Self-Stabilizing Structured Ring Topology P2P Systems

Ayman Shaker and Douglas S. Reeves
Departments of Electrical and Computer Engineering and Computer Science
N. C. State University
{ashaker,reeves}@ncsu.edu

Abstract

We propose a self-stabilizing and modeless peer-to-peer(P2P) network construction and maintenance protocol, called the Ring Network*(RN) protocol. The RN protocol, when started on a network of peers that are in an arbitrary state, will cause the network to converge to a structured P2P system with a directed ring topology, where peers are ordered according to their identifiers. Furthermore, the RN protocol maintains this structure in the face of peer joins and departures.*

The RN protocol is a distributed and asynchronous message-passing protocol, which fits well the autonomous behavior of peers in a P2P system. The RN protocol requires only the existence of a bootstrapping system which is weakly connected. Peers do not need to be informed of any global network state, nor do they need to assist in repairing the network topology when they leave. We provide a proof of the self-stabilizing nature of the protocol, and experimentally measure the average cost (in time and number of messages) to achieve convergence.

1 Introduction

P2P systems are decentralized and have several attractive properties like fault tolerance, load-balancing and self-scaling[3]. P2P systems construct and maintain overlay networks with different network topologies, depending on application requirements. *Structured* P2P systems impose specific local relationships between the peers, which gives rise to a global structure. The advantage of this structure is that it can then be used by the P2P system for efficient data placement, search, and retrieval.

Current protocols for the construction and maintenance of specific structures generally rely on simplifying assumptions that are either undesirable or unrealistic. First, most such systems require or assume a specific initial configuration of the peers. Second, the construction and maintenance protocols assume that the network remains in an ideal or close to ideal topology at all times. If the topology of the network deviates from this ideal, then these construction and maintenance protocols may cease to function correctly. Third, a dynamic network, in the sense that peers arrive and depart continuously, can become partitioned. If this happens, the construction and maintenance protocols generally have no way to bring the network back to the desired topology. One result of these limitations is the difficulty of merging two structured P2P networks efficiently and smoothly. Existing construction and maintenance protocols generally do not allow such merging, even if each network individually has the desired structure.

In this paper we propose the *Ring Network*(RN) protocol; a distributed construction and maintenance protocol that gives rise to a sorted *Ring Network*, starting from an arbitrary topology. This protocol is able to correctly merge two P2P networks into a single ring, or fix a network that is somehow partitioned. Furthermore, the protocol does not require the manual creation of a "seed" network, unlike most structured P2P systems. The protocol also does not require peers to be informed of or monitor the global state of the network, before deciding what updates to perform. Each peer simply runs the proposed construction and maintenance protocols, without specific knowledge of the current state of the network topology, and independently of other peers.

Existing methods of constructing structured P2P overlay networks generally adopt a top-down view of topology construction and maintenance. These methods may require central coordination, may have difficulty keeping up with the rate of change of typical P2P systems, and/or may not tolerate or recover from errors or failures. Efforts to contain or eliminate these problems sometimes lead to complex protocol "fixes". Our method, in contrast, may be thought of as a bottom-up approach to constructing structured networks, in a way that is simple, robust, and fully distributed.

The next section presents the network model and a formal definition of the problem. Section 3 is a review of related work. Section 4 describes the proposed protocol. Section 5 evaluates the correctness and the performance of the protocol. The paper is concluded in section 6.

2 Problem Statement

A P2P network is a logical overlay network formed on top of a fixed infrastructure network, such as the Internet. Therefore, any peer u in the network can communicate or send a

message to any other peer v directly, if u knows the network address of v. In such a case, v is said to be a *neighbor* of u.

A P2P network topology is defined by the set of peers in a network, and the neighbor relationships between those peers. This topology can be represented by a graph, with vertexes representing peers, and a directed edge from vertex u to vertex v if v is a neighbor of u. Let the neighbors of peer u be denoted as $u.\Gamma$. [1] We further denote the ith neighbor of peer u by $u.\Gamma_i$. The 0th neighbor, $u.\Gamma_0$, is a special neighbor of u and is also called the *successor* of u. If v is the successor of u, then u is a *predecessor* of v. If v is the ith neighbor of u, $v = u.\Gamma_i$, then let $u.index(v) = i$.

Each peer has a unique identifier or ID, in the form of a nonnegative integer. Let the largest possible ID assignable to a peer be $m - 1$. We assume each peer knows the value of m. The distance from a peer u to a peer v is defined by a distance function, $d_m(u, v) = v - u$, where subtraction is performed modulo m.[2] For simplicity of presentation, we omit the subscript m from the distance function in this paper. The smaller the value of $d(u, v)$ the *closer* v is said to be to u. For the purposes of this paper we define a P2P network as being a *ring network* if for each peer u in the network, u's successor, v, is the peer closest to u from the set of all peers and $v \neq u$ (unless the system consists of a single peer, in which case the successor of u will be itself). Note that the *ring network* is directed. If v is the neighbor of u, then the opposite is not necessarily true.

We refer to a peer v as being *between* a peer u and a peer w if v is closer to u than w is, or if $v = w$. As an alternative representation, we will write $u < v \leq w$ when v is *between* u and w. This definition is not dependent on the topology of the network; it is simply the relative ordering of three peers, according to their identifiers.

To fully define a P2P system, one has to take into consideration *bootstrapping*, which is the method by which peers are initially made aware of other peers in the system. Let the peers known to the bootstrapping system, and whose addresses are returned by the bootstrapping system in response to queries, be termed the *bootstrapping peers*. The bootstrapping peers and the neighbor connections between them must form a *weakly connected graph*. A weakly connected graph is a directed graph in which there is a path between any two vertexes, ignoring the directionality of the edges. In a strongly connected graph, there is a path between any two vertexes, considering directionality of the edges. All strongly connected graphs are also weakly connected, but the reverse is not true. We now prove that weakly connected bootstrapping systems are necessary and sufficient to ensure that all peers will be able to communicate once they join together in a P2P network topology.

Proposition 2.1. *For all peers to be able to communicate by means of the topology created by a P2P system, if each peer only queries the bootstrapping system once it is necessary for the bootstrapping system to be weakly connected.*

Proof. If the bootstrapping system is not weakly connected, there there will be at least two peers u and v for which there is neither a path from u to v, nor a path from v to u. It is therefore impossible for u and v to communicate with each other through any sequence of neighbors. □

Proposition 2.2. *For all peers to be able to communicate by means of the topology created by a P2P system, it is a sufficient condition for the bootstrapping system to be weakly connected.*

Proof. Assume that the every peer is connected to at least one bootstrapping peer, and the bootstrapping peers are weakly connected. If every peer u then notifies each of its neighbors v of its address, and v forms a connection to u, the resulting network will be strongly connected. Therefore, every peer will be able to communicate with every other peer through some set of peers. □

Therefore, a weakly connected bootstrapping system is the simplest and most general form of bootstrapping. We call such a bootstrapping system a *minimal bootstrapping system*. Ensuring the bootstrapping system is weakly connected in general is not difficult. For instance, connecting the bootstrapping peers together in an unordered chain will make them weakly connected. In some previous works, the bootstrapping system is replaced by a *seed network*. Most seed networks are required to have a minimum size and/or a specific structure, and this structure may be highly constrained. In fact, the seed network may need to be an instance of the target topology which the construction protocol seeks to create and maintain. In contrast, a *minimal bootstrapping system* imposes no such requirement, and is appropriate for constructing arbitrary topologies.

We next give the definition of a self-stabilizing P2P system. We define a self-stabilizing P2P protocol as one which can give rise to a desired topology starting from the simplest of necessary and sufficient assumptions.

Definition 2.3 (Self-Stabilizing P2P system). *A P2P system that constructs a desired topology, starting with each non-bootstrapping peer having an arbitrary set of neighbors (including no neighbors). To achieve this, the P2P system relies only on the existence of a minimal bootstrapping system.*

Some P2P systems may function when the bootstrapping system is minimal, but impose requirements on the initial neighbor relationships of non-bootstrapping peers. Such P2P systems will not be self-stabilizing, because they depend on a specific initial topology. In addition, a P2P system that assumes the bootstrapping system is strongly connected is not a self-stabilizing P2P system.

In this paper, we assume an asynchronous, reliable message-passing model. This means a message sent from peer u to peer v is eventually delivered in a finite but unbounded time. Furthermore, we assume that messages sent between two peers in the same direction follow the FIFO rule, as would

[1] When referencing a variable v stored by peer u, we will write $u.v$.

[2] The distance function is unidirectional. The distance from v to u is not necessarily the same as the distance from u to v.

be the case over a single TCP connection. We allow peers to execute steps of the protocol in an asynchronous fashion. A synchronous protocol would assume protocol execution occurs at the same rate (or even at the same time) on all peers, and that message transmission time is bounded. In many cases, this simplifies the analysis of the protocol. On the other hand, asynchronous protocols assume only that transmission time is finite, and do not make any assumption about the relative speed of protocol execution on different peers. This leads to the following definition:

Definition 2.4 (Distributed Asynchronous P2P system). *A P2P system in which the form of communication between peers is message-passing, where message transmission time is finite but unbounded. Furthermore, there is no shared memory or shared clock(s) between peers. All peers execute the same protocol.*

In our network model peer failures and departures follow the fail-stop model as specified in [14]: peers are able to perfectly detect the failure or departure of other peers in the P2P network. We define a modeless P2P system as follows:

Definition 2.5 (Modeless P2P protocol). *A modeless P2P protocol is one in which it is not required for every peer to be notified of each peer arrival or departure, or to be aware of the total number of peers.*

Modeless P2P protocols are highly desirable. They avoid the need for global communication among peers, or the need for periodic protocol synchronization, and are therefore faster and have lower overhead.

We now define our problem with reference to above properties: *Given are n peers, each with an arbitrary set of neighbors (or no neighbors), and a minimal bootstrapping process. Design a modeless and distributed asynchronous P2P protocol that is self-stabilizing, and that results in the construction of a ring network.*

3 Related Work

The problem of organizing nodes in a distributed system into a desired topology has been approached in many ways. This review of related work focuses on two properties: (i) whether the methods are self-stabilizing, and (ii) whether the protocols are modeless or not.

P2P research has addressed the organization of peers into a desired topological structure. Many structured P2P systems are based on distributed hash tables (DHTs) [9]. One of the most widely-cited such P2P system is Chord [17]. In [14], Liben-Nowell et al. describe maintenance protocols for Chord that may be run by each peer in order to keep the topology from deviating substantially from a chordal ring. The Chord protocols, however, always assume that the topology has some partial structure. In the absence of such minimal structure, the Chord protocols cease to guarantee convergence to the correct topology. Therefore, they do not qualify as self-stabilizing.

In [1], Angluin et al. solve the problem identified in this paper, i.e., starting from a weakly connected network, build an ordered ring network. Though the protocols give rise to a structured P2P network with low complexity, their solution does not address failures or joins of peers. Furthermore, all peers are required to start execution of the protocols at the same time. Therefore, although the protocol presented in this work is self-stabilizing, it is neither asynchronous nor modeless.

Topology construction using gossip-based protocols has been introduced in the T-Man protocol [7]. In this work, peers are organized according to a ranking function to produce a desired network topology, such as ring or torus topologies. T-man assumes that the network starts out with a topology close to a random graph, which is a stronger requirement than a weakly connected bootstrapping system. In addition, global knowledge of the system size and loose synchronization are required.

Li et al. propose Ranch [13], a set of protocols for the construction of P2P systems with ring topologies. The main contribution of this paper is the introduction of a provably correct, asynchronous protocol that can handle concurrent arrivals and departures of peers. However, the method assumes that departing peers always perform clean-up protocols prior to their departure. Furthermore, Ranch is not able to guarantee the correct topology will be constructed if the network is started from an arbitrary state.

Other structured P2P systems (CAN [5], Pastry [16], Tapestry [19], and others [15, 8, 2, 10]) likewise cannot construct the desired network topology starting from an arbitrary initial state, and/or are synchronous rather than asynchronous.

Another related field is resource discovery. One approach [12] is a distributed protocol that will construct a star topology, given an initial weakly connected network. This method requires one node to have special capacities, and is completely dependent on the correct and timely functioning of this central node. Kutten and Peleg [11] later introduced an asynchronous protocol with the same goal. Their method is self-stabilizing but is not modeless, and still targets the star topology, with the drawbacks mentioned above.

4 The Ring Network(RN) Protocol

This section presents the RN protocol, which constructs a *ring network* starting from an arbitrary state. We start with an overview, followed by a description of the protocol, and conclude with detailed pseudocode.

The RN protocol is a distributed protocol in which every peer independently and asynchronously executes the same procedure, and peers communicate by asynchronous message passing. In this protocol, each peer periodically initiates a search for a successor candidate. Peers that assist in this search will make note of the information contained in any message they propagate. Using information returned by successor searches, the bootstrapping process, or gleaned from mes-

sage propagation, each peer will independently select a (closest) successor node. This process repeats indefinitely, allowing rapid adaptation to changes in the set of peers.

Local information stored by each peer includes:

- Γ: the set of current neighbors of the peer.
- W: the set of peers returned by *Closer-Peer Searches*.
- B: the set of peers learned by the *Search Monitor* protocol.
- s: a peer selected randomly from the current successor, and peers returned by the bootstrapping system.

The three steps of the protocol are now described in more detail.

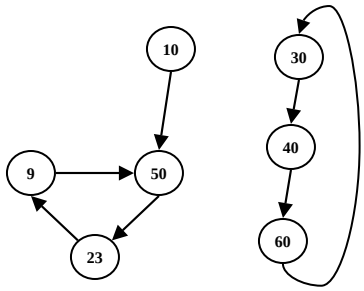
4.1 The Closer-Peer Search

Each peer x will periodically initiate a search for a peer that is closer to it than its current successor. x contacts for this purpose a peer s that is randomly chosen from its current successor $x.\Gamma_0$, or a peer returned by the bootstrapping service. The peer s receiving this request forwards it to that neighbor of s to which x is closest. The receiver of this request message propagates this request in a similar manner, in an attempt to get closer and closer to x. When a peer u receiving such a request finds that the initiator x of the request is closer to u than any of u's neighbors, the search terminates. u then sends to the initiator x the address and identity of its own successor $u.\Gamma_0$, which x adds to its set W. The frequency of this search affects only the performance of the protocol, not its correctness. The evaluation of performance factors is addressed in section 5.2

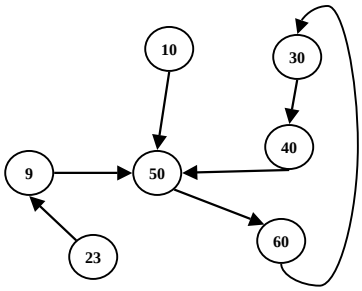
The result of this search will depend on the current topology of the network. If the network is already a ring network, a Closer-Peer Search will terminate at the peer that initiated the search, and no change will occur. Otherwise, the search may terminate at a peer u to which x is close. In such a case, x will be between u and $u.\Gamma_0$, and $u.\Gamma_0$ will be a promising candidate to be x's successor. As an example, in figure 1(a) peer 50 starts a Closer-Peer Search starting at peer 30. The search terminates at peer 40, which notifies peer 50 about its successor 60. Peer 50 then adds the new successor candidate 60 to peer 50's set W.

4.2 The Search Monitor

A peer u will monitor each *Closer-Peer Search* request that it receives. A search request initiated by x ($x \neq u$) and terminating at u means that x is closer to u than u's current successor $u.\Gamma_0$. u as a result adds the address and identity of x to its set B. In the previous example, peer 40 in figure 1(a) adds 50 to its set B, because the Closer-Peer Search initiated by 50 terminated at 40.



(a) A P2P network



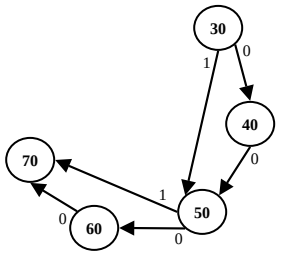
(b) Closer-Peer Search

Figure 1. Examples of a P2P network in the process of converging into a *ring network*. Circles represent peers; numbers inside circles are peer IDs. Edges represent successor connections. (a) A topology consisting of 7 peers. (b) A possible result of the execution of the *Closer-Peer Search* by peer 50, starting from peer 30.

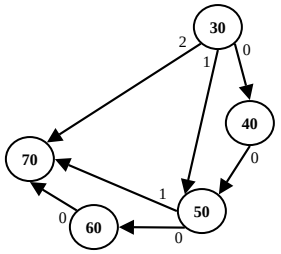
4.3 Neighbor Update

Each peer u will frequently check whether it has learned of a closer peer than its current successor $u.\Gamma_0$. It examines for this purpose its current list of neighbors, a bootstrapping peer returned by the bootstrapping process, the set W of candidates returned by completed Closer-Peer Searches, and the set B of candidates it compiled while propagating search request messages on behalf of other peers. The peer closest to u from among the union of these is chosen to be the new successor, $u.\Gamma_0$. In figure 1(a), after W and B have been updated, peer 40 and peer 50 update their successors as shown in figure 1(b).

In parallel with successor updating, each peer u will update its list of other neighbors, starting from $u.\Gamma_1$, proceeding to $u.\Gamma_2$, and so forth. u sends a message to neighbor $u.\Gamma_i$ asking it to return the identity of $u.\Gamma_i$'s ith neighbor. When $u.\Gamma_i$ responds with the identity of such a peer v, u will use this as the



(a) Before neighbor update.



(b) After the neighbor update.

Figure 2. A possible outcome of a *neighbor update* performed by peer 30. Edges represent neighbor connections. An edge labeled with the number i at its tail, points to the ith neighbor.

new value of $u.\Gamma_{i+1}$. However, if v is not between $u.\Gamma_i$ and u, the value of $u.\Gamma_{i+1}$ will not be updated. This update process is a form of *pointer jumping*, as proposed in [6]. The purpose of these neighbor connections is to improve the speed of searching, by reducing the number of hops that must be traversed. They are similar, but not identical, to the fingers employed by Chord[17].

An example is demonstrated in figure 2. Peer 30 would like to update its 2nd neighbor in figure 2(a). For that, it will query its 1st neighbor, peer 50, for the 1st neighbor of peer 50. When peer 50 responds, peer 30 will set as its 2nd neighbor peer 70, as shown in figure 2(b). This is because peer 70 is between peer 50 and peer 30, i.e., $50 < 70 < 30$. Peer 30 is therefore able to discover a peer that is 4 hops away from it, using two messages.

4.4 Detailed Pseudocode

We now present the pseudocode for the RN protocol. An asynchronous, message-passing model is assumed. The pseudocode is written in the AP notation introduced in [4]. In this notation, a peer p is specified by its **const**, **input** and **var** variables, and its actions. An action is of the form $\langle guard \rangle \rightarrow \langle statement \rangle$. Actions are delimited by two square brackets, []. The statement of an action can be executed only if the guard for that action evaluates to true. An execution step for the protocol is as follows. All guards of all actions of all peers are evaluated, after which only one statement of an action whose guard evaluated to true is executed. The choice of statement to execute, if there is more than one whose guard evaluates to true, is random. This ensures that each enabled action will eventually be executed, but the order of execution is arbitrary.

We choose this notation for describing the RN protocol because it facilitates an analysis of correctness, for an arbitrary interleaving of actions by the peers. This analysis model has been widely adopted by the distributed processing community [4]. Since the protocol is asynchronous, the need for interrupts is eliminated when using this notation, making the protocol presentation clearer. In section 5.2, we assume a specific execution rate by the peers, and measure experimentally the resulting convergence time of the algorithm.

Using this notation, the pseudocode for the RN protocol is shown in figure 3.

```
peer u
const
   T : set of bootstrapping peers
input
   w : a peer (successor candidate)
   x : peer being searched for
   c : index of received neighbor
   z : new neighbor
   s : a bootstrapping peer
var
   S : Set of peers
   B : Set of successor candidates
   W : Set of successor candidates
   Γ_i : ith neighbor

(a1)  true →
        S := {s} ∪ W ∪ B ∪ Γ
        Γ_0 := arg min_{k∈S} d(u, k)
        B := W := ∅
  []
(a2)  true →
        s:= Get random peer from {T ∪ Γ_0}
        send closer-peer search(u) to s
  []
(a3)  receive closer-peer search(x) from q →
        if x is closer to u than any neighbor ∈ Γ
        then B := B ∪ {x}
             send successorCandidate(Γ_0) to x
        else
             send closer-peer search(x) to arg min_{k∈Γ} d(k, x)
  []
(a4)  receive successorCandidate(w) from q →
        W := W ∪ {w}
  []
(a5)  true →
        for each h in Γ do
        send getNeighbor(index(h)) to h

  []
(a6)  receive getNeighbor(j) from q →
        if Γ_j exists
        then send neighbor(Γ_j, j) to q
  []
(a7)  receive neighbor(z, c) from q →
        if Γ_c ≤ z < u
        then Γ_{c+1} := z
      else Γ_{c+1} := NIL
```

Figure 3. Pseudocode of the RN protocol for a peer u.

5 Evaluation

In this section, we will first analyze the RN protocols with respect to self-stabilization. After that, we present the results of experiments measuring message and time complexity, and show the behavior under dynamic conditions.

5.1 Analysis

Without loss of generality, assume there are n peers, whose identities are restricted to the range $\{0 \ldots (n-1)\}$. Also, assume that no peer arrivals or departures occur while the network is converging into a ring network. Proof of convergence of any distributed protocol is generally only possible if the set of inputs does not change during convergence, since in most cases a different set of inputs will lead to a different solution. Given a proof of correctness when starting from an arbitrary state, it is clear that correctness will be recovered following any change (peers joining or leaving), as long as sufficient time for convergence to the new solution is allowed.

The goal of this section is to prove convergence in the static case. In the following section, we provide experimental results showing how long the convergence time is, and the effect of dynamic joins and leaves.

Lemma 5.1. *If the result of a peer x initiating a Closer-Peer Search is the return of a peer $v \neq x$, then v's predecessor must update its successor to be a closer peer.*

Proof. According to the action (a3) in the pseudocode, a value is returned to x when a node x is between a node u and u's successor v. Also in action (a3), it may be seen that u stores a reference to x. Therefore, when u next runs action (a1), it is guaranteed to change its successor. □

Lemma 5.2. *If the P2P system is not in a ring network topology, then at least one peer will find a closer peer than its current successor when it performs a Closer-Peer Search.*

Proof. Let G_s be the graph induced by the successor neighbors, so that the outdegree of every vertex is 1. Therefore, $G_s = (V, E_s)$, where V is the set of all peers and there is an edge $(u, v) \in E_s$ if, and only if, $v = u.\Gamma_0$. G_s can be either disconnected or weakly connected. By disconnected we mean that there exist a pair of peers, u and v, such that no path exists between them in either direction.

For the first case, if G_s is disconnected, then there exist at least two disjoint graph components. Since, the bootstrapping system is guaranteed to be weakly connected, then, by extension, it is guaranteed that at least one peer u in one component will have another peer, v, in another component as its bootstrapping peer. Therefore, if a Closer-Peer Search is performed by u starting at v then u will find a peer in another component. According to lemma 5.1 this will cause one peer to change its successor to a closer peer.

For the case when G_s is weakly connected, there are 3 subcases:

1. All the peers are part of a single cycle. In this case, since the network is not in a ring network topology, there exists at least one peer u that does not have as its successor the closest possible peer. Therefore, if u performs a Closer-Peer Search starting at u's successor, then it is guaranteed that the search will find a closer peer to u, and the P2P system continues to converge.
2. The graph is acyclic. This means the graph is a tree. The leaves of the tree have an in-degree of zero. This implies that if a leaf performs a Closer-Peer Search starting with its successor then it will find a different peer than itself. According to lemma 5.1 the P2P system will keep converging.
3. There is a single cycle in the graph that does not contain all the peers. Since the out-degree for the graph is 1, then it is impossible to have more than one cycle in the graph. Furthermore, the peers that are not part of the cycle will induce a tree graph. As stated in subcase 2 above, the P2P system will continue to converge. □

Theorem 5.3. *(Correctness) Once the P2P system is a ring network, it will remain so (assuming there are no changes in the set of peers).*

Proof. As can be seen from the pseudocode, the only location where a peer u might change its successor is in action (a1). A successor is replaced with another peer if, and only if, a closer peer to u is found. Once the network is a ring network, by definition every peer's successor is the closest peer to it in the system. Therefore, u will never encounter a peer that is closer to it than its current successor. □

Theorem 5.4. *(Convergence) Starting from any state, the P2P system will converge to a ring network.*

Proof. As can be seen from the pseudocode, the only location where a peer u might change its successor is in action (a1). A successor is replaced with another peer if, and only if, a closer peer to u is found. Otherwise, u will keep its current successor. Therefore, the convergence function will never increase the sum of the distances from all peers to their successors. From lemma 5.2 we also see that as long as the network is not in a ring network topology, then at least one peer will find a successor candidate that is closer than its current successor. Since distances are finite, as long as at least one distance to a successor decreases, convergence is eventually guaranteed. □

5.2 Experiments

We used simulation to evaluate the overhead and scalability of the RN protocol. In the first set of experiments, the set of peers was static. The second set of experiments allowed peers to join and leave dynamically. Assumptions were as described in section 2. We simulated the P2P network using Netlogo, an agent-based simulator [18]. The simulation model was identical in implementation to the pseudocode in figure 3, with one exception. Action $a5$ was modified so that only one

send occurs. Neighbors are contacted in order, as shown in $a5$, but only one per execution of the statement, rather than all neighbors in one execution. This change reduces the number of messages that are sent, but does not affect the correctness of the protocol.

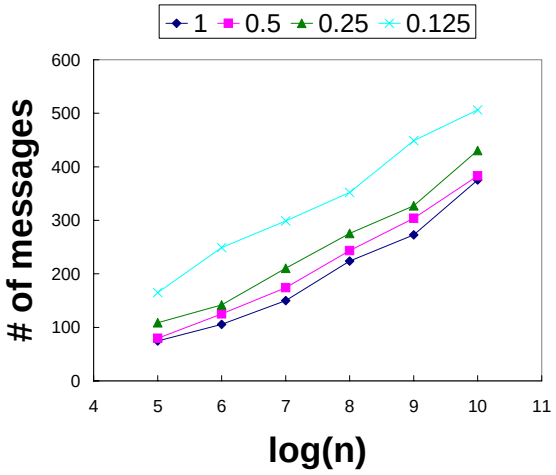
The configuration of the bootstrapping system can potentially affect performance. In our experiments, a random peer from the bootstrapping set was returned for each request. The size of the set of bootstrapping peers was varied to measure the impact of this parameter, from 12.5% up to 100% of the peers in the system. Bootstrapping peers were initially connected in an unordered chain, which is a simple form of a weakly connected graph.

The unit of time measured in the first two experiments was the maximum time to deliver a message. We fixed this time for purposes of illustration; our protocol is asynchronous and does not depend on this assumption for correctness. Communication was non-blocking and peers did not wait for a response to any message they sent. The simulation proceeded by randomly interleaving the actions of all peers for which the guard evaluated to true.

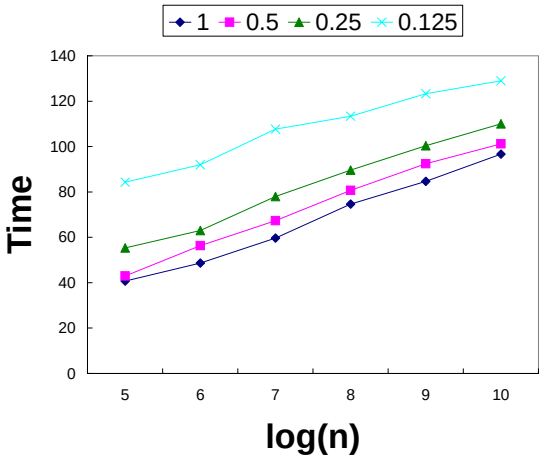
The first experiment measured the average number of messages sent per peer during construction of the ring network, starting with peers having no neighbors. Figure 4(a) shows the result for P2P networks of size 2^i, for $i \in \{5, \ldots, 10\}$. For each point, the 95% confidence interval was less than 5% of the measured value. The second experiment measured the time to converge on a ring network topology, under the same conditions, and with similar confidence intervals. Both number of messages and time to convergence grow logarithmically with the number of peers in these experiments, which indicates the protocol should scale well to large networks.

The third experiment investigated the effect of concurrent peer arrivals and departures in the P2P system. The simulation was started with a P2P system of 1024 peers already configured in a ring network. Peer arrival and departure rates were varied from 2 per unit time to 12 per unit time, where the unit of time was taken as the time to execute one statement at each peer. This reflects then a very high peer churn rate. Peer lookups were performed concurrently with peer arrivals and departures. For each lookup, a randomly-selected peer searched for another randomly-selected peer in the network. A search could fail if the two peers were not connected, or if the set of peer successors had not converged to the correct configuration.

The results of this experiment are presented in figure 5. For searches which succeeded, the hop-count for the search was measured. Without dynamic updates, for a converged system the failure rate should be zero, and the average search hop-count should be $(1/2)\log(n)$, where n is the size of the network. From the results, the hop-count for successful searches grows slowly as the number of peers leaving or arriving per unit of time grows to a very high level. The search failure rate is moderately high under these conditions, but with no attempt to exploit redundancy. If the RN protocol is modified to maintain multiple successors per peer, as in Chord [17], the failure rate would be substantially lower and path lengths shorter, at the expense of greater complexity and higher maintainence overhead.



(a) Message Complexity



(b) Time Complexity

Figure 4. Time and number of messages exchanged per peer for a network of size n to converge into a ring. The label for each line represents the fraction of peers that serve as bootstrapping peers.

We also measured the time for the RN protocol to restore a system of 1024 peers to a ring topology, after a varying number of peer arrivals and departures. Starting from a converged ring network, the arrivals and departures were modeled as occuring instantaneously and simultaneously, followed by normal execution of the RN protocol. The results (not shown due to space constraints) indicate that convergence time is logarithmic in

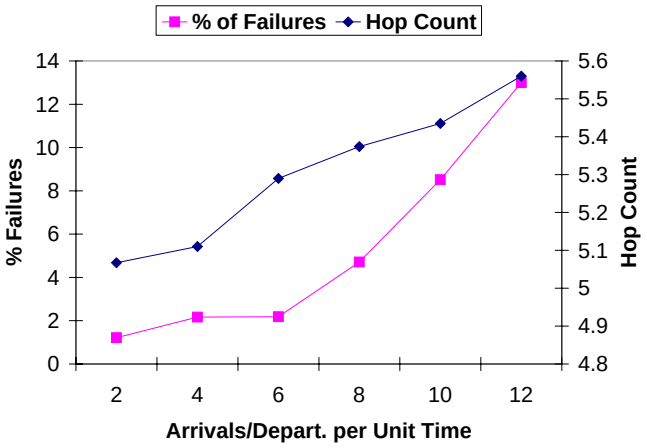


Figure 5. Failure lookup rate and hop count for searches for random peers in a dynamic P2P network.

the number of peers arriving / departing, varying from 25 time units to recover from one arrival/departure, to 80 time units to recover from 1000 arrivals/departures. This is in constrast to other P2P protocols that either cannot recover from such a high number of peer changes, or which must reconstruct the entire topology from scratch, regardless of the number of changes that occur. To emphasize, no modification of the RN protocol is required in the case of dynamic arrivals and departures, which justifies our assertion that the RN protocol is modeless.

6 Conclusion

This paper investigated the problem of organizing a set of peers into a structured P2P network, in a self-stabilizing way, starting from an arbitrary state. The RN protocol was introduced for this purpose to construct a ring network. The RN protocol is simple and self-contained. There is only a single procedure which all peers run, regardless of the state of the network, and regardless of changes to the set of peers. In particular, the peers do not have to estimate the size of the network, or exchange other information globally, unlike most other P2P protocols. The protocol is asynchronous, and represents a practical means for creating structured P2P networks in a robust, highly adaptable way.

In future work, we would like to establish non-trivial bounds for the convergence rate of the protocol.

References

[1] D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *17th ACM Symposium on Parallelism in Algorithms and Architectures(SPAA 2005)*, Las Vegas, NV, USA, 2005.

[2] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant routing in peer-to-peer systems. In *ACM PODC*, 2002.

[3] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM*, 2003.

[4] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley and Sons, 1998.

[5] M. Handley, P. Francis, R. Karp, S. Shenker, and S. Ratnasamy. A scalable content-addressable network, 18 2001.

[6] W. D. Hillis and J. Guy L. Steele. Data parallel algorithms. *Commun. ACM*, 30(1):78–78, 1987.

[7] M. Jelasity and O. Babaoglu. T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy, May 2004.

[8] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, 2003.

[9] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[10] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (IPTPS05)*, 2005.

[11] S. Kutten and D. Peleg. Asynchronous resource discovery in peer to peer networks. In *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, page 224, Washington, DC, USA, 2002. IEEE Computer Society.

[12] S. Kutten, D. Peleg, and U. Vishkin. Deterministic resource discovery in distributed networks. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 77–83, New York, NY, USA, 2001. ACM Press.

[13] X. Li, J. Misra, and G. Plaxton. Active and concurrent topology maintenance. In R. Guerraoui, editor, *Distributed Algorithms*, volume 3274/2004 of *Lecture Nodes in Computer Science*, pages 320–334, Oct 2004.

[14] D. Liben-Nowell, H. Balakrishnan, and D. R. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the twenty-first Annual Symposium on Principles of Distributed Computing (PODC-02)*, pages 233–242, New York, 21–24 2002. ACM Press.

[15] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS*, 2003.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 329-350 2001.

[17] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11, 2003.

[18] U. Wilensky. NetLogo: Center for connected learning and computer-based modeling, Northwestern University., 1999.

[19] B. Y. Zhao, J. Kubiatowicz, and J. Joseph. Tapestry: an infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, 2001.