Internet Engineering Task Force (IETF)
Request for Comments: 7363
Category: Standards Track
ISSN: 2070-1721

J. Maenpaa
G. Camarillo
Ericsson

September 2014

Self-Tuning Distributed Hash Table (DHT) for REsource LOcation And Discovery (RELOAD)

Abstract

REsource LOcation And Discovery (RELOAD) is a peer-to-peer (P2P) signaling protocol that provides an overlay network service. Peers in a RELOAD overlay network collectively run an overlay algorithm to organize the overlay and to store and retrieve data. This document describes how the default topology plugin of RELOAD can be extended to support self-tuning, that is, to adapt to changing operating conditions such as churn and network size.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7363.

Table of Contents

1. Introduction

REsource LOcation And Discovery (RELOAD) [RFC6940] is a peer-to-peer signaling protocol that can be used to maintain an overlay network and to store data in and retrieve data from the overlay. For interoperability reasons, RELOAD specifies one overlay algorithm, called "chord-reload", that is mandatory to implement. This document extends the chord-reload algorithm by introducing self-tuning behavior.

DHT-based overlay networks are self-organizing, scalable, and reliable. However, these features come at a cost: peers in the overlay network need to consume network bandwidth to maintain routing

state. Most DHTs use a periodic stabilization routine to counter the undesirable effects of churn on routing. To configure the parameters of a DHT, some characteristics such as churn rate and network size need to be known in advance. These characteristics are then used to configure the DHT in a static fashion by using fixed values for parameters such as the size of the successor set, size of the routing table, and rate of maintenance messages. The problem with this approach is that it is not possible to achieve a low failure rate and a low communication overhead by using fixed parameters. Instead, a better approach is to allow the system to take into account the evolution of network conditions and adapt to them.

This document extends the mandatory-to-implement chord-reload algorithm by making it self-tuning. The use of the self-tuning feature is optional. However, when used, it needs to be supported by all peers in the RELOAD overlay network. The fact that a RELOAD overlay uses the self-tuning feature is indicated in the RELOAD overlay configuration document using the CHORD-SELF-TUNING algorithm name specified in Section 9.2 in the topology-plugin element. Two main advantages of self-tuning are that users no longer need to tune every DHT parameter correctly for a given operating environment and that the system adapts to changing operating conditions.

The remainder of this document is structured as follows: Section 2 provides definitions of terms used in this document. Section 3 discusses alternative approaches to stabilization operations in DHTs, including reactive stabilization, periodic stabilization, and adaptive stabilization. Section 4 gives an introduction to the Chord DHT algorithm. Section 5 describes how this document extends the stabilization routine of the chord-reload algorithm. Section 6 describes how the stabilization rate and routing table size are calculated in an adaptive fashion.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses terminology and definitions from the RELOAD base specification [RFC6940].

numBitsInNodeId: Specifies the number of bits in a RELOAD Node-ID.

DHT: Distributed Hash Tables are a class of decentralized distributed systems that provide a lookup service similar to a regular hash table. Given a key, any peer participating in the

system can retrieve the value associated with that key. The responsibility for maintaining the mapping from keys to values is distributed among the peers.

Chord Ring: The Chord DHT uses ring topology and orders identifiers on an identifier circle of size 2^numBitsInNodeId. This identifier circle is called the Chord ring. On the Chord ring, the responsibility for a key k is assigned to the node whose identifier equals to or immediately follows k.

Finger Table: A data structure with up to (but typically less than) numBitsInNodeId entries maintained by each peer in a Chord-based overlay. The ith entry in the finger table of peer n contains the identity of the first peer that succeeds n by at least 2^(numBitsInNodeId-i) on the Chord ring. This peer is called the ith finger of peer n. As an example, the first entry in the finger table of peer n contains a peer halfway around the Chord ring from peer n. The purpose of the finger table is to accelerate lookups.

n.id: In this document, this abbreviation is used to refer to the Node-ID of peer n.

O(g(n)): Informally, saying that some equation f(n) = O(g(n)) means that f(n) is less than some constant multiple of g(n). For the formal definition, please refer to [Weiss1998].

Omega(g(n)): Informally, saying that some equation f(n) = Omega(g(n)) means that f(n) is more than some constant multiple of g(n). For the formal definition, please refer to [Weiss1998].

Percentile: The Pth (0<=P<=100) percentile of N values arranged in ascending order is obtained by first calculating the (ordinal) rank n=(P/100)*N, rounding the result to the nearest integer and then taking the value corresponding to that rank.

Predecessor List: A data structure containing the first r predecessors of a peer on the Chord ring.

Successor List: A data structure containing the first r successors of a peer on the Chord ring.

Neighborhood Set: A term used to refer to the set of peers included in the successor and predecessor lists of a given peer.

Routing Table: Contents of a given peer's routing table include the set of peers that the peer can use to route overlay messages. The routing table is made up of the finger table, successor list, and predecessor list.

3. Introduction to Stabilization in DHTs

DHTs use stabilization routines to counter the undesirable effects of churn on routing. The purpose of stabilization is to keep the routing information of each peer in the overlay consistent with the constantly changing overlay topology. There are two alternative approaches to stabilization: periodic and reactive [Rhea2004]. Periodic stabilization can either use a fixed stabilization rate or calculate the stabilization rate in an adaptive fashion.

3.1. Reactive versus Periodic Stabilization

In reactive stabilization, a peer reacts to the loss of a peer in its neighborhood set or to the appearance of a new peer that should be added to its neighborhood set by sending a copy of its neighbor table to all peers in the neighborhood set. Periodic recovery, in contrast, takes place independently of changes in the neighborhood set. In periodic recovery, a peer periodically shares its neighborhood set with each or a subset of the members of that set.

The chord-reload algorithm [RFC6940] supports both reactive and periodic stabilization. It has been shown in [Rhea2004] that reactive stabilization works well for small neighborhood sets (i.e., small overlays) and moderate churn. However, in large-scale (e.g., 1000 peers or more [Rhea2004]) or high-churn overlays, reactive stabilization runs the risk of creating a positive feedback cycle, which can eventually result in congestion collapse. In [Rhea2004], it is shown that a 1000-peer overlay under churn uses significantly less bandwidth and has lower latencies when periodic stabilization is used than when reactive stabilization is used. Although in the experiments carried out in [Rhea2004], reactive stabilization performed well when there was no churn, its bandwidth use was observed to jump dramatically under churn. At higher churn rates and larger scale overlays, periodic stabilization uses less bandwidth and the resulting lower contention for the network leads to lower latencies. For this reason, most DHTs, such as CAN [CAN], Chord [Chord], Pastry [Pastry], and Bamboo [Rhea2004], use periodic stabilization [Ghinita2006]. As an example, the first version of Bamboo used reactive stabilization, which caused Bamboo to suffer from degradation in performance under churn. To fix this problem, Bamboo was modified to use periodic stabilization.

In Chord, periodic stabilization is typically done both for successors and fingers. An alternative strategy is analyzed in [Krishnamurthy2008]. In this strategy, called the "correction-on-change maintenance strategy", a peer periodically stabilizes its successors but does not do so for its fingers. Instead, finger pointers are stabilized in a reactive fashion. The results obtained in [Krishnamurthy2008] imply that although the correction-on-change strategy works well when churn is low, periodic stabilization outperforms the correction-on-change strategy when churn is high.

3.2. Configuring Periodic Stabilization

When periodic stabilization is used, one faces the problem of selecting an appropriate execution rate for the stabilization procedure. If the execution rate of periodic stabilization is high, changes in the system can be quickly detected, but at the disadvantage of increased communication overhead. Alternatively, if the stabilization rate is low and the churn rate is high, routing tables become inaccurate and DHT performance deteriorates. Thus, the problem is setting the parameters so that the overlay achieves the desired reliability and performance even in challenging conditions, such as under heavy churn. This naturally results in high cost during periods when the churn level is lower than expected, or alternatively, poor performance or even network partitioning in worse than expected conditions.

In addition to selecting an appropriate stabilization interval, regardless of whether or not periodic stabilization is used, an appropriate size needs to be selected for the neighborhood set and for the finger table.

The current approach is to configure overlays statically. This works in situations where perfect information about the future is available. In situations where the operating conditions of the network are known in advance and remain static throughout the lifetime of the system, it is possible to choose fixed optimal values for parameters such as stabilization rate, neighborhood set size and routing table size. However, if the operating conditions (e.g., the size of the overlay and its churn rate) do not remain static but evolve with time, it is not possible to achieve both a low lookup failure rate and a low communication overhead by using fixed parameters [Ghinita2006].

As an example, to configure the Chord DHT algorithm, one needs to select values for the following parameters: size of successor list, stabilization interval, and size of the finger table. To select an appropriate value for the stabilization interval, one needs to know the expected churn rate and overlay size. According to

[Liben-Nowell2002], a Chord network in a ring-like state remains in a ring-like state as long as peers send Omega(square(log(N))) messages before N new peers join or N/2 peers fail. Thus, in a 500-peer overlay churning at a rate such that one peer joins and one peer leaves the network every 30 seconds, an appropriate stabilization interval would be on the order of 93 s. According to [Chord], the size of the successor list and finger table should be on the order of log(N). Already a successor list of a modest size (e.g., log2(N) or 2*log2(N), which is the successor list size used in [Chord]) makes it very unlikely that a peer will lose all of its successors, which would cause the Chord ring to become disconnected. Thus, in a 500-peer network each peer should maintain on the order of nine successors and fingers. However, if the churn rate doubles and the network size remains unchanged, the stabilization rate should double as well. That is, the appropriate maintenance interval would now be on the order of 46 s. On the other hand, if the churn rate becomes, e.g., six-fold and the size of the network grows to 2000 peers, on the order of 11 fingers and successors should be maintained and the stabilization interval should be on the order of 42 s. If one continued using the old values, this could result in inaccurate routing tables, network partitioning, and deteriorating performance.

3.3. Adaptive Stabilization

A self-tuning DHT takes into consideration the continuous evolution of network conditions and adapts to them. In a self-tuning DHT, each peer collects statistical data about the network and dynamically adjusts its stabilization rate, neighborhood set size, and finger table size based on the analysis of the data [Ghinita2006]. Reference [Mahajan2003] shows that by using self-tuning, it is possible to achieve high reliability and performance even in adverse conditions with low maintenance cost. Adaptive stabilization has been shown to outperform periodic stabilization in terms of both lookup failures and communication overhead [Ghinita2006].

4. Introduction to Chord

Chord [Chord] is a structured P2P algorithm that uses consistent hashing to build a DHT out of several independent peers. Consistent hashing assigns each peer and resource a fixed-length identifier. Peers use SHA-1 as the base hash function to generate the identifiers. As specified in RELOAD base [RFC6940], the length of the identifiers is numBitsInNodeId=128 bits. The identifiers are ordered on an identifier circle of size 2^numBitsInNodeId. On the identifier circle, key k is assigned to the first peer whose identifier equals or follows the identifier of k in the identifier space. The identifier circle is called the Chord ring.

Different DHTs differ significantly in performance when bandwidth is limited. It has been shown that when compared to other DHTs, the advantages of Chord include that it uses bandwidth efficiently and can achieve low lookup latencies at little cost [Li2004].

A simple lookup mechanism could be implemented on a Chord ring by requiring each peer to only know how to contact its current successor on the identifier circle. Queries for a given identifier could then be passed around the circle via the successor pointers until they encounter the first peer whose identifier is equal to or larger than the desired identifier. Such a lookup scheme uses a number of messages that grows linearly with the number of peers. To reduce the cost of lookups, Chord maintains also additional routing information; each peer n maintains a data structure with up to numBitsInNodeId entries, called the finger table. The first entry in the finger table of peer n contains the peer halfway around the ring from peer n. The second entry contains the peer that is 1/4th of the way around, the third entry the peer that is 1/8th of the way around, etc. In other words, the ith entry in the finger table at peer n contains the identity of the first peer s that succeeds n by at least 2^(numBitsInNodeId-i) on the Chord ring. This peer is called the ith finger of peer n. The interval between two consecutive fingers is called a finger interval. The ith finger interval of peer n covers the range [n.id + 2^(numBitsInNodeId-i), n.id + 2^(numBitsInNodeId-i+1)) on the Chord ring. In an N-peer network, each peer maintains information about O(log(N)) other peers in its finger table. As an example, if N=100000, it is sufficient to maintain 17 fingers.

Chord needs all peers' successor pointers to be up to date in order to ensure that lookups produce correct results as the set of participating peers changes. To achieve this, peers run a stabilization protocol periodically in the background. The stabilization protocol of the original Chord algorithm uses two operations: successor stabilization and finger stabilization. However, the Chord algorithm of RELOAD base defines two additional stabilization components, as will be discussed below.

To increase robustness in the event of peer failures, each Chord peer maintains a successor list of size r, containing the peer's first r successors. The benefit of successor lists is that if each peer fails independently with probability p, the probability that all r successors fail simultaneously is only p^r.

The original Chord algorithm maintains only a single predecessor pointer. However, multiple predecessor pointers (i.e., a predecessor list) can be maintained to speed up recovery from predecessor failures. The routing table of a peer consists of the successor list, finger table, and predecessor list.

5. Extending Chord-Reload to Support Self-Tuning

This section describes how the mandatory-to-implement chord-reload algorithm defined in RELOAD base [RFC6940] can be extended to support self-tuning.

The chord-reload algorithm supports both reactive and periodic recovery strategies. When the self-tuning mechanisms defined in this document are used, the periodic recovery strategy is used. Further, chord-reload specifies that at least three predecessors and three successors need to be maintained. When the self-tuning mechanisms are used, the appropriate sizes of the successor list and predecessor list are determined in an adaptive fashion based on the estimated network size, as will be described in Section 6.

As specified in RELOAD base [RFC6940], each peer maintains a stabilization timer. When the stabilization timer fires, the peer restarts the timer and carries out the overlay stabilization routine. Overlay stabilization has four components in chord-reload:

1. Update the neighbor table. We refer to this as "neighbor stabilization".

2. Refreshing the finger table. We refer to this as "finger stabilization".

3. Adjusting finger table size.

4. Detecting partitioning. We refer to this as "strong stabilization".

As specified in RELOAD base [RFC6940], a peer sends periodic messages as part of the neighbor stabilization, finger stabilization, and strong stabilization routines. In neighbor stabilization, a peer periodically sends an Update request to every peer in its connection table. The default time is every ten minutes. In finger stabilization, a peer periodically searches for new peers to include in its finger table. This time defaults to one hour. This document specifies how the neighbor stabilization and finger stabilization intervals can be determined in an adaptive fashion based on the operating conditions of the overlay. The subsections below describe how this document extends the four components of stabilization.

5.1. Update Requests

As described in RELOAD base [RFC6940], the neighbor and finger stabilization procedures are implemented using Update requests. RELOAD base defines three types of Update requests: 'peer_ready',

'neighbors', and 'full'. Regardless of the type, all Update requests include an 'uptime' field. The self-tuning extensions require information on the uptimes of peers in the routing table. The sender of an Update request includes its current uptime (in seconds) in the 'uptime' field. Regardless of the type, all Update requests MUST include an 'uptime' field.

When self-tuning is used, each peer decides independently the appropriate size for the successor list, predecessor list, and finger table. Thus, the 'predecessors', 'successors', and 'fingers' fields included in RELOAD Update requests are of variable length. As specified in RELOAD [RFC6940], variable-length fields are on the wire preceded by length bytes. In the case of the successor list, predecessor list, and finger table, there are two length bytes (allowing lengths up to 2^16-1). The number of NodeId structures included in each field can be calculated based on the length bytes since the size of a single NodeId structure is 16 bytes. If a peer receives more entries than fit into its successor list, predecessor list, or finger table, the peer MUST ignore the extra entries. A peer may also receive less entries than it currently has in its own data structure. In that case, it uses the received entries to update only a subset of the entries in its data structure. As an example, a peer that has a successor list of size 8 may receive a successor list of size 4 from its immediate successor. In that case, the received successor list can only be used to update the first few successors on the peer's successor list. The rest of the successors will remain intact.

5.2. Neighbor Stabilization

In the neighbor stabilization operation of chord-reload, a peer periodically sends an Update request to every peer in its connection table. In a small, low-churn overlay, the amount of traffic this process generates is typically acceptable. However, in a large-scale overlay churning at a moderate or high churn rate, the traffic load may no longer be acceptable since the size of the connection table is large and the stabilization interval relatively short. The self-tuning mechanisms described in this document are especially designed for overlays of the latter type. Therefore, when the self-tuning mechanisms are used, each peer only sends a periodic Update request to its first predecessor and first successor on the Chord ring; it MUST NOT send Update requests to others.

The neighbor stabilization routine is executed when the stabilization timer fires. To begin the neighbor stabilization routine, a peer sends an Update request to its first successor and its first predecessor. The type of the Update request MUST be 'neighbors'. The Update request includes the successor and predecessor lists of

the sender. If a peer receiving such an Update request learns from the predecessor and successor lists included in the request that new peers can be included in its neighborhood set, it sends Attach requests to the new peers.

After a new peer has been added to the predecessor or successor list, an Update request of type 'peer_ready' is sent to the new peer. This allows the new peer to insert the sender into its neighborhood set.

5.3. Finger Stabilization

Chord-reload specifies two alternative methods for searching for new peers to the finger table. Both of the alternatives can be used with the self-tuning extensions defined in this document.

Immediately after a new peer has been added to the finger table, a Probe request is sent to the new peer to fetch its uptime. The 'requested_info' field of the Probe request MUST be set to contain the ProbeInformationType 'uptime' defined in RELOAD base [RFC6940].

5.4. Adjusting Finger Table Size

The chord-reload algorithm defines how a peer can make sure that the finger table is appropriately sized to allow for efficient routing. Since the self-tuning mechanisms specified in this document produce a network size estimate, this estimate can be directly used to calculate the optimal size for the finger table. This mechanism is used instead of the one specified by chord-reload. A peer uses the network size estimate to determine whether it needs to adjust the size of its finger table each time when the stabilization timer fires. The way this is done is explained in Section 6.2.

5.5. Detecting Partitioning

This document does not require any changes to the mechanism chord-reload uses to detect network partitioning.

5.6. Leaving the Overlay

As specified in RELOAD base [RFC6940], a leaving peer SHOULD send a Leave request to all members of its neighbor table prior to leaving the overlay. The 'overlay_specific_data' field MUST contain the ChordLeaveData structure. The Leave requests that are sent to successors contain the predecessor list of the leaving peer. The Leave requests that are sent to the predecessors contain the successor list of the leaving peer. If a given successor can identify better predecessors (that is, predecessors that are closer to it on the Chord ring than its existing predecessors) than are

already included in its predecessor lists by investigating the predecessor list it receives from the leaving peer, it sends Attach requests to them. Similarly, if a given predecessor identifies better successors by investigating the successor list it receives from the leaving peer, it sends Attach requests to them.

6. Self-Tuning Chord Parameters

This section specifies how to determine an appropriate stabilization rate and routing table size in an adaptive fashion. The proposed mechanism is based on [Mahajan2003], [Liben-Nowell2002], and [Ghinita2006]. To calculate an appropriate stabilization rate, the values of three parameters must be estimated: overlay size N, failure rate U, and join rate L. To calculate an appropriate routing table size, the estimated network size N can be used. Peers in the overlay MUST recalculate the values of the parameters to self-tune the chord-reload algorithm at the end of each stabilization period before restarting the stabilization timer.

6.1. Estimating Overlay Size

Techniques for estimating the size of an overlay network have been proposed, for instance, in [Mahajan2003], [Horowitz2003], [Kostoulas2005], [Binzenhofer2006], and [Ghinita2006]. In Chord, the density of peer identifiers in the neighborhood set can be used to produce an estimate of the size of the overlay, N [Mahajan2003]. Since peer identifiers are picked randomly with uniform probability from the numBitsInNodeId-bit identifier space, the average distance between peer identifiers in the successor set is (2^numBitsInNodeId)/N.

To estimate the overlay network size, a peer computes the average inter-peer distance d between the successive peers starting from the most distant predecessor and ending to the most distant successor in the successor list. The estimated network size is calculated as:

$$N = \frac{2^{numBitsInNodeId}}{d}$$

This estimate has been found to be accurate within 15% of the real network size [Ghinita2006]. Of course, the size of the neighborhood set affects the accuracy of the estimate.

During the join process, a joining peer fills its routing table by sending a series of Ping and Attach requests, as specified in RELOAD base [RFC6940]. Thus, a joining peer immediately has enough information at its disposal to calculate an estimate of the network size.

6.2. Determining Routing Table Size

As specified in RELOAD base [RFC6940], the finger table must contain at least 16 entries. When the self-tuning mechanisms are used, the size of the finger table MUST be set to max(ceiling(log2(N)), 16) using the estimated network size N.

The size of the successor list MUST be set to a maximum of ceiling(log2(N)). An implementation can place a lower limit on the size of the successor list. As an example, the implementation might require the size of the successor list to be always at least three.

The size of the predecessor list MUST be set to ceiling(log2(N)).

6.3. Estimating Failure Rate

A typical approach is to assume that peers join the overlay according to a Poisson process with rate L and leave according to a Poisson process with rate parameter U [Mahajan2003]. The value of U can be estimated using peer failures in the finger table and neighborhood set [Mahajan2003]. If peers fail with rate U, a peer with M unique peer identifiers in its routing table should observe K failures in time K/(M*U). Every peer in the overlay maintains a history of the last K failures. The current time is inserted into the history when the peer joins the overlay. The estimate of U is calculated as:

$$U = \frac{k}{M * Tk},$$

where M is the number of unique peer identifiers in the routing table, Tk is the time between the first and the last failure in the history, and k is the number of failures in the history. If k is smaller than K, the estimate is computed as if there was a failure at the current time. It has been shown that an estimate calculated in a similar manner is accurate within 17% of the real value of U [Ghinita2006].

The size of the failure history K affects the accuracy of the estimate of U. One can increase the accuracy by increasing K. However, this has the side effect of decreasing responsiveness to changes in the failure rate. On the other hand, a small history size

may cause a peer to overreact each time a new failure occurs. In [Ghinita2006], K is set to 25% of the routing table size. Use of this value is RECOMMENDED.

6.3.1. Detecting Failures

A new failure is inserted to the failure history in the following cases:

1. A Leave request is received from a neighbor.

2. A peer fails to reply to a Ping request sent in the situation explained below. If no packets have been received on a connection during the past 2*Tr seconds (where Tr is the inactivity timer defined by Interactive Connectivity Establishment (ICE) [RFC5245]), a RELOAD Ping request MUST be sent to the remote peer. RELOAD mandates the use of Session Traversal Utilities for NAT (STUN) [RFC5389] for keepalives. STUN keepalives take the form of STUN Binding Indication transactions. As specified in ICE [RFC5245], a peer sends a STUN Binding Indication if there has been no packet sent on a connection for Tr seconds. Tr is configurable and has a default of 15 seconds. Although STUN Binding Indications do not generate a response, the fact that a peer has failed can be learned from the lack of packets (Binding Indications or application protocol packets) received from the peer. If the remote peer fails to reply to the Ping request, the sender should consider the remote peer to have failed.

As an alternative to relying on STUN keepalives to detect peer failure, a peer could send additional, frequent RELOAD messages to every peer in its connection table. These messages could be Update requests, in which case they would serve two purposes: detecting peer failure and stabilization. However, as the cost of this approach can be very high in terms of bandwidth consumption and traffic load, especially in large-scale overlays experiencing churn, its use is NOT RECOMMENDED.

6.4. Estimating Join Rate

Reference [Ghinita2006] proposes that a peer can estimate the join rate based on the uptime of the peers in its routing table. An increase in peer join rate will be reflected by a decrease in the average age of peers in the routing table. Thus, each peer maintained an array of the ages of the peers in its routing table sorted in increasing order. Using this information, an estimate of the global peer join rate L is calculated as:

$$L = \frac{N}{Ages[floor(rsize/2)]},$$

where Ages is an array containing the ages of the peers in the routing table sorted in increasing order and rsize is the size of the routing table. It has been shown that the estimate obtained by using this method is accurate within 22% of the real join rate [Ghinita2006]. Of course, the size of the routing table affects the accuracy.

In order for this mechanism to work, peers need to exchange information about the time they have been present in the overlay. Peers receive the uptimes of their successors and predecessors during the stabilization operations since all Update requests carry uptime values. A joining peer learns the uptime of the admitting peer since it receives an Update from the admitting peer during the join procedure. Peers learn the uptimes of new fingers since they can fetch the uptime using a Probe request after having attached to the new finger.

6.5. Estimate Sharing

To improve the accuracy of network size, join rate, and leave rate estimates, peers share their estimates. When the stabilization timer fires, a peer selects number-of-peers-to-probe random peers from its finger table and send each of them a Probe request. The targets of Probe requests are selected from the finger table rather than from the neighbor table since neighbors are likely to make similar errors when calculating their estimates. The number-of-peers-to-probe is a new element in the overlay configuration document. It is defined in Section 7. Both the Probe request and the answer returned by the target peer MUST contain a new message extension whose MessageExtensionType is 'self_tuning_data'. This extension type is defined in Section 9.1. The 'extension_contents' field of the MessageExtension structure MUST contain a SelfTuningData structure:

```
struct {
  uint32                  network_size;
  uint32                  join_rate;
  uint32                  leave_rate;
} SelfTuningData;
```

The contents of the SelfTuningData structure are as follows:

network_size

The latest network size estimate calculated by the sender.

join_rate

The latest join rate estimate calculated by the sender.

leave_rate

The latest leave rate estimate calculated by the sender.

The join and leave rates are expressed as joins or failures per 24 hours. As an example, if the global join rate estimate a peer has calculated is 0.123 peers/s, it would include in the 'join_rate' field the ceiling of the value 10627.2 (24*60*60*0.123 = 10627.2), that is, the value 10628.

The 'type' field of the MessageExtension structure MUST be set to contain the value 'self_tuning_data'. The 'critical' field of the structure MUST be set to False.

A peer stores all estimates it receives in Probe requests and answers during a stabilization interval. When the stabilization timer fires, the peer calculates the estimates to be used during the next stabilization interval by taking the 75th percentile (i.e., third quartile) of a data set containing its own estimate and the received estimates.

The default value for number-of-peers-to-probe is 4. This default value is recommended to allow a peer to receive a sufficiently large set of estimates from other peers. With a value of 4, a peer receives four estimates in Probe answers. On the average, each peer also receives four Probe requests each carrying an estimate. Thus, on the average, each peer has nine estimates (including its own) that it can use at the end of the stabilization interval. A value smaller than 4 is NOT RECOMMENDED to keep the number of received estimates high enough. As an example, if the value were 2, there would be peers in the overlay that would only receive two estimates during a stabilization interval. Such peers would only have three estimates available at the end of the interval, which may not be reliable enough since even a single exceptionally high or low estimate can have a large impact.

6.6. Calculating the Stabilization Interval

According to [Liben-Nowell2002], a Chord network in a ring-like state remains in a ring-like state as long as peers send Omega(square(log(N))) messages before N new peers join or N/2 peers fail. We can use the estimate of peer failure rate, U, to calculate the time Tf in which N/2 peers fail:

$$Tf = \frac{1}{2*U}$$

Based on this estimate, a stabilization interval Tstab-1 is calculated as:

$$Tstab\text{-}1 = \frac{Tf}{square(log2(N))}$$

On the other hand, the estimated join rate L can be used to calculate the time in which N new peers join the overlay. Based on the estimate of L, a stabilization interval Tstab-2 is calculated as:

$$Tstab\text{-}2 = \frac{N}{L * square(log2(N))}$$

Finally, the actual stabilization interval Tstab that is used can be obtained by taking the minimum of Tstab-1 and Tstab-2.

The results obtained in [Maenpaa2009] indicate that making the stabilization interval too small has the effect of making the overlay less stable (e.g., in terms of detected loops and path failures). Thus, a lower limit should be used for the stabilization period. Based on the results in [Maenpaa2009], a lower limit of 15 s is RECOMMENDED, since using a stabilization period smaller than this will with a high probability cause too much traffic in the overlay.

7. Overlay Configuration Document Extension

This document extends the RELOAD overlay configuration document by adding one new element, "number-of-peers-to-probe", inside each "configuration" element.

self-tuning:number-of-peers-to-probe: The number of fingers to which Probe requests are sent to obtain their network size, join rate, and leave rate estimates. The default value is 4.

The RELAX NG grammar for this element is:

```
namespace self-tuning = "urn:ietf:params:xml:ns:p2p:self-tuning"

parameter &= element self-tuning:number-of-peers-to-probe {
xsd:unsignedInt }?
```

This namespace is added into the <mandatory-extension> element in the overlay configuration file.

8. Security Considerations

In the same way as malicious or compromised peers implementing the RELOAD base protocol [RFC6940] can advertise false network metrics or distribute false routing table information for instance in RELOAD Update messages, malicious peers implementing this specification may share false join rate, leave rate, and network size estimates. For such attacks, the same security concerns apply as in the RELOAD base specification. In addition, as long as the amount of malicious peers in the overlay remains modest, the statistical mechanisms applied in Section 6.5 (i.e., the use of 75th percentiles) to process the shared estimates a peer obtains help ensure that estimates that are clearly different from (i.e., larger or smaller than) other received estimates will not significantly influence the process of adapting the stabilization interval and routing table size. However, it should be noted that if an attacker is able to impersonate a high number of other peers in the overlay in strategic locations, it may be able to send a high enough number of false estimates to a victim and therefore influence the victim's choice of a stabilization interval.

9. IANA Considerations

9.1. Message Extensions

This document introduces one additional extension to the "RELOAD Extensions Registry":

Extension Name	Code	Specification
self_tuning_data	0x3	RFC 7363

The contents of the extension are defined in Section 6.5.

9.2. New Overlay Algorithm Type

This document introduces one additional overlay algorithm type to the "RELOAD Overlay Algorithm Types" registry:

Algorithm Name	Reference
CHORD-SELF-TUNING	RFC 7363

9.3. A New IETF XML Registry

This document registers one new URI for the self-tuning namespace in the "ns" subregistry of the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:p2p:self-tuning

Registrant Contact: The IESG

XML: N/A, the requested URI is an XML namespace

10. Acknowledgments

The authors would like to thank Jani Hautakorpi for his contributions to the document. The authors would also like to thank Carlos Bernardos, Martin Durst, Alissa Cooper, Tobias Gondrom, and Barry Leiba for their comments on the document.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC6940] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", RFC 6940, January 2014.

11.2. Informative References

[Binzenhofer2006]
Binzenhofer, A., Kunzmann, G., and R. Henjes, "A Scalable Algorithm to Monitor Chord-Based P2P Systems at Runtime", In Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1-8, April 2006.

[CAN] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and S. Schenker, "A Scalable Content-Addressable Network", In Proceedings of the 2001 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications, pp. 161-172, August 2001.

[Chord] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", IEEE/ACM Transactions on Networking, Volume 11, Issue 1, pp. 17-32, February 2003.

[Ghinita2006]
Ghinita, G. and Y. Teo, "An Adaptive Stabilization Framework for Distributed Hash Tables", In Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 29-38, April 2006.

[Horowitz2003]
Horowitz, K. and D. Malkhi, "Estimating Network Size from Local Information", Information Processing Letters, Volume 88, Issue 5, pp. 237-243, December 2003.

[Kostoulas2005]
Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K., and A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups", In Proceedings of the 4th IEEE International Symposium on Network Computing and Applications, pp. 41-48, July 2005.

[Krishnamurthy2008]
Krishnamurthy, S., El-Ansary, S., Aurell, E., and S. Haridi, "Comparing Maintenance Strategies for Overlays", In Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 473-482, February 2008.

[Li2004] Li, J., Strinbling, J., Gil, T., Morris, R., and M. Kaashoek, "Comparing the Performance of Distributed Hash Tables Under Churn", Peer-to-Peer Systems III, Volume 3279 of Lecture Notes in Computer Science, Springer, pp. 87-99, February 2005.

[Liben-Nowell2002]
Liben-Nowell, D., Balakrishnan, H., and D. Karger, "Observations on the Dynamic Evolution of Peer-to-Peer Networks", In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), pp. 22-33, March 2002.

[Maenpaa2009]
Maenpaa, J. and G. Camarillo, "A Study on Maintenance Operations in a Chord-Based Peer-to-Peer Session Initiation Protocol Overlay Network", In Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1-9, May 2009.

[Mahajan2003]
Mahajan, R., Castro, M., and A. Rowstron, "Controlling the Cost of Reliability in Peer-to-Peer Overlays", In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), pp. 21-32, February 2003.

[Pastry] Rowstron, A. and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems", In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, pp. 329-350, November 2001.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[Rhea2004]
Rhea, S., Geels, D., Roscoe, T., and J. Kubiatowicz, "Handling Churn in a DHT", In Proceedings of the USENIX Annual Technical Conference, pp. 127-140, June 2004.

[Weiss1998]
Weiss, M., "Data Structures and Algorithm Analysis in C++", Addison-Wesley Longman Publishing Co., Inc., 2nd Edition, ISBN 0201361221, 1998.

Authors' Addresses

Jouni Maenpaa
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Jouni.Maenpaa@ericsson.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: Gonzalo.Camarillo@ericsson.com