Load Balancing for Structured P2P Networks Using the Advanced Finger Selection Algorithm (AFSA)

Jani Hautakorpi
Ericsson Research NomadicLab
FI-02430 Jorvas, Finland
jani.hautakorpi@ericsson.com

Jouni Mäenpää
Ericsson Research NomadicLab
FI-02430 Jorvas, Finland
jouni.maenpaa@ericsson.com

ABSTRACT

Structured Peer-to-Peer (P2P) networks, such as networks based on Distributed Hash Tables (DHTs), can be enhanced by using load balancing mechanisms. Current load balancing mechanisms are either trying to achieve even distribution of objects among nodes, make the address space as evenly populated as possible, or both. However, we have taken a different approach to load balancing in this paper and we have defined the Advanced Finger Selection Algorithm (AFSA) which is not focused on balancing the objects among the nodes and does not require evenly populated address space. AFSA is an algorithm which changes the way how nodes are selected as fingers to the overlay routing tables in structured P2P networks. We implemented the AFSA algorithm for both Chord and Bamboo and we evaluated it with simulations.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed Networks*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*

General Terms

Algorithms, Design, Performance

Keywords

Peer-to-peer, distributed hash table, load balancing, simulation, Chord, Bamboo

1. INTRODUCTION

A number of different load balancing mechanisms have been developed for structured Peer-to-Peer (P2P) networks, such as for networks utilizing Distributed Hash Table (DHT) algorithms. When the current load balancing mechanisms are examined from a high abstraction level, it can be said that they are trying to achieve one or both of the two following features. The first feature is to attempt to achieve an even distribution of objects among the nodes. This can be done, for example by selecting the less loaded node out of two when storing an object to a P2P network [4]. The second feature is to make the address space evenly populated. That can be done, for example by splitting the largest partition of the P2P network and placing a joining node in the middle of that partition [11].

We have taken a different approach to load balancing and we have defined a novel algorithm, called the Advanced Finger Selection Algorithm (AFSA), that will change the way how nodes are selected to overlay routing tables. The nodes in an overlay routing table represent outgoing connections from one node to another, and in this paper we refer to them as outgoing fingers (or *out-fingers*). AFSA does not balance objects among the nodes, and it does not create an evenly populated address space. Instead, AFSA balances the number of incoming fingers (from hereon referred to as *in-fingers*) among the nodes in a P2P network, because our simulations have shown that an uneven distribution of in-fingers correlates with an uneven distribution of load.

The design of AFSA does not prevent the use of other load balancing algorithms in addition of using AFSA (e.g. for balancing the objects). AFSA also works well in an environment where the address space is unevenly populated. Furthermore, unlike many other load balancing algorithms, AFSA requires only one node-id (i.e. the key a node has) per physical node and it preserves complete freedom in choosing it. The ability to use a single arbitrary node-id is a desired property, because then node-ids can, for example be generated by hashing the public key that presents the identity of a node. This is similar than the cryptographically generated addresses in IPv6 [1], but above the network layer on a P2P network level.

By balancing the number of in-fingers AFSA also spreads the load relatively evenly among the nodes in a P2P network. The load AFSA is balancing is the number of packets a node has to handle from other nodes. These packets are, for example application level requests from other nodes that the node has to forward. When we, from hereon, use the word *load*, we mean application level packets from other nodes in the P2P network. AFSA does not address the problems related to popular files or services, but nothing prevents it from being used in conjunction with other mechanisms that do address those problems.

The even distribution of in-fingers is also a desirable prop-

SAC'10 March 22-26, 2010, Sierre, Switzerland.

erty from the viewpoint of a designer who is designing applications on top of P2P networks. For example, the even distribution of in-fingers can be utilized when designing a wildcard search mechanism for P2P Session Initiation Protocol (P2PSIP) [5] networks.

The evaluation of AFSA was done with simulations. We implemented AFSA for both Chord [19] and Bamboo [17] algorithms. The implementations were used in simulations. The results of the simulations show that AFSA balances the load well among the nodes in a P2P network, and that it produced relatively even distribution of in-fingers among the nodes. Even though AFSA is currently implemented only for Chord and Bamboo, we see no reason why it could not be implemented also for many other distribution mechanisms used in structured P2P networks.

The remainder of this paper is organized as follows. Section 2 briefly introduces the existing load balancing algorithms. Section 3 identifies the research problem we are solving in this paper. Section 4 describes the algorithm we have designed, AFSA. Section 5 presents the simulation environment and the simulation results. Section 6 contains discussion related to AFSA and the results. Section 7 gives directions for future work, and finally Section 8 concludes the paper.

2. RELATED WORK

In the following we present some of the current load balancing algorithms. Later on we make a comparison between AFSA and these algorithms in Section 6.

The concept of *virtual servers* is used as a basis for many load balancing algorithms, such as [19, 7, 15, 8, 9]. The virtual servers, which was first introduced in [19], is a concept where a single physical node is mapped to multiple virtual nodes where each virtual node has an unrelated node-id. Some of these algorithms are more suitable to static networks, such as [19, 7, 15], and some of them, such as [8, 9], are suitable also to dynamic networks. Furthermore, there are also other differences between the algorithms, such as some of them accommodate the heterogeneity of physical nodes, and some of them accommodate skewed distribution of object keys (e.g. in a situation where object keys carry semantics). However, all of these algorithms have some common properties, such as, they require more fingers than DHTs without virtual servers, and they require one physical node to have multiple node-ids.

Many load balancing algorithms, such as [13, 11, 4], are attempting to *minimize the variation between partition sizes*. In other words, they try to create a situation where nodes have equally sized partitions.

Two of the above mentioned algorithms, [13] and [11], are working in a non-continuos manner. When a new node joins, these algorithms try to find, by using probes, a relatively large partition from the P2P network, split it, and place the new node into that partition. In addition, [13] also balances partitions when a node leaves from a P2P network by possibly changing a node-id of a node in the P2P network. These algorithms require that nodes have to accept arbitrary node-ids.

Bienkowski et al. have presented an infinite and continuous process [4] for minimizing the variation between the partition sizes. Each node in the network executes this process which dynamically balances the network by migrating the nodes responsible for relatively short partitions to points in the address space where nodes are responsible for relatively long partitions. It also copies the objects between the migrating nodes. This process requires that nodes have to accept arbitrary node-ids.

Karger and Ruhl have presented a load balancing algorithm [10] which can be seen as a a hybrid between the virtual server concept and an attempt to minimize the variations between partition sizes. [10] uses two different mechanisms. In the first mechanism each node has multiple possible positions (i.e. multiple node-ids) where it can choose from and change its position dynamically. In other words, one physical node has multiple virtual servers but only one of them is active at a given time. The goal of this mechanism is to occupy the address space evenly. The second mechanism is such that each node can dynamically have an arbitrary place in the address space and the goal is to have good load balancing properties even in a scenario where objects have a skewed distribution of keys.

Byers et al. [6] use the *power of two choices* paradigm [2, 14] for load balancing in DHTs. Their load balancing is focused on creating an even distribution of objects in a P2P network despite the differences in partition sizes. The load balancing works in a manner where objects are stored to the less loaded of two (or more) nodes in the P2P network. All of the nodes that would potentially be storing a given object maintain a redirection pointer to a node that is actually storing the given object. These redirection pointers speed up the lookups.

3. IDENTIFYING THE PROBLEM

Our research work for this paper began when we noticed that the implementation of Chord in the OverSim [3] simulator had an uneven distribution of in-fingers. In other words, the number of in-fingers varied significantly from node to node in a Chord network. Later we discovered that Chord as an algorithm, and not just its specific implementation in OverSim, does produce an uneven distribution of in-fingers by using a second independent implementation of Chord. The second implementation was a P2PSIP overlay simulator which is implemented in Java (in contrast, Chord in OverSim is implemented in C++), and it is based on the implementation used in [12].

The distributions of in-fingers in these two implementations is illustrated in Fig. 1. The figure illustrates the distributions of in-fingers in a setting where there are 10000 nodes and all of the nodes have been idling for 3600 seconds after the time all the nodes have joined to the network. With idling we mean a situation where a node is just running a DHT algorithm, but is not sending or receiving application level packets. It can be seen from the figure that the distributions are quite wide. For example, the network using OverSim's Chord has 123 nodes that have more than 50 in-fingers and 1178 nodes that have less than 3 in-fingers.

The average number of in-finger in experiments was 13.6 with the OverSim implementation and 13.0 with Java-based implementation. So, if the number of in-fingers would be evenly distributed, then all the nodes would have either 13 or 14 in-fingers in both experiments. Given the fact that there is a linear relation between the number of in-fingers and the number of forwarded application level packets, Chord leaves a lot to desire on its load balancing properties.

We have focused our examination to the number of in-fingers as the indicator of uneven load balance, because it

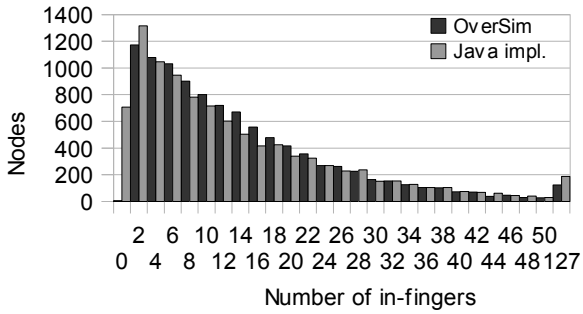

Figure 1: Histograms of the in-finger distributions in two separate Chord implementations

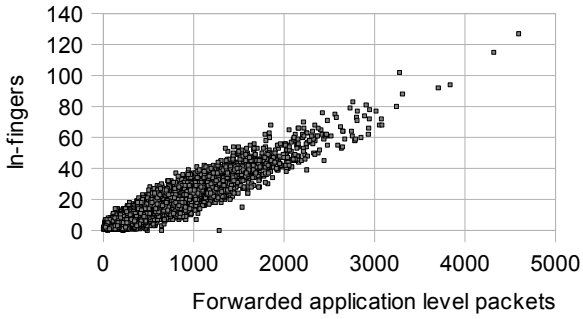

Figure 2: Scatter plot showing the relation between the number of in-fingers and the number of forwarded application level packets in Chord

was straightforward to measure it from the above mentioned implementations, and the following simulation showed that there is a linear relation between the number of in-fingers and number of forwarded application level packets (i.e. load).

In order to examine the relation between the number of in-fingers and the number of forwarded application level packets we did a simulation run with unmodified Chord using the OverSim simulator (the equivalent simulation with Bamboo is described in Section 4.2). The simulation contained 10000 nodes and all the nodes were idling for 1800 seconds after all the nodes had joined. After the idling period, all the nodes started sending application level packets, 84 in total per node, on 20 second intervals. 1800 second after the idling period stopped, the simulation ended and we examined the number of in-fingers and the number of forwarded application level packets in each node. The results show that there is positive linear correlation between the number of in-fingers and the number of forwarded application level packets in Chord. The correlation is illustrated as a scatter plot in Fig. 2.

The figure shows that the number of forwarded application level packets increase as the number of in-fingers increase. The Pearson product-moment correlation coefficient (from hereon referred to as Pearson's correlation) between the number of in-fingers and the number of forwarded application level packets is 0.94.

4. THE ADVANCED FINGER SELECTION ALGORITHM

The Advanced Finger Selection Algorithm (AFSA) is inspired by the *power of two choices* paradigm, but unlike Byers et al. [6] we use the paradigm for selecting out-fingers, and not for selecting target nodes for storing objects. Next we will explain AFSA on an abstract level, and then in the following two sections we will explain more concretely how it is built as a part of Chord and Bamboo.

The biggest change AFSA induces to structured P2P networks is that it changes the way how nodes are selected as out-fingers to overlay routing tables. The main idea in AFSA is that each out-finger is selected from a set of out-finger candidates in a manner that results to an even in-finger distribution in a structured P2P network. There are two different modes in AFSA which we will describe next.

The first mode is called an *implicit mode* where the selection between out-finger candidates is based on probabilities. The probability of a candidate being selected is calculated from some quantifiable property (f) that a candidate node has. There is requirement that f has to have a positive linear correlation to the number of in-fingers. Implicit mode works in a manner where the probability of the node getting selected as an out-finger decreases as f increases. The probability that an i:th candidate gets selected (P_i) as an out-finger to an overlay routing table in a scenario where there are c out-finger candidates is calculated with the following equation:

$$P_i = \frac{1 - \frac{f_i}{\sum_{i=1}^{c} f_i}}{c - 1}$$

The second mode is called an *explicit mode* where the selection between out-finger candidates is done by accepting an out-finger candidate that has the lowest number of in-fingers at the time of the selection. It is noteworthy that in explicit mode nodes need to keep track of how many in-fingers they have at all times and they have to have an ability to communicate their in-finger counts to other nodes.

Both modes have an optional parameter, c_m, that defines the maximum number on out-finger candidates ($c \leq c_m$). The c_m is two at minimum (if it would be one, it would effectively mean that AFSA is disabled).

4.1 AFSA in Chord

Overlay routing tables (also know as *finger tables*) are refreshed periodically in Chord [19]. The refreshing is done by periodically calling the *fix_fingers* function. AFSA modifies the *fix_fingers* function by adding new logic to it.

We did a simulation run with an unmodified Chord using the OverSim simulator. The simulation contained 10000 nodes. When all the nodes had joined, the nodes were at idle for 3600 seconds. After 3600 seconds of idling the simulation ended and then we examined the number of in-fingers and partition (i.e. the node-free address space counter-clockwise from the node) sizes of the nodes in the simulation. The results show that there is positive linear correlation between the number of in-fingers and partition sizes of the nodes. The correlation is illustrated as a scatter plot in Fig. 3. Partition sizes are presented in the figure by translating the 30 most significant bits of the partition size into an integer value. It can be seen from the figure that when the partition size increases, so does the number of in-fingers. The Pearson correlation between the number of in-fingers and partition sizes is 0.94.

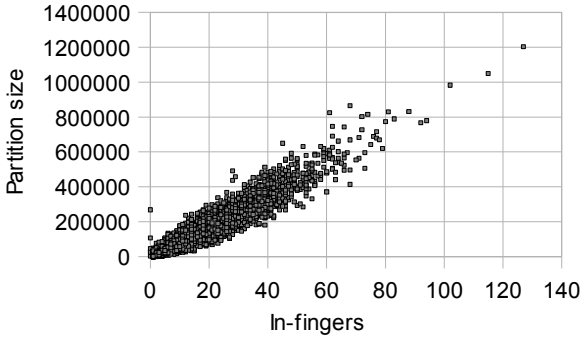

Figure 3: Scatter plot showing the relation between the number of in-fingers and the partition sizes of the nodes in Chord

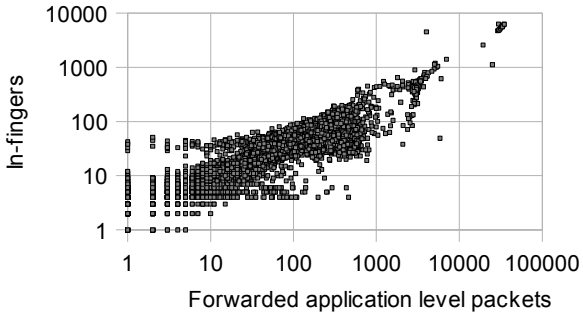

Figure 4: Scatter plot showing the relation between the number of in-fingers and the number of forwarded application level packets in Bamboo (note that the scales are logarithmic)

The implicit mode of AFSA in Chord utilizes this correlation. The partition size of a node is used as the quantifiable property, f (refer to Section 4). In other words, an out-finger candidate with a small partition size is more likely to be selected as an out-finger than a candidate with a larger partition size. The reasoning is such that the nodes with a small partition size are probabilistically less frequently in out-finger selection situations than the nodes with a larger partition sizes and AFSA attempts to corrects this imbalance. Implicit mode is straightforward to implement to Chord, since the partition size of a node does not have to be explicitly calculated. Furthermore, there is no need to explicitly convey partition size information between nodes, since nodes performing the selection between out-finger candidates can derive partition sizes from node-ids. Partition sizes can be derived from node-ids, because out-finger candidates are consecutive nodes in the overlay address space.

The explicit mode does not use partition sizes. It performs the out-finger candidate selection based on in-finger counts. So, Chord has to have some additional capabilities in order to support the explicit mode. Each node has to keep track of how many in-fingers they have and each node has to be able to convey the number of in-fingers to other nodes, because the selection is done based on that information.

In Chord, the implicit mode is easier to implement than the explicit mode, because the implicit mode requires only minor changes to the Chord algorithm. The design of the Chord algorithm is such that it can benefit significantly from having more than two out-finger candidates (i.e. $c_m > 2$).

4.2 AFSA in Bamboo

In Bamboo [17], the overlay routing table is refreshed by using two separate processes, *local* and *global tuning*. In the local tuning process the overlay routing table is refreshed, in a way, one row at a time, and in the global tuning process a single entry in the overlay routing table is being refreshed. AFSA modifies both of these processes.

We did a simulation run with an unmodified Bamboo using the OverSim simulator. The simulation contained 10000 nodes and all the nodes were idling for 1800 seconds after all the nodes had joined. After the idling period, all the nodes started sending application level packets, 84 in total per node, on 20 second intervals. 1800 second after the idling period stopped, the simulation ended and we examined the number of in-fingers and the number of forwarded application level packets in each node. The results show that there is positive linear correlation between the number of in-fingers and the number of forwarded application level packets in Bamboo. The correlation is illustrated as a scatter plot in Fig. 4. The figure shows that the number of in-fingers increase as the number of forwarded application level packets increase. The Pearson's correlation between the number of in-fingers and the number of forwarded application level packets is 0.96. It is noteworthy that there is no correlation between the number of in-fingers and the partition sizes nor there is correlation between the number of in-fingers and the link delays. In the former case the Pearson's correlation was 0.03 and -0.23 in the latter. Link delays for each node in the simulation were taken randomly between 30 and 70 milliseconds (ms).

The implicit mode of AFSA in Bamboo utilizes the correlation between the number of in-fingers and the number of forwarded application level packets. The sliding sum of the forwarded application level packets by the node is used as the quantifiable property, f (refer to Section 4). With the sliding sum we mean a calculation where only packets from the last t_{ss} seconds are calculated into the sum. In other words, an out-finger candidate which has not forwarded a lot of application level packets in the last t_{ss} seconds is more likely to be selected as an out-finger then a candidate which has forwarded more packets. Unlike in Chord, the implicit mode is not straightforward to implement to Bamboo. In Bamboo, the implicit mode requires continuous tracking of application level packets and an ability to convey the sliding sum of forwarded application level packets to other nodes.

The explicit mode does not use the sliding sum of the forwarded application level packets. It performs the out-finger candidate selection based on the in-finger counts. So, as in Chord, also Bamboo has to have some additional capabilities in order to support the explicit mode. Each node has to keep track of how many in-fingers they have and each node has to be able to convey the number of in-finger to other nodes.

There are few differences in the manner how AFSA is built as a part of Chord and Bamboo. First, unlike in Chord, the implicit mode is not easier to implement than the explicit mode in Bamboo. Second, unlike in Chord, the design of the Bamboo algorithm is such that it can not significantly benefit from having more than two out-finger candidates (i.e. the utilization of c_m is not needed). The reason is that in

Bamboo the global tuning process could benefit from more than two out-finger candidates, but it would be impractical to implement it to the local tuning process. It would be impractical, because the local tuning would require fresh information from a relatively large number of nodes (e.g. from 90 nodes) in order to make a selection between out-finger candidates. This fresh information can be either recent sliding sum of forwarded application level packet or recent number of in-finger candidates. In contrast, Chord's *fix_fingers* function, and Bamboo's global tuning need fresh information only from a relatively small set of nodes (e.g. from 2-6 nodes).

5. SIMULATIONS

We used the OverSim [3], which is built on top of the OMNeT++ [20], simulator for conduction the simulations. More specifically, we implemented AFSA as an extension to Chord and Bamboo implementations in OverSim. It is noteworthy that we used the existing Chord implementation as a starting point, but the Bamboo implementation required few bug fixes, most notably the fixing of the global tuning process.

The purpose of the simulations was to evaluate how well AFSA works with Chord and Bamboo. AFSA is regarded as working well when it balances the load and produces relatively even distribution of in-fingers among the nodes. The purpose of the simulations was not to make a comparison between Chord and Bamboo.

All the simulations had 10000 nodes with random nodeids. The random numbers for the simulations were produced by Mersenne Twister random number generator with a cycle length of 2^{19937}. All the simulations were 4600 simulated seconds long. When we from hereon mention seconds or milliseconds, we are talking about the time in the simulation, not wall-clock time. First, the nodes joined within 100 ms intervals, and so all the nodes had joined after 1000 seconds. Then, all the nodes were on idle for 1800 seconds. After that, all the nodes started sending application level packets to random keys. Each node sent 84 packets in 20 second intervals. The simulations ended when 4600 seconds had passed.

The simple underlay network model of OverSim was used for all simulations. There was no churn in the simulations. Both Chord and Bamboo used *semirecursive* routing type (as it is defined by OverSim). The *fixfingers* function in Chord was called every 120 seconds. The Bamboo's global tuning process was executed in 30 second intervals, and the local tuning process in 60 second intervals.

When we refer to application level packets from here on, we refer to packets that were captured between 4000 and 4010 seconds (which was 5019 packets per simulation run on average). The topology of the simulations was such that all the nodes in the network had a link to the same single point in the network, and the delays of those links were taken randomly between 30 and 70 ms. The delays remained the same thorough the simulation. It is noteworthy that the topology and the delay distribution used in the simulations does not correspond the topology and the delay distribution of the Internet.

5.1 Results

We executed 15 simulation runs with different parameters. Seven of those runs were different Chord-based runs, and three of them were based on Bamboo. In addition, we did five extra runs to measure the repeatibility of the simulation results. It is noteworthy that the simulation runs had long execution times, because of their relatively large scale, in our hardware.

The Chord-based simulation runs contained one run without AFSA (i.e. just unmodified Chord) and both implicit and explicit modes of AFSA had three runs each. On those three runs the optional parameter c_m was varied between 2, 4, and 6. Bamboo-based simulation runs contained one run without AFSA, one run with the implicit mode of AFSA, and one run with the explicit mode of AFSA. Optional parameter c_m was not used in the Bamboo-based simulation (refer to Section 4.2). The implicit mode of AFSA in Bamboo calculated the sliding sum of forwarded application level packets from the last 300 seconds, so t_{ss} was 300.

When we analyzed the results from the simulations we focused on three things: to the number of in-fingers, number of forwarded application level packets (i.e. load), and end-to-end characteristics of application level packets. For the number of in-fingers and the number of forwarded application level packets we examined four key statistical values: average, standard deviation, 5th percentile, and 95th percentile. These statistical values present the distribution of in-fingers and load in the network. In addition to the statistical values, we also examined selected histograms presenting the in-finger distributions in the network. The end-to-end characteristics, namely the average delay and average end-to-end hop count of application level packets, were examined so that we could evaluate the impact AFSA had on application level packet forwarding.

All the key statistical values are presented in Table 1. The table uses the "Imp.", "Exp.", "Avg.", and "Std." abbreviations which correspond to Implicit, Explicit, Average, and Standard. The topmost four data rows show the key statistical values regarding the number of in-fingers. For example, the average number of in-fingers in nodes was 45.97 with the implicit AFSA in Bamboo. It can be seen from the table that AFSA enhances significantly the in-finger distribution, for example by lowering the standard deviation from 11.72 (unmodified Chord) to 2.15 (Chord with the explicit AFSA, c_m=6). All the statistical values improve when AFSA is used. In general, the explicit modes produce better results than the implicit modes, and this is especially true for Bamboo. Furthermore, it can be said that the evenness of in-finger distribution increases when c_m increases in Chord.

The next four data rows in the table show the key statistical values regarding the number of forwarded application level packets. All the statistical values describing the number of forwarded application level packets behave much in the same way as the statistical values describing the number of in-fingers. All the statistical values get significantly better when AFSA is used. For example, the standard deviation is lowered from 1333.48 (unmodified Bamboo) to 52.79 (Bamboo with explicit AFSA).

We have shown earlier in this paper that there is a positive linear correlation between the number of in-fingers and the number of forwarded application level packets in Chord (see Fig. 2) and in Bamboo (see Fig. 4). This can also be seen in the standard deviation values in the Table 1. We can see that the variance (i.e. standard deviation) in the number of forwarded application level packets decreases when the variance in the number of in-fingers decreases.

Table 1: Results from the simulations

		Chord							Bamboo		
		No AFSA	Imp. AFSA			Exp. AFSA			No AFSA	Imp. AFSA	Exp. AFSA
			c_m=2	c_m=4	c_m=6	c_m=2	c_m=4	c_m=6			
Number of in-fingers	Average	13.62	13.70	13.86	13.94	13.54	13.18	12.88	46.05	45.97	45.94
	Std. deviation	11.72	10.51	6.62	5.47	6.22	3.10	2.15	228.19	12.78	1.38
	5th percentile	1	3	5	6	5	9	10	1	27	45
	95th percentile	37	34	26	24	24	19	17	111	69	47
Forwarded application level packets	Average	546.61	534.78	532.80	529.70	547.64	549.91	552.59	218.48	225.74	226.32
	Std. deviation	456.70	390.28	230.53	180.49	286.23	185.72	154.25	1333.48	154.92	52.79
	5th percentile	66	102	214	269	165	286	333	0	109	143
	95th percentile	1440	1302	955	852	1067	890	834	493	406	312
Packet info	Avg. delay	0.35	0.35	0.34	0.34	0.35	0.34	0.34	0.13	0.17	0.18
	Avg. hops	6.96	6.90	6.84	6.81	6.90	6.87	6.80	3.47	3.50	3.51

The average delay and average hop count of application level packets is shown in the lowest two data rows in the table. The hop count means how many nodes an application level packet has to travel through before reaching its destination, and the average delay means how long it takes for the packet to reach its destination. It can be seen that AFSA has only a relatively small impact to the average hop count. Likewise Chord has only a relatively small impact to the average delay. However, AFSA has an somewhat degrading impact to the average delay in Bamboo. This is not surprising, because unmodified Bamboo chooses the out-fingers based on Round-trip Times (RTTs) of out-finger candidates. In other word, in an unmodified Bamboo the out-finger candidate (note that there is only one candidate) with lowest RTT get selected as an out-finger. It is quite natural to expect low RTTs when the out-fingers are selected based on RTTs, and not on something else. There is more related discussion in Section 7.

Three selected in-finger distributions of the Chord-based simulations are presented as histograms in Fig. 5. The selected distributions are unmodified Chord (dark gray), the implicit AFSA where c_m=6 (gray), and the explicit AFSA where c_m=6 (light gray). It can be seen from the histograms that the in-finger distribution gets considerably narrower when AFSA, and especially the explicit AFSA, is used. It is noteworthy that an unmodified Chord has a very uneven in-finger distribution where there are 1178 nodes having less than 3 in-fingers and 123 nodes having more that 50 in-fingers (and there is even one node with 127 in-fingers).

In-finger distributions of the Bamboo-based simulations are presented as histograms in Fig. 6. The distributions are unmodified Bamboo (dark gray), the implicit AFSA (gray), and the explicit AFSA (light gray). It can be seen from the histograms that the in-finger distribution gets considerably narrower when AFSA, and especially the explicit AFSA, is being used. It is noteworthy that unmodified Bamboo has an extremely uneven in-finger distribution where there are 5563 nodes having less than 21 in-fingers and 259 nodes having more that 300 in-fingers (and there is even one node with 6279 in-fingers). Bamboo with explicit AFSA reaches significantly better result and it has 9993 nodes that have more than 40 but less than 61 in-fingers. Furthermore, it can be seen from Table 1 that 90% of the nodes with explicit AFSA in Bamboo have their in-finger counts between 45 and 47 (while average is 45.94).

As a summary from the experiments it can be said that

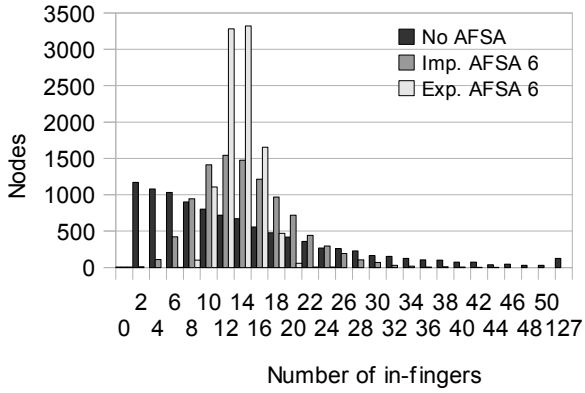

Figure 5: Histograms of the in-finger distributions in Chord

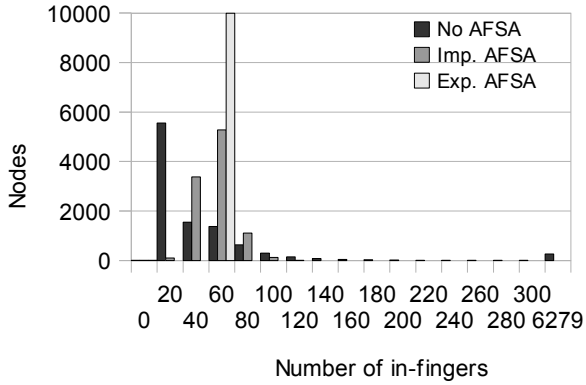

Figure 6: Histograms of the in-finger distributions in Bamboo

AFSA enhances the in-finger distribution, and therefore also the load balancing, significantly. In general, the explicit modes produce better results than the implicit modes, but also the enhancements produced by the implicit modes are significant. The increase of c_m yields better results in Chord. Given how small changes the implicit AFSA mode in Chord requires, it performs remarkably well when c_m=6.

5.2 Repeatibility of the Results

The fundamental decision in our simulation work was to do a number of relatively large scale (10000 nodes, 4600 seconds) simulations runs, as opposed to a greater number of simulation runs with smaller scale. Given the limited hardware resources, we did not run all the measurements multiple times due to long execution times. However, to get an idea of the repeatibility of the measurements, we did run one simulation, Chord with explicit AFSA, c_m=6, six times.

We calculated the standard error for the key statistical values regarding the number of in-fingers. The average values of the key statistical values from the six simulation runs are presented in the following (standard error is presented in parenthesis after each value): average 12.88 (0.00), standard deviation 2.20 (0.01), 5th percentile 9.83 (0.15), and 95th percentile 17.17 (0.15). From these values we can deduct that the results stay relatively unchanged among multiple simulation runs.

6. DISCUSSION

AFSA is inherently scalable as an algorithm. This is due to the fact that AFSA requires two or more (up to c_m) out-fingers candidates per overlay routing table entry in order to do load balancing. When a structured P2P network is relatively small, then there are less possible out-finger candidates per an overlay routing table entry in the network. For example, in both Chord and Bamboo there is a short supply of the possible out-finger candidates to those entries in overlay routing tables that require a long prefix match. In Chord those are the closest out-fingers and in Bamboo those are the out-fingers in the highest rows. However, the number of possible out-finger candidates for those overlay routing table entries grow when the network grows. Furthermore, load balancing is usually not a problem in relatively small P2P networks.

As a general statement it can be said that a node in a structured P2P network with AFSA consumes more bandwidth than a node without AFSA. The increased bandwidth usage is due to the fact that multiple out-finger candidates needs to be are gathered and some extra information has to be conveyed through the network (e.g. in-finger counts in the explicit AFSA mode). We did not measure the explicit increase in the bandwidth consumption for two reasons. The first reason was that a part of the AFSA signaling is in the packets that are sent to the network even in cases where AFSA is not used. For example, the in-finger counts in the explicit AFSA in Chord are conveyed in packets that are created by the *fixfingers* function of unmodified Chord. Another example from real networks is one where a part of the AFSA signaling could be incorporated to otherwise needed NAT keep-alive packets, such as to *Binding* methods of Session Traversal Utilities for NAT (STUN) [18]. The second reason was that the increase in bandwidth usage is highly dependent on the implementation factors. For example, the gathering of multiple candidates in the implicit mode of AFSA in Chord can take either only a single packet exchange with one out-finger candidate (thanks to Chord's *successor list*), or packet exchanges with all the candidates.

In this paper we examined AFSA in an environment where all the nodes had AFSA capabilities and there was no churn. The current implementation of AFSA does not have such failure case handlers and retransmission timers that would be needed for evaluating incremental deployability and churn resistance. However, our assumption is that at least the implicit mode of AFSA in Chord is well suited to environments where only a portion of nodes support AFSA and where the rest are running unmodified Chord. Furthermore, we do not see any theoretical reasons why a given structured P2P network with AFSA would have worse churn resistance that the same structured network without AFSA.

The fact that AFSA balances the number of in-fingers among all the nodes in the network might not be an ideal solution for all environments. For example, an uneven in-finger distribution, which could be based on node capabilities, might make the P2P network more efficient and robust in heterogenous environments.

AFSA has some benefits and differences when compared to the existing load balancing algorithms. First, AFSA does not increase the number of out-fingers (which can be indirectly seen by observing the average number of in-fingers in Table 1) like the load balancing mechanisms based on *virtual servers* concept [19, 7, 15, 8, 9] do. Second, AFSA preserves the complete freedom in choosing node-ids unlike the load balancing mechanisms that are *minimizing the variation between the partition sizes* [13, 11, 4]. Third, AFSA does not mandate that a single physical node has more than one node-id, unlike the load balancing mechanism presented by Karger and Ruhl [10]. And lastly, AFSA is not focused on creating an even distribution of objects unlike the load balancing algorithm by Byers et al. [6]. Due to this difference, it might be beneficial to use AFSA in conjunction with [6].

7. FUTURE WORK

One of the desirable properties of AFSA is that it produces a relatively even in-finger distribution to a P2P network. We believe that this property can be used, and is essential, for creating probabilistic wildcard searches on top of P2P networks. With wildcard searches we mean, for example a "*catering*" query in a P2PSIP network [5] which would return the contact information of catering services in a probabilistic manner.

Simulation results in Section 5.1 show that AFSA increases the end-to-end delay of the application level packets when compared to unmodified Bamboo. The reason for this is that the unmodified Bamboo selects its out-fingers based on RTT and does not care about load balancing. There might be environments where it would be beneficial to have an algorithm that takes both, RTT and load balancing, into consideration when selecting between the out-finger candidates. This kind of hybrid algorithm could be, for example implemented as an extension to AFSA.

In theory, there should be no reasons why the AFSA algorithm could not be applied also to other structured P2P networks, in addition to Chord and Bamboo, which have out-fingers pointing to distant locations in the address space of the overlay. However, AFSA is not suitable to such structured P2P networks where the out-fingers of the nodes are

pointing only to the immediate neighbors, such as vanilla version of Content-Addressable Network (CAN) [16]. We believe that AFSA is especially easy to implement to such structured P2P networks that are using periodic stabilization, as opposed to reactive stabilization, because periodic stabilization usually makes the gathering of multiple out-finger candidates quite convenient.

Given the promising simulations results, it would be beneficial to build a real implementation (i.e. not a simulation) of AFSA. The real implementation could be used for validating the AFSA by conducting experiments in real networks, such as in Internet.

8. CONCLUSIONS

A novel load balancing algorithm, the Advance Finger Selection Algorithm (AFSA), was presented in this paper. AFSA is an inherently scalable algorithm that produces a relatively even distribution of in-fingers and load among the nodes in a structured P2P network. We believe that the relatively even in-finger distribution can be utilized for building new services, such as a wildcard search mechanism, on top of structured P2P networks. AFSA was implemented for both Chord and Bamboo algorithms in the OverSim simulator. The simulation results from OverSim show that AFSA works well. Even though AFSA was implemented only for Chord and Bamboo, we believe that it would be feasible to implemented it to other structured P2P networks as well. A notable property of AFSA is that it requires only a single node-id per physical node and it preserves complete freedom in choosing it. Furthermore, AFSA does not increase the number of fingers in a P2P network.

9. ACKNOWLEDGMENTS

The authors would like to thank Ari Keränen for fruitful discussions related to the load balancing mechanisms.

10. REFERENCES

[1] T. Aura. Cryptographically Generated Addresses (CGA). In *Proc. of the 6th Information Security Conference (ISC)*, pages 29–43, Oct. 2003.

[2] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced Allocations (extended abstract). In *Proc. of the 26th Annual Symposium on Theory of Computing (STOC)*, pages 593–602. ACM, 1994.

[3] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proc. of the 10th IEEE Global Internet Symposium (GI) in conjunction with IEEE INFOCOM 2007*, pages 79–84, May 2007.

[4] M. Bienkowski, M. Korzeniowski, and F. Meyer auf der Heide. Dynamic Load Balancing in Distributed Hash Tables. In *Proc. of the 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS)*, volume 3640 of *LNCS*, pages 217–225. Springer, 2005.

[5] D. A. Bryan and B. B. Lowekamp. Decentralizing SIP. *ACM Queue*, 5(2), Mar. 2007.

[6] J. Byers, J. Considine, and M. Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 31–35, Feb. 2003.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area Cooperative Storage with CFS. *SIGOPS Operating System Review*, 35(5):202–215, 2001.

[8] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Dynamic Structured P2P Systems. In *INFOCOM 2004. 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2253–2262, Mar. 2004.

[9] P. Godfrey and I. Stoica. Heterogeneity and Load Balance in Distributed Hash Tables. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 596–606, Mar. 2005.

[10] D. R. Karger and M. Ruhl. Simple Efficient Load-Balancing Algorithms for Peer-to-Peer Systems. *Theory of Computing Systems*, 39(6):787–804, 2006.

[11] K. Kenthapadi and G. S. Manku. Decentralized Algorithms Using both Local and Random Probes for P2P Load Balancing. In *Proc. of the 17th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 135–144. ACM, 2005.

[12] J. Mäenpää and G. Camarillo. A Study on Maintenance Operations in a Chord-based Peer-to-Peer Session Initiation Protocol Overlay Network. In *Proc. of the 6th International Workshop on Hot Topics in P2P Systems (HotP2P)*, Jan. 2009.

[13] G. S. Manku. Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables. In *Proc. of the 23rd Annual Symposium on Principles of Distributed Computing (PODC)*, pages 197–205. ACM, 2004.

[14] M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The Power of Two Random Choices: A Survey of Techniques and Results. In *Handbook of Randomized Computing*. Springer, 2001.

[15] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM Conference*, Aug. 2001.

[17] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. Technical Report UCB/CSD-03-1299, University of California Berkeley, Dec. 2003.

[18] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, Internet Engineering Task Force, Oct. 2008.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the ACM SIGCOMM Conference*, pages 149–160, Aug. 2001.

[20] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proc. of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SimuTools)*, Mar. 2008.