Effective Manycast Messaging for Kademlia Network

Lubos Matl
Czech Technical University
Charles square 13, 121 35
Prague 2, CZ
matllubo@fel.cvut.cz

Tomas Cerny
Czech Technical University
Charles square 13, 121 35
Prague 2, CZ
tomas.cerny@fel.cvut.cz

Michael J. Donahoo
Baylor University
Waco, TX, US
jeff_donahoo@baylor.edu

ABSTRACT

Peer-to-peer (P2P) communication plays an ever-expanding role in critical applications with rapidly growing user bases. In addition to well-known P2P systems for data sharing (e.g., BitTorrent), P2P provides the core mechanisms in VoIP (e.g., Skype), distributed currency (e.g., BitCoin), etc. There are many communication commonalities in P2P applications; consequently, we can factor these communication primitives into overlay services. Such services greatly simplify P2P application development and even allow P2P infrastructures to host multiple applications, instead of each having its own network. Note well that such services must be both self-scaling and robust to meet the needs of large, ad-hoc user networks. One such service is Distributed Hash Table (DHT) providing a dictionary-like location service, useful in many types of P2P applications. Building on the DHT primitives for search and store, we can add even more powerful group communication services to increase network capabilities with nodes-group formation and messaging (manycast, anycast, multicast, etc.). We begin by providing a survey of DHT networks and their group communication extensions. Next, we propose extensions for the Kademlia DHT to allow group communication and compare its properties with existing group communication services in the Pastry network. We place particular on manycast as it is a more generalized form of communication that has received little attention from the research community. Using these empirical results, we show which network is best suited to particular communication situations.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed networks—*Network communication, Network topology, Store and forward networks*; C.2.2 [**Network Protocols**]: Routing protocols; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Enhancement*; C.4 [**Performance of systems**]: [Design studies, Fault tolerance, Reliability, availability, and serviceability]

SAC'15 April 13 - 17 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695903

General Terms

Measurement, Reliability, Performance, Experimentation

Keywords

Distributed systems, performance, communication, P2P, DHT, key-based search

1. INTRODUCTION

P2P networking is a widely-known architectural pattern for distributed systems. P2P has been applied to a variety of applications, including file sharing (BitTorrent [10]), video or audio streaming (SplitStream [5], Coolstream [22]), parallel computation, online payment systems (Bitcoin [16]), or voice-over-IP services (Skype). All these applications demonstrate that the architecture is important and useful for multiple domains.

In P2P information collecting, computation, resource sharing, problem solving, etc. execute in a distributed manner. There are several types of P2P networks that we can distinguish according to its structure from unstructured networks, whose connections are formed randomly (Gnutella, Gossip, Kazaa), structured networks that are organized into a specific topology. Such networks consist of nodes, which cooperate to find a correct receiver for a submitted message.

One special type of structured P2P networks is a Distributed Hash Tables (DHT), such as Pastry, Kademlia, CAN or Tapestry [15, 17, 18, 23]. DHT primarily provides a hash-table service in a distributed approach. These tables are used for storing pairs $(key, value)$ and efficiently retrieving values according to a given key; therefore a DHT network is an ideal tool for data search engines. Good DHT employ self-organization, fault-tolerance and scalability. Some DHTs even include user anonymity.

We can use DHT services to enable group formation and communication. A peer can contact some single group member (anycast), all group members (multicast) or selected number of group members (manycast). DHT frameworks are usually extended with group communications as it enables much broader applicability. For instance, Scribe [4, 6] is group communication extension to Pastry DHT [18]; similarly SplitStream is another such extension for data streaming.

In this paper we consider Kademlia DHT designed by Maymounkov and Mazieres in 2002 [15]. It provides value lookup similar to other DHTs, with the benefit of proximity-based routing. Searches in Kademlia find the peer containing the key/value pair with the shortest path from the search

origin. Unfortunately, Kademlia only allows contacting a single network peer, and thus there is no option for group communication. In our work, we describe an extension to the Kademlia network that provides group communication. We particularly emphasize manycast communication and consider two types of manycast messages: anycast [6] and multicast [4]. The manycast communication gives the network an ability to contact N-group members, and since Kademlia provides proximity-base routing, the contacted group members are the closest among all the group members with respect to time delay between the communication initiator and the group members. This gives the requester an ability to consume services from members close to it on the network; furthermore, it reduces network communication delay and thus improves the quality of service.

We implement a prototype extension that provides manycast and compare its abilities and qualities with the Pastry groups implementation. We compare the extensions in terms of delivery time, network load and fault tolerance. We are aware that there is no silver bullet in P2P networks, but the comparison gives insight on which network best suits to a particular situation.

The rest of the paper is organized follows. Section 2 provides background for DHT, Kademlia network and group communication. Section 3 contains related work. In Section 4, we show our design of Kademlia group communication. We present comparison test results in Section 5, and finally Section 6 provides our conclusions.

2. BACKGROUND

This section first provides a brief description of P2P networks and DHTs. Further, briefly introduces Scribe, group communication extension for Pastry. Finally, is presented the Kademlia network where we discuss its differences from other P2P networks, its advantages and disadvantages.

As mentioned previously, the peers in P2P networks cooperate to provide some service. Unlike the client-server architecture where the client sends requests and the server replies after completing the task, in P2P there is no distinction between clients and servers. Every autonomous peer is both server and client; therefore a peer has two roles: requester and solver. This architecture brings several benefits like self-scalability, anonymity, reliability and decentralization. Self-scalability is natural because while more peers mean more requesters and more solvers as well. Anonymity can be achieved by hiding the source address during the request and the response message path though the network. Services are more reliable because a service is still available from another peer when a provider fails. Finally, the decentralized nature of P2P networks provides greater robustness. P2P networks still contain several limitations in security and usability, which provides a great opportunity for research.

2.1 Distributed Hash Table (DHT)

A DHT is a structured P2P network that enables storing data inside connected nodes. This system provides a similar service to a hash table, but with the exception that the stored values are distributed among network nodes. First, every connected node must have a node identifier (NID). The NID is used for storing and retrieving data from the network. Anyone who wants to store data inside the hash table must have a pair $(key, value)$ and the stored value can be found using the particular key. The value is very often stored in one or more nodes with NIDs closest to the key; consequently, the key may be used for finding nodes storing the value.

Unfortunately, a DHT is more complicated than typical hash table because every node cannot have knowledge of the whole network. It is not possible to connect a large number of nodes having full network knowledge because the knowledge base would be too large and would quickly grow stale. Therefore, every node has information only about some part of the network, and nodes must cooperate to find the closest node to the searching key. To achieve this, network peers form a specific network structure and execute a protocol, which finds the right node with the lowest number of network hops. There exist several networks that provide different structures and searching mechanisms with finding a particular node with a logarithmic number of hops depending on the number of nodes. For example consider CAN [17], Pastry [18], Tapestry [23] or Kademlia [15]. These networks have the desired properties for practical use such as fault tolerance, self-scalability and decentralization.

2.2 Group Communication

DHTs may be extended to provide group communication. First, nodes may create groups where every node can arbitrarily connect to or disconnect from a group. For example, a group may present nodes, which have the same sort of information or provide the same service. Like DHT nodes, every group must have a Group Identifier (GID). Network nodes may contact the group using the GID in the tree ways: manycast, anycast and multicast. Anycast is a message delivered to a single group member; multicast delivers to all group members; and manycast delivers to N-group members, where the N is defined by the sender. It may seem that anycast and multicast are special type of manycast message, but in terms of realization manycast message is very different.

The group implementation usually uses tree architecture where the root of the tree is the node whose NID is closest to the GID from the network. The other group members are connected to the root or its descendants. Such tree construction may include a technique for tree balancing, which further improves the message delivery process. The group is found using the same algorithm used for message search in the DHT. If a message is being forwarded by a node that is a group member the node does not forward it and instead receives it. A Group-Join message and anycast message works this way; therefore these messages are received by the first group member, which forwards it or by the group root.

A multicast message is delivered to the first receiver the same way as anycast, but this receiver must also forward message inside the group tree. The node resends message to its parent and descendants. This procedure is recursively repeated (forwarder only omits node from which the message received) until the all group nodes receive the multicast.

2.3 The Manycast Extension

A manycast message is received with a sender-defined number of group receivers. The network determines which subgroup of N nodes receives the message, and the selected group members receive messages with identical content. Manycast can be used as a security feature where a sender sends the same requests and evaluates whether the responses are identical. The content of a manycast message does not need

to be completely identical; it can contain non-identical control information. The control information may be, for example, the index of a manycast message. Therefore every node receives a manycast message with different index from 1 to N where N is the number of desired receivers. An application example of manycast index can be file sharing where user wants to download a file from multiple sources. The receiver decides according to the manycast index, which part of file should be sent back to the requester. This strategy could significantly decrease download time.

Scribe supports anycast and multicast messages but not general manycast message. There is only one way to send manycast message, which is to resend the message from one node to another as an anycast message inside the group tree. Every manycast receiver must resend new anycast messages until the message is delivered to the desired number of receivers. The problem with this solution is that the delivery time depends on the number of manycast receivers, and the group tree may be suboptimal because the message is forwarded from one part of the group tree to another.

Two methods have been designed for manycast message delivery. With N-anycast [13] the sender sends N-anycast messages with the same content and different indices. When the messages follow the same routing path, there is a chance that only one group member receives all manycast messages. For this reason, the first hop of each message is different. Therefore the sender sends every message through a different node that is gathered from neighborhood set, which every Pastry node owns. The neighborhood set contains the nodes that are closest to the owner in terms of network delay, and thus N-anycast preserves the Scribe property of the closest recipients of the anycast message [7].

Tree distribution [13] uses only a single message that is sent to a Scribe group in the same manner as an anycast message. The receiver of the message resends the message to K nodes. These nodes are selected from its descendants and parent inside the tree. The value of K is often selected as 2 but may be given a greater value. The tree distribution method is very similar to Scribe's manycast distribution, but the manycast message may be delivered much faster with the balanced tree and properly set parameter K.

These two solutions raise other problems such as how to avoid more receivers than desired and how to deliver message to the receiver only once. These problems are beyond the scope of this paper, and details can be found in [13].

2.4 Kademlia

Kademlia is a DHT based on an XOR metric. The network has several applications with more than a million nodes deployed on the Internet. Mainline DHT and Azures DHT are applications based on Kademlia network that work as peer discovery for BitTorrent. KAD is used for peer and content discovery in a widely used file-sharing application called eModule. According to Measuring Large-Scale Distributed Systems (MLDHT) [20], the number of BitTorrent users per day is between 15 and 27 in millions. This huge number of users shows that Kademlia is able to achieve great results in a large number of client applications.

Similarly to other DHTs, every Kademlia node owns a unique 160-bit integer identifier. This identifier may be generated using a peer's IP address, randomly generated by the node itself, or assigned by a central authority. The distance, d, between two nodes of a Kademlia network is computed using an XOR operation. This means that distance between nodes a and b is $d = a\,XOR\,b$. The XOR metric is also unidirectional, which means that if there exists some point x and distance d, there is only one point y where $d = x\,XOR\,y$. Together with the routing algorithm, ensures that every routing hop will be closer to the message key, and the message is delivered to the node with the closest identifier.

Unlike Pastry, Kademlia uses lookups for discovering a message receiver, and thus a sender first finds the correct message receiver using lookup messages and then directly sends the message to the node. A sender must cooperate with other network nodes to find the message receiver and use its network information. For this purpose, every node owns 160 lists that are called k-buckets.

Every record stored inside the k-bucket list contains the triple $< ipaddress, port, identifier >$. Records are stored to the correct k-bucket according to distance. The k-bucket with index i contains only nodes whose distance is between 2^i and 2^{i+1} from the bucket's owner. To ensure finding the message recipient, no k-bucket can be empty if the node with particular distance from k-bucket owner exists. Records stored inside each k-bucket list are sorted by the access time as well. The last visited node is stored as the last node of the k-bucket and is removed if a new node belonging to the k-bucket is found. A node receiving lookup request finds the closest nodes to the lookup key inside its buckets and sends the nodes back. The receiver recursively contacts the nodes until it finds the closest recipient.

The Kademlia network has several great properties. Scalability is the biggest; a message is received within $log(N)$ iterations where N is total number of network nodes. The mathematical proof can be found in [15]. The second is load balancing because load is equally distributed among all network nodes. Another property is fault tolerance. A failed node is automatically removed from the k-bucket, and it is replaced with a different node if it exists. The advantage over the Pastry network is that Kademlia uses only one algorithm to find the closest node. The Pastry network has two phases. The first phase finds an identifier, which is half as far to the message key, and the second finds the final receiver with a little bit different algorithm. The first phase uses the routing table and the second the leaf set. Nodes found in the first phase may be far away (geographic distance) from the second phase, which may cause extra delay. Therefore the Kademlia may have shorter message delivery time. For shorter paths, Pastry constantly improves its routing tables according to delay. Kademlia does not need to do this; the fastest path is found in a more natural way, because the sender always uses the first response for the lookup.

Kademlia has some disadvantages as well, for instance flooding the network. It is necessary to send many more lookup messages to find the correct receiver. In contrast Pastry sends only single message, which is directed to the receiver. Lookups are tiny messages, and Kademlia does not need to send many control messages for routing table updates like the Pastry does. While Kademlia may flood the network with lookups, it does not flood the network with control messages; however, this together may generate more data traffic.

3. RELATED WORK

The most used DHTs are Pastry, Chord, CAN, Tapestry and Kademlia [15, 17, 18, 19, 23]. These networks are

built from a large number of nodes where each node has a unique identifier. These identifiers are usually randomly spread through the network to ensure high quality of message routing. Nodes must own one or more routing tables that are used for routing the message via network to the right receiver. Every message must contain a key, and it is delivered to a node with the closest identifier to the key. The nodes have only limited knowledge about the network structure, and thus nodes must cooperate to deliver the message. There are two ways for message delivery: forwarding and lookup. Either the message is forwarded in the network from one node to another until it arrives to the right recipient, or the sender first finds the message receiver, with a lookup messages, and then sends the message directly to the receiver.

Many DHTs [15, 18] deliver the message within $O(log(n))$ routing algorithm iterations, where the n is the total number of network nodes. Some networks can improve the delivery time; for example the Pastry network can select message path according to network delay between nodes.

Scribe [4] is a Pastry extension that provides anycast and multicast messaging. The extension adds the ability to create groups of network nodes and to send messages to these groups. The anycast message is received by one group member; the multicast is received by all group members. Groups have a tree-based architecture where the root node has the closest identifier to the group identifier.

Another Pastry extension, SplitStream [5], uses multiple Scribe overlay trees for data streaming. With SplitStream data are split and every piece of the data is distributed through a different tree. The identifiers of nodes and groups are selected in such a way that each node can be an internal tree node only once. Therefore the streaming load is optimally distributed between users.

P2P networks are usually complex to use because their implementations are "low-level". This problem is addressed by the ELISA framework [14] that hides the low-level P2P implementation details with a high-level API. This brings two advantages: First, the usage is easier. Second, this approach brings the uniform interface for various low-level network frameworks and thus its portability. ELISA provides improvements to Scribe and introduces manycast group messages with two strategies: N-anycast and Tree distribution [13].

Among the many uses of manycast messages, we mention Cooperative Web Cache (CWC) [8]. The CWC is a distributed P2P web cache for storage of static data from web pages. It can be seen as a self-scalable, free-of-charge Content-Delivery Network (CDN). The study in [8] shows that the distribution of static data forms more than 90% of web pages. Sharing this static data inside a P2P network may be advantageous in terms of delivery time on the client-side and web server load. The delivery time may be reduced when the delay among network nodes is lower than it is between the server and the client. The server load is reduced because a large number of files are received from the neighbor P2P network nodes. The manycast message is used to download large files from more then one node. This solution can be comparable to BitTorrent.

Alternative approaches that provide manycast messages in computer networks is message flooding or scoped-flooding [3]. Both these solutions are very primitive and flood the network with a large number of request messages. All receivers, which are able to respond, send back the response. The receiver may receive more responses than initially expected. This is an advantage because the node may choose the correct recipients. On the other hand, the network load may be large, compared with solutions that build on DHT, which use a structured P2P network and thus reduce network load.

PPmcast's [12] aim is to create group communication on top of the Kademlia network. The group architecture of PPmcast is very similar to Scribe. Groups form a tree structure with the root node whose identifier is closest to the group identifier. In [12] the authors compare their test results with Scribe focusing on multicast messages. The main difference with our work is the different group tree structure and the implementation of manycast method that could work both as anycast and as multicast in our solution.

4. KADEMLIA GROUP EXTENSION

For the Kademlia DHT extension to enable group communication, we consider a tree architecture similar to Scribe. The root of the tree has the identifier closest to the group identifier. All messages are sent to the root node; if a message is delivered to a node that is connected to the group, the node does not forward the message but simply receives it.

The Kademlia network has several differences compared to Scribe. For message delivery, it does not use direct message forwarding but instead uses a lookup. Therefore when a node A receives a lookup for a group X that A is a member of, it does not continue with the routing algorithm. Instead A sends back a message stating that A is the receiver. This approach works for both anycast and join messages.

4.1 Tree Creation/Join Message

First, it is necessary to build the tree. To join a node simply sends a join message. The join message is sent like a normal Kademlia message, and thus it must contain a key, which is used to route the message to the group root. Since Kademlia uses lookups for the root finding, it is probable that during the search one of the group members is contacted. At this point, it is possible to interrupt the lookup algorithm and send a join message directly to the found group member that will serve as a join point.

The tree efficiency depends on the strategy that is used for its construction. The easiest solution is to join new nodes to the first node that receives a join message, but this may create very unbalanced tree structures. Another strategy always delivers a join request to the root, and the join request traverses the multicast tree until the most suitable place for the new node is found. In both strategies, a tree node is selected, and the new node is joined as its child.

Both strategies have advantages and disadvantages. The first strategy generates less load on the root and a smaller delay between the neighbor nodes in the tree, but the second strategy may use superior techniques for manycast delivery, because tree root has more information about the tree structure. Because we target the smallest delay between tree nodes, we use the first described method where nodes are connected to the first node that is contacted. This solution has $O(log(n))$ complexity (number of lookups) where n is number of peers in the network because it uses the same routing algorithm as DHT search message and it may only decrease the number of lookups when another group member is contacted than the tree root.

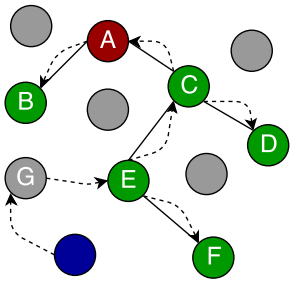

Figure 1: Multicast

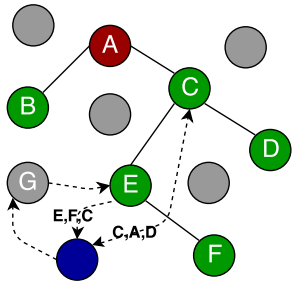

Figure 2: Manycast

4.2 Anycast Message

An anycast message is sent with the same way as the join message. The first nodethat receives this message is the anycast receiver. This node may resend the message to another tree node if is not possible to respond from some reason, for instance load distribution or performance. Because the routing algorithm is the same, the join message complexity is $O(log(n))$.

4.3 Multicast Message

A multicast message is received by all group members. The first phase of sending a multicast message is same as the anycast or join message. The second phase distributes the message inside the group tree. Fig. 1 shows both phases where the sender (blue node) wants to deliver a multicast message to 5 receivers. The message is first delivered to the group member E with the standard lookup process. Group member E sends the tree message to group members F and C. Every other group member forwards message to its descendants and its parent, ignoring the node from which it received the message. This process repeats until all group members receive the same message. The manycast message has complexity $O(log(n) + m)$ where m is number of tree members and n is number of peers in the network.

4.4 Manycast Message

More complexity comes with the manycast message. For the purpose of manycast delivery, we utilize lookups inside the tree. The message delivery is divided into two phases. The first phase again works like the anycast message; therefore a message is delivered to one tree member who starts the second phase. The second phase finds a desired number of manycast receivers, which is achieved by the lookup. Fig. 2 shows the algorithm. In the first phase, node E is found. This node sends back information about its neighbors inside a group tree, specifically nodes C and F. The manycast sender sends a lookup to node C and receives nodes A and D. Next the sender has enough nodes for the manycast and sends the message to nodes C, D, E, F and A. If the number of desired receivers is m, then the complexity of the algorithm is $O(log(n) + m)$.

4.5 Tree Maintenence

The tree must be fault-tolerant, and it is not easy to maintain. Therefore every tree member must periodically send a control message to its parent. If the parent is not reachable, the node must find another group member who is not its descendant. The join message becomes more complicated for this case, and a node that receives such join message must verify that the sender is not its ancestor in the tree. This adds more message overhead for the tree maintenance.

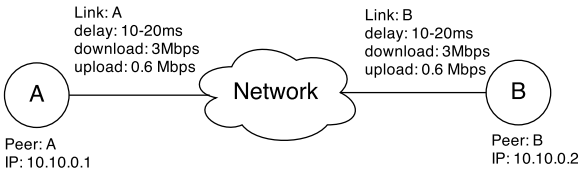

Figure 3: Network settings

The major problem in tree maintenance is when the root member fails. The root node is the group bottleneck, because a failed root means the unavailability of the group for other network peers. Therefore the tree must be rebuilt as soon as possible, but the unavailable root may be detected too late. This weakness can be addressed by providing multiple tree roots for a group. The more nodes that serve as the tree roots, the lower is the probability of not finding the group. Of course the root nodes identifiers must be the closest to the group identifier from the network. If the node with the closest identifier to the group fails, then the message is received by the node with second closest identifier. Since this node is root of the tree as well, the message is correctly delivered. Unfortunately, more root members mean more control communication, but this is only a small price for greater fail-tolerance of the group.

5. CASE STUDY

In this section, we evaluate our Kademlia group extension and compare it with Scribe in terms of network load, network delay and fault tolerance. For this purpose, we created two implementations. The first implementation uses the FreePastry framework [11], which contains the Scribe extension with N-anycast and Tree distribution improvements. The second implementation uses the Kademlia group extension, which is built on top of TomP2P [2] that implements the Kademlia network. Both of these frameworks use Java 7; therefore we can use the ELISA API [14], which provides the indirection layer for both Kademlia and Pastry implementations, bringing equivalent group creation and message submission mechanisms.

All evaluation tests were performed on the same computer with Linux Ubuntu 14.04, so network peers were executed on the same machine with a CPU of 8 cores of 2.93GHz and 12GB memory. To address the high memory requirements, the peers were executed inside a single Java Virtual Machine (JVM). The case study aim is to compare both networks according to network conditions, and thus we had to find a way to set connections between network peers. For this purpose, we created virtual interfaces with different IP address for each peer. Next, we were able to set network delay and bandwidth between peers interfaces through the Netlink interface. For this purpose, we use the ShaPy library [8].

We considered the use of test environments like EmuLab or PlanetLab, but there are several reasons why we did not use them [9, 21]. EmuLab provides network creation according to given specification and prepares network of computers connected with pre-set network delay and bandwidth, but the number of computers is very limited so it is not possible to test more than 80 peers on EmuLab. PlanetLab provides a larger number of resources, but the PlanetLab environment is very unstable with respect to delay, and it is not possible to repeat tests with the same settings.

Table 1: Scribe - message delay [ms]

	net 1%	net 2%	net 5%	net 10%
multicast	309	431	544	687
manycast 20%	478	462	586	773
manycast 40%	497	556	829	1180
manycast 60%	472	692	942	1486
manycast 80%	539	643	992	1501
manycast 100%	551	772	1221	1857

Table 2: Scribe - transferred data [MB]

	net 1%	net 2%	net 5%	net 10%
network set up	201	187	212	208
multicast	510	535	544	687
manycast 20%	478	462	586	773
manycast 40%	406	492	550	638
manycast 60%	430	571	596	694
manycast 80%	481	548	662	734
manycast 100%	512	633	667	797

5.1 Test Environment

All the tests contain 2800 peers connected to a single network. The link settings between nodes is shown Fig. 3. Network peers were connected to the network, where every link has limited download/upload to 3Mbps/0.6Mbps. These values are based on Speed Matters Report [1]. The delays between peers and the network were selected randomly from the interval of 10-20ms. Because the delays between peers are summed, the maximum delay is 40ms, and the minimum is 20ms.

The case-study focuses on manycast messages, and thus their distribution over network peers is mainly measured. We create tests with multiple percentages of peers in the group: 1%, 2%, 5% and 10% peers in the group. The manycast messages were tested with more manycast setups. We used five settings with the number of receivers: 20%, 40%, 60%, 80% and 100% peers in the group.

Every P2P group settings is tested with the same network setup and repeated 3 times. The results are averaged over the measured values. For all the tests, we leave a 1 hour gap for peer settings so Pastry has enough time to improve peers tables according to the network delay among nodes.

5.2 Message Delay

The test results for Scribe and Kademlia extension are shown in Table 1 and 3. We first measured the time of multicast message delivery. The resulting times are very similar, although Scribe has slightly better delivery times for multicast message. The reason is that the lookup message of the Kademlia network does not contain any data, and unlike Pastry where data is sent together with the message, it is necessary to send the data after the first tree member is found. For the manycast message, the situation is very similar, but the difference is much smaller. For the manycasts with more than 50% receivers of the group members Kademlia is even faster. The reason is that Scribe uses the Tree distribution method, and thus requires more network hops inside the tree for finding all distinct manycast receivers. The problem of the N-anycast and the Tree distribution distinct receivers is discussed in [13].

Table 3: Kademlia extension - message delay [ms]

	net 1%	net 2%	net 5%	net 10%
multicast	342	447	622	775
manycast 20%	455	434	767	976
manycast 40%	508	568	877	1457
manycast 60%	509	570	961	1648
manycast 80%	519	576	1060	1674
manycast 100%	513	579	1136	1632

Table 4: Kademlia extension - transferred data [MB]

	net 1%	net 2%	net 5%	net 10%
network set up	57	60	52	59
multicast	24	30	46	59
manycast 20%	23	34	71	105
manycast 40%	31	52	104	146
manycast 60%	31	51	124	173
manycast 80%	32	52	140	175
manycast 100%	31	55	139	173

5.3 Transfered Data

The next comparison is based on the amount of transferred data. The test results were measured by the iptables command. We measure the whole amount of transferred data during the tests in MB. The results are in Table 2 and 4. The difference between networks is significant at first glance. The amount of data transferred over the Pastry network is much higher. The transferred data are more than 60% higher for Pastry network for the network set up. During message submission, the difference is even much higher. The reasons for higher data transfer for the Pastry network are two: 1) Pastry sends more control messages among peers to update routing tables according to the delay and 2) a Pastry message that is forwarded through the peers contains data. In the contrast Kandemlia uses lookup messages and sends data only to the final receiver. Although Kademlia use more messages for lookup, the control messages of Pastry appear as much higher overhead.

5.4 Summary

According to results, the Pastry has better time results for fewer manycast receivers and Kademlia for more manycast receivers. The Kademlia network has lower network load; the difference in data traffic is much higher than in the delivery time. Therefore Kademlia implementation is better for clients who have bigger connection limits.

One problem with these results is that it is valid only for FreePastry and Kademlia extension implementations. Simulation could be better than emulation because there are no implementation dependencies, although the tests are only emulation, we believe that Pastry must have higher traffic. The aim of the paper is to compare our implementation of group extension with the popular implementation of Pastry/Scribe. The results demonstrate that is possible to create group extension on top of Kademlia network and shows which network is better for a particular purpose.

6. CONCLUSION

In this paper, we provide a survey of P2P network architectures and DHTs. Next, we show the usual message delivery extensions to DHT and discussed their design details.

We describe existing extensions in Pastry DHT and suggest and implement group message extension to Kademlia DHT. Next, we provide a manycast message service that brings many opportunities for improvements in application communication. We also provide an evaluation comparing the message communication extension in Pastry and Kademlia DHTs and discuss our results regarding network load, network delay and fault tolerance. Furthermore, we implement Kandemila network under the ELISA API. This provides simplification for P2P application development that gives a high-level indirection to multiple underlying network frameworks. When using the ELISA API, future changes to low-level frameworks or their extension will not impact application implementations, providing platform independence.

In future work, we aim to consider manycast advantages for distributed databases, transactions, confirmation protocols. Next, we aim to consider other DHT implementations under ELISA API and provide suggestion based on empirical evaluations.

7. ACKNOWLEDGMENTS

This paper is supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS OHK3-038/14. We would also like to thank Baylor University for their helpful support in therms of research area.

8. REFERENCES

[1] Speed matters, internet speed report, 2010.

[2] T. Bocek. Tomp2p-a distributed multi map, 2009.

[3] C. Carter, S. Yi, P. Ratanchandani, and R. Kravets. Manycast: Exploring the space between anycast and multicast in ad hoc networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 273–285. ACM, 2003.

[4] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. *SIGOPS Oper. Syst. Rev.*, 37(5):298–313, Oct. 2003.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. In B. Stiller, G. Carle, M. Karsten, and P. Reichl, editors, *Group Communications and Charges. Technology and Business Models*, volume 2816 of *Lecture Notes in Computer Science*, pages 47–57. Springer Berlin Heidelberg, 2003.

[8] T. Cerny, P. Praus, S. Jaromeska, L. Matl, and M. Donahoo. Towards a smart, self-scaling cooperative web cache. In *SOFSEM 2012: Theory and Practice of Computer Science*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, July 2003.

[10] B. Cohen. The bittorrent protocol specification, 2008.

[11] P. Druschel, E. Engineer, R. Gil, Y. C. Hu, S. Iyer, A. Ladd, et al. Freepastry. *Software available at http://www. cs. rice. edu/CS/Systems/Pastry/FreePastry*, 2001.

[12] Z.-J. Han, R.-C. Wang, and Y. Wang. Ppmcast: a novel p2p-based application-level multicast using kamulia. *The Journal of China Universities of Posts and Telecommunications*, 16:114–119, 2009.

[13] L. Matl and T. Cerny. Effective manycast messaging for overlay networks. In *Proceedings of Student Research Forum. 40th Conference on Current Trends in Theory and Practice of Computer Science*, 2014.

[14] L. Matl, V. Kloucek, V. Bohdal, J. Kubr, and T. Cerny. Elisa: Extensible layer for internet services and applications. In C. L. M. S. C. Linger H.; Fisher J.; Barnden A.; Barry, editor, *Proceedings of the 2012 International Conference on Information Systems Development*, Lecture Notes in Computer Science. Springer, 2013.

[15] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[16] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, Aug. 2001.

[18] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, 2001. Springer-Verlag.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17 – 32, feb 2003.

[20] L. Wang and J. Kangasharju. Measuring large-scale distributed systems:case of bittorrent mainline dht. *Proc. of IEEE P2P, 2013*, 2013.

[21] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[22] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2102–2111. IEEE, 2005.

[23] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.