Security & Privacy in **P2P** Networks

Niels Olof Bouvin

Overview

- **Aspects of security***
- **Venues of attack**
- **Techniques for anonymity & censorship resistance**
- **Securing a DHT**

*This is *not* the interesting part to talk about during the exam

Dangers of distributed systems

- **Trust**
  - who can you trust?
- **Identity theft**
  - pretending to be you (or someone you trust)
- **Privacy**
  - preventing others listening in on the conversation
- **Censorship & attacks**
  - denying you the right to know

The Internet

- **The Internet is vast and not at all safe**
  - data packets going from machine to machine before they reach you
- **Many standards and protocols established back in safer days**
  - SMTP, NNTP, ftp, telnet, ...
- **There are plenty of criminals, who would delight in taking over your machine and stealing your data**
  - see iloveyou, Code Red, SQL Slammer, SoBig.F, Swen, Storm, NotPetya, WannaCry, etc.
  - not to mention DDoS, industrial espionage, etc.

Who can you trust?

- **Surely you can trust well-established Web sites?**
- **Several important open source ftp servers have been 'owned' over the years**
  - thus leaving black hats free to insert code of their own in popular open source projects... (example: savannah.gnu.org)
- **This also happened for Microsoft some years ago**
- **Numerous sites have been hacked for credit card numbers etc.**
- **Spoofing of URLs: `www.paypa1.com`**
  - Unicode URLs have made everything more *interesting*

Cryptography

- **Fact: Messages can be intercepted. But intercepted data is worthless, if the interceptor cannot read it**
  - (the people involved are traditionally known as Alice, Bob, and Carol)
- **Cryptography is *very* old, and has been based on a long number of techniques**
  - today, cryptography is based on advanced, hard-to-solve mathematical problems
- **Regardless of the method used, a *key* is used to signify how the plain text is transformed into cipher text**
  - and for some reason, it always involves Alice and Bob trying to communicate securely with Carol trying to eavesdrop…

Symmetric cryptography

- **The same key is used to encrypt and decrypt the message**
- **Advantages**
  - symmetric cryptography is *fast*
- **Disadvantages**
  - the key must be securely exchanged between Alice and Bob
  - if the key is compromised, the entire communication is instantly readable

Asymmetric cryptography

- **Keys come in pairs:**
  - a public key known to all
  - a private (secret) key known *only* by the user
- **A message encrypted with the public key can be decrypted *only* by the private key**
  - so if Alice encrypts a message with Bob's public key, only Bob can decrypt it with his private key
- **A message signed with the private key can be verified *only* by the public key**
  - so if Alice signs a message with her private key, all can verify (using Alice's public key) that Alice is the author

Asymmetric cryptography

- **Advantages**
  - as the private key is never shared, the system is secure
  - the system can also be used to authenticate (or "digitally sign") messages
- **Disadvantages**
  - only as secure as the private key...
  - significantly slower than symmetric cryptography
    - not as much a drawback as you might think

Establishing trust

- **How does Alice know Bob is really Bob, and not Carol claiming to be Bob?**
- **Asymmetric cryptography often relies on CAs – Certification Authorities**
  - these, using out-of-band methods, establish the correct identity of Bob, and assigns a (signed) certificate to Bob
  - Alice can then verify that some CA has vouchsafed Bob, and if she trusts the CA, she can trust Bob
- **A problem with these certificates is the cost…**
  - at least until **Let's Encrypt** emerged (https://letsencrypt.org)

Establishing trust

- **A less centralised approach is taken by PGP (Pretty Good Privacy), where Bob relies on associates to confirm his identity**
  - users sign signatures of people they know (and have verified)
  - if Alice knows (and trusts) any of these associates, she can trust Bob's identity
  - “small-world” experiments show typically at most six degrees of separation between any two persons
  - trust decreases over distance
- **GPG is the open source equivalent**

Symmetric/asymmetric cryptography

- **Asymmetric cryptography is used for the initial communication to establish identity and (securely) exchange a randomly generated symmetric key**
- **This is the method used by TLS used in, e.g., `https`**
  - the Web server provides the Web browser with its CA signed certificate (the browser checks this against its installed CA root certificates)
  - the browser generates a random key, encrypts it with the server's public key, and returns it to the server
  - as only the server can decrypt the key, the server and browser can initiate a securely symmetric (i.e., *fast*) encrypted session

Secure hashes

- **Secure (or cryptographic) hashes are used to verify the integrity of a message**
  - most common used to be MD5 (128 bits) and SHA-1 (160 bits)
- **It is thought computationally infeasible to create two different messages with identical secure hash codes (it requires brute force and 2^{128} or 2^{160} are *big*)**
  - This is no longer true...
    - MD5 and SHA-1 have both been weakened. Neither are fatally compromised, but methods have been devised to generate messages matching a given hash code. Use SHA256 or WHIRLPOOL instead

Secure hashes

- Thus, if the (secure) hash code of a message is known, we can check whether the message has been modified by computing the hash code of the message ourselves and comparing the results
- Given the quality of the secure hash, it is just as good (and *much* faster) to sign the (compact) hash code with your private key for authentication as signing the entire message

Security – a purely technical problem?

- **Security can be addressed through a number of technical means**
- **However, these valiant efforts are all for naught**
  - in the face of inexperience and nigh terminal cluelessness
- **Some of the most successful black hat hackers have operated, not through absurd Hollywood computer guru excellence, but through social engineering**
  - (hacking being considerably easier, if you can get people to tell you their password)

Overview

- Aspects of security
- **Venues of attack**
- **Techniques for anonymity & censorship resistance**
- **Securing a DHT**

How to attack a P2P system?

- **Attacks against P2P systems can broadly be divided into**
  - (Distributed) Denial of Service
    - requesting
    - pushing
  - Malicious peers
  - Sybil
  - Shadow

(Distributed) Denial of Service

- **Overload the system**
  - often using a swarm of captured machines *(botnet)*
- **Difficult to resist, if attackers are resource rich**
- **Defences:**
  - minimise cost of losing *any* individual peers
  - make it difficult to identify important peers
  - optimise traffic so that only minimal part of network is affected
  - do not let new (bogus) data overwrite old (good) data

Malicious peers

- **Malicious peers can**
  - reroute traffic in wrong directions
  - claim other peers are down
  - poison routing tables of others
  - corrupt transferred data
  - create a high churn rate
  - time out to decrease overall performance
- **Defences**
  - do not rely on only one path or line of inquiry
  - verify peers and data
  - favour long living peers

Sybil attack

- **Create a *lot* of fake peers and join the network**
  - easy to do, if you let a machine masquerade as many
- **Using all these these peers in concert, traffic can be subverted or surveilled**
- **Defences**
  - make joining expensive
  - ensure that paths on the overlay network involve multiple subnets
    - sybils are likely to originate from the same subnet

Eclipse attack

- **Peers are eclipsed by other, malicious peers that insert themselves between good peers and the network**
  - the good peers' contribution to the network is subverted
  - good peers seem to disappear from the network
- **Defences**
  - ensure that a peer cannot freely choose its position on the network
  - have several paths available to the network

Overview

- Aspects of security
- Venues of attack
- **Techniques for anonymity & censorship resistance**
- **Securing a DHT**

Crowds: defeating Web tracking

- **A number of members participate in a crowd, and they are known to each other**
  - if a member, Bob, wishes to retrieve a Web page, Bob sends a request for the URL to a random member, Carol (using symmetric encryption). Carol can then choose to retrieve the Web page or randomly forward the request to another crowd member, Alice, and so on. Eventually a member chooses to retrieve the Web page, and the Web page is returned along the request's path

Mix networks: defeating traffic analysis

- Mix networks are used to ensure that a sender and receiver cannot both be known
- A mix network consists of a number of known *mixers* —routers with asymmetric key pairs

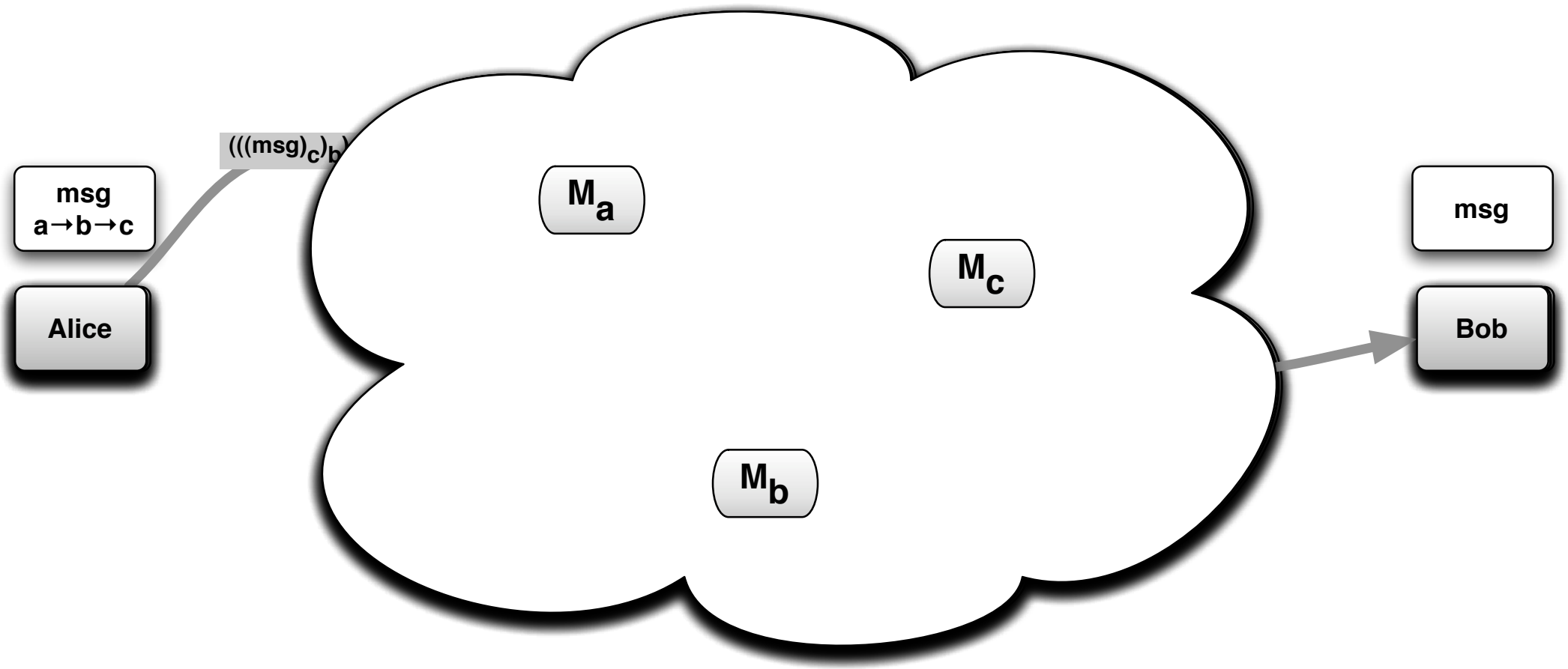
Mix networks: defeating traffic analysis

- A sender chooses a path through the mix network (m_1, ..., m_n), and encrypts the message (with some final destination) with m_n's public key, encrypts this message (with $m_{n-1} \rightarrow m_n$) with m_{n-1}'s public key and so on
- The message is then sent to m_1, who decrypts the message using its private key, and sends it to the next mixer, who repeats the process
- This is also known as *onion routing*

Mix networks: defeating traffic analysis

- Eventually, the message makes it to m_n, who can then forward the message onwards to its final destination
- Only m_1 knows the sender and only m_n knows the receiver and neither knows the route of the message (not even their own position on the path)

Mix networks – an example



Problems with existing mix networks

- **The original mix networks relied on a "cloud" of established, known mixers**
  - thus, easy to block (deny any access to the mixers)
  - a malicious mixer would recognise sender/recipient, if at the edge of the connection
  - cover traffic makes traffic analysis difficult within the cloud, but what about the edges?
  - edge traffic analysis becomes feasible (if expensive)
- **If the message leaving the network is in clear text, it is exposed to the last node on the path**
  - some protocols leave sensitive data in headers (e.g., IP address of sender)
- **Sophisticated alternative found in *Tarzan***

Tarzan

- **Goals**
  - P2P: All participants can mix
  - Robustness against malicious peers
  - Ensured anonymity
  - Look like IP to applications (just a library)
- **Characteristics**
  - P2P network
  - Mimics: generating secure cover traffic

Tarzan is a P2P network

- **Defeating blocking**
  - Tarzan is a scalable P2P network
  - thus, thousands of peers can participate
  - this makes it unfeasible to block everyone suspected of being a mixer
- **Traffic analysis**
  - everybody is a mixer
  - cover traffic among all peers
  - no clear point for edge traffic to analyse

Discovery – joining the network

- A new peer starts by retrieving a peer list from a known peer
- The peer can then ping the other peers (thus validating their IP address), validate their public key, and retrieve their lists
- This process is repeated until the peer is satisfied
- Later, peers gossip among themselves
  - thus, a good coverage of the network is gained over time

Mimics

- **Peers exchange cover traffic**
- **Cover traffic is between validated peers**
- **Cover traffic is**
  - encrypted
  - sent at a uniform data rate (but adjusted when there is real traffic)
  - uniform – all packets are the same size
- **Every peer exchanges mimic traffic with *k* other peers**

Defense against malicious peers

- **A malicious peer could spawn many (virtual) peers to increase its chance of being selected for tunneling**
  - but peers must be validated to be a part, and you cannot fake your IP return address
- **Most likely, a malicious peer will only control a subpart of the IP address space**
  - Tarzan therefore randomly selects between sub-domains of the IP address (spreading the participants over the Internet)

Establishing a secure tunnel

- **The originator iteratively selects peers (across IP domains) towards its target using the mimics of the peers along the route**
  - the originator either already knows the mimics from its own discovery, or can validate them independently
- **Thus, the message is continually under the traffic cover**
- **All exchanges are encrypted**

Through the tunnel

- **The message is NAT'ed (given a private IP address)**
  - the message is covered in encryption layers (one per hop)
- **All traffic is padded and shipped using UDP (and protected by the cover traffic)**
  - forwarded (and stripped) along the tunnel
- **The destination PNAT peer NATs again to public alias address**
  - PNAT contacts the destination service
- **Responses returned similarly**

Characteristics

- **Scalability**
  - Overhead is unavoidable, but looks reasonable – no hotspots or SPoF
  - Though best suited for fairly low bandwidth jobs, if to be hidden behind cover traffic
- **Fairness**
  - Peers are chosen at random, cover traffic is set at a fair pace
- **Integrity and security**
  - Difficult to subvert
- **Anonymity, deniability, censorship resistance**
  - Quite strong

Summary

- Secure if enough peers participate
- P2P: A good case to blur the distinction between clients and servers
- Spans domains to make Sybil attacks difficult
- Dynamically adjusted cover traffic over mimic pairs makes it difficult to analyse traffic
- Neat to provide Tarzan as infrastructure – use the library as you would IP

Freenet

- **Objective**
  - to build a virtual file space across peers that cannot be easily attacked and that provides a high degree of protection against censorship
- **Decentralised architecture**
- **Built-in redundancy – popular files are replicated across the network**
- **High security and plausible deniability – nodes have encrypted file spaces**
  - have found use in mainland China where censorship is real

Freenet

- No authentication (to real world identities) as such, but can authenticate pseudonyms, allowing e.g., only the original author to update a document
- Each resource in a Freenet node space is encrypted and integrity checked with SHA-1 hash
- Network traffic is encrypted link to link
- Routing is performed in a way to foil surveillance

Characteristics

- **Globally Unique Identifiers (GUIDs) are crucial in Freenet – these are SHA-1 hashes (160 bit)**
  - Content-hash keys (CHK) : Hashes calculated over files inserted into Freenet
  - signed-subspace keys (SSK): Hashes calculated from a **public key** and a **textual description**. The signified file is signed with the private key and can therefore only be modified by the owner. These ("indirect") files are intended to contain directory listings with GUIDs on other files
- **To participate in Freenet, a node must dedicate some disk space**

Architecture

- **Freenet nodes know only their immediate neighbors**
  - traffic may have originated from the neighbor, or the neighbor might only be passing it on
  - this makes it difficult to pinpoint whence a file originated
  - this also means that files get transferred over a number of nodes before reaching the destination
    - ...which might be bad for performance
- **Nodes maintains a table of known GUIDs and the peers thought to hold the associated resource (maybe itself)**

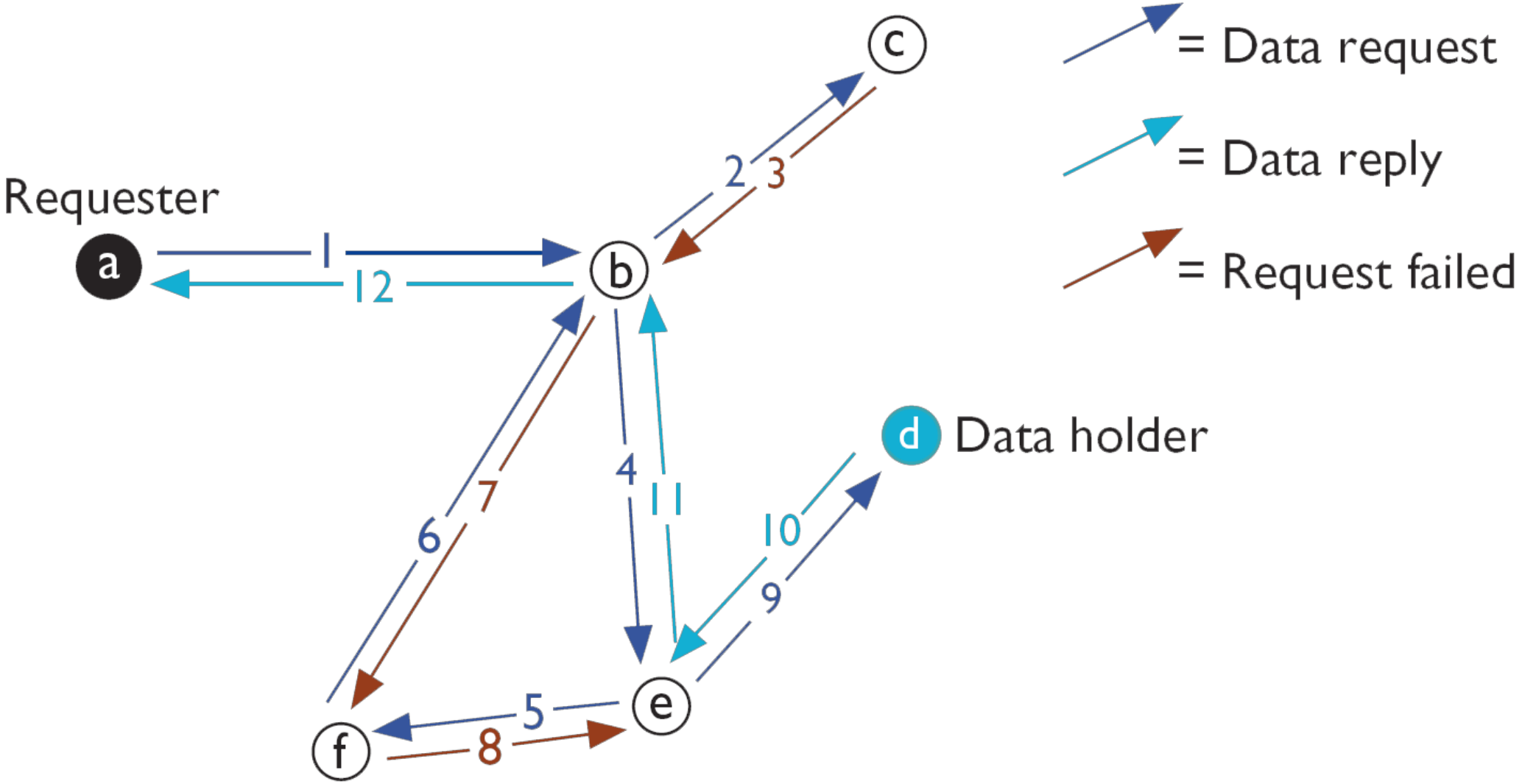
Requesting a file

- A user knows (somehow) the GUID (and key) of a desired resource
- This query is checked against the local node's store. If not found, the query is forwarded to the known peer with the closest GUID, and this process is repeated until the resource is located or TTL runs out
- If the resource is located, it is *returned by the same route* to the originator (who is the only one who *knows* it is the originator). Along the route back, nodes stores the GUID and location, and may even cache the resource

Requesting a file – security measures

- **Along the way, peers may alter the message by setting themselves as the data holder and possibly caching it**
  - to thwart attacks against a data holder
- **Peers may also alter the value of TTL**
  - to thwart analysis of TTL
- **Thus, popular resources and their GUIDs are replicated across the network**
  - this makes DoS attacks of resources self defeating

Requesting a file



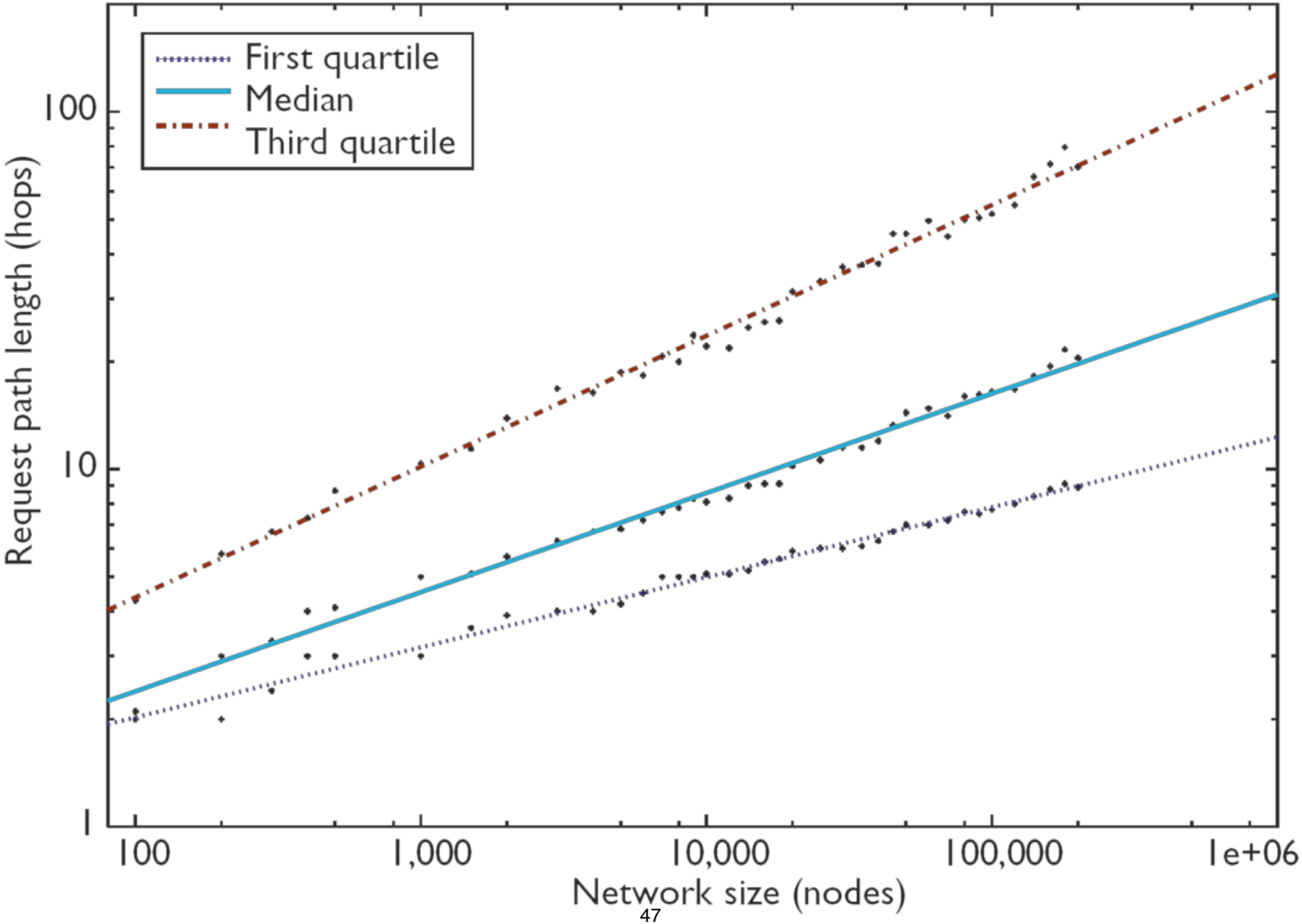
Storing a file on Freenet

- **The originator hashes the resource and sends the GUID out on the network with a TTL**
  - Other nodes check the GUID for uniqueness and forwards it to the nearest (in ID space) neighbor until TTL runs out. The final peer sends 'all clear' following the route back to the originator
- **The originator can now publish the file. It is verified at each peer along the route, routing tables are updated, copies are cached, and the file ends up at the final peer on the route**
- **Unpopular files will eventually be reclaimed by the system to make room for more popular files**

Joining Freenet

- **A new node joins Freenet by making an announcement (containing a public key, an IP address and TTL) to a (somehow) known node.**
  - The nodes forward the announcement randomly until TTL and these nodes generate a GUID in concert for the new node
  - The GUID is then the responsibility of the new node and requests close to the GUID are forwarded to the node
- **As inserts and requests matching the GUID of the new peer are directed towards it, it will gradually learn its delegated part of the key space**

Search performance



Experiences

- Searching is so far somewhat missing – this is handled elsewhere (and this, of course, presents an excellent target for censorship)
- Resources are encouraged to be encrypted by the creator, allowing readers (who know the key) to decrypt it. (How are these keys safely distributed?)
- The safety of the system means that resources may travel some distance before reaching their destination. OTOH replication of resources and updates in routing tables improves performance

Characteristics

- **Scalability**
  - Simulations look good (caching would be expected to help), but in use Freenet is reportedly fairly slow
- **Fairness**
  - Caching will relieve overworked peers – peers will accumulate and serve data over time
- **Integrity and security**
  - The SHA-1 should keep files intact (though not any more)
- **Anonymity, deniability, censorship resistance**
  - High marks – though only as long as there is a safe method of distributing the keys

Overview

- Aspects of security
- Venues of attack
- Techniques for anonymity & censorship resistance
- **Securing a DHT**

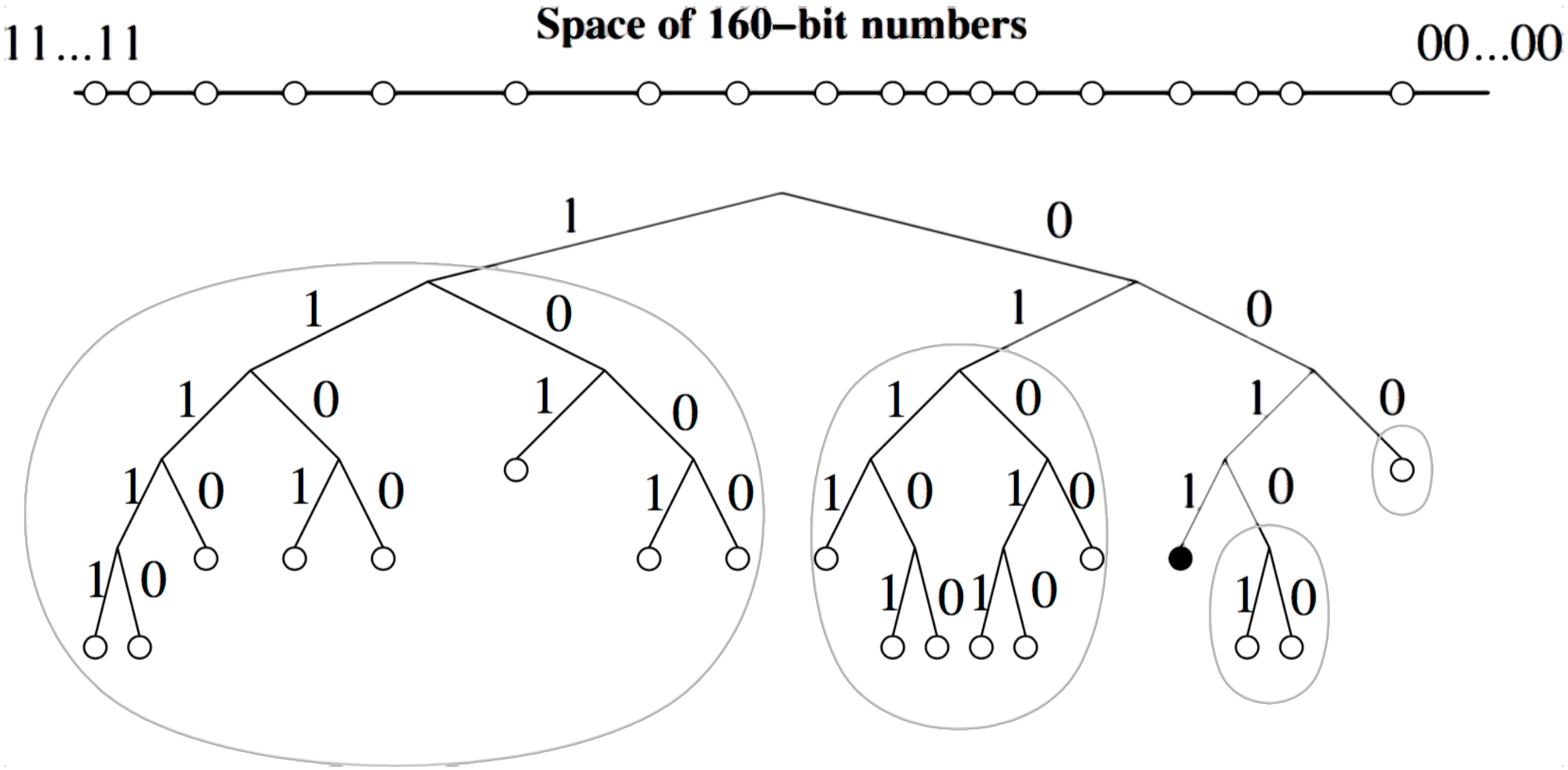
Are DHTs secure?

- **Structured P2P networks may well seem vulnerable**
  - deterministic routing mechanism
  - crucial routing information kept at peers
  - peer ID determines position in network
  - values kept at peer with closest key

Aspects of Kademlia

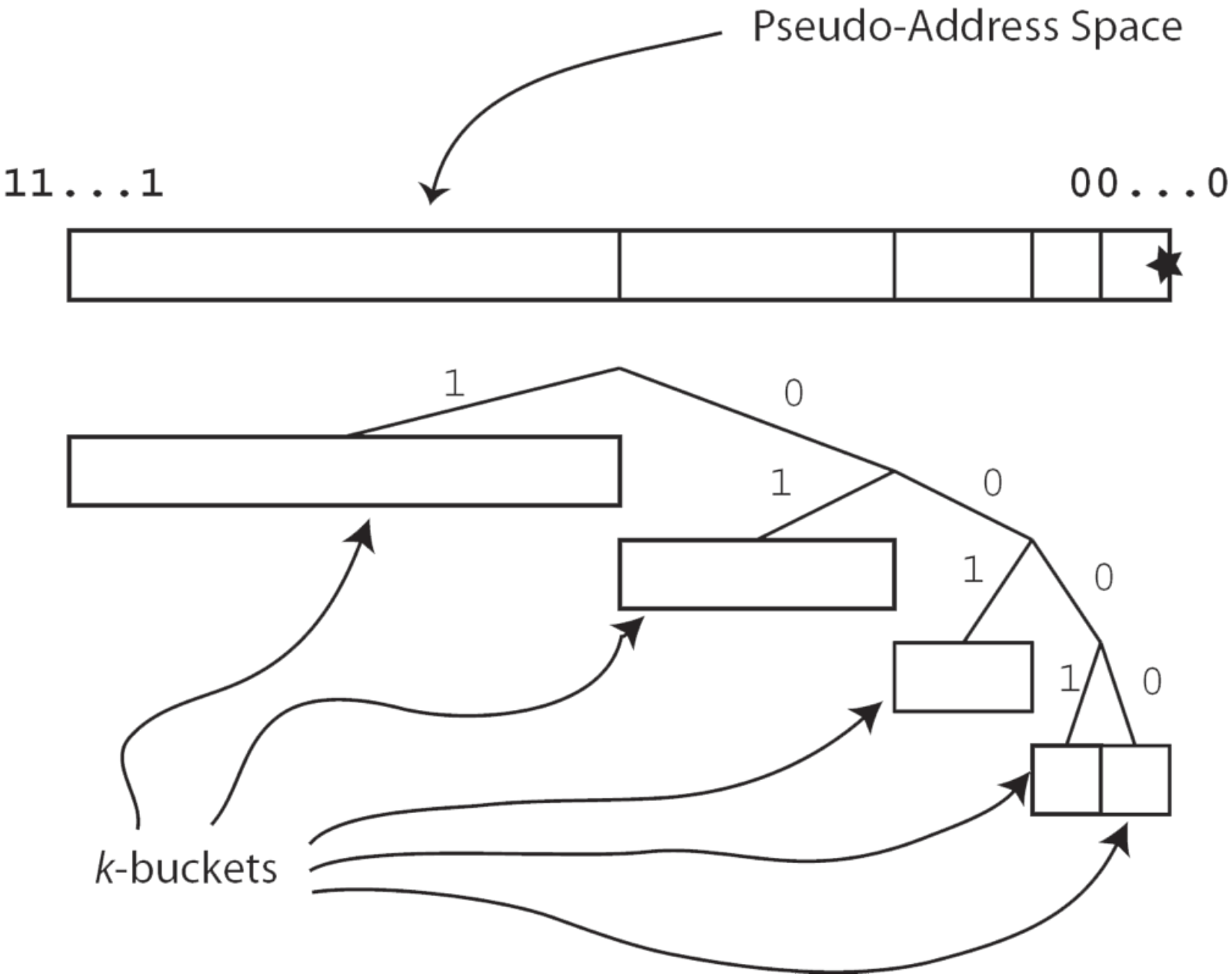
- **All IDs are 160 bits long, random or found with SHA-1**
  - i.e., uniform distribution, etc
- **To navigate this key space, Kademlia uses XOR**
  - $d(X, Y) = X \text{ XOR } Y$; $d(X, Y) = d(Y, X)$
  - intuition: higher order difference = longer distance
- **A Kademlia routing table stores 160 k-buckets**
  - the i^{th} k-bucket contains nodes within a XOR distance of 2^i to 2^{i+1} from itself (so the i^{th} bit is significant)
  - up to k nodes in each bucket, ordered by liveness (most recently seen at tail)
    - thus, once again, more complete knowledge of 'close' peers, but still knowledge about the rest of the world

Kademlia routing table

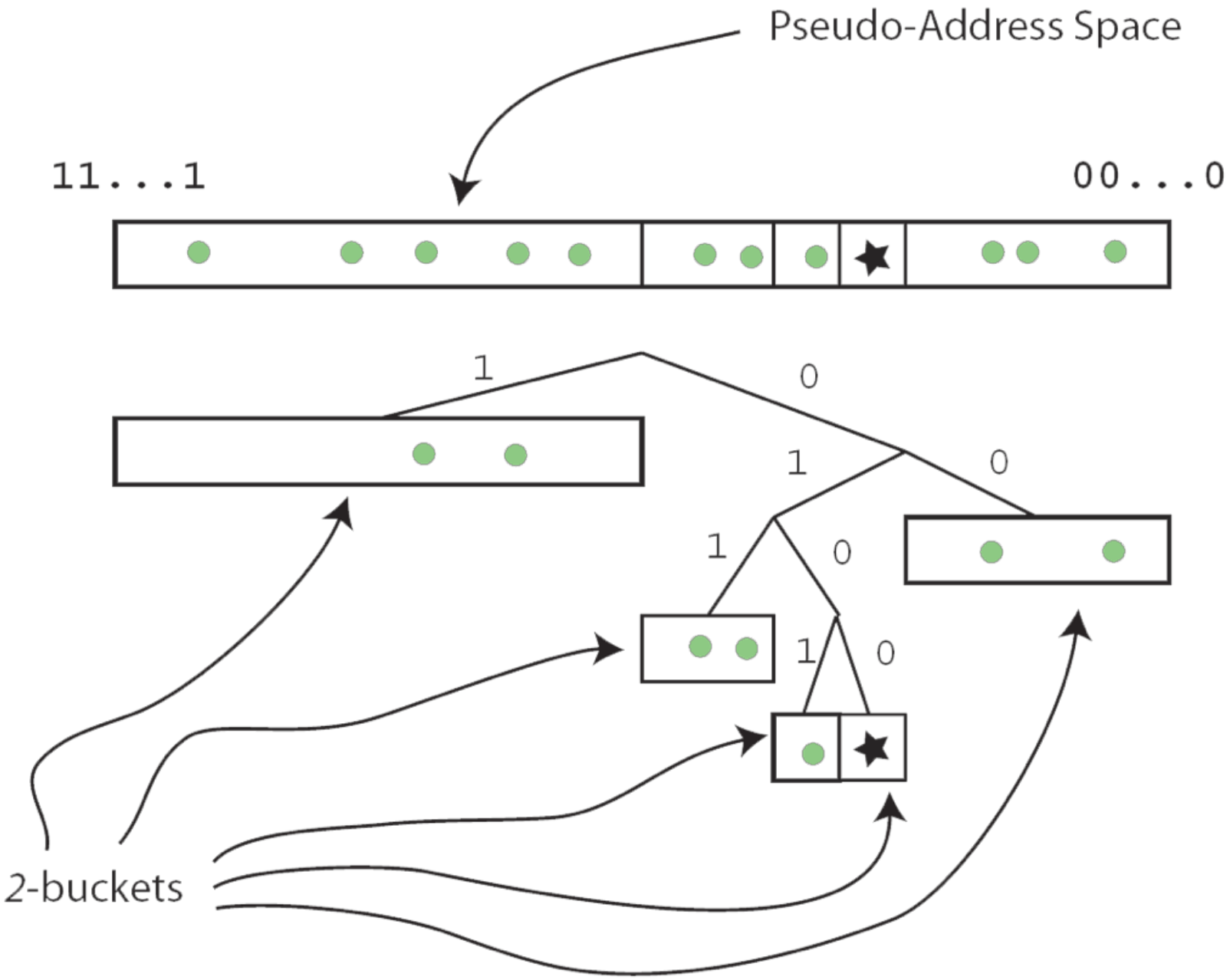


- **Peer 0011 (•) must know some peers in the highlighted groups — all different prefixes to itself**

Kademlia routing table



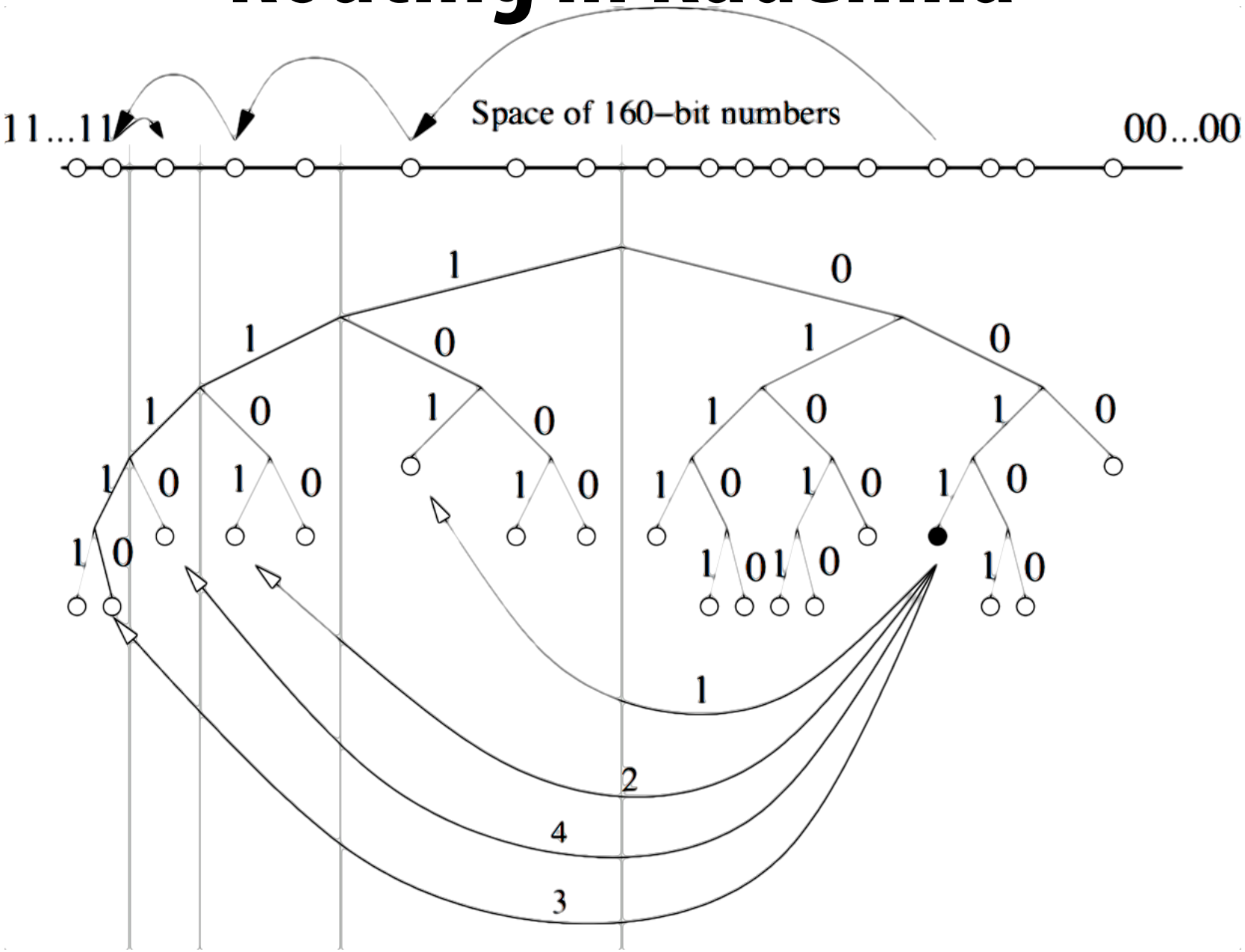
Kademlia routing table



Locating a destination

- Given a destination, use the (XOR) distance from ourselves to find the matching k-bucket
- Contact nodes in that k-bucket to get even closer nodes
  - if there are not enough nodes in the bucket, use the nearest
- Repeat until the k closest nodes have been found

Routing in Kademlia



- Reaching 1110 from 0011. 0011 knows initially 101

Operations in Kademlia

- **PING**
- **STORE**
- **FIND_NODE**
- **FIND_VALUE**

FIND_NODE

- **FIND_NODE$_n$(id)**
  - returns the k closest nodes to an ID that n knows
- **Iterative process:**
  - n_0 = origin
  - N_1 = FIND_NODE$_{n_0}$(ID)
  - N_2 = FIND_NODE$_{n_1}$(ID)
  - …
  - N_m = FIND_NODE$_{n_{m-1}}$(ID)
- **The node can choose any peer among the returned k nodes for the next step**
- **Lookup terminates when k closest nodes have responded**

FIND_VALUE

- **FIND_VALUE$_n$(key)**
  - works like FIND_NODE, unless n knows the value in which case the value is returned
  - if one of the k closest nodes does not have the value, the requester will store it there

Maintaining routing tables

- **Upon communication with another node**
  - Check the appropriate k-bucket
    - if already there, move to tail
    - if there is room, insert at tail
    - if new, and least recently seen node is unresponsive, replace with new node (and move to tail)
    - else: ignore node
  - Thus, the routing tables are populated, and old, active nodes are given preferential treatment
  - Implementation optimization: keep new peers in cache replacement list; replace only member of k-bucket if unresponsive during normal operations

Maintaining routing tables

- **Why prefer old nodes?**
  - Studies show that the longer a peer *stays* online, the higher the probability is that it will *remain* online
  - Makes it difficult to flood the network with bogus peers
- **As SHA-1 is uniform, a Kademlia node will receive messages from nodes with IDs uniformly distributed across the key space**
  - Thus, all traffic is valuable and increases knowledge

Parallelism in Kademlia

- At each step in the lookup process, `FIND_NODE`/`FIND_VALUE` queries α nodes in parallel
- The node can then choose the quickest peer and move on
- Ensures locality and takes advantages of the strongest peers
- The system does not have to wait until a node times out as with other systems
  - this makes, e.g., a *slowloris* attack infeasible

Redundancy in Kademlia

- Each (key, value) pair is republished every hour and stored at k locations close to the key
- (key, value) expires after 24 hours, so old data is flushed
- But, original publisher republishes (key, value) every 24 hour, so valuable information is maintained
- Whenever a peer A observes a new peer B with an ID closer to some of A's keys, A will replicate these keys to B

Joining the network

- **Bootstrapping**
  - compute an ID
  - (somehow) locate a peer in the network
  - add that peer to the appropriate k-bucket
  - find neighbours by doing `FIND_NODE` on own ID
  - populate the other k-buckets by performing `FIND_NODE` on random IDs within those buckets
- **This process (due to the reflected nature of Kademlia) ensures that the new peer is known across the network**

Failure in Kademlia

- Unlikely: Routing tables are continually refreshed due to ordinary traffic
- As SHA-1 is uniform, the k-buckets will be evenly updated
- If there is no traffic, a peer will regularly explicitly refresh oldest k-bucket
- Parallelism in queries ensures that a failing peer is
  - detected
  - routed around

Kademlia

- **Most popular DHT ⇒ biggest target for attacks**
- **Weaknesses**
  - deterministic routing along converging path
  - sybils can saturate the network with malicious peers
  - eclipse peers can collude to produce poor routing
- **Strengths**
  - prefers long living peers, so churn attacks are inefficient
  - routing information is continually refreshed — no specific operation to target

S/Kademlia

- **All peers have public/private keys**
- **Securing Kademlia through**
  - expensive NodeId generation
  - sibling broadcast
  - routing over disjoint paths
  - verifiable messages using public/private keys

Secure Node Identifiers

- Sybils rely on cheap/home-made/unverifiable NodeId generation
- Ids created as public key hashes
- Weak signatures on (IP, port, timestamp)
  - `PING`, `FIND_NODE`
- Strong signatures on whole messages
  - man in the middle made difficult
  - message contains nonce, so replay is impossible

Generating Ids for S/Kademlia

- **Central authority**
  - can co-sign peers' certificates
  - can control/limit the growth of sybils
  - but, centralised/SPoF
- **Crypto-puzzles**
  - no central authority, but computationally expensive
  - given a crypto hash function H (e.g., SHA1, SHA256, etc.) and $\oplus$=XOR
  - static: Generate key so that c_1 first bits of H(H(key)) = 0
    - NodeId = H(key) (*so Nodeld cannot be chosen freely*)
  - dynamic: Generate X so that c_2 first bits of H(key $\oplus$ X) = 0
    - increase c_2 over time to *keep Nodeld generation expensive*
  - verification is O(1) — creation is $O(2^{c1} + 2^{c2})$

Sibling broadcast

- **Standard Kademlia uses**
  - *k* buckets, *k* redundant copies of key/values *(siblings)*
- **The number of redundant copies increases integrity**
  - but marries network connectivity (*k*-bucket) to redundancy (*k* copies)
- **S/Kademlia adds**
  - *s* redundant copies of key/values
  - sibling lists of a size to ensure that a peer knows *s* siblings with high probability
    - similar to leaf sets from Pastry

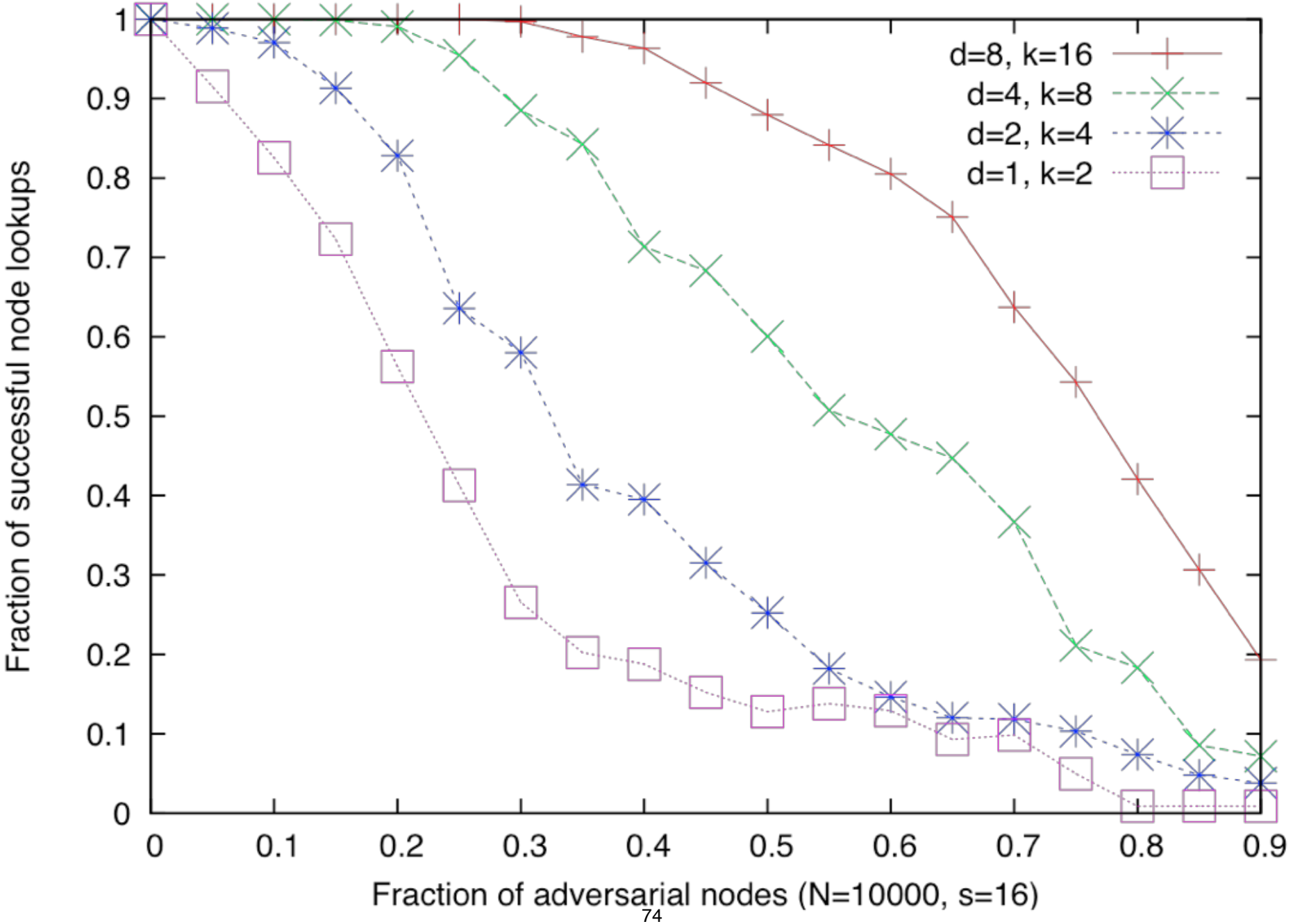
Populating the k-buckets

- **Actively valid nodeIds:**
  - signed, responses to RPCs
  - added if there is room (as usual in Kademlia)
- **Valid nodeIds**
  - signed
  - only added if the prefix is sufficiently different from the peer's own
    - makes a targeted attack more difficult
- **Unsigned nodeIds**
  - ignored

Querying in S/Kademlia

- **We need to ensure that a malicious peer cannot steer the query into a territory of malicious peers**
  - ordinary Kademlia queries use a single list of nodes, refined over queries. Malicious peers could drown out the good results in this single list
- **S/Kademlia issues queries over *d* paths, that are kept disjoint, and where every peer is queried only once**
- **This increases the odds for not all searches going into malicious territories**

Results


d=8, k=16
d=4, k=8
d=2, k=4
d=1, k=2
Fraction of successful node lookups
Fraction of adversarial nodes (N=10000, s=16)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Results

- **Making attacks harder (not impossible) by**
  - limiting NodeId generation with crypto-puzzles
  - accepting only signed NodeIds into k-buckets
  - distributing queries across a wider set of the network
- **Unfortunately at the cost of having good peers solve crypto-puzzles**

Characteristics

- **Scalability**
  - nearly as scalable as Kademlia — signing is an overhead, but network messages are small
- **Fairness**
  - as fair as Kademlia, and if you don't sign, you are ignored
- **Integrity and security**
  - malicious peers are less likely to subvert the network
- **Anonymity, deniability, censorship resistance**
  - not easy to subvert routing in order to suppress key/values

Conclusions

- **Reputation and trust on the Internet is *hard***
- **A number of good techniques exist – often based on a central authority**
  - but can you trust the authorities?
- **P2P makes everything *worse***
  - no central authority makes designs challenging
- **P2P can make many things *better***
  - by making it difficult for the central authority to eavesdrop