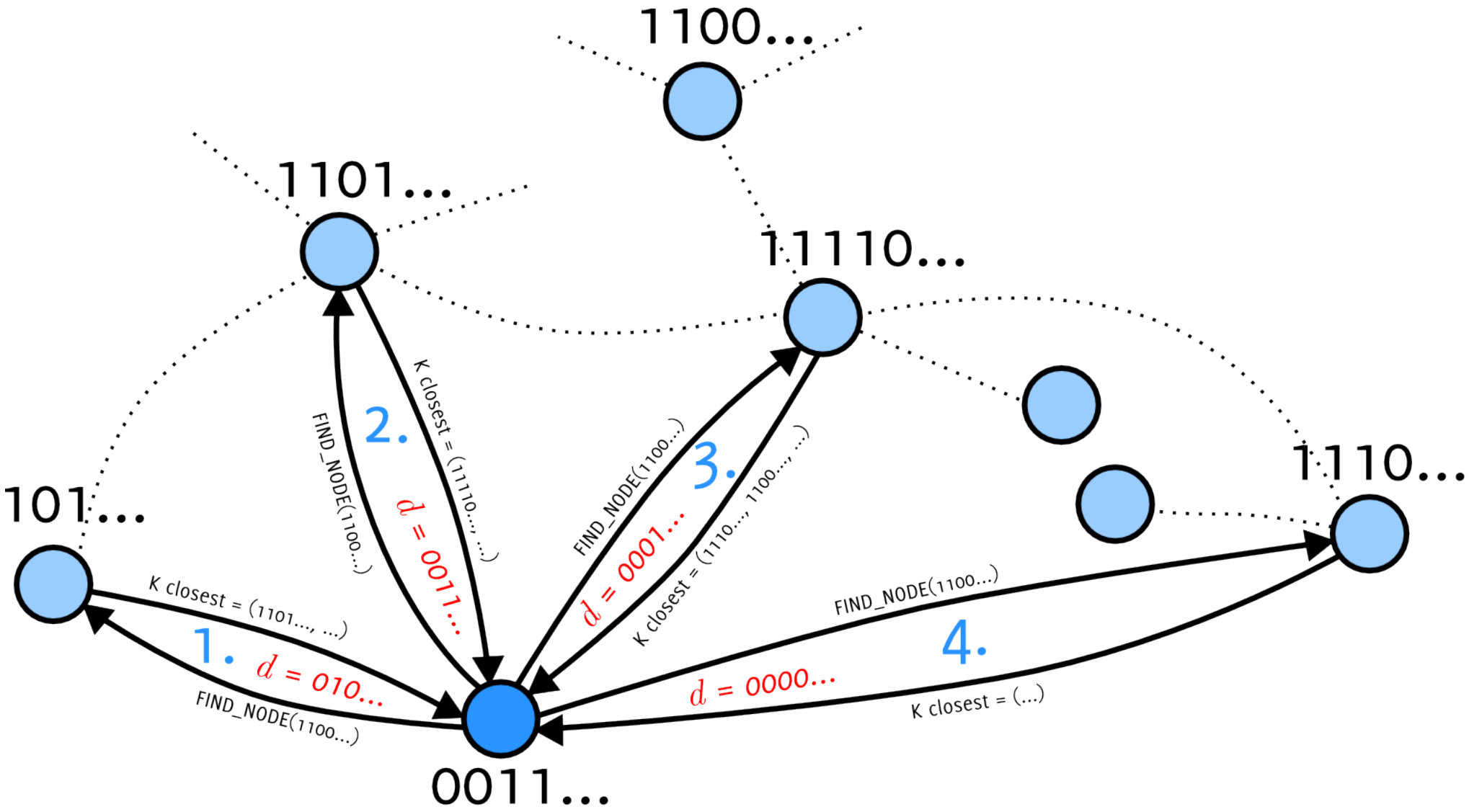
Query-Process of a Browser Client

3. Implementation



Kademlia's FIND_NODE-Operation

2. Application Design



The node 0011. . . tries to locate the identifier 1100. . .

Example: Behind the `put`-Operation

3. Implementation

```
/* Find K closest nodes */
List<VerifiedNode> closestNodes = new FindNodeOperation(key, ..).execute();

/* Send STORE requests to each of them */
for (VerifiedNode recipientNode : closestNodes) {
    StoreRequest storeRequest = new StoreRequest(key, value);
    ...
    messenger.addResponseListener(storeRequest.getId(), recipientNode, this);
    messenger.send(recipientNode, storeRequest);
}
```