DHT CHAT SYSTEM

Bjørn Elvheim, Jon Berg, Ronny Jensen
{ bjornel, jberg, ronnyj }@stud.cs.uit.no

Abstract:
We describe in this paper the design and implementation of a distributed chat system using DHT overlay networks for routing of messages. Our chat system is censorship-resistant and, being based on DHT, routes messages efficiently. In addition shows two methods for doing search in DHT.

Keywords:
DHT, Pastry, Distributed Chat, Search

1. Introduction

There is an ever-increasing demand for quick and easy communication in today's modern society. Computer users expect to be able to communicate with friends, employees and others with as little hassle as possible. Traditional approaches such as IRC [2] is loosing acceptance, whilst instant messaging services like ICQ and MSN / Windows Messenger are becoming more and more popular. But what all of these have in common, is that they are all based on a central server/servers to provide the service. Therefore they are vulnerable to censorship, being shut down by authorities, DDOS-attacks from hostile users of the internet, and so forth.

The different implementations of chat services deliver varying degrees of anonymity to the users. Today's popular instant messaging-services require users to register with an email-address to be able to use the service. This partially enforces that those who talk to each other know each other beforehand, and also makes sure that the service provider knows who is using their service. Anonymity is therefore not an option with the most popular instant messaging programs. Furthermore, this information can be sold to spammers / advertising companies, and help them specially tailor commercials for the users based on their habits while chatting. While this is very efficient for those advertising, we feel that registering and using data about users for this purpose is a violation of the users' right to privacy. In this respect, we regard this use just as bad as spyware / data miners.

IRC [2] has a little different approach. Users are not required to register before participating in chats with other users. Users thus have an increased degree of anonymity, but it's always possible for other users to extract the IP-address of the machine other users use. In addition, the central servers know the IP-address of every user connected. Unless they use a proxy to hide their IP address. A little different approach is used in i.e. the Invisible IRC Project [1]. But even here, the central server must know the IP-addresses of those connected.

A centralized system can have scalability problems when it comes to the amount of users a system can handle. Scalability in an open distributed system such as DHT is archived by evenly distributing the network and storage load on the users of the system. Making a system on top of DHT is appealing when the system must scale well in number of users it can handle.

Since all traffic in a central server based chat service goes through a central server, all traffic can be easily monitored. This means that governments or the companies that runs the servers can restrict the topics that can be discussed, and also censor and partly decide what users are allowed to talk about. Users can be denied access to the network by those who runs the servers. Additionally, users have little privacy, since all they talk about can be logged on the servers. This can be used against the users. Even if a user only participates in a chat just for fun, and says things he doesn't really mean, it can be registered permanently on the users' record.

By having a distributed system we want to increase the fault tolerance. Tolerating groups of nodes to disconnect without disrupting the functionality of the system would be important. This applies to nodes disconnecting intentionally and when parts of the Internet would suddenly fail. IRC users often suffer when the central server they are connected to goes offline or is split from the rest of the network. This is usually not a very big problem for Messenger / ICQ – users, but it still happens. This is due to central servers being particularly vulnerable to Denial of Service Attacks. To be able to deliver a robust service that is not so vulnerable to DDOS attacks, an obvious solution is distributing the servers, and in addition make sure that each server has replicas that is geographically far from each other.

Our solution to these problems is simply to remove all central servers, replacing them with p2p-clients that also handle the server side job. Using the Pastry DHT implementation as an overlay network, routing is fast and efficient, and scalable. It still requires one or more trusted central servers for the bootstrap-process, but once a node has joined, it may never have to communicate with the bootstrap server again. In our implementation, no single node knows the IP-addresses of more than a small subset of nodes, thus making it much more difficult to find the IP-address of a user. Users decide if they want to supply their real name when entering a discussion, and no central registering is required to use the chat system. In addition, if two or more users feel that they don't want to involve other nodes in routing / chatting, they can start their own private network to ensure that only friendly participants are present. Seeing that our solution is totally distributed, no single service provider has the responsibility to run a central server. This means that no single point of failure exists, and in a big network, it is impossible to monitor the traffic of the network. It is also impossible to deny a node to connect to the network. This is inherently so since it can bootstrap to any node in the network as long as it knows its IP address, even if someone in control of official bootstrap-nodes denied him access to the network. Additionally, logging and registering of user chats is much harder, as each node only serves a small subset of users, and usually only knows the IP-address of a small subset of participants. By making it hard / impossible to monitor which channels a user is logged on to, its exceedingly hard to monitor individual users chat subject and make a profile of them. Also, all users except those that connect as neighbour nodes to the node being monitored are anonymous. By distributing all servers, and replicating them efficiently, it's exceedingly hard to successfully DDOS a distributed network. Using DHT and employing an efficient algorithm for distributing the replicas, the attackers can't tell where the replicas are placed. It does not suffice to DDOS an IP-range, because the ring-topology of DHT ensures that the replicas are placed on geographically distant nodes.

2. Related work

2.1 Searching

Searching in large amount of information is approached in a number of ways. In Google [3] you have a central authority that collects information. Google describes a way of making the result relevant by using the page rank algorithm and various ways to make the architecture scale. Google have shown to be able to index a large number of web pages, but it is still only a fraction of the total number of documents estimated to be available on the web. Google currently have 4.2 billion documents indexed and it is estimated to be around 550 billion documents available. Another approach is to do the searching in a distributed way. A decentralized solution could be more resistant to censorship and manipulated search results. A number of P2P systems implements keyword search by flooding queries to its' peers. Gnutella [4] and KaZaA [5] are example of systems that use this mechanism. P2P indexing and search [6] have many difficulties when it comes to resource constraints in network bandwidth and storage capacity on the nodes. [6] To make a P2P search system within these constraints optimizations like various caching and compression must be done.

2.2 Chat

By now there exists a number of different chat and instant messenger services, like IRC [2]. These systems are normally based on a server – client architecture. The one that owns the server also controls what happens there etc. Also these systems have no natural anonymity, since all communication is done right on top of the internet.

2.3 DHT

In large P2P networks routing can be troublesome. Nodes can not know of all the other nodes since there potentially could be many of them, and a P2P system should try to avoid having servers, since that interferes with the concept of all nodes being peers. Distributed hash table systems are systems that solves the routing in P2P networks by introducing hashing and hash tables. Every node and object in the system has it own id, all ids are inside a key space. The nodes have responsibility for a part of the key space. There are currently many implementations of DHT systems; Pastry [7], Chord [8], Can [9] and Tapestry [10]. These implementations do the routing between nodes based on keys, where the goal is to find the node with id most similar to the key.

3. Design

To implement this chat system we decided to use the DHT implementation Pastry. The Pastry implementation we use are FreePastry 1.3.2 from Rice University. The reason for choosing FreePastry is first of all that it is written in Java, which makes it platform independent and fairly easy to use. It is reasonable good documented, and it supports replication. It is also not bundled together with any other software, like Chord [10].

This implementation consists of several parts. The first part is made up of the user interface, the sending and receiving of messages, and the session concept. This part is what makes it possible to just chat. The session concept, also called group or channel, is the way to let several users chat together. The plan is that the name of the session will be a kind of hint about what the participants in the session are chatting about. The hashed value of the name is the session's id; pastry will use this id to find out which node this session is to be placed on.

The next part is the part which lets users find sessions by the session names, a kind of search functionality. There are two types of search. The first type is based on keywords, and the hashed value of these keywords. The hashed value is the id that belongs to the keyword, and it is used to find out which node to ask.

The second type is more of a flood based type where you ask N number of random nodes. All nodes that rout or in any other way see that search message will also process it.

The third part is the replication functionality. When a session is placed on a node there is no guarantee that the node will stay online. It can go down because of an error, or because the user logs of, either way will the session be lost. Replicating the sessions means that every session is places on more than one node, and replicating the session will not stop them from getting lost, just lower the probability for that to happen, since more nodes has to go down.

3.1 Chatting functionality

This functionality is what makes chatting between people possible. The key here is that all the people that are chatting with each other are part of the same session. In principle all they have to know about the session is the session name. Pastry can with the means of hashing give a session an id based on the name of the session, and then route to the nodes that has the responsibility for that particular id. This node will be responsible for that session. The session has a list of all the participants. When a person sends a message to that session, he or her will be added to that list if not already there. Then the node that has that session will send the message to all the participants in that list. For this to work a GUI has been implemented that lets users read and write to sessions.

Also the user has to have a way to give configuration input to the system, and preferably it will be there the next time the user starts the application. Configuration input is which node and port the application should bootstrap to. The bootstrapping is used to give the new node some information about some of the other nodes in the system, so that the new node can build up its routing table. Also which port on the computer to use, and the user's nickname is information the system needs.

For ease of use, these configurations are stored to a configuration file in the users' home-directory on the computer he is using.

3.2 Replication

In this system there are no guarantees on when or how nodes go offline. Any node can crash or be turned of at any time without warning. If that node is the holder of one or more chat sessions, then they will be lost. To reduce this problem we use replication. Every session is replicated to k other nodes, this means that k+1 nodes are used to store every session. In our implementation k is currently set to 4. Pasty support replication to a certain degree, it can tell the replicating nodes that the root node got a new id to store, and that they should fetch it. The application programmer has to match the id to the correct session. Pastry, as far as we could se, did not help with updating objects that was already replicated; this had to be implemented by the programmer.

3.3 Session search

To join a session you will need to know the name of the session. An important function of a chat system is locating relevant session names for something you are interested in. In a large system there will be a very large number of sessions and these will not be visible to the user because listing what is stored on all nodes in DHT is difficult. For a user to find a session to discuss a certain topic it will be very difficult without searching. The name will give some indication of what the participants in this session discuss. But the actual text typed in the session will give a more accurate indication of what is being discussed in the session. The initial problem with finding something in a DHT system is that there is no support for searching or listing the content that is stored at the nodes. The only way to address an object in the DHT is by its exact name. In our system we have sessions that we access with a name. The process of searching in a DHT is finding a name of a session that is stored at some node that matches a criterion. The problem with this is that hashing is mapping an object directly to another object. Searching can be seen as mapping an object to another object that is somewhat similar to a criterion. In our system we have come up with two approaches to solve the problem with searching in DHT. These two methods are a flood-based querying of session names and a hash mapping of keywords to session names.

3.4 Flood based search

The flood based search is a search type that is similar to the Gnutella protocol in the way that messages are sent to a set of nodes; they process the message and also forward it to other nodes that do the same. The messages have a counter that keeps them from going for ever. Every node that gets this type of search message will compare the search name with the names of all the session this node have. If any such session is found the name of that session is sent back to the original sender of the search message. The return message is not sent the same way as the search message, it is sent directly to the node through pastry. This search is not based on hashing, that means that it is possible to use fuzzy search words, in other terms, search can return session names that are similar but not exactly the same as the search word. When a node sends out a search message it will chose a set of random IDs and send to them. This means that it can happen that the same node receives more than one copy of the same message, if more than one of the random IDs are in that nodes ID space. This is not a problem since the node can just ignore all but the first message. The way to choose who to send the messages to could have been done another way. Every node in pastry has a list of its neighbours both in address space and in distance over the internet. The reason for not using these sets is that the sets of some neighbours are very similar, they overlap, node A's neighbours are almost the same as node B's neighbours if A and B are neighbours. The closer neighbours are, the more the sets overlap. This means that many of the nodes would end up with getting the same message over and over again. Therefore the search message is propagated out in the net very slowly. So using neighbours in flood based searching on top of pastry has it limitations.

In our implementation it is not the receiver of the search message that processes it. It is all the nodes that forward the message on its way to the receiver. The counter that stops the message from going forever is set to 6 in our implantation, when this reaches 0 the message will not be forwarded any more, even if the message has not reached the receiver. When the receiver gets the message it does not do anything with it. Instead of letting the nodes remember who they got search messages from and return the answer to that node, we stores the id of the original sender of the search message within the search message it self. When a node has something that the searcher may be looking for, the answer is sent directly to the searcher. The reason for doing it this way is that it is easier and takes less time.

3.5 Search with hash mapping of keywords to session names

In this method we build a search using the underlying functionality of lookup that a DHT gives. The process of doing this consists of two parts, the building of the distributed index, and the search part.

In building the distributed index, a keyword that map to a session name will be routed to a node responsible for the hash of that keyword. These keywords will act as pointers to the session name. Each node will maintain a list of these mappings. When a node in the system gets one of these messages that sends out a mapping of key to session name it is added to the nodes local list. All words are converted to lowercase to avoid mismatch between words that are the same but with some letters in different case.

Searching for a session name in the system is done by constructing a search-message with a search-word, this message is routed to the node that is responsible for the hash of this search-word. When his node receives the search-message it will search its list of keywords. If the keyword is found a result message is constructed with the mapping of search-word to session name and routed in DHT back to the sender of the search-message.

The initial idea was that when the first user in a session joins or other users join the session they will enters a string they think are relevant and split up into individual keywords. Then the keywords are inserted in the distributed index as explained. After implementing it this way we thought that this was not the optimal way to do things. First it was a bit annoying for the user to must type in these keywords, and second it could be that the content of the discussion after a while drifted in other directions than originally thought when the keywords were typed.

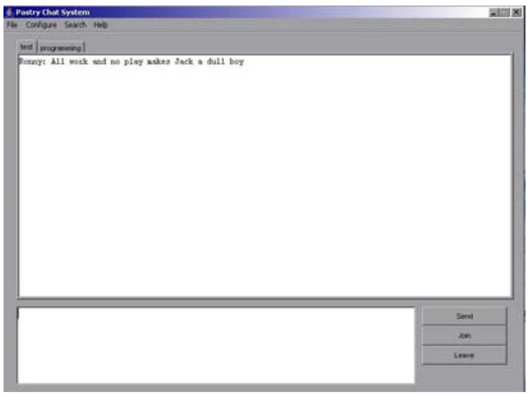
A new concept *"YouTypeWeIndex"* for building the distributed index was then implemented. This way is a more aggressive way of generating the keywords. When the users type lines in a session this text is also used for indexing. For every line a user type in a session it is split up into individual keywords. These are then checked if they are among the 300 most commonly used words in the English language. This is to reduce the amount of keywords like "is", "am" and "was" that will not have any use in searching. All words that pass the common word test will then be used to generate keywords in the same way as before. This will mean that the list that maps keywords to session names out on the nodes will grow as more text is typed in sessions. To avoid the list to grow indefinitely a garbage collection mechanism for deleting these keywords is implemented. When a keyword to session-name mapping is created on a node it will also be marked with the time it was created. On each node a method will be called regularly that goes through the list and deletes the ones that are older than a given time. In our implementation we have chosen to keep keywords for 15 min. In our implementation both the manual entering of keywords and the dynamically indexing while a user types are present.

3.6 Comparison of the two search methods

When the system only have keyword mapping to the general topic as original thought this limits the search to synonyms or related words for a topic. When we generate keywords dynamically while the user types we extend the search functionality with some content aware search capability. When searching for sessions the user is really interested in a search that reflects the actual things being discussed. A session that discusses completely different things than what was set as keywords and topic is not as relevant.

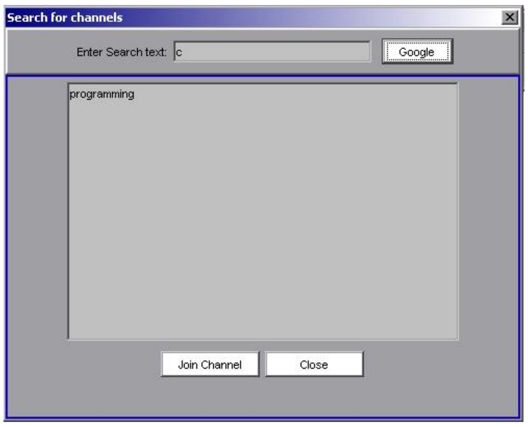
In hash mapping of keywords to session names the generation of keyword will generate a lot of messages while building up the distributed index's on all the nodes. Potentially all words typed in a session must be sent out to different nodes. Processing and routing of this will of course demand some processing, memory and network consumption. But on the other hand searching will be very effective, this only involves one message for a search-word and one reply-message that must be routed by DHT. Processing involved is only searching the list on the node that receives the search message. If a keyword exists for a topic the user that search for this is guaranteed to find it.

In flood based search there is no initial setup cost to prepare the nodes in the net. The actual search involves a lot of messages when sending the search query out. The nature of flooding is to reach out to as many nodes as possible while limiting the search by a hopcount variable that is decremented each time a jump is done to another node. The number of actual messages sent out is given by how many random nodes you send out to at each node and the initial size of the hopcount variable. This method does not give a guarantee to find what you search for if it indeed exists because of the random nature of the algorithm and that the limitations of how many nodes it is forwarded to. The flood search will give results on keywords that only partially match, ex. car will match cars.



Ronny: All work and no play makes Jack a dull boy

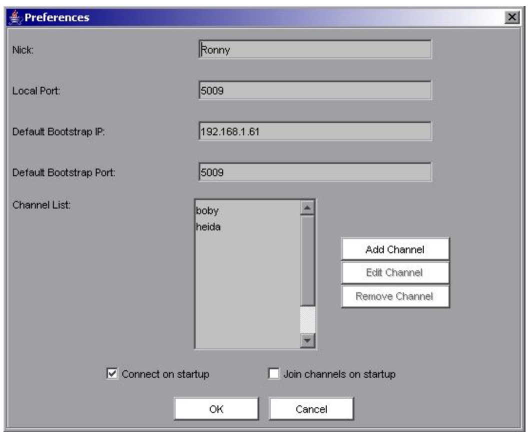
Fig 1. Screenshot of the main chatting window.



Enter Search text: c

programming

Fig 2. Dialog for entering search queries and displaying search result.



Nick: Ronny

Local Port: 5009

Default Bootstrap IP: 192.168.1.61

Default Bootstrap Port: 5009

Channel List: boby, heida

[✓] Connect on startup

[ ] Join channels on startup

Fig 3. Configuration dialog.

4. Future work

Some work in implementing the system still remains. Replication of the list that stores keyword to session name is not implemented. This would be done in roughly the same way as the replication of sessions is now done. What happens now without replication of this list is that when nodes disconnect the data in the list is lost. This means that some keywords can arbitrarily be deleted before the garbage collection was supposed to delete it. We would implement the flood-based search to also do search on the content not just the topic. This could easily be done by doing it very much like the hash mapping of keywords to sessions does it by maintaining what has been said in a sessions for some time and searching this text. With a chat system that is very open it is desirable to have a way of blocking or banning people that behaves badly in a session. This could be done by having some sort of voting mechanism. If a majority bans this user his messages will not be forwarded. Another way could be to have superuser in each session that can to this. Now there is no way of seeing who are on the list of participants in a session. A GUI part with this information would be useful. Additional extensions of this system could be support for voice or video conversation, sharing of files and multi user drawing.

5. Conclusion

In this article we have proven that it is possible to build a chatting system on top of a DHT based overlay network. In addition we have also proven that it is possible to implement search functionality in DHT based systems. We have implemented two types of search, one based on keyword hashing and one based on flooding. They both have pros and cons, and their use will vary with the situation. One drawback with the flooding search is that it actually reduces the anonymity in the system, since a node can see who is searching. Anonymity is one of the things we have said that the system should support, but since the lack of this comes from our programming, and not from the concept of DHT, we do not se this as a major drawback. In other words it can be fixed. We are not in a position to tell which type of search works best, since we have not tested them thoroughly.

Another problem that arises with p2p is that nodes can come and go as they want. This problem is addressed with replication. We believe that the problem is not removed with replication, but it is decreased. How much it is decreased will vary with how many replication nodes one use. In this application there are 4 replication nodes, this mean that there are 5 copies of every session in the system. If a session is to disappear, all 5 nodes storing that session have to go down before a certain time. That time is the time it takes from the first node goes down until the systems finds that out and starts replication to another node. This time is set in the system, and in our implementation it is 30 sec. It is our opinion that the number of replicas and the time between replica update is enough to reduce the chance of loosing session considerable. This has only been tested superficiality, so the benefits are only what we think.

6. References

[1] [Online] http://sourceforge.net/projects/invisibleip/

[2] [Online] http://www.irchelp.org/irchelp/altircfaq.html

[3] The Anatomy of a Large-Scale Hypertextual Web Search Engine. Sergey Bin, Lawrence Page. In Proc. of the Seventh World Wide Web Conference, 1998. http://www-db.stanford.edu/pub/papers/google.pdf

[4] The Gnutella Protocol Specification v0.4. Clip2. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[5] [Online] Kazza. http://kazaa.com/

[6] On the Feasibility of Peer-To-Peer Web Indexing and Search. Jinyang Li, Boon Thau Loo, Joe Hellerstein, Frans Kaashoek, David R. Karger, Robert Morris, On the Feasibility of Peer-to-Peer Web Indexing and Search, 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Feb 2003. http://www.pdos.lcs.mit.edu/~rtm/papers/search_feasibility.pdf

[7] Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, A. Rowstron and P. Druschel. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001 http://research.microsoft.com/~antr/Past/Pastry.pdf

[8] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160. http://www.pdos.lcs.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[9] A Scalable Content-Addressable Network, Sylvia Ratnasamy Ph.D. Thesis, October 2002
In Proceedings of ACM SIGCOMM 2001
http://www.icir.org/sylvia/thesis.ps

[10] Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Ben Y. Zhao, John Kubiatowicz and Anthony Joseph
UCB Tech. Report UCB/CSD-01-1141