Xor-overlay Topology Management Beyond Kademlia

Erick Lavoie, Laurie Hendren
McGill University, Montreal, Canada,
Email: erick.lavoie@mail.mcgill.ca
Email: hendren@cs.mcgill.ca

Miguel Correia
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Lisboa, Portugal
Email: miguel.p.correia@tecnico.ulisboa.pt

***Abstract*—Kademlia is a widely successful Distributed Hash Table (DHT) implementation, better known for its use in the BitTorrent protocol. In this paper, we revisit its overlay management separately from the DHT operations to allow it to be used with other distributed abstractions by providing an accurate and consistent view of the k-closest nodes to any given identifier. We then present invariants that avoid the irregularities that are necessary with the original k-bucket design and improve on later published solutions by not needing an additional parameter to tune. We then provide a specification of a distributed abstraction that provides the automatic management of routing tables using our invariants and a pseudo-code implementation of that abstraction. We finally sketch how our abstraction may be used for a security mechanism and how it could be combined with other known distributed abstractions to provide other services than file sharing in xor-based overlay networks enabling xor-based network overlays to go beyond their original Kademlia design.**

1. Introduction

Some self-organizing peer-to-peer overlay networks such as BitTorrent are built with Kademlia [1], a widely successful Distributed Hash Table (DHT) implementation. At the heart of Kademlia lies an elegant overlay network that organizes its peers using the xor distance and is simple and efficient in the number of messages it uses. Its efficiency comes from combining overlay management and DHT operations to update the topology as a side-effect of the latter. Its design makes the overlay management efficient for exchanging immutable data, such as torrents (files) in BitTorrent. It is easy to use in other applications that require only weak consistency amongst replicas[1].

The distributed system field has produced many abstractions (and algorithms to implement them) for reliably broadcasting messages, implementing shared registers or mutable files, and establishing consensus, for example. These abstractions are mature and can be studied and implemented from a textbook such as the one by Cachin et al. [2]. These abstractions require a known group of nodes in which individuals may fail as the algorithm progresses. A self-organizing peer-to-peer overlay network can potentially enable multiple copies of these abstractions to be run in parallel on disjoint subsets of nodes and to perform automatic load-balancing amongst them in the same way the ownership of the keys in a DHT is spread between the participating nodes. However, the original Kademlia implementation does not guarantee an *accurate*[2] and *consistent*[3] view of the nodes that are part of the group used for replicating items, e.g. the k-closest nodes to a target identifier [3], [4]. It only ensures that enough of them execute correctly long enough without crashing or disconnecting to maintain at least one copy available for retrieval (and the regeneration of replicas every hour). It does not define properties for defining a group in such a way that they all agree on the group composition (even eventually). That therefore precludes the easy implementation of existing distributed abstractions on top of those groups.

In this paper, we revisit the overlay management separately from the DHT operations to enable its usage with other existing distributed abstractions, and additionally make it easier to understand separately of the original DHT application domain. Our key contribution is the development of new invariants on routing tables' subtrees (k-buckets) that enable all nodes in the network to obtain an accurate and consistent view of the region of the identifier space that is close to a given identifier, such as their own, independently of the particular distribution of identifiers. Our invariants avoid the irregularities which are a result of the original k-bucket design and introduce no additional parameter to tune.

Then, we provide a specification of a Xor-overlay Topology Manager (XTM) distributed abstraction that manages automatically the routing tables using the new invariants, and a pseudo-code implementation of that abstraction. The abstraction provides a list of peers that can be used with the original lookup algorithm. Therefore, the abstraction and implementation could be used for pedagogical purposes to introduce the underlying xor-based topology management

1. If the same key is updated with a different value, it provides no guarantee that later retrievals will only see the newer value. Because file sharing applications usually distribute immutable files, the problem does not happen for them.

2. *Def.* The actual k-closest nodes to a target identifier are all found.

3. *Def.* All nodes within the close region of the target identifier compute the same k-closest nodes to a target identifier using their routing table.

operations of Kademlia before explaining how they are combined with DHT operations, providing a layered presentation of the protocol. They may also serve as a reference for language-specific implementations.

Finally, the improved accuracy and consistency of the computation of the k-closest nodes enables new applications. We sketch how the insights of the paper enable the implementation of (1) the filtering of messages that are going to the wrong destination, (2) a leader-based chat room that provides ordering of messages with replication in case the leader fails, and (3) a quorum-based uniform reliable broadcast primitive than can be used to build decentralized storage.

To summarize, in this paper we make the following contributions:

- we provide additional insights about the original design that were not explained or emphasized in the original paper, in particular how the file sharing domain that was targeted could minimize the number of messages and how the lookup algorithm leverages complementary unordered and ordered views of the identifier space (xor-space);
- we derive new invariants on routing tables to eliminate the irregularities in the original k-bucket design and provide accurate and consistent computation of the k-closest nodes;
- we provide the specification of a Xor-overlay Topology Manager (XTM) that maintains the topology, independently of the original DHT operations;
- we provide a pseudo-code implementation of the XTM abstraction in a crash-stop partially synchronous distributed system;
- we sketch how the XTM can be used to improve the security of existing systems and to build peer-to-peer services other than file-sharing by combining it with other well-known distributed algorithms.

The remainder of the paper is structured as follows. In Section 2, we revisit and summarize the key properties of Kademlia and make explicit some implicit design decisions of the original paper. In Section 3, we revisit the lookup algorithm of Kademlia and explain how it leverages different views of the identifier space during its different phases of operation. In Section 4, we present our new invariants. In Section 5, we present the specification and implementation of a distributed abstraction that manages the topology in a xor-based overlay using our invariants. In Section 6, we sketch a new security mechanism and the application of existing distributed algorithms to a xor-based overlay. Finally, we present related work in Section 7 and conclude in Section 8.

2. Kademlia: Distributed Hash Table With Minimum Management Overhead

In this section, we list the key properties of Kademlia [1] in a concise manner for those who are not familiar with the protocol already. We start with the xor distance and a presentation of the subtrees that underlie the routing tables. We then present the key properties of the protocol by separating those that derive from the topology from those that are required by the application (Distributed Hash Table). Moreover we present separately those that were introduced for fault tolerance and those for optimizing the number of messages. A complete presentation can be found in the original paper [1]. Those familiar with the protocol may skip directly to Section 2.6. In that section, we explicit some design decisions that were not explained in the original paper that then motivate the rest of the paper.

2.1. Identifiers, Xor Distance, and Subtrees

Every node in Kademlia has an identifier (X) chosen randomly and composed of b bits. The distributed hash table (DHT) keys are also composed of b bits and represent the identifiers of the $(key, value)$ pairs. The pairs are stored in the k nodes with the smallest distance to the key. The distance is computed by performing a bitwise exclusive-or (xor or $\oplus$) between identifiers and keys, and interpreting the distance as an integer. Formally:

$$\begin{aligned} x_i =& \quad \{0,1\} \\ X =& \quad (x_1, x_2, ..., x_b) \\ x_i \oplus y_i =& \quad 0 \text{ if } x_i = y_i, 1 \text{ otherwise} \\ d(X,Y) =& \quad \textstyle\sum_1^b 2^{i-1}(x_i \oplus y_i) \end{aligned}$$

The entire identifier space comprises 2^b identifiers with values between 000...0 and 111...1 which can be organized in a binary tree in which an identifier belongs to the left subtree if the leading bit is 0, or the right subtree if the leading bit is 1, and recursively for the following bits.

In order to scale while requiring each node to only know about a manageable number of peers, each node keeps track of a bounded number of peers in each subtree with a prefix that differs by one bit from its identifier. This is sufficient to ensure full connectivity (see Chapter 2 in Cai's master thesis [5]). The subtrees are created and populated as peers are discovered, by splitting the node's subtree (initially at the root of the binary tree and covering the entire identifier space).

The binary tree and subtrees for routing purposes are illustrated in Figure 1. We now list some of the key properties and ideas of Kademlia. They are regrouped by topic to make them easier to consider separately. Each topic has its own section and each property is numbered. To refer to a given property in the text, we write *Topic Section Number - Property Number* (ex: 2.2-1).

2.2. Topology (Routing Tables and Peer Lookup)

The properties listed hereafter are related to the finding and management of peers by each node. The list of peers a node maintains is alternatively called routing tables or topology in different papers. We present the original properties

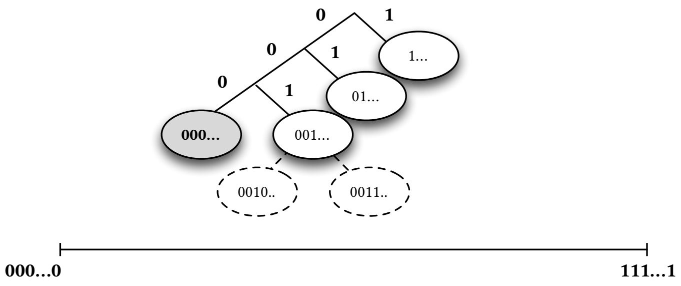

Figure 1. Subtrees (K-buckets) in Kademlia, grouping sets of identifiers from the entire identifier space (shown with a line that goes from 000...0 to 111...1). The grayed subtree represents the node 000...0's subtree and all others are subtrees containing peers that make up the routing table. The closest neighboring subtree is split also (shown with dotted circles) if there are more than k peers to ensure all peers that are close in the identifier space are known (2.2-5).

to contrast them with the new invariants we will introduce in Section 4:

1) Every node knows of at least one node in each of its subtrees (k-bucket), if that subtree contains a node. This is necessary and sufficient to ensure convergence of the lookup procedure;
2) A subtree contains up to k peers, others are ignored. k is chosen so the probability that all peers failing within a one-hour period is sufficiently small to ensure 2.2-1 stays true;
3) The current state of a slice of the network is obtained by performing a lookup for the closest nodes to a given identifier. It provides the k closest nodes that represent a fix-point amongst all the nodes that were contacted in a converging path towards a target identifier (the k closest nodes computed from the routing tables they return are the k closest nodes themselves). We write k-closest nodes as a shorthand for the result of performing a lookup for the k closest nodes. Empirical evidence suggests that the lookup is not completely accurate, i.e. it fails to correctly locate some of the actual k closest nodes [3]. As a side-effect of finding the k-closest nodes, all other nodes discovered along the path are used to populate or refresh the subtrees of the topology (routing tables). Symmetrically, the nodes queried during the lookup will populate or refresh their k-buckets with the information of the node performing it (if they have space in their routing table for it);
4) To join the network, a node lookups its own ID (implicitly making itself known because of 2.2-3);
5) Irregularity: the closest neighboring subtree (to the node ID's subtree) is recursively split to contain all neighbors (shown with dotted circles in Figure 1), which contradicts 2.2-2. We explain why it was needed in Section 4.

2.3. Distributed Hash Table (DHT) Operations

A distributed hash table implements lookup, storing, and deletion of $(key, value)$ pairs by key. We explain here how those operations are built using peers obtained from the Topology (Section 2.2):

1) Storing an immutable value is performed on the k-closest nodes, after finding them using the lookup procedure. As for 2.2-2, k is chosen so that the probability of all k nodes failing within one hour is small. Storing a value in the k-closest nodes provides automatic load-balancing under the assumption that the keys are uniformly distributed throughout the identifier space;
2) Retrieving an immutable value is performed on the first of the k-closest nodes that answers. This is possible because the integrity of the data retrieved is guaranteed by other means (ex: its key is the hash of the value) and the same data[4] should therefore be on any of the k-closest nodes;
3) Deleting a value is done by stopping its republication (see 2.4-1).

2.4. Fault Tolerance

Throughout the design k nodes are used to have multiple replicas or paths available to ensure the system stays available even in the presence of failure or sudden disconnection of individual nodes:

1) Replicas of key-value pairs may disappear (because nodes fail) or end up in nodes that are not in the k-closest (because new nodes closer to the ID have joined). Therefore, to maintain k copies in the k-closest nodes, the pairs are republished every hour by one of the k-closest nodes to all the other $k-1$;
2) K-buckets implement a least-recently seen eviction policy, except that live nodes are never removed. Old nodes tend to stay longer and the scheme resists massive joining attacks therefore providing stability.

2.5. Topology Management and Optimizations

As much as possible, topology management piggybacks as a side-effect of the lookup operation used by DHT operations to minimize extra configuration messages. However some situations cannot be managed as a side-effect of it and others can be optimized to reduce the number of messages:

1) If a key is not looked up, the routing information will never be updated. Therefore when some part of the identifier space covered by the corresponding subtree has not been looked up for an hour, a node will randomly look for an ID within that range to refresh its information about its peers within that region;

4. Similar data if weakly consistent mutable data is allowed.

2) When one of the k-closest nodes republishes a key, all other $k-1$ closest nodes will not republish it for the next hour by noticing it comes from another k-closest node;
3) When republishing all its pairs, a node first refreshes all the k-buckets in a subtree before republishing all the pairs that belong to that subtree so the lookup is amortized over all store operations.

2.6. Implicit Design Decisions

The original paper left implicit some design decisions that were made to leverage opportunities that come from the file sharing application domain to simplify the design and reduce the number of messages exchanged. We make them explicit here to better understand the limitations of the design:

1) *Only the node performing a lookup obtains an accurate view of the state of (a slice of) the network and that view is valid for a limited but unspecified amount of time*. The information is obtained as late as possible and as much as possible as part of a DHT operation. This minimizes the number of maintenance messages that nodes have to send between each other to maintain the overlay;
2) *Only weakly consistent or immutable data is stored*. If storing mutable data with strong consistency was supported, additional messages would be needed to keep the copies synchronized. Said differently, if a key is updated with a different value it is not specified whether the old value might be retrieved again. The caching mechanism (explained in the original paper [1]) certainly suggests it may;
3) *Eventual consistency of the network and DHT is sufficient.* A node may store a new value not exactly on the k-closest nodes without preventing correct retrieval of the value in most cases. Its replication still guarantees fault tolerance and a lookup will converge to the values anyway (probabilistically). Failure of peers in the routing table need not be resolved in a timely fashion, it can be handled only when a new peer from the same subtree has announced itself through a lookup (2.2.3) or no node from the same subtree has performed a lookup in the last hour (2.5.1);

3. Two Views of the Identifier Space

In this section, we revisit the lookup protocol of Kademlia to provide an additional explanation of why it works. This explanation will make the invariants we introduce in Section 4 easier to understand.

The main purpose of the lookup protocol is to compute the k-closest nodes to a given identifier. If all nodes knew about all their peers, the procedure would be simple: a node would simply have to sort them according to the xor distance to the target identifier. However, in a network containing millions of nodes in which nodes are constantly joining and leaving, it is not practical for a node to maintain accurate information on all the nodes.

Therefore a scalable scheme is used. Each node maintains information about a *slice* of the entire network keeping track of 1 to k nodes in each of its subtrees (if it contains at least one node, Section 2.1). At each step of the lookup procedure, the closest nodes found so far answer with their own k-closest peers, until a fix-point is reached (the k-closest nodes return themselves). As explained in the original paper, the reason the lookup procedure converges in a logarithmic number of steps (compared to the number of nodes in the network) is because if each node knows at least one node in each of its subtrees, then at each step, the nodes returned are at least one-bit closer to the target identifier [1].

A key insight behind the scheme, which is not emphasized in the original paper, is that for convergence of the lookup procedure it does not matter which nodes of the subtrees are remembered in the routing tables. Any k nodes would do. The original paper chooses to prioritize the longest running nodes under the assumption that their probability of failure is lower the longer a node has been running without crashing [1]. However, it may happen that a long running network may have all nodes use most of the same oldest nodes in their routing tables. This may in turn make the network strongly dependent on their continued reliability and otherwise fragile if they eventually fail. But other schemes are possible because the k nodes' identifiers in a subtree *need only to belong to the set of identifiers that correspond to the region of the identifier space covered by that subtree*. Their actual ordering according to the target identifier (or the originating node) does not influence the worst case behavior of the lookup procedure. This is an *unordered view of a region of the identifier space and only requires knowledge of some nodes in the region of the identifier space that is covered by the corresponding subtree*.

This insight may be used to provide different properties in other circumstances. For example, if all nodes were equally reliable, a different scheme could be used in which the k-closest peers to a node's id are kept instead. This would ensure a uniform distribution of links between peers through the entire network. Alternatively, the peers with the longest matching IP-address prefix can be preferred to minimize latency [6].

However, once the lookup procedure converges towards a region that is close to the target identifier, which we will define more precisely in the next section, the ordering of identifiers according to the target *does matter*. And if some nodes in that close region are not known then the computation of the k-closest nodes will be incorrect. Therefore the correct computation of the k-closest nodes requires *an ordered view of the close region to a target identifier and complete knowledge of all the nodes in the region that is close to the target identifier*.

Both views are therefore complementary. In the next section we use this insight to derive different invariants on Kademlia subtrees.

4. New Invariants for Accurate and Consistent Computation of the K-Closest Nodes

The presence of an irregularity in the construction of subtrees (Property 2.2-5 mentioned previously), i.e. the splitting scheme of k-buckets is not applied uniformly throughout the entire identifier space, should arise suspicions about the organization principle that is used. Section 2.5 of the original paper [1] mentions that "[nodes] ensure they have complete knowledge of a surrounding subtree with at least k nodes" while seemingly contradicting the statement in Section 2.2 by writing that a k-bucket (subtree) will grow to up to size k. In this section, we propose different invariants that remove the irregularities by making the intuition that justified their introduction explicit.

Remember that the goal of the lookup procedure is to correctly compute the k-closest nodes to a given target identifier by contacting only a slice of all the nodes in the network. Moreover, as explained in the previous section and in Section 2.5 of the original paper, complete knowledge of all nodes in the region close to the target identifier is necessary for that computation. However, a node may not know by itself if it is one of the k-closest to the target identifier. *It is only in relationship to the other nodes present that it may compute so.* This is why the node that is performing a lookup is gathering information from different nodes and in the process acquires a more accurate view of the relevant slice of the entire network (2.6-1).

We realized that while the original Kademlia invariants on routing tables are sufficient to ensure good enough convergence of the lookup procedure in most cases, they are insufficient (or insufficiently specified) to ensure the correct computation of the k-closest nodes under any possible distribution of node identifiers.

Our new invariants are based on the following insight. A node may be definitely sure that *it is not part of the k-closest nodes to any identifier within a region of the identifier space covered by a subtree if it knows at least k other nodes with identifiers that belong to that subtree and are closer than its own.*

Reformulating the same invariant from a different perspective, an identifier is k-closest to any k nodes within a given subtree (that also contains the identifier) than to nodes in a disjoint subtree (not a parent).

This insight leads to the following new invariants for splitting and merging subtrees (k-buckets) in the routing table. Initially, the node is in the root subtree. The subtree is split if and only if the two resulting subtrees each have at least k nodes in them. Otherwise, a node keeps all contacts within the subtree. The subtree that contains the node represents the *close region* and all other subtrees represent *far regions*. If after a crash was detected a subtree contains less than k nodes, it is merged with the nearest sibling subtree (and all subtrees it may contain) to ensure all subtrees have at least k nodes at all time.

These invariants ensure that if the network is stable, all nodes will eventually have a consistent view of the close region because they will all know about all other nodes in it. Moreover, regardless of the actual distribution of nodes in the space, nodes will always compute the k-closest nodes correctly in the close region, and know for sure that they are not one of the k-closest in the far region.

The actual size of the close region and therefore the number of nodes that are kept in a subtree depend on the actual distribution of nodes' identifiers throughout the entire identifier space. For example, a non-uniform distribution with less than k nodes in one of the root's subtree would force the close region to cover the entire identifier space. However, the probability of that happening in practice with identifiers generated with a uniform random number generator decreases exponentially with the number of nodes and is therefore not a problem in practice even with a small number of nodes (ex: a hundred).

We may see now that the original irregularity in the original paper was necessary because the subtree was split too early. The irregularity was introduced to ensure complete knowledge of all nodes in the close region but was insufficiently precise about how many neighboring subtrees should be split to ensure complete knowledge. Our invariants give a precise definition of the close region in which complete knowledge of all the nodes is required and ensure that all nodes will eventually compute the same close region. Moreover, the list of peers our invariants produce can be used for lookup in the same way the original did. Additionally, a node may actively maintain subtrees to compute the k-closest nodes not only to its own identifier but also to other identifiers in the network (presumably to monitor nodes or specific keys).

5. Xor-Overlay Topology Management

In this section, we present the Xor-Overlay Topology Manager (XTM), a peer-to-peer algorithm for maintaining the overlay topology as nodes join and leave the network. We first give an intuitive overview of the algorithm. We then specify the system model we are using, introduce a formal abstraction with key properties that together show it can be used to compute the k-closest nodes, and finally describe the algorithm. The presentation is inspired by the textbook approach of Cachin and al [2].

5.1. Overview

When joining the network, a new node initializes its close region to the entire identifier space and will therefore keep all other nodes it discovers as its peers. As it learns about enough peers, some regions of the space become far and the node removes extra peers, keeping a fixed number per far region. While keeping only k peers would be enough for correctness, our implementation actually keeps an extra h peers to tolerate less than h peers suddenly crashing[5]. This improves the availability of the network in the presence of peers joining and leaving.

5. The extra h peers delay the splitting of a region, we therefore view them as a hysteresis parameter, hence the usage of the letter h

As the network continues to grow, the far regions will cover more and more of the space and the node's close region will become smaller and smaller. This automatically balances the load across all nodes.

When nodes leave, by crashing or not, they may trigger reconfiguration of other nodes' peers. If the number of peers in a far region drops below $k + h$, a node obtains new peers to maintain knowledge of $k+h$ peers from that region. If the number of peers drops below k in a region, a node merges its close region with adjacent far regions up to and including the far region with $< k$ peers. The close region will again be reduced when new peers replace those that have left.

5.2. System Model

This xor-overlay topology can be implemented for different distributed system models because it follows from the topology invariants (Section 4), not the assumptions made on the distributed system. In the next sections, we provide the abstraction and implementation of the Xor-overlay Topology Manager (XTM) for one particular case of a distributed system by using the following assumptions:

- Identifiers for nodes are uniformly random within the identifier space;
- Nodes have access to an eventually perfect failure detector [7];
- Nodes fail by crash-stopping and therefore cooperate until they fail.

All nodes are physically connected through an IP network, therefore any node can send a message to any other node using its IP address and port. Each node maintains a set of *peers* with whom it has active connections, and sends them periodic heartbeat messages for failure detection.

One possible scheme to obtain identifiers that are uniformly random is to derive the identifier from the hash of the IP address and port of a node. The random uniformity comes from using a good hashing function.

5.3. Abstraction

The xor-overlay topology manager (XTM) abstraction defined in Module 1 executes on each node and provides a well-defined interface to obtain the current set of peers. The set of peers can be used by another module to correctly compute whether the current node is part of the k-closest nodes to a target identifier or find all other peers that are closer to it. To do so, the other module sorts the peer set, augmented with the node's identifier, with the target identifier id. If the node's identifier is in the first k elements then it is part of the k-closest. Otherwise, the closest identifier(s) can be used to contact closer nodes to id using the regular Kademlia lookup algorithm (Property 2.2-3 and Section 3).

The abstraction is configured with parameters that are fixed for the entire execution. k is the size of the number of replicas (the k-closest nodes), b is the number of bits used for identifiers, and h is the hysteresis factor that delays splitting of regions to make the overlay topology more stable in the presence of nodes leaving and joining.

The abstraction is first initialized by providing an id coming from a uniform pseudo-random number generator and a set of bootstrap nodes qs with the *Join* request. From the bootstrap set of nodes, the XTM will discover more peers until at least k are known and a first set of peers is provided with the *Peers* indication. As more nodes are discovered new *Peers* indications are delivered. The regions computed from the set of peers in later *Peers* indications will eventually converge towards the regions that would be computed if all peers were known[6]. At some point the *Unstable* indication may be delivered if too many peers leave at once and there is not enough time for other peers to replace them and maintain the abstraction properties. Following an *Unstable* indication, a new *Peers* indication means the properties are reestablished.

We decompose the properties required from the XTM implementation such that they ensure collectively that the same k-closest nodes would be computed if a node had known the entire set of peers in the network. The properties are guaranteed for *correct* nodes.

The *eventual regions accuracy* property (XTM1) is a liveness property. It ensures that once enough peers have been discovered a node will compute the same regions as if it would know all other nodes in the network. Moreover, by construction because of the definition of far regions, the number of regions should be logarithmic compared to the number of correct nodes.

The *close nodes accuracy* property (XTM2), is a liveness property. It ensures that all correct nodes in the close region will be known. It is necessary for computing the k-closest nodes in the close region.

The *eventual stability* property (XTM3), is a liveness property. It ensures that even if an *Unstable* event happens, as long as there are enough correct nodes ($\geq k$) in the network, a new set of peers will be provided.

The three liveness properties together ensure that the abstraction will eventually provide a set of peers that is useful for computing the k-closest nodes in the close region.

The *completeness* property (XTM4) is a safety property. It ensures that there are enough peers in the far regions to *safely* ignore most other peers in them.

The *scaling* property (XTM5) is also a safety property that follows by construction of the topology. Since a subtree contains at most $k+h$ peers and there are at most b subtrees that correspond to each bit of the identifiers, there are therefore at most $(k + h) * b$ peers in the routing table. As all subtrees fill probabilistically uniformly as new nodes join, then the number of peers in the set of peers grows logarithmically compared to the number of nodes.

We chose those properties to be as *weak* as possible while making it easy to build stronger versions of the same abstraction using ours. For example, Module 1 provides

6. The initial convergence can be made faster if the abstraction is initialized by providing a bootstrap set qs obtained by performing a Kademlia lookup (Property 2.2-3 and Section 3) for id. It is not needed for correctness.

Module 1 Xor-Overlay Topology Manager

Module:

Name: XorOverlayTopologyManager, **instance** *xtm*

Parameters:

k: Number of replicas to use.
b: Number of bits in the identifier space.
h: Hysteresis factor to control when a region splits in smaller regions.

Events:

Request: $\langle xtm, Join \mid$ *id,qs*$\rangle$: Joins the close region to id using qs as bootstrap nodes.
Indication: $\langle xtm, Peers \mid$ *ps*$\rangle$: Provides the latest set of peers (excluding the node's id).
Indication: $\langle xtm, Unstable \mid$ *region*$\rangle$: Invalidates the previous set of peers because the properties of the xor-overlay topology may not be satisfied in $region$.

Properties:

XTM1: *Eventual regions accuracy*: The far and close regions computed with the node identifier id from the latest set of peers will eventually match the regions computed from the set of all correct nodes with a number of replicas between k and $k + h$ inclusively.
XTM2: *Eventual close nodes accuracy*: The set of peers will eventually contain all correct nodes that are in id's close region.
XTM3: *Eventual stability*: Once the Unstable event has been raised, if there exists at least k correct nodes, a new set of peers will be eventually provided.
XTM4: *Completeness*: Each region, close and far, computed from the latest set of peers contains at least k nodes. (By definition, the union of the close and the regions covers the entire identifier space.)
XTM5: *Scalability*: By construction of the peer set, the set of peers contains at most $(k + h) * b$ nodes. As in Kademlia, because the identifiers are uniformly distributed, the number of peers grows logarithmically with the total number of nodes.

eventual accuracy both for the close nodes and the regions. It means that no explicit indication is provided when it is actually achieved. A stronger version (*eventually strong accuracy*) could provide such an indication by synchronizing with all peers in the close region to make sure everyone has the same set of nodes in it and verify with them if they compute the same far regions. That stronger version would therefore need to wait for all nodes in the close region to agree before delivering the set of peers. However, the overhead of synchronization may not be necessary for many applications (such as Kademlia's DHT) so we prefer the weaker version here.

5.4. Implementation

The implementation is provided in Algorithm 1 (at the end of the paper). It follows the properties and intuitions about xor-space in a straightforward way. It is written in an event-driven style and assumes the internal data structures, such as sets and dictionaries raise events when modified. Therefore predicates on them can be used to trigger operations. Notice also that some events are triggered explicitly for modules that are encapsuled by the implementation and do not appear in the abstraction definition. These events are prefixed with the module they reference. For example, **trigger** $< ps, SetSubscriptions|peers >$ triggers the event *SetSubscriptions* on the ps instance of the PeerSampling module, with the set of peers as argument.

We use PeerSampling, a module that obtains samples of nodes from the entire network at regular intervals using gossiping [8]. We use a special *SetSubscriptions* request for PeerSampling that is used to bias the sampling node towards regions of the space that are closer. Otherwise, if samples were uniformly drawn from the entire space, the probability of obtaining new samples in the close region would become smaller as the number of nodes would grow. The sampler therefore provides new peers from a mix of peers of the node's current peers and a set of random peers from the entire network. The topology is updated with the new samples, or when a failure is suspected.

Depending on the number of nodes in the different regions of the topology, regions may be split or merged. The split is straightforward and simply means the close region is split in a close and a far region once enough nodes are known. The merge is slightly more complicated, as the merge may need to happen with a far region that is not the closest. In that case all regions in-between need also to be merged with the close region. If after merging there are not enough nodes in the newer region to satisfy the completeness property, then the Unstable event is raised.

In far regions, we only keep the $k + h$ closest nodes. This spreads the number of connections evenly between all nodes because no node might become a peer of a high fraction of all nodes. Other schemes would be possible. For example, we could keep the longest running node, as done by Kademlia (Property 2.5-2).

6. Use Cases

The Xor-Overlay Topology Manager (XTM) enables new security mechanisms and applications to be built on a xor-based overlay network. In this section we sketch three of them.

6.1. Filtering Operations to the Wrong Destination

In the Kademlia design, because only the node performing a lookup has the most up-to-date information about the running nodes (2.6.1), when a node is the target of a store operation it cannot know if it is part of the k-closest nodes to the target identifier or not. By default it should therefore store the information. Using the XTM a node may compute whether it is effectively the k-closest and filter out operations for target identifiers for which it is not.

6.2. Building a Decentralized Chatroom using Leader Election

Imagine a decentralized messaging application with multiple clients exchanging messages between each other. Clients are all in a chat room with a binary identifier r (presumably the hash of the name of the room). The node closest to r becomes a chat room server s and decides the ordering of messages, replicates them on the $k-1$ closest nodes, and sends the newest messages to all clients. If s fails then the clients reconnect to the other closest transparently. The newer server then continues on with the replicated history. If a new node tries to connect with an identifier that would be closer to the chat room identifier than r's identifier, it is asked to reconnect with a different one by the k-closest.

In effect, this scheme implements automatic leader election using only the node identifiers, without any extra synchronization mechanisms, by exploiting an ordered view of the k-closest nodes. It may also be used to implement a more general replicated state machine to implement other kinds of stateful server services.

6.3. Building Decentralized Storage using Quorum-based Algorithms

Imagine a decentralized storage system that stores user information using git repositories. The individual objects that compose the repository are immutable but the pointer that represents the head of the repository history is mutable so that it tracks the latest update. Moreover, the location of the repository on the network should be stable to be found by its user(s) and could presumably derive from a user identifier.

An unordered view of the k-closest nodes to a target identifier may be used to implement quorum-based algorithms in which a majority of nodes need to agree for an operation to succeed. Suppose then that amongst the k-closest nodes to an identifier, f are likely to fail during its execution. Then, for example, a Uniform Reliable Broadcast (URB) primitive may be implemented by ensuring $k > 2f$ (See the Module 3.3 in Cachin et al. [2]). In turn the URB primitive may be used to build a consistent replication scheme in which if a message m is delivered by one of the k-closest nodes (whether correct of faulty), then m is delivered by all k-closest nodes. That replication scheme may then serve as the basis for updating the head of the repository.

7. Related Work

Researchers have identified accuracy problems with the lookup procedure in Kademlia and suggested various solutions. In 2009, Kang and al. [3] studied why lookups fail in Kad, a Kademlia implementation used by eMule (a BitTorrent client). Their work shows clear empirical evidence that the k-closest nodes found are not accurate because almost half of replica roots are never located for rare objects and almost ten percent of search queries never find the replica roots immediately after publishing. They attributed the problem to the high-level of similarity of routing tables and the fact that “only half of the nodes near a specific ID are alive”. They proposed extra steps in the lookup procedure that increase the accuracy while increasing the latency. Three years later, Liu and al. [4][7] performed a similar study again and found that additional factors influenced the lookup reliability: selective denial-of-service (malicious) nodes, misses when an operation timeouts after having found the roots but did not have time to perform an operation on them before the timeout occurred, in addition to misses that happen when the roots are not found. They suggested to perform operations on the *θ-neighborhood* of an identifier that is defined as the region of the space that shares a fixed length prefix θ and comprises all nodes within it. Rather than modifying the lookup operation, we suggest instead to change the invariants that decide which peers are kept in the routing table and ensure nodes know about all their peers in the region close to itself. Our approach adds no latency on lookups and instead places the trade-off between the frequency at which nodes send each other heartbeat messages to detect failure (bandwidth used) and accuracy. Moreover, our close region definition is similar to the θ-neighborhood but the size of the neighborhood (close region) shrinks automatically as the network size grows rather than being fixed, removing the need for tuning the θ parameter.

Other researchers have proposed ways of implementing newer abstractions on top of a Kademlia overlay. Chazapis and al. [9] proposed an additional *update* operation for a

7. We reference the more complete journal paper that was published later.

DHT that mutates the value of a key and guarantees the update to be consistent and atomic on all replicas in a Byzantine distributed system model using a quorum based algorithm. For the correct operation, they "assume that each peer has a good knowledge of its close peers and thus will know the quorum members of each data item it stores." They prudently mention that 'depending on the way each particular DHT implementation manages routing tables, this may require an extra messaging step". Our work provides such guarantees while not requiring extra message steps (if eventual consistency is sufficient). Czirkos and al. [10] proposed different algorithms for broadcasting messages to all nodes of the network that leverage the tree structure of the routing tables for efficiency. As their approach is probabilistic, it doesn't need more accuracy or consistency than what is already provided by the original design of Kademlia so our work is not necessary for those applications.

8. Conclusion and Future Work

In this paper, we first provided insights about the original Kademlia design and made explicit some design decisions that were implicit in the original paper. We re-explained the elegance of the lookup procedure in terms of its combined use of ordered and unordered views of the identifier space. We explained an insight that enable different invariants on routing tables and a precise definition of the close region of a node. Both enable accurate and consistent computation of the k-closest nodes to any target identifier of the identifier space. Our invariants avoid the irregularities of the original k-bucket design, do not require extra lookup operations and are compatible with the original lookup procedure, and dynamically adjust to any network size while requiring one less parameter to tune than the θ-neighborhood approach. We provided a clear and precise specification and implementation of a Xor-based Topology Manager (XTM) that maintains routing table using the invariants. We finally sketched how the XTM could be leveraged to build new applications that go beyond the file sharing domain of Kademlia by adapting known distributed abstractions to leverage the unique topology of xor-based overlay networks.

Our work could be extended by formulating a precise implementation of a strongly consistent variant of our XTM, study the messaging overhead, and implement it in various distributed system models. It could be used to reformulate the original Kademlia implementation in layers to make it easier to learn, and separating the procedures made for correctness from those required for optimizations. Our XTM implementation could also be formalized and automatically checked for correctness to ensure it meets the abstraction specification. Finally, empirical studies to verify its effect on accuracy and consistency of k-closest nodes lookup should ensure the practice meets the theory.

We hope this paper will motivate peer-to-peer application developers to implement other existing distributed abstractions on Kademlia networks and use the resulting system as a foundation for more sophisticated self-organizing systems.

Acknowledgment

Inspiration for this work came from trying to understand and clearly specify how the k-closest nodes were computed in the Safe Network implementation, itself based on Kademlia and developed by MaidSafe Incorporated. This culminated in finding new invariants for accurate and consistent computation of the k-closest nodes, which are described in this paper.

We would like to thank MaidSafe Inc. for providing accommodations during a short visit in Troon (Scotland) in 2015, as well as David Irvine and Bejamin Bollen for insightful discussions and the rest of the employees for an enjoyable stay.

We would also like to thank other collaborators such as Thanh Nguyen for initial guidance in research methodologies in distributed algorithms, Andreas Fackler for comments on a previous version of this paper, and reviewers for their comments and suggestions, which all helped greatly improve the quality of our work.

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013, by the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Fonds de recherche du Québec Nature et technologies (FRQNT).

References

[1] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 53–65.

[2] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer-Verlag Berlin Heidelberg, 2011.

[3] H. J. Kang, E. Chan-Tin, N. J. Hopper, and Y. Kim, "Why Kad Lookup Fails," in *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE, 2009, pp. 121–130.

[4] B. Liu, T. Wei, C. Zhang, J. Li, and J. Zhang, "Improving lookup reliability in Kad," *Peer-to-Peer Networking and Applications*, vol. 8, no. 1, pp. 156–170, 2015.

[5] X. Cai and L. Devroye, "A Probabilistic Analysis of Kademlia Networks," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 8283, pp. 711–721.

[6] C. Pornavalai *et al.*, "Proximity neighbor selection using IP prefix matching in Kademlia-based Distributed Hash Table," in *Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 2010 International Conference on*. IEEE, 2010, pp. 671–675.

[7] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[8] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," in *Engineering Self-Organising Systems*. Springer, 2005, pp. 1–15.

[9] A. Chazapis and N. Koziris, "XOROS: A Mutable Distributed Hash Table," in *Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2007), Vienna, Austria*, 2007.

[10] Z. Czirkos and G. Hosszú, "Enhancing the Kademlia P2P Network," *Periodica Polytechnica Electrical Engineering*, vol. 54, no. 3-4, pp. 87–92, 2012.

Algorithm 1 Xor-Overlay Topology Manager

```
Implements:
    XorOverlayTopologyManager, instance $xtm$.

Uses:
    PeerSampling, instance $ps$;
    EventuallyPerfectFailureDetector, instance $fd$.

upon event $\langle xtm, Init\rangle$ do
    $id$ := 0; $peers$ := $\emptyset$; $topology$ := $\{\top : \emptyset\}$;      ▷ A topology maps regions (keys) to members (values);
                                                        ▷ Initialized to the entire identifier space ($\top$

upon event $\langle xtm, Join \mid id'$,$qs\rangle$ do $peers$ := $qs$; $id$ := $id'$;

upon $peers$ changed do trigger $\langle ps, SetSubscriptions \mid peers\rangle$;

upon $topology$ changed do
    if $\forall members \in topology.values, |members| \geq k$ then      ▷ topology.values provides all values in the topology
        trigger $\langle xtm, Peers \mid peers\rangle$;

upon event $\langle ps, Sample \mid qs\rangle$ do                         ▷ Update topology and groups
    for all $q \in qs$ do
        for all $region \in topology.keys$ do              ▷ topology.keys provides the set of all keys in the topology
            $members$ := $topology[region]$;      ▷ topology[region] provides the members set associated to the region
            if $q \in region$ and $q \notin members$ then
                $topology[region]$ := $members \cup \{q\}$;      ▷ topology[region] := ... replaces the members set of a region
                $peers$ := $peers \cup \{q\}$;

upon event $\langle fd, Suspect \mid q\rangle$ do                          ▷ Handle failures
    $peers$ := $peers - \{q\}$;
    for all $members \in topology.values$ such that $q \in members$ do
        $members$ := $members - \{q\}$;                ▷ False suspicions are reintroduced through Peer Sampling

                                                                            ▷ Split close region
upon $topology$ changed and $\exists region \in topology$ with $members$ and $id \in region$ such that $members$ can be split
in two smaller groups and the smallest group size $> k + h$ and they both cover half of the topology region do
    $close\_hood$ := xor-closest half-split of region to $id$;
    $far\_hood$ := xor-furthest half-split of region to $id$;
    $close\_members$ := $\{m|m \in members$ and $id(m) \in close\_hood\}$;
    $far\_members$ := $\{m|m \in members$ and $id(m) \in far\_hood\}$;
    $topology$.remove($region$);                      ▷ topology.remove removes the region and associated members
    $topology[close\_hood]$ := $close\_members$;          ▷ topology[region] := ... implicitly creates the association
    $topology[far\_hood]$ := $far\_members$;

upon $topology$ changed and $\exists region \in topology$ with $members$ such that $|members| < k$ do      ▷ Merge region(s)
    for all $region \in topology$ with $members$ from biggest to smallest region such that $|members| < k$ do
        $members$ := $topology[region]$;                                    ▷ Merge smaller regions
        for all $region' \in topology.keys$ such that $region' \neq region$ and $|region'| \leq |region|$ do
            $members'$ := $topology[region']$;
            $members$ := $members \cup members'$
            $topology$.remove($region'$)
        $topology[region]$ := $members$;
        break;
    if $\exists region \in topology$ with $members$ such that $|members| < k$ then
        trigger $\langle xtm, Unstable \mid region\rangle$;                  ▷ Next samplings should add new members

                                                               ▷ Rem. extra peers in far regions
upon $\exists region \in topology$ with $members$ such that $|members| > k + h$ and $id \notin region$ do
    $excluded\_members$ := $\{m|m \in members$ and $id(m)$ is not in the closest $k + h \in region\}$;
    $topology[region]$ := $members - excluded\_members$;
    $peers$ := $peers - excluded\_members$;
```