A bandwidth study of a DHT in a heterogeneous environment

Olof Rensfelt, Lars-Åke Larzon
IT Department
Uppsala University
olofr@it.uu.se, lln@it.uu.se

Abstract

We present a NS-2 implementation of a distributed hash table (DHT) modeled after Bamboo. NS-2 is used to evaluate the bandwidth costs involved in using a DHT in heterogeneous environments. Networks are modeled as mixed networks of desktop machines and 3G cellphones. We also document the modifications of NS-2 that were needed to simulate churn in large networks.

1 Introduction

In the design of distributed applications, there has been a strong trend during the last decade to use the Internet mainly for connectivity and build an *overlay network* with its own node identifier space on top of IP. This effectively deals with problems that could otherwise occur due to dynamics in IP address allocations. By not using the IP address as an identifier for the service itself, the service can continue to function as long as Internet connectivity is maintained even if IP addresses change over time.

A common approach to introduce a new identifier space is to use *distributed hash tables* (DHTs). A DHT is a distributed data structure that functions much like an ordinary hash table, except that the key space is distributed over several nodes rather than kept together at one single node. When querying a value in a DHT, the query is routed to the node that maintains the corresponding part of the key space. Nodes continually exchange data to keep track of how the responsibility is divided among them. Most DHTs include some degree of replication to deal with nodes that may disappear without prior notice.

Most existing evaluations of DHTs are done using simulators written for that specific purpose to enable proper simulation of the data structure itself with a focus on how queries are carried out. Modeling of the communication between nodes that collaborate in a DHT tend to be quite simplistic at best - sometimes it is assumed that all messages sent are instantaneously received by the receiver. While this assumption can be argued to be a reasonable simplification in an environment with fast computers communicating over a fixed Internet connection with high bandwidth, it does not hold for more heterogeneous network environments.

In a prior study of Pastry [2], we found that the management traffic needed to keep a DHT functioning is non-negligible. It is therefore motivated to study what happens in a DHT network when introducing it to a non-ideal network environment consisting of nodes with large variations in terms of connectivity and performance. In this report,

we study how the widely used DHT implementation *Bamboo*[11] behaves under such conditions. To be able to use a more detailed networking model, we have implemented Bamboo in the NS-2 network simulator and conducted experiments that involve both mixed types of nodes and unpredictable network dynamics.

2 Overlays

A recent trend is to use overlays to deploy functionality that the existing network infrastructure does not provide. An overlay network is a logical network built on top of the existing network with its own addressing scheme using the Internet as a link layer. Building an overlay network makes it possible to deploy functionality not available in the current Internet architecture, or to create services that are provided by the users of the service. A user provided service is for instance BitTorrent where a group of users cooperate to provide efficient mass-distribution of large files. In some sense the users pay for the service by participating in the overlay. The common use of overlays is as the communication module of an application, sometimes referred to as BYOI (Bring Your Own Infrastructure). BYOI has proved very successful in file-sharing applications and in some sense in VoIP with Skype. Parts of the research community are however suggesting to have overlays as services for multiple applications to share, provided by companies [1]. The benefit would be that the price of management overhead can be shared among more participants. Also, such a service could be assumed to be more stable than a service where only the users collaborate to provide the overlay. Another benefit, or draw back, depending on conviction, is that a payment system needs to be added to the system because users no longer pay for the service by participating in it.

3 Distributed Hash Tables

A common service provided by overlay networks is a lookup service handling flat identifiers with a ordinary query-response semantic. Such a service is often implemented using DHTs (Distributed Hash Tables) [9, 16, 13, 18, 11] . A DHT allows you to insert values connected to keys much like ordinary hash tables. A key is typically a hash of the value stored or alternatively a hash of some meta data of the value. When the key is inserted it is routed through the overlay network until it reaches the node that is responsible for storing the key. The key can later be used to retrieve the value from the DHT.

The flat address structure often used in overlays, and especially DHTs, is appealing for cases when you want addressing differentiated from your physical location in the network. Such a differentiation can for instance be a building block in systems supporting mobile nodes [15] where identifiers should remain the same regardless of the location of the node.

Despite the flat address space structure on the DHT level, it is still possible to add some form of hierarchy in the application. E.g in [17] we embed the geographic location of information in the key itself. Other have also built hierarchy on top of a DHT [3].

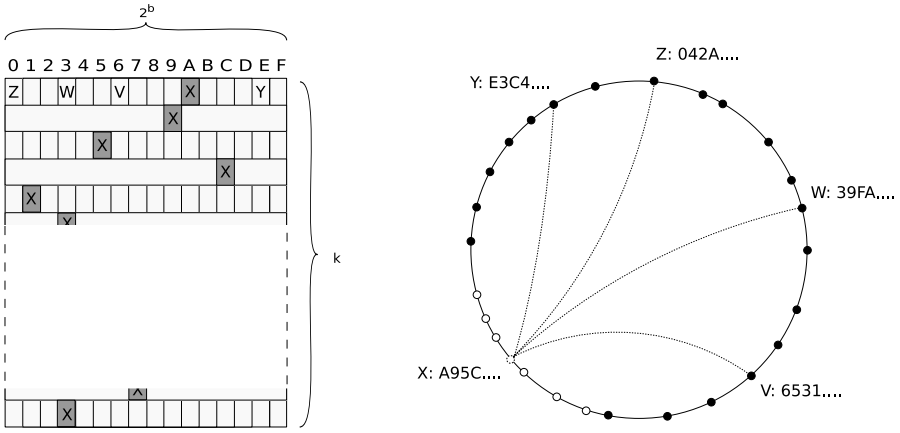


Figure 1: The routing table. The white nodes are the middle white node's leafset if the leafset size is configured according to l=3. The dotted arcs show the routing table entries

3.1 Bamboo

Bamboo is a DHT implementation first presented in [11]. It is referred to as a third generation DHT, where lessons learned from previous systems have been incorporated in the design. The Bamboo implementation has proved stable when used in OpenDHT [12] ,where it serves a system with good uptime.

To continue the earlier studies[2] of how an overlay network behaves in a heterogeneous environment, we chose to implement a DHT in NS-2[5]. We believe that a lot of the problems seen in [2] is addressed with Bamboo. For example, the problems encountered with Pastry in heterogeneous networks were mainly caused by management traffic congesting nodes, and a new approach to management traffic were presented in [11].

3.1.1 Network structure

Bamboo uses the routing logic of Pastry but has more developed mechanisms for maintaining the network structure in a dynamic environment. A big part of network dynamics is that nodes leave and new nodes join the network, which is called churn. Bamboo maintains two sets of neighbor information in each node (figure 1). The leafset consists of successors and predecessors that are the numerically closest in key space. When routing a query, it is forwarded to a node which has the key in its leafset. Using the leafset is enough to ensure correct lookups. However if only the leafset was used when doing lookups, a lookup complexity of $log(n)$ is all that could be achieved. To improve the lookup complexity, a routing table is used. The routing table is populated with nodes that share a common prefix, and routing table lookups are ordinary longest prefix matching.

The major difference between Pastry and Bamboo is how they handle management traffic. In Pastry, management is initiated when a network change is detected, while in Bamboo all management are periodic regardless of network status. The approach to use periodic updates has been showed to be beneficial during churn [11] since it does not cause management traffic bursts during congestion. Such traffic bursts can further

increase network disturbances.

The Bamboo system has been evaluated both in simulation and as a deployed system on PlanetLab[4]. However the evaluations have not taken bandwidth or other node specifics into account, only network delay. This is not a major problem if you want to evaluate scalability and lookup delays in noncongested networks. The nodes in PlanetLab are typically very strong machines on academic or other types of very stable, high bandwidth networks, and therefor they are not suited for studying the scenario we are investigating.

3.2 Management traffic

In order for a DHT to be able to serve requests and maintain a consistent network view among its nodes, it needs to perform network maintenance. This maintenance consists of network messages sent between nodes. In this section we will describe the different types of maintenance performed by Bamboo. Periodic management traffic occurs in all layers of the Bamboo system (figure 2). In the data transfer layer, ping messages are used to measure RTTs (Round Trip Times) to peers. Routing table and leafset information are exchanged and databases are synchronized. We have used [11, 10] as design documents as well as the Java source code from [6].

3.2.1 Neighbor ping

The most basic management traffic type is to make sure that you can still reach your one-hop neighbors in the overlay. This is normally done with an echo/reply type of communication. In Pastry it is called *probes*, and other systems have the same function with different names. The messages sent are not ICMP pings but UDP echo and reply packets. The major design decisions regarding neighbor pings are the interval which is used to ping and the number of unanswered pings that should cause a node to treat a neighbor as unreachable or, as in Bamboo, as possibly down. In Bamboo the neighbor pings are also used to maintain a RTT estimate used for retransmission timeout calculations.

The reason why UDP is the preferred transport protocol in Bamboo is that the overhead of connection oriented communication does not justify the benefits of reliable transfer. A DHT also has a non symmetric nature regarding neighbor knowledge between nodes, meaning that the fact that node A has node B in its neighbor set does not necessarily mean that node B's neighbor set include node A. Because of this asymmetry the number of nodes that know a certain node will increase with the network size. If TCP is used as the transport protocol, the state that a node needs to keep increases significantly as TCP needs both the receiving and the sending nodes to keep state information. A DHT could benefit from using a transport protocol with properties like DCCP [7] as mentioned in [11]. DCCP offers a UDP-like, nonreliable datagram transfer with congestion control.

3.2.2 Leafset updates

Changes in node leafsets are propagated using an epidemic approach. Every node periodically chooses a random node from its leafset and performs a leafset push followed by a leafset pull in response. Both messages involve sending the complete leafset to the synchronizing node where the information is incorporated. It is important to both

push and pull leafsets. Otherwise there might arise situations where nodes are missed in the leafsets of its neighbors [10].

3.2.3 Local routing table updates

When a node has another node in its routing table, those two nodes per definition share one level. The local routing table updates are used to exchange the node information in that level. If a node gets information about other nodes that fits into the routing table it probes the nodes to test reachability and to get a RTT estimate. If a node is reachable and fits into an empty field in the routing table, it gets added. If the matching routing table entry is occupied, the node with the lowest latency is chosen. Other optimization schemes could be considered, such as optimizing for uptime, but optimizing for latency is the most common approach used. Having an optimized routing table does not influence lookup correctness, only lookup latency.

3.2.4 Global routing table updates

Local routing table updates can only improve routing table levels that are not empty. To improve that, you need to exchange routing table information with nodes that you do not yet know of. To find such nodes the routing functionality of Bamboo is used. To optimize a certain routing table entry, a lookup is made for a key which shares prefix with that entry. If a suitable node exists in the network the request will be routed to it, and that nodes is a candidate for the routing entry. Unlike with local updates, global updates can be used to optimize a specific routing table entry.

3.2.5 Data storage updates

When data is stored in the DHT using the PUT command, the data is routed through the DHT to the node primarily responsible for storing the data. When the responsible node gets the data, it caches it within its leafset at 'desired_replicas' neighbors in each direction. The caching does not occur immediately, but is performed by the periodic replication functionality described below. The value 'desired_replicas' is a configure parameter, and with the default settings there are 7 copies of the data within the system. When nodes disappear or joins, the subset of nodes that should store a certain value changes. Therefore there is a need for a mechanism to try to restore the distributed storage to the wanted state. The default setting of 'desired_replicas', and the resulting 7 copies of each data units within the system, causes demands for storage space. If all nodes have equal amounts of keys to store, every node needs to store seven times that amount.

The first maintenance operation made is that a node periodically picks a random node in its leafset and synchronizes the stored keys with it. A synchronization operation starts with a node picking a node to synchronize with and requests a synchronization. The other node calculates the set among its stored keys that it believes should also be stored at the initiating node and send those keys and the hash values of the data. The other node receives the keys and hash values, and matches them to what it has stored. If a certain data unit received is not already stored it requests that data unit from the initiating node.

The second maintenance operation performed by the data storage layer is to move values that are not longer within a nodes storage range. If a node has such a value stored, it performs a new PUT to the place it should be stored before deleting it.

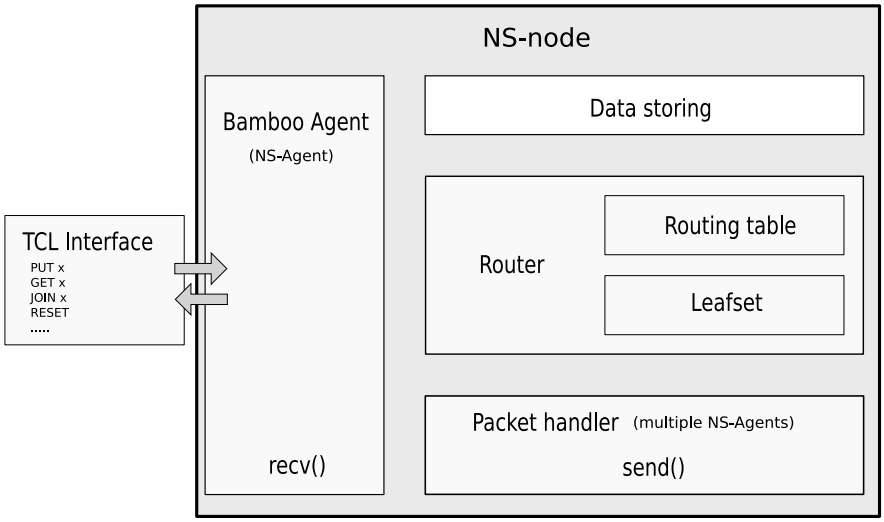


Figure 2: Block diagram of the Bamboo-NS2 implementation

4 Implementation

We have implemented a DHT in NS-2 [5] and, in what we believe to be the relevant properties, made it as similar to Bamboo as we could. However, since we did not run the Java code in simulation, differences might exist that we are not fully aware of. We will state the known differences when describing the different parts of the system. During the implementation work we have used the technical report [14] as a reference as well as the source code, and later the doctoral thesis[10] when it became available. In the following text we will refer to our implementation as Bamboo-NS2, and the original implementation as Bamboo.

The NS2 implementation consists of multiple modules that are constructed to fit the design of NS2, rather than the design of Bamboo (figure 2). There are however many similarities between which modules Bamboo and Bamboo-NS2 are divided into.

4.1 NS-2 specifics

To be able to simulate big networks, we needed to make some simplifications. One simulation specific method is that we have the possibility to build the overlay network before the actual simulation starts. We will refer to this as building the network offline. In section 5 we will further discuss how this influences the evaluation.

As previously mentioned we have not implemented storage of real data in order to save memory, and instead of a faked hash value we use a globally unique id on every data item that exists in the DHT. Since we have control of all data that is inserted into the DHT, we believe this to be a valid approach.

When we started to simulate churn, we ran into some problems with memory leaks trying to free NS-2 objects. This lead us to reuse the same NS-agents with multiple overlay nodes. First we tried to have multiple NS-nodes for each overlay node, so that when an overlay node went down and a 'new' overlay node came up, it came up on a different NS-node. The reason that we did not simply use the same NS-node for the new overlay node is because of the node information about the old node that is

still in the system. This would cause a new node to receive traffic meant for an old node which would take up link bandwidth. We call this kind of traffic 'stale traffic'. We did not want to filter out traffic to no longer active nodes at the sending node, because in a real life deployment there is no way of knowing whether a node is active or not. The approach with multiple NS-nodes meant that we needed to simulate much bigger networks since many more physical nodes than overlay nodes where needed. Even when we used three physical nodes per overlay node stale traffic still turned up at newly joined nodes. Therefore we needed to find an other method of getting rid of stale traffic.

The second method involved giving every overlay node another globally unique id (GID), apart from its overlay address, and introducing a directly indexed lookup table with connection status. Then we modified the NS2 routing function to compare next hop IP from the routing logic to the end destination IP of the packet, and if they are equal it makes a status lookup to see if the destination overlay node is active. If it is not active the packet is simply dropped after it has been logged as stale traffic, and will therefore not stress the last hop link of a new node.

4.2 Packet handler

The packet handler at a Bamboo-NS2 node consists of a list of known neighbors. Bamboo implements reliable transfer on top of UDP, using acknowledgments which are also used for RTT measurements. If traffic is not flowing between nodes, periodic probes are sent to keep the estimated RTT accurate. In Bamboo-NS2 we use the NS-2 class agent, which we connect between nodes. Agents are closest matched by UDP sockets in Bamboo. To keep the memory usage low, we connect agents dynamically when needed. We encountered problems when we tried to free memory after the agents were not needed anymore. A workaround was to implement an agent pool, which we could request agents from in order to reuse them. An agent pair is only used to send data one way, because there where implementation benefits from having all traffic to a node go through one agent. We call the sender-side agents *bamboo_send_agent*, because they are of a different class compared to the receiving side type described in 4.4.

We did not use cumulative acknowledgments since we did not want to keep state at the receiver for every node that communicates with us. We do however need to keep a *bamboo_send_agent* for each node we communicate with, so the benefit of not using accumulative acknowledgments is limited. In a real deployment, the approach would be more beneficial.

4.3 Router

The Bamboo-NS2 router consists of three modules; The routing table, the leafset, and the routing logic. The routing table consists of information about nodes spread over the key space, as well as functions to maintain and lookup node information. When we use the term node information, we refer to a structure which apart from a key value also consists of information of the network connection point of the node.

The leafset consists of ordered node information about the numerically closest nodes in key space which are the white nodes in figure 1, and functions to insert and remove nodes from the list. As previously mentioned, the routing table works like in Pastry. The routing table and leafset are used by the routing logic to lookup the next hop node when a key is looked up. When a routing request of a key is made to the routing logic, it first checks whether that key falls within the leafset. If the key is within the

leafset, the numerically closest node is found, and the nodes information is returned as the next hop. If the looked up key is not within the leafset, a request to the routing table is made, which returns the closest node outside the leafset. If no such node exists, the next hop node is the numerically closest node of the two leafset nodes that are furthers away, and then the information about the closest node is return by the routing logic.

4.4 Agent

The Bamboo-NS2 Agent is both the listening agent in NS-2 as well as the interface to the TCL scripts used to run simulations. It is the connection details for the listening agent which is spread through the network for other nodes to connect to.

From the TCL script that defines the simulation, the behavior of the Bamboo-NS2 node can be controlled. You can set the word and key length, make PUTs and GETS, connect and disconnect etc. It is in the listening agents $recv()$ function that all incoming traffic to a node enters. If a new packet is an acknowledgment, the packet handler is called to remove the acknowledged packet from its buffer, as well as to calculate a RTT estimate. If the packet is not an acknowledgment the packet handler acknowledges the packet and checks whether it is a new packet or not. If it is a old packet or a PING the only action taken is the acknowledgment. If it is new packet, it is sent to the router to calculate the next hop and generate a new packet to send. If the next hop returned by the router is not *null* and not the node itself, the agent sends the new packet to the next hop node with the help of the packet handler module.

When a Bamboo-NS2 node is connected to a NS-2 network node, and it has joined the overlay network using the join command to the agent, PUTs and GETs can be issued to the agent from the TCL script. A PUT takes a key, an id, and the data size as arguments. The key is where the value is stored, the id is instead of an hash of the data, and the size is how big the data is. No actual data is put into the system but the size field is used to set the correct size of network packets during simulation, and the id is used to distinguish between different values. The GET command takes the key value requested and records the time. If a GET matches multiple values in the DHT only one is returned. This is not how Bamboo behaves; Bamboo would return values together with a pointer. The pointer can be used to retrieve the remaining values that matches the GET with repetitive GETs.

To support different measurements of the system, two different GET behaviors are implemented. The first is the one resembling Bamboo with keys stored and cached, as is later described in the section on the data storing. The second is a special GET where you lookup exact nodes in the network to evaluate the pure routing functionality of the system without the noise of key management.

4.5 Data storing

The data storing module in our system does not implement all the functionality present in Bamboo. The synchronization between nodes is initialized by a node when it sends a list of its keys to another node. The receiving node builds a list of the keys in the received message it does not have, and sends that list to request those keys. Keys in the systems have a TTL, but that is a function we do not use during our tests. A good study of the storage problem is [10].

In Bamboo an improved synchronization method is used. It is based on Merkle trees [8] and it involves building a tree of hash values over the stored key values. The best case for this method is when the nodes are completely synchronized, which will

result in the need to exchange one hash value to determine that. According to [10] the worst case of the Merkle tree approach is only $O(n)$, were n is then number of keys. However, there is no evaluation of the time aspect of synchronization.

4.6 Other differences to Bamboo

Bamboo uses a concept of possibly down nodes. That is nodes that have not responded to 4 succeeding pings. The set of possibly down nodes are still periodically pinged with a greater period and are considered unreachable. If a node in the set answers to ping it becomes a known neighbor again. A big advantage of this is that it can rejoin a partitioned overlay network. If for instance the connection between two continents is cut off, two different overlay networks will be formed and their knowledge of each other will fade away with 4 succeeding pings. With the addition of possibly down nodes, that you keep trying to reach for a long time, the partition of the network can be healed. We have not implemented support for treating nodes as possibly down in our implementation, since we have not been interested in studying the influence of the intermediate network on the overlay, only to study the influence of connection technologies.

Our implementation does not handle multiple PUTs to the same key in the same way as Bamboo does. However, we believe that for the sake of evaluating the performance in heterogeneous environments, the benefit from such a complete implementation is limited, compared to the need for it in a deployed system.

5 Evaluation

To evaluate the system in heterogeneous environments, we have set up scenarios in NS-2. In this section we will first describe the simulation setup, then describe our evaluations of the impact of different network variables and present the results of each evaluation.

5.1 Physical network layout

The physical network layout used in our simulations is modeled with the nodes in clusters connected with very high bandwidth links with long delays (figure 3). The reason not to use a more advanced topology created by a topology generator is that we wanted to keep the variables influencing the simulation as static as possible. By using the same delay on the links between the clusters, we have had an easier task to realize their influence on the total delay on for example lookups. We used very high bandwidth links between clusters so that they would not introduce packet loss, but only delay. Overlay nodes are not connected to the cluster nodes, only to the NS-nodes, with links into the cluster nodes. The node characteristics are set at the last hop link into the cluster.

5.2 Overlay network layout

The overlay is built “offline” in order to have a fixed, well known starting state. During the building of the network every node has knowledge about every other node. This will create a network where a node has as many nodes as possible in its routing table. The routing table is however not optimized for proximity since RTT measurements are

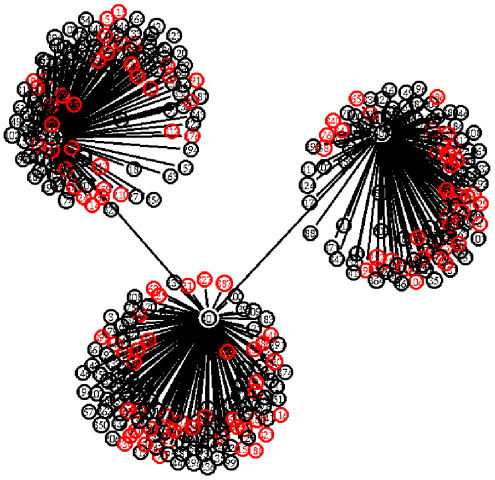


Figure 3: A NS-2 network layout with 3 clusters presented using NAM, where the clusters model continents

	Strong link	Weak link	Link between clusters
downlink	10 Mb	384 Kb	100 Gb
uplink	10 Mb	64 Kb	100 Gb
delay	5 ms	115 ms	50 ms

Table 1: Physical network specifications

not done before the simulation starts. When the simulation starts, a node pings all the nodes it has in its routing table and leafset. This causes an initial burst of traffic that needs to be taken into account. We decided to not collect data until the system was stable.

5.3 Stabilization time

The fact that we build the network offline indicates that we start the simulation in an unrealistic state. The main factor of initial instability is that nodes need to ping their neighbors in order to calculate RTTs. To study how long it takes for the system to stabilize we periodically performed GETs on a stable system and then plotted a moving average of the lookup times. The stabilization time is of course dependent on management traffic settings, as well as overlay network behavior, but we decided to use the settings from [14] since they were tweaked for a system under churn(table 2). From figure 4 we decided that that the initial 80 seconds of the simulation should be considered start up time. We also looked at the simulation runs with churn and we concluded that 80 seconds still seemed to catch the initial turbulence (figure 5).

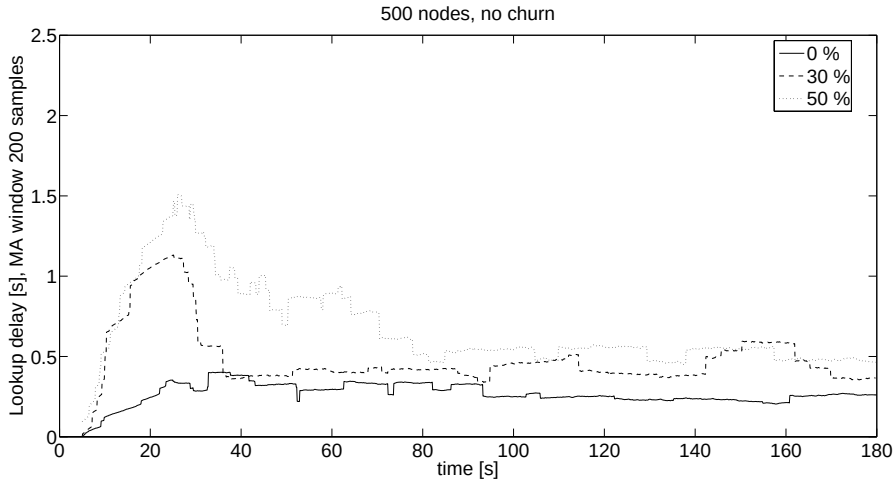


Figure 4: Smoothed lookup delay over the time of simulations for different percentages of weak nodes

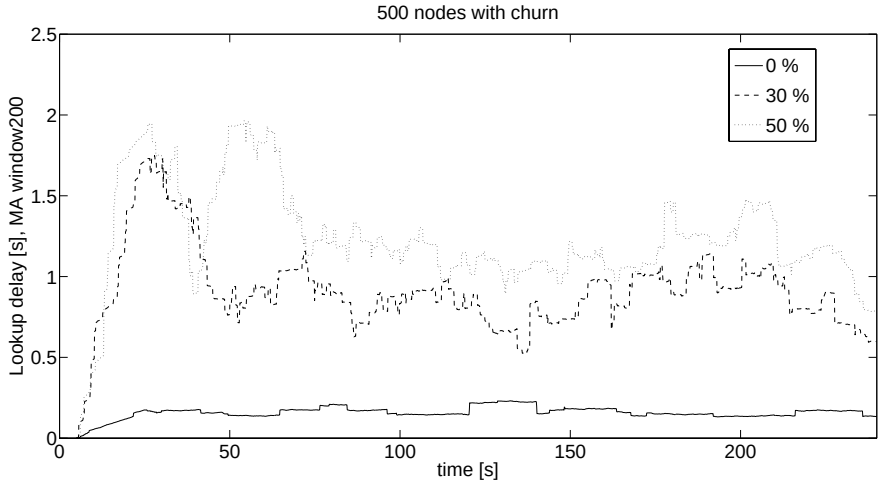


Figure 5: Smoothed lookup delay over the time of simulations for different percentages of weak nodes with churn

5.4 Measurements

Making measurements of a DHT is not as straightforward as it might first seem. The first problem is that the measurement traffic influences the systems performance by adding extra load. It is on the other hand not a realistic scenario to have a DHT without lookups that influence performance. There are two directions we could have taken with the lookups. One way is to try to model store and lookup traffic realistically, for instance by using a stochastic process that causes bursts in network utilization. However, the bursts would make analysis of lookup delays harder as it would be hard to compare two different samples in time. It would be hard because of the difference in measurement environment. We choose to use a periodic probing scheme to simplify

	technical report	OpenDHT
Neighbor ping period	4	20
Leafset maintenance	5	10
Local routing table maintenance	5	10
Global routing table maintenance	10	20
Data storing maintenance	10	1

Table 2: Management traffic periods in seconds

the analysis of the data.

In the tests performed on Bamboo in [11, 10], a majority procedure was used to decide if a lookup was successful or not. 10 nodes requested the same key, and if they received different answers the minority was considered to be wrong. Since we have global knowledge in simulation we have used single lookups. Another problem with deciding on success or failure is whether to use a timeout. If a timeout is used you will remove information about how lookup times are distributed and move it to the failure statistics. With a very long timeout you will get a high success ratio but a higher mean lookup time. In [12] 60 minutes is used, but we have set a timeout of 60 seconds, since we do not believe that a lookup that exceeds a minute can be considered a success.

The next decision to make about the lookup is whether you should lookup nodes or keys. If you make lookups aiming at nodes, you do not need to introduce the extra complexity of a data storing system. On the other hand you need to make sure that the requested node is available for lookups during the right time. If you do not, you might decide that a lookup has failed when there is no way of success. Having to take transit lookups into account when simulating churn complicates matters. Therefore we use the data storing to allow us to simulate churn more freely and we make lookups for keys rather than nodes.

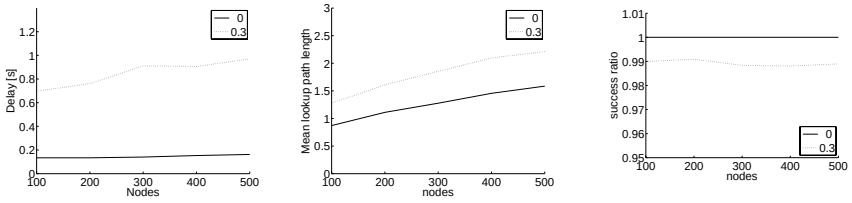
5.5 Simulation specifics

Simulations were made with management traffic according to [11], where churn was targeted, as well as with settings matching the ones used in the deployed DHT service OpenDHT [12]. During simulation, 10 of the strong nodes were used as bootstrap nodes. The first scenario used is that nodes are distributed over 3 clusters as seen in figure 3. The links between the clusters are modeled as having extreme high bandwidth but with a intercontinental delay. The nodes are connected to one of the clusters with a link that either is a 10Mb/s, low delay link (strong node) or a link with specifications according to measurements made of 3G connectivity (weak node). The weak nodes has a down link bandwidth of 384 Kb/s, an uplink bandwidth of 64 Kb/s and a link delay of 110 ms. Weak nodes are uniformly distributed over the network.

With the choice of NS-2, we sacrificed the possibility to study large networks (more than approximately 500 nodes), but it does allow us to simulate link bandwidth and link queue drops.

6 Network variables

There are multiple variables that influence the characteristics of the network. In a dynamic overlay, these variables will change over time, but in order to study how they



(a) Lookup delay (b) Mean lookup path length (c) Success ratio

Figure 6: System performance as a function of network size, with 0 percent and 30 percent weak nodes

influence network performance we have kept them fixed during the course of one simulation. We will present the simulation setup and result for each network variable.

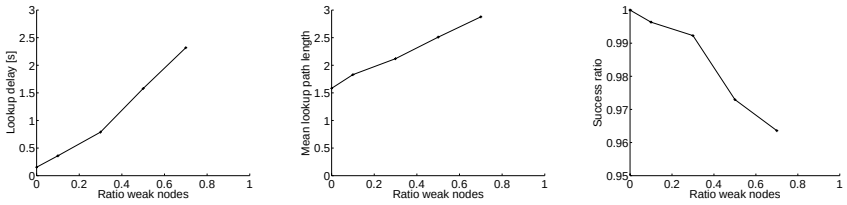
6.1 Size

The network size is the number of participating nodes at a measured time. How the size of a DHT impacts performance is evaluated previously, both in simulation and on testbeds using emulation[13]. We only study how size influences the network up to 500 nodes. The reason for varying the size in our initial simulations is to justify our decision to use a fixed network size of 500 nodes when studying bandwidth usage. The lookup path length complexity of $O(log(n))$ ensures good scalability properties. The results from the simulations are presented in figure 6 where we use lookup delay, lookup path length, and lookup success ratio as measures of the systems performance.

From figure 6(a) we can conclude that the added weak nodes, and the resulting churn, affects the lookup times much more than the size of the network. The lookup delay for networks with only strong nodes and no churn is only marginally affected by size, which might seem non-intuitive when figure 6(b) shows a increase in lookup path length. We believe it to be caused by the routing table being optimized for communication latency in combination with how we model the core network. When communication within a cluster is very cheap compared to between clusters, and when the routing tables are optimized for network proximity, an extra overlay hop might not increase the total lookup delay significantly. For instance in the simulation with 500 nodes, where the nodes are randomly distributed among the three clusters, every node should have more than three candidates for each top level routing table entry. With three candidates per top level entry, on average one of them should be in the same cluster and thus chosen when the routing table is optimized.

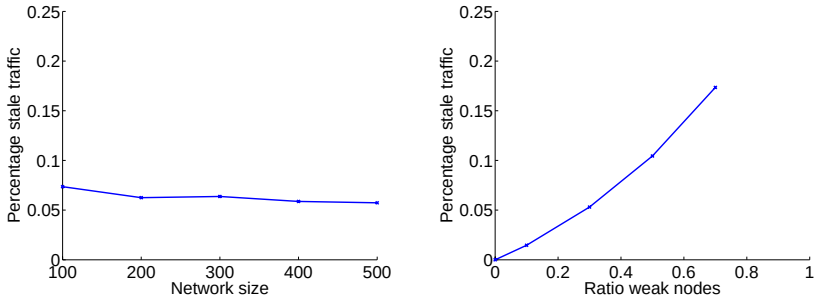
When weak nodes are introduced a small increase in lookup delay can be seen (figure 6(a)) but when the size of the network reaches 300 nodes it levels out. We believe it to be caused by the same mechanism as in the case of a static network. The information of the weak nodes does not spread through the network fast enough to make a big impact, and even when the information reaches other nodes it is unlikely that a weak node is the best candidate in a routing table.

In figure 6(b) we can see that having weak nodes in the network increases the mean lookup path length. Since the latency and bandwidth of links should not influence lookup path length, the difference is probably caused by the introduction of churn in the network. Churn causes routing tables to be non optimal, which should cause increased lookup path lengths.



(a) Lookup delay (b) Mean lookup path length (c) Success ratio

Figure 7: Influence of heterogeneity on system performance in a 500 nodes network



(a) Stale traffic vs. size (b) Stale traffic vs. ratio of weak nodes

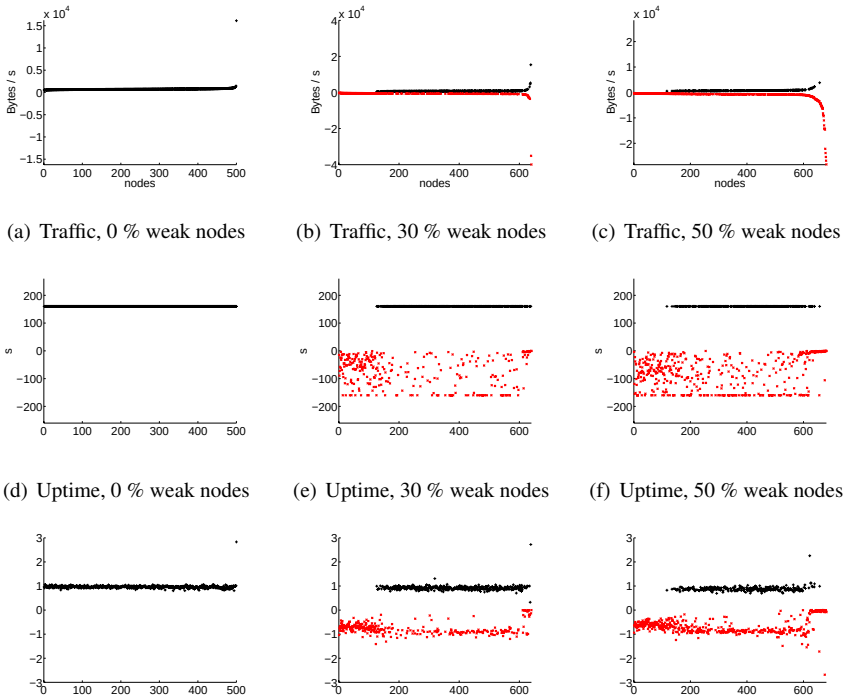
Figure 8: How percentage of stale traffic depends on size and ratio of weak nodes

Finally we we can see from figure 6(c) that the success ratio of lookups is constant 100 % for a static network which is what should be expected in a noncongested network. We use a low request rate so the network is not congested during these experiments.

6.2 Node capacities

A common assumption, both in simulation and in real world tests, is that all nodes are created equal. That assumption does not follow the trend of networks where the heterogeneity increases. We choose to make a simplification of network heterogeneity by introducing what we call weak and strong nodes. A weak node is modeled from a UMTS cellphone, as such phones are probably the first mobile devices that it makes sense to have as members in an overlay. The strong nodes are modeled from desktop computers with broadband connections. We use the term ratio to describe how many percent of the nodes that are weak.

In these simulations we keep network size fixed and vary the ratio of weak nodes. More weak nodes does not only lead to more weak links but also to a more dynamic network. A more dynamic network increases the risk of lookups being lost in transit. Failed lookups have two different reasons. First a lookup can be lost if a node leaves the network while the lookup is routed through it. Second a lookup fails if it reaches the destination node when that node has recently joined and the destination nodes data storage has not yet been synchronized. Bamboo has caching optimizations but we have not implemented them because we believe that they hide the true performance in an experimental evaluation. Nevertheless they make complete sense in a deployed



(a) Traffic, 0 % weak nodes (b) Traffic, 30 % weak nodes (c) Traffic, 50 % weak nodes

(d) Uptime, 0 % weak nodes (e) Uptime, 30 % weak nodes (f) Uptime, 50 % weak nodes

(g) Send / received , 0 % weak nodes (h) Send / received , 30 % weak nodes (i) Send / received , 50 % weak nodes

Figure 9: Traffic distribution, uptime and send / received ratio among nodes for various ratios of weak nodes

system.

As we can see in figure 7(c) all lookups succeed when no weak nodes are present in the network. This is expected because it means that the network is static. It seems that the success rate has a close to linear relation to the ratio of weak nodes which is promising.

Regarding lookup delays (figure 7(a)) there is a weak tendency of nonlinearity in the results, which we have also seen in other simulations. We believe that with a small amount of weak nodes in the network, the weak nodes are unlikely to end up in routing tables, but as the ratio increases more weak nodes starts to forward traffic.

In figure 7(c) we can see that even for 50 % weak nodes the succes ratio is well over 95 % which seems quite good, considered the introduced churn.

6.3 Churn rate

A system that is distributed over the Internet will experience churn. The churn can be caused by many different things like network problems, node crashes or nodes that join and leave in a controlled fashion. We only simulate single nodes going up and down.

Whenever a node leaves the network it leaves silently, meaning that all state is left in the network. Only having silent leaves is the worst case scenario, but it is also how Bamboo handles leaves. When nodes leaves silently the node information related to those nodes will continue to spread throughout the network for some time. It will

however fade out when nodes that receive the information unsuccessfully tries to ping the dead node. The ping traffic to dead nodes, as well as neighbors that try to perform maintenance with dead nodes, cause what we call stale traffic within the network. We define stale traffic as traffic that is destined for a node that is no longer a member of the network. In figure 8(a) the percentage of stale traffic is plotted against the size of the network. The figure shows that the percentage of stale traffic is not increasing with the size of the network. There might have been an increase in stale traffic if nodes did not try to ping neighbors before adding them to leafsets and routing tables, but since they do, information about down nodes are not redistributed through the network.

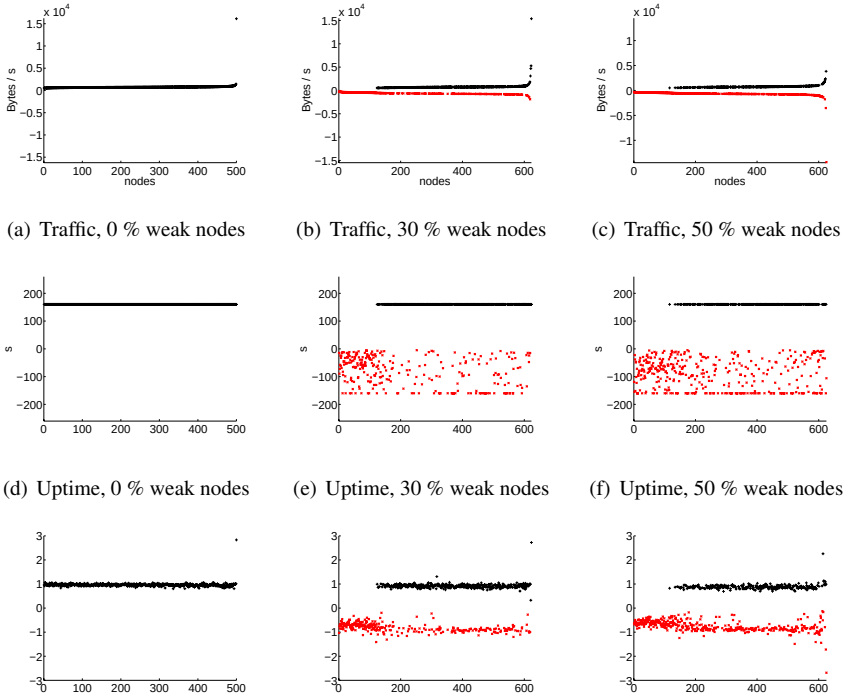
We have simulated networks with churn and different ratios of weak nodes. The size of the network in the simulations presented here is at most 500 nodes, which is close to the upper limit of what it is feasible to simulate with the methods and tools we have chosen. Even if larger networks would be interesting to study we believe that 500 nodes is enough to study the performance of the system, since an initial deployment of a DHT might for instance be on PlanetLab with some 200 nodes.

Weak nodes come and go in the system while strong nodes are static. How long a weak node is connected is determined by a Poisson process. The inter-arrival times model a mean online period of three minutes. Three minutes is a very short period of time but as we model cell phones used by mobile users, we believe that it is unlikely with many weak nodes that are online for extended time periods.

In figure 9 we present a visualization of the results gathered during three simulation rounds with different ratios of weak nodes. Each column of plots presents information about one run. All negative values are weak nodes and all positive values are strong nodes. The top plot shows the mean bandwidth utilization of all nodes in the simulation. The nodes are sorted on the utilization, and a completely even distribution of used bandwidth would look like a horizontal line. By studying the columns, some relations can be seen. When all nodes are strong the distribution is almost even but with weak nodes that introduce churn the distribution becomes less even. We can see that there are two major clusters of weak nodes at the extremes of utilized bandwidth. From studying the uptime plot we can see that the nodes that use the least bandwidth are nodes that have an uptime less than the maximum uptime. This means that those nodes have joined during the simulation and we believe that the reason for them to have a smaller load is that the information about them has not yet spread through the system, which could be very beneficial for a heterogeneous system. Weak nodes are typically connected shorter periods and could then get a smaller workload. The other extreme are the weak nodes that have the highest bandwidth utilization and the uptime plot gives us the information that they have typically been online for a very short period of time. From the bottom plot we can make the observation that the ratio between received and sent bytes are very close to zero which indicates that these nodes have just gone online, sent initial probes but that they have not yet received much response.

Because of the nature of a Poisson process some very short uptimes will occur, but extremely short uptime of nodes is not very realistic. A node might join the network in order to make a request and then leave, but we believe it to be unlikely that a node will take the cost of sending probes without getting the benefit of the response. To minimize the effect of the very short lived nodes in our analysis we added the condition that a node must have an uptime greater than 5 seconds to be presented, and then plotted the same data as in figure 9 in figure 10.

In figure 10(e) we still see a cluster of nodes that does not seem to be influenced by the extra condition on uptime.



(a) Traffic, 0 % weak nodes (b) Traffic, 30 % weak nodes (c) Traffic, 50 % weak nodes

(d) Uptime, 0 % weak nodes (e) Uptime, 30 % weak nodes (f) Uptime, 50 % weak nodes

(g) Send / received , 0 % weak nodes (h) Send / received , 30 % weak nodes (i) Send / received , 50 % weak nodes

Figure 10: Traffic distribution, uptime and send / received ratio among nodes for various ratios of weak nodes

7 Discussion

When we realized that a DHT's management traffic could pose a problem in heterogeneous environments [2], we realized that to evaluate how big the problem was , we needed to be able to simulate bandwidth. The most common approach when simulating DHTs is to only simulate a static delay between nodes. Such a network model will not introduce packet drops, reordering or delay variations due to congestion in the network. To be able to model a more dynamic network, we needed a more expressive simulator. Our choice of NS-2 was based on the fact that it is a de facto standard within the community, and also that it has good supporting tools like topology generators etc.

The choice to simulate a DHT in NS-2 showed time consuming. Implementing the DHT functionality from scratch was needed to be able to make simplifications, and simplifications where needed to be able to study somewhat large networks. As memory is a constraint in simulations, we needed to implement some extra support for dynamic allocation of simulation resources. Our experience is that NS-2 is not a tool that fits simulations of large, dynamic, overlay networks well. For example, we have not been able to simulate more than 60 minutes of real time when simulating a 200 nodes network with 30 % weak nodes on a 2 GB Ram machine. Fortunately, that time has been enough for the networks to stabilize so we have been able to get data from stable networks. The way we set up the networks offline 5.2 also shortened stabilization times compared to real experiments.

In conclusion we think that more bandwidth studies of DHTs are needed as they are becoming more common as building blocks in distributed systems. The simulation approach can be valuable but a simulator which is less detailed compared to NS-2 and more complex than the delay only simulators would be a good tool for such analysis.

References

[1] H. Balakrishnan, S. Shenker, and M. Walfish. Peering peer-to-peer providers, 2005.

[2] Fredrik Bjurefors, Lars Åke Larzon, and Richard Gold. Performance of pastry in a heterogeneous system. In *Proceedings of the fourth IEEE International Conference on Peer-to-Peer Computing*, 2004.

[3] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered dht applications, 2005.

[4] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[5] Sally Floyd and Steve McCanne. ns network simulator. Online: http://www.isi.edu/nsnam/ns.

[6] The bamboo distributed hash table. Online: http://bamboo-dht.org/.

[7] E. Kohler, M. Handley, and S. Floyd. Designing dccp: Congestion control without reliability, 2003.

[8] R Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Proceedings of the annual international cryptology conference(CRYPTO)*, pages 369–378. Springer-Verlag, 1988.

[9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM SIGCOMM conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[10] Sean Rhea. *OpenDHT: A Public DHT Service*. PhD thesis, University of California, Berkeley, August 2005.

[11] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[12] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. *SIGCOMM Comput. Commun. Rev.*, 35(4):73–84, 2005.

[13] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[14] Sean Rhea and Dennis Geels and Timothy Roscoe and John Kubiatowicz. Handling churn in a DHT. Technical Report UCB/CSD-03-1299, University of California, Berkeley, December 2003.

[15] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure, 2002.

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[17] Sven Westergren. Notenet - range queries in a dht. Master's thesis, Uppsala University, 2007.

[18] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.