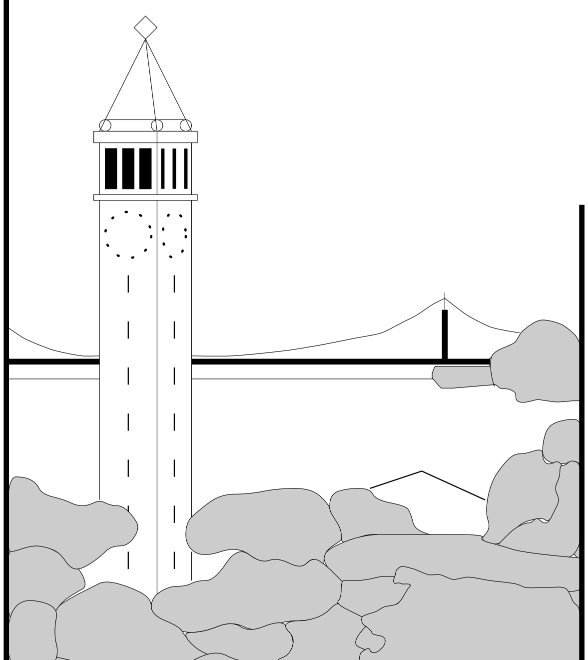
MultiChord: A Resilient Namespace Management Protocol

Nancy Lynch
MIT

Ion Stoica
UC Berkeley



Report No. UCB/CSD-04-1306

Computer Science Division (EECS)
University of California
Berkeley, California 94720

MultiChord: A Resilient Namespace Management Protocol

Nancy Lynch — MIT
Ion Stoica — UC Berkeley

Abstract

MultiChord is a new variant of the Chord namespace management algorithm [7] that includes lightweight mechanisms for accommodating a limited rate of change, specifically, process joins and failures. This paper describes the algorithm formally and evaluates its performance, using both simulation and analysis. Our main result is that lookups are provably correct—that is, each lookup returns results that are consistent with a hypothetical ideal system that differs from the actual system only in entries corresponding to recent joins and failures—in the presence of a limited rate of change. In particular, if the number of joins and failures that occur during a given time interval in a given region of system are bounded, then all lookups are correct. A second result is a guaranteed upper bound for the latency of a lookup operation in the absence of any other lookups in the system. Finally, we establish a relationship between the deterministic assumptions of bounded joins and failures and the probabilistic assumptions (which are often used to model large scale networks). In particular, we derive a lower bound for the mean time between two violations of the deterministic assumptions in a steady state system where joins and failures are modeled by Poisson processes.

1 Introduction

This paper describes *MultiChord*, a new, more resilient variant of the Chord namespace management algorithm [7]. The main innovation is that MultiChord includes lightweight mechanisms for accommodating a limited rate of change, specifically, process joins and failures.

The contributions of this paper include (a) techniques for improving the performance and resiliency of peer-to-peer namespace management algorithms, and (b) methods of analyzing performance for such algorithms in the presence of a bounded rate of change.

Building in resiliency: We improve the performance and resiliency of Chord by adding additional entries to processes' routing (finger) tables, and by delaying a process from joining until its finger table is properly populated. This demonstrates an approach to building peer-to-peer namespace management services in which resiliency to a bounded rate of change is built in from the beginning. The method we use is to design the ideal communication infrastructure with enough redundancy to accommodate a bounded rate of change without reducing latency, and to maintain this redundant structure using gossiping. Newly-joining processes should not participate fully in the system until they have been fully incorporated into the communication infrastructure. This general approach should extend to other communication infrastructures such as those proposed in [1, 4, 5, 6, 7].

Formal modeling and analysis: We present the algorithm precisely, using high-level, nondeterministic timed I/O automata pseudocode. We analyze its performance conditionally, assuming a limited rate of change. This demonstrates how peer-to-peer namespace management algorithms can be modeled using state machines and subjected to proofs and analysis. In particular, it demonstrates that interesting performance results can be obtained for such algorithms using conditional analysis, conditioned on the "normal case" assumption that changes happen at a bounded rate. This kind of analysis should be useful in comparing different namespace management algorithms.

Our method of analysis is quite different from the probabilistic style used by Liben-Nowell *et al* [2]. Our claims are not probabilistic, but rather, worst-case bounds under restricted circumstances. Our assumptions about the rate of change are rather strong. However, as we discuss in Section 3, we can relax these assumptions by adding probabilistic assumptions, while still obtaining our stronger latency bounds.

1.1 Overview

The original Chord protocol [7] assumes a circular identifier space (called the Chord ring) of size $N = 2^n$. With each process i is associated a unique logical identifier in this space. Each process i maintains a routing table (known as a finger table). The k-th entry in this table, called the *k-th finger* of process i, contains a reference to the first process whose logical identifier follows process i's logical identifier by at least 2^k in the clockwise direction on the Chord ring, where $0 \leq k < n$. In the remainder of this paper we refer to these fingers as the power-of-two fingers of i. The *successor* of a logical identifier id represents the first process whose logical identifier follows id in the clockwise direction on the Chord ring, or the process with logical identifier id if such a process exists. We redefine the notion of successor in the context of MultiChord in Section 1.2.

In MultiChord, process i maintains, in addition to the finger table like that used in Chord, information about its "b-block" (i.e., its own b successors and b predecessors) and all b-blocks of its power-of-two fingers. The value of b is chosen based on an assumed upper bound on the "normal" rate of change. When the algorithm is in an ideal state, each process' finger table contains its b-block, as well as a b-block for each of its power-of-two fingers. However, this information can degrade from an ideal state as a result of process joins and failures.

MultiChord includes lightweight mechanisms, based on periodic background gossiping, for maintaining the system in a nearly ideal state in the face of limited change, i.e., limited joins and failures. Each process i continually sends its own b-block to its b successors and b predecessors, which allows them to update their finger tables. In addition, process i continually "pings" its power-of-two fingers, who respond by returning their own b-blocks. These periodic exchanges of information between a process and the processes in its finger table allow the system to gravitate back toward an ideal state in the face of changes. Like Chord, MultiChord does not differentiate between a process failure and departure. When a process i fails or leaves, processes who maintain process i in their finger tables will remove it when it expires.

When a new process i joins the system, it first populates its finger table with its b-block, and the b-blocks of its power-of-two fingers. Like in Chord, a process i uses the lookup operation to find its power-of-two fingers. There are two other instances when a process i invokes a lookup: (i) when a client at location i explicitly invokes a lookup operation for a specified target, and (iii) when it decides to refresh its finger table.

Like Chord, MultiChord implements the lookup operation in an iterative fashion. Consider a process i that performs a lookup on value x. At every iteration (stage), process i sends a query to the best known predecessor for x. Let process k be this predecessor. Upon receiving the query, process k checks whether it knows the process responsible for x–that is, whether its immediate successor is responsible for x—, and if yes, it sends the answer back to process i. Otherwise process k sends its best known predecessor for x to i. MultiChord generalizes this procedure: at every stage process i sends $c > 1$ queries to the best known c predecessors for x. In turn, process k responds with its best known c predecessors of x. As we will show this redundancy increases the resilience of the lookup in the face of changes.

The value of c is chosen to be larger than the number of changes that "normally" occur in a "small" interval of time, in a limited region of the ring. The length of this small interval of time is assumed to be sufficient for the system to recover from a limited number of changes in the relevant region of the ring. The admissible rate of change is quantified in Section 3.

1.2 Notations

Notation	Comments
PId	the set of *physical process identifiers* (e.g., IP address and port number)
XId	the set of *logical identifiers*; $0 \leq i_k < N$, for any $i_k \in XId$
GId	the set of *general identifiers* of the form $g_k = (i_k, x_k)$, where $i_k \in PId, x_k \in XId$
$XtoP,\ PtoX$	one-to-one correspondence from XId to PId, and its inverse.
$succ(x, k, R)$	the k^{th} successor of x in ring R
$pred(x, k, R)$	the k^{th} predecessor of x in R
$succset(x, k, R)$	successor set of x; $succset(x, k, R) = \{succ(x, \ell, R) : 0 \leq \ell \leq k\}$
$psuccset(x, k, R)$	proper successor set of x; $psuccset(x, k, R) = \{succ(x, \ell, R) : 1 \leq \ell \leq k\}$
$predset(x, k, R)$	predecessor set of x; $predset(x, k, R) = \{pred(x, \ell, R) : 0 \leq \ell \leq k\}$
$ppredset(x, k, R)$	proper predecessor set of x; $predset(x, k, R) = \{pred(x, \ell, R) : 1 \leq \ell \leq k\}$
$block(x, k, R)$	block of x; $block(x, k, R) = succset(x, k, R) \cup predset(x, k, R)$

Table 1: Notations used in this paper.

Table 1 shows the main notations used in this paper. Each process is identified by a physical identifier (e.g., IP address and port number), and a logical identifier in identifier space $0..2^n - 1$, where $N = 2^n$. A *ring* R is a nonempty subset of logical identifiers (XId), ordered in a clockwise direction.

The k^{th} successor of x in R is denoted by $succ(x, k, R)$. For $k = 0$, $succ(x, 0, R) = x$ if $x \in R$, and is otherwise undefined. If $k \geq 1$ then $succ(x, k, R)$ is the k^{th} value encountered when moving clockwise in $R - x$ starting from the position of x, if $|R - x| \geq k$, and is otherwise undefined. The k^{th} predecessor of x is defined similarly (see Table 1).

2 The MultiChord Protocol

In this section we present the details of the MultiChord protocol.

2.1 Process Automaton: Signature

For the rest of this section, we fix a physical address $i \in PId$, and describe the process automaton for location i, $MultiChord_i$. Throughout this section, we use me as an abbreviation for the general identifier $GId\ g$ with $g.phys = i$ and $g.log = PtoX(i)$, where $g.phys$ and $g.log$ denote the physical identifier, and the logical identifier of g, respectively. Formally, $MultiChord_i$ is a timed I/O automaton, as defined in Chapter 23 of [3].

The signature of $MultiChord_i$ is given in Figure 1. The external signature describes the inputs and outputs (primarily, client invocations and responses) by which the MultiChord service interacts with its environment. The external signature includes join, lookup and receive inputs and corresponding acknowledgments. We do not include special "leave" requests and responses in this paper; instead, we treat leaves as failures. We do not consider rejoining after a failure. The internal signature consists of transitions that implement join and lookup protocols, and maintain the finger tables in the face of a limited rate of change.

Input:
$\mathsf{join}(J)_i$, J a finite subset of $PId - \{i\}$
$\mathsf{lookup}(x)_i$, $x \in XId$
$\mathsf{receive}(m)_{j,i}$, $m \in Msg$, $j \in PId$

Output:
$\mathsf{join\text{-}ack}_i$
$\mathsf{lookup\text{-}ack}(H)_i$, $H \subseteq GId$
$\mathsf{send}(m)_{i,j}$, $m \in Msg$, $j \in PId$

Internal:
$\mathsf{join\text{-}ping}_i$
$\mathsf{neighbor\text{-}refresh}_i$
$\mathsf{chord\text{-}ping}_i$
$\mathsf{stabilize}(x)_i$, $x \in XId$
$\mathsf{garbage\text{-}collect}(f)_i$, $f \in Finger$

Time-passage:
$\mathsf{time\text{-}passage}(t)$, $t \in R^+$

Figure 1: $MultiChord_i$: Signature

2.2 Process Automaton: Data Types and Constants

Table 2 shows the data structures and the message formats used by the MultiChord protocol. In addition we define two operations on sets of fingers:

1. $update(F, F')$, which computes $F \cup F'$; if a finger f belongs to both sets of fingers F and F' then f inherits the highest expiration time, $exptime$, that it has in the two sets.
2. $truncate(F, t)$, which bounds the $exptime$ of each finger $f \in F$ to t, i.e., $f.exptime := \max(f.exptime, t)$.

Notation	**Comments**
$Finger$	finger data structure; consists of fields: $(gid \in GId,\ exptime \in R^{\geq 0} \cup \{\infty\})$
$ReqId$	*request identifier* set, partitioned into subsets $ReqId(i)$, $i \in PId$; used to identify lookup instances
$Request$	used to implement one lookup stage; consists of fields: $(id \in ReqId, stage \in \mathbb{N}^+, target \in XId)$
$JoinRecord$	used to keep track of progress in a process' attempt to join the system; consists of fields: $(reqids \subseteq ReqId, comp \subseteq ReqId, acktime \in R^{\geq 0} \cup \{\infty\})$
$ClientRecord$	used to keep track of client-initiated lookup requests at a particular location; consists of fields: $(reqids \subseteq ReqId, comp \subseteq ReqId, acked \subseteq ReqId(i))$
$LookupMsg$	lookup message; consists of fields: $(tag = \mathsf{lookup}, id \in ReqId, stage \in \mathbb{N}^+, target \in XId)$
$LookupRespMsg$	lookup response message, $(tag = \mathsf{lookup\text{-}resp}, id \in ReqId, stage \in \mathbb{N}^+, preds \in Set[Finger])$
$LookupCompMsg$	lookup completion msg., $(tag = \mathsf{lookup\text{-}completion}, id \in ReqId, stage \in \mathbb{N}^+, block \in Set[Finger])$
$PingMsg$	ping message used to refresh finger information, $(tag = \mathsf{ping}, \mathsf{ping})$
$BlockMsg$	message used to send a block to another message, $(tag = \mathsf{block}, block \in Set[Finger])$
T_e	the timeout value for expiration of entries in the finger table
T_g	the time between scheduling gossiping messages, i.e., $PingMsg$ and $BlockMsg$ messages
T_j	the time from when a joining process has received all its responses until it responds to its client
b	number of proper predecessors and successors that a process maintains about itself and its power-of-two fingers
c	number of responses that a client returns in response to a lookup request; $c < b$

Table 2: Data structures and message formats used in MultiChord.

MultiChord uses only five types of messages: $Lookup$, $LookupRespMsg$ and $LookupRespCompletion$ to implement join and lookup operations, and $PingMsg$ and $BlockMsg$ to maintain the finger tables in the face of changes.

In addition, MultiChord uses the following time constants: (1) T_g, the time between scheduled gossiping messages, (2) T_e, the timeout value for expiration of entries in the finger table, and (3) T_j, the time from when a joining process has completed its systematic collection of responses until it responds to its client.

Finally, MultiChord uses two constants b and c. Constant b represents the level of redundancy used by a process to maintain routing information. In particular, each process maintains its b proper successors and b proper predecessors, and b proper successors and b proper predecessors of each of its power-of-two fingers. Constant c represents the level of redundancy used to perform lookups, the basic operations in MultiChord. During lookup operations, each process issues c concurrent queries, which makes it highly likely that at least one process will respond. The value of c is chosen to be larger than the number of changes that are likely to occur in an arc of the ring, in intervals of some reasonable length. The length of this interval should be sufficiently long to allow recovery from recent changes. The value of b is usually larger than c; c must be large enough to ensure a response under "normal" conditions (with bounded changes), while b must be large enough to support the infrastructure maintenance protocol.

2.3 Process Automaton: State

The state of $MultiChord_i$ consists of the state variables listed in Figure 2. Note that our initializations of these variables assign tuples to record-valued variables. We use the convention that the order of the components in the tuples is the same as the order presented in the definitions of the record types.

State variables:
$status \in \{\mathsf{idle}, \mathsf{joining}, \mathsf{active}\}$, initially idle
$join \in JoinRecord$, initially $(\emptyset, \emptyset, \infty)$
$client \in ClientRecord$, initially $(\emptyset, \emptyset, \emptyset)$
$used\text{-}reqids \subseteq ReqId(i)$, initially $\emptyset$
$requests \in Set[Request]$, initially $\emptyset$
$fingers \in Set[Finger]$, initially $\{(me, \infty)\}$
$out\text{-}queue$, a sequence of $Msg \times PId$, initially empty
$ping\text{-}time \in R^{\geq 0} \cup \infty$, initially ∞
$nbr\text{-}refresh\text{-}time \in R^{\geq 0} \cup \infty$, initially ∞
$failed$, a Boolean, initially false

Derived variables:
$local\text{-}ring = \{x \in XId : \exists f \in fingers[f.log = x]\}$
For $x \in XId$, $k \geq 0$:
$f\text{-}succset(x, k) = \{f \in fingers : f.log \in succset(x, k, local\text{-}ring)\}$
$f\text{-}psuccset(x, k) = \{f \in fingers : f.log \in psuccset(x, k, local\text{-}ring)\}$
$f\text{-}predset(x, k) = \{f \in fingers : f.log \in predset(x, k, local\text{-}ring)\}$
$f\text{-}ppredset(x, k) = \{f \in fingers : f.log \in ppredset(x, k, local\text{-}ring)\}$
$f\text{-}block(x, k) = \{f \in fingers : f.log \in block(x, k, local\text{-}ring)\}$

Figure 2: $MultiChord_i$: State

The $status$ variable keeps track of the state of process i. The $join$ variable keeps track of the progress of the joining protocol for process i, and the $client$ variable keeps track of the progress of all client-initiated lookups at location i. The *fingers* variable contains a set of fingers, which represent process i's best knowledge of the current members of the ring (including their expiration times). The $used\text{-}reqids$ variable keeps track of which request identifiers in $ReqId(i)$ have already been used; it is used to model the generation of unique identifiers. The $requests$ variable keeps track of the set of requests that have been initiated at location i; these may be generated on behalf of the local joining protocol, local client lookup requests, or heavyweight stabilization. The $out\text{-}queue$ variable is a buffer for messages that process i has generated and has not yet sent.

The $nbr\text{-}refresh\text{-}time$ and $ping\text{-}time$ variables are used to schedule the gossip messages; $nbr\text{-}refresh\text{-}time$ is used by process i to schedule sending of its own block to its nearby neighbors, whereas $ping\text{-}time$ is used by process i to schedule "ping" messages to request block information from other processes. Finally, the $failed$ variable is a flag saying whether process i has failed.

Process i also maintains some derived variables, which also appear in Figure 2. The derived variable *local-ring* is defined to be the set of logical identifiers that appear in i's *fingers* variable, that is, *local-ring* represents i's current local view of the global ring. Other derived variables are defined to give various successor and predecessor sets, with respect to the *local-ring*. For example, $f\text{-}succset(x, k)$ is defined to be the set of fingers in the current *finger* set whose logical identifiers are among the k successors of x in the current *local-ring*; if x appears in *local-ring* then this set include x itself.

2.4 Process Automaton: Transitions

In this section we present the main transitions in MultiChord. Section 2.4.1 describes the basic transitions such as message sending, garbage-collection, and time-passage transitions. Section 2.4.2 shows the transitions involved in the joining protocol, and Section 2.4.3 presents the transitions involved in the stabilization protocol. Finally, Section 2.4.4 describes transitions involved in the client lookup protocol.

2.4.1 Basic Transitions

Figures 3(a)-(c) shows three basic transitions: sending, garbage-collection, and time-passage transitions.

A send transition simply removes the first *Msg* from *out-queue* and sends it to the indicated destination, using an assumed point-to-point network. Process i can do this only if it has at least begun the protocol, and has not failed. A garbage-collect transition removes an entry from its *fingers* set when the entry's $exptime$ has been reached. A time-passage transitions advances the time until the next event, i.e., scheduling times of pinging, acknowledging the client, or neighbor-refreshing, and the $exptime$ of any finger in the *fingers* set. Time may not pass at all if the $out\text{-}queue$ is nonempty; this implies that messages in the $out\text{-}queue$ are sent out immediately, without any time passage.

Output $\mathsf{send}(m)_{(i,j)}$
Precondition:
  $\neg failed$
  $status \neq \mathsf{idle}$
  $(m, j) = head(out\text{-}queue)$
Effect:
  remove $head(out\text{-}queue)$

(a)

Internal $\mathsf{garbage\text{-}collect}(f)_i$
Precondition:
  $\neg failed$
  $status = \mathsf{active}$
  $f \in fingers$
  $f.exptime \leq now$
Effect:
  $fingers := fingers - \{f\}$

(b)

$\mathsf{time\text{-}passage}(t)$
Precondition:
  if $\neg failed$ then
    $now + t \leq ping\text{-}time$
    $now + t \leq join.ack\text{-}time$
    $now + t \leq nbr\text{-}refresh\text{-}time$
    $\forall f \in fingers : now + t \leq f.exptime$
    $out\text{-}queue$ is empty
Effect:
  $now := now + t$

(c)

Figure 3: (a) Sending transitions; (b) Garbage-collection transition; (c) Time passage transitions.

2.4.2 Transitions Involved in the Joining Protocol

Like Chord, in MultiChord a process uses lookups to populate its finger table when it joins the system. Where the two protocols differ is in the amount of state required to join the system. Whereas in Chord a process is required to know only a set of successor processes, in MultiChord a process is required to know a set of processes (i.e., a b-block) for *each* of its power-of-two fingers. As we will show in Section 3 this redundancy increases the resilience of the protocol in the face of changes.

Next, we present the details of the transitions involved when process i joins the system. These include:

1. The join_i transitions, by which the client at location i requests to join.
2. The receive transitions for lookup, lookup-resp, and lookup-comp messages, which are involved in initially populating process i's *finger* set.
3. The join-ping transitions and the receive transitions for ping messages; these are used to complete the *finger* set before process i responds to the client.
4. The $\mathsf{join\text{-}ack}_i$ transitions, by which process i responds to its client.

Figure 4 shows the join and join-ack transitions. In a $\mathsf{join}(J)_i$ transition, processor i initiates joining by submitting a set J of PIds of other processes that should already be members of the system. Process i handles the join request only if it has not failed and has not previously begun joining. To handle the join request, the process first sets its $status$ to joining and schedules its $ping$ task. If $J = \emptyset$, the process is already done and schedules its response to the client. Otherwise, if $J \neq \emptyset$, process i launches a set of lookup requests, one for itself and one for each of its power-of-two successors.

When all these requests have completed, and when sufficient additional time has passed (as determined by a scheduled $ack\text{-}time$ being reached), process i can report back to the client with a $\mathsf{join\text{-}ack}_i$ transition. When it does so, it converts its $status$ to active and schedules its $nbr\text{-}refresh$ task.

As in Chord, MultiChord implements an iterative lookup protocol. The processing of a lookup request involves three types of transitions, which appear in Figure 5. When process i receives a lookup message, it handles this message only if it is already active, that is, if it has completed its joining protocol. In order to handle the lookup message, it sends either a lookup-resp or a lookup-comp message, depending on whether it thinks that the search has reached its goal. The test for completion is that, according to i's current information, target x is among the c proper predecessors of the target. In the case of a lookup-comp message, process i sends back its best information about the target's block of radius b. In the case of a lookup-resp message, process i sends back its c best proper predecessors for the target.[1]

When process i receives a lookup-resp message for the current stage of a current request, it updates its *finger* table with the information contained in the $preds$ field of the incoming message. Then because the request is not completed, process i generates a new batch of lookup messages for the next stage of the same request. This next stage

[1] In either case, process i first truncates all fingers' $exptime$s to now plus the maximum timeout value T_e; this is because i's entry for itself has $exptime = \infty$, but we do not want others to record $exptime = \infty$ for i.

Input $\mathsf{join}(J)_i$
Effect:
  if $\neg failed$ then
    if $status = \mathsf{idle}$ then
      $status := \mathsf{joining}$
      $ping\text{-}time := now$
      if $J = \emptyset$ then $join.acktime := now$
      else
        for $x \in \{me.log\} \cup \{me.log + 2^k : 0 \le k \le n-1\}$ do
          choose $rid \in ReqId(i) - used\text{-}reqids$
          $used\text{-}reqids := used\text{-}reqids - \{rid\}$
          $join.reqids := join.reqids \cup \{rid\}$
          $requests := requests \cup \{(rid, 1, x)\}$
          for $j \in J$ do
            add $((\mathsf{lookup}, rid, 1, x), j)$ to $out\text{-}queue$

Output $\mathsf{join\text{-}ack}_i$
Precondition:
  $\neg failed$
  $status = \mathsf{joining}$
  $join.reqids \subseteq join.comp$
  $join.acktime = now$
Effect:
  $status := \mathsf{active}$
  $nbr\text{-}refresh\text{-}time := now$

Figure 4: Client-level transitions related to joining

Input $\mathsf{receive}(\mathsf{lookup}, r, s, x)_{j,i}$
Effect:
  if $\neg failed$ then
    if $status = active$ then
      if $me.log \in ppredset(x, c, local\text{-}ring)$ then
        $block := truncate(f\text{-}block(me.log, b), now + T_e)$
        add $((\mathsf{lookup\text{-}comp}, r, s, block), j)$ to $out\text{-}queue$
      else
        $preds := truncate(f\text{-}ppredset(x, c), now + T_e)$
        add $((\mathsf{lookup\text{-}resp}, r, s, preds), j)$ to $out\text{-}queue$

Input $\mathsf{receive}(\mathsf{lookup\text{-}resp}, r, s, F)_{j,i}$
Effect:
  if $\neg failed$ then
    $new\text{-}fingers := \{f \in F : f.exptime \ge now\})$
    $fingers := update(fingers, new\text{-}fingers)$
    if $\exists x[(r, s, x) \in requests)]$ then
      choose x where $(r, s, x) \in requests$
      $requests := requests - \{(r, s, x)\} \cup \{(r, s+1, x)\}$
      for $f \in f\text{-}ppredset(x, c)$ do
        add $((\mathsf{lookup}, r, s+1, x), f.gid.phys)$ to $out\text{-}queue$

Input $\mathsf{receive}(\mathsf{lookup\text{-}comp}, r, F)_{j,i}$
Effect:
  if $\neg failed$ then
    $new\text{-}fingers := \{f \in F : f.exptime \ge now\})$
    $fingers := update(fingers, new\text{-}fingers)$
    if $r \in join.reqids$ then
      $join.comp := join.comp \cup \{r\}$
      if $join.reqids \subseteq join.comp$ and $join.acktime = \infty$ then
        $join.acktime := now + T_j$
    if $r \in client.reqids$ then
      $client.comp := client.comp \cup \{r\}$

Figure 5: Transitions of the lookup protocol

has the next-higher stage number, which is recorded in the request record. The messages for the new stage are sent to the c currently-known best proper predecessors of the target. Note that the number of messages does not increase exponentially at each stage; the protocol limits the the number of messages to c.

When process i receives a lookup-comp message for the current stage of a current request, it updates its *finger* table with the information in the *block* field of the incoming message. As in the lookup-resp case, process i increments the request's stage number, to register the fact that some response for this stage has arrived. If the current request is part of i's joining protocol, then the completion of this request is recorded in the *join* record; if this represents the completion of the last request, then process i also schedules the client acknowledgment. On the other hand, if the request is being done on behalf of a client-initiated lookup, the completion is recorded in the *client* record (see Appendix A).

During the joining protocol, process i periodically pings its power-of-two *fingers* for their b-blocks. The relevant transitions are the join-ping transitions and the receive transitions for ping messages and their responses (see Figure 6).

Process i performs a join-ping transition while it is joining, whenever $ping\text{-}time$ is reached. When it does so, it sends ping messages to the c-blocks of all targets for which lookup requests have already completed. This allows process i to augment and refresh its information about completed requests while finishing the joining protocol. When process i receives a ping message, it responds by sending back its b-block, in a block message. When process i receives a block message, it updates its *finger* table with the new information.

Internal $\mathsf{join\text{-}ping}_i$
Precondition:
  $\neg failed$
  $status = \mathsf{joining}$
  $ping\text{-}time = now$
Effect:
  for $r \in requests$ where $r.id \in join.comp$ do
    for $f \in f\text{-}block(r.target, c)$ do
      add $((\mathsf{ping}), f.gid.phys)$ to $out\text{-}queue$
  $ping\text{-}time := now + T_g$

Input $\mathsf{receive}(\mathsf{ping})_{j,i}$
Effect:
  if $\neg failed$ then
    if $status = \mathsf{active}$ then
      $block := truncate(f\text{-}block(me.log, b), now + T_e)$
      add $((\mathsf{block}, block), j)$ to $out\text{-}queue$

Input $\mathsf{receive}(\mathsf{block}, F)_{j,i}$
Effect:
  if $\neg failed$ then
    if $status = \mathsf{active}$ then
      $new\text{-}fingers := \{f \in F : f.exptime \geq now\}$
      $fingers := update(fingers, new\text{-}fingers)$

Figure 6: Transitions related to pinging during the join protocol

2.4.3 Transitions Involved in Stabilization

Once process i is active, it performs several types of transitions to maintain its finger table. The protocol includes two kinds of stabilization: normal case, lightweight stabilization, and a heavier-weight stabilization.

In the lightweight stabilization protocol, process i periodically sends its b-block to its nearby neighbors (the members of its b-block), and periodically pings processes in the vicinity of its power-of-two successors, so that they send i their current b-blocks. The transitions involved in this lightweight stabilization protocol are the neighbor-refresh transitions, the chord-ping transitions, and the receive transitions for ping and block messages. Note that the pseudocode for ping and block transitions has already been presented in Figure 6, while the pseudocode neighbor-refresh and chord-ping transitions appears in Figure 7(a)-(b).

Internal $\mathsf{neighbor\text{-}refresh}_i$
Precondition:
  $\neg failed$
  $status = \mathsf{active}$
  $nbr\text{-}refresh\text{-}time = now$
Effect:
  for $f \in f\text{-}block(me.log, b)$ do
    add $((\mathsf{block}, f\text{-}block(me.log, b)),$
      $f.gid.phys)$ to $out\text{-}queue$
  $nbr\text{-}refresh\text{-}time := now + T_g$

(a)

Internal $\mathsf{chord\text{-}ping}_i$
Precondition:
  $\neg failed$
  $status = \mathsf{active}$
  $ping\text{-}time = now$
Effect:
  for $k \in \{0, \ldots, n-1\}$ do
    for $f \in f\text{-}block(me.log + 2^k, c)$ do
      add $((\mathsf{ping}), f.gid.phys)$ to $out\text{-}queue$
  $ping\text{-}time := now + T_g$

(b)

Internal $\mathsf{stabilize}(x)_i$
Precondition:
  $\neg failed$
  $status = \mathsf{active}$
Effect:
  choose $rid \in ReqId(i) - used\text{-}reqids$
  $used\text{-}reqids := used\text{-}reqids - \{rid\}$
  $requests := requests \cup \{(rid, 1, x)\}$
  for $f \in f\text{-}ppredset(x, c)$ do
    add $((\mathsf{lookup}, rid, 1, x), f.gid.phys)$
      to $out\text{-}queue$

(c)

Figure 7: Transition related to stabilization.

In the heavyweight stabilization protocol is similar to the Chord stabilization protocol (see Figure 7(c)). Process i (for any reason, unspecified here) may try to obtain new information about any target x. Most commonly, such a target will be one of its power-of-two successors. For example, process i might execute $\mathsf{stabilize}(x)_i$ for each x of the form $PtoX(i) + 2^k$, at regular intervals, or when it suspects that its information is out-of-date.

2.4.4 Transitions Related to Client Lookups

The transitions related to client-initiated lookup operations include the receive transitions already described, plus the lookup and lookup-ack transitions. These last two appear in Figure 8.

When process i receives a client-initiated lookup request, it handles it in much the same way it handles a request in the joining protocol. Namely, it chooses and records a request identifier, and sends a lookup message to each of the c best proper predecessors it knows for the target identifier. An exception: If process i believes it is one of the c best predecessors, it does not bother sending out any lookup messages, but simply records the fact that the lookup is done. A lookup-ack can occur when a request is done but not yet acknowledged to the client. In this case, the response includes information about process i's current c best predecessors for the target.

Input $\mathsf{lookup}(x)_i$
Effect:
  if $\neg failed$ then
    if $status =$ active then
      choose $rid \in ReqId(i) - used\text{-}reqids$
      $used\text{-}reqids := used\text{-}reqids - \{rid\}$
      if $me.log \in ppredset(x, c, local\text{-}ring)$ then
        $client.comp := client.comp \cup \{rid\}$
      else
        $client.reqids := client.reqids \cup \{rid\}$
        $requests := requests \cup \{(rid, 1, x)\}$
        for $f \in f\text{-}ppredset(x, c)$ do
          add $((\mathsf{lookup}, rid, 1, x), f.gid.phys)$ to $out\text{-}queue$

Output $\mathsf{lookup\text{-}ack}(H)_i$
Precondition:
  $\neg failed$
  $r \in requests$
  $r.id \in client.comp - client.acked$
  $H = \{f.gid : f \in f\text{-}ppredset(r.target, c)\}$
Effect:
  $client.acked := client.acked \cup \{r.id\}$

Figure 8: Transitions for client lookup

3 Summary of Analysis Results

In this section we give a short and informal summary of our analysis results. Appendix A presents the proofs of these results.

We make the following assumptions about the environment: (1) all processes are time-synchronized, (2) the message delay is bounded above by d, and there is no message loss, (3) during an interval of time $T_g + 2d$, the number of join-ack events among processes in an "arc" of the ring containing at most $b + 1$ processes is at most $joinbd$, and (4) during an interval of time T_e, the number of failed processes in an "arc" of the ring containing at most $b + 1$ processes is at most $failbd$.
Then we show that if these assumptions hold, and furthermore, if the following constraints are satisfied:
1. $T_j > T_g + 2d$ and $T_e > 5(T_g + 2d)$
2. $c > 7joinbd + 4failbd$
3. $b \geq 2c + 3joinbd + \max(2joinbd, failbd)$

we prove that all lookup operations are correct. In particular we prove the following result:

Theorem 3.1. *Every good execution α satisfies $2T_g + 6d$-lookup-correctness.*

The notion of *e-Lookup-correctness* is defined as follows: suppose that a $\mathsf{lookup\text{-}ack}(H)_i$ event occurs in β at time t, in response to a prior $\mathsf{lookup}(x)_i$. Let β' be the portion of β ending with the given $\mathsf{lookup\text{-}ack}(H)_i$ event. Then there exists a ring R such that:

1. $R \subseteq aug\text{-}ring(\beta')$,
2. $global\text{-}ring(\beta') - \{PToX(j) : \mathsf{join\text{-}ack}_j \text{ occurs at a time } \geq t - e\} \subseteq R$, and
3. $H = ppredset(x, c, R)$.

Furthermore, we show that in the absence of any other events in the system the lookup latency is bounded. More formally, we prove the following result:

Theorem 3.2. *Suppose that α is a good execution, α' a finite prefix of α containing at least $2c + 1$ join-ack events. Suppose that:*

1. *The final step of α' is a lookup_i step in which i initiates request r, with target x.*
2. *No other requests (on behalf of joins, client lookups, or stabilizes) are active at any time $\geq \ell time(\alpha') - T_e$.*

Then request r terminates with a receive(lookup-comp) *step, at a time that is $\leq \ell time(\alpha') + 4(\log N + 1)d$.*

In order to prove these results, in Appendix A we first prove a series of results asserting that the basic routing infrastructure is maintained correctly by the joining and refresh protocols.

While the deterministic assumptions on the bounded number of joins ($joinbd$) and failures ($failbd$) allow us to prove strong analytical results, these assumptions are not always realistic. We consider this issue in Appendix B,

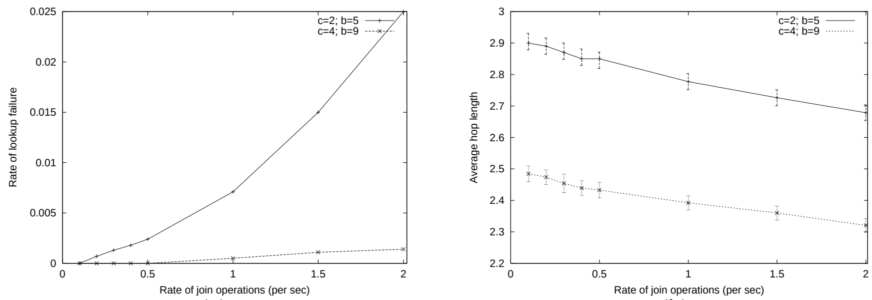


Figure 9: (a) The lookup failure versus the rate of change; (b) the average path length and the 90-th percent confidence interval as a function of change rate.

where we give bounds on the probability that these assumptions hold in a steady state system in which processes join according to a Poisson process and have a lifetime drawn from an exponential distribution. In particular, we compute the mean time between two violations of these assumptions as

$$T_f > \frac{T_j}{N}\ e^{\frac{(c/3-\overline{\lambda}T_j(b+1))^2}{4\overline{\lambda}T_j(b+1)}}, \tag{1}$$

where $\overline{\lambda}$ represents the normalized rate of change (i.e., the rate of change in the entire system divided by the number of processes N in the system), $c/3 > \overline{\lambda}T_j(b+1)$, and $b \geq 13c/6$.

4 Simulation Results

In this section we evaluate our algorithm by simulation. Our goal is twofold. First, we want to get a sense of how much we can push the protocol in practice before it breaks, i.e., before we start to see lookup failures. Second, we want to see how the protocol performs on the average case. We use the average number of stages in a lookup as the metric to evaluate the performance of MultiChord.

We have developed an event driven simulator that accurately implements the protocol at the message level. In all simulations, we use $T_g = 10$ sec, $T_j = 11$ sec, and $T_e = 55$ sec. The message propagation delay is bounded by $d = 50$ ms. Note that these values satisfy the constraints presented in Section 3, i.e., $T_j > T_g + 2d$ and $T_e > 5(T_g + 2d)$. Each process schedules heavy stabilization every 60 sec.

We consider a network with $1,000$ processes, in which processes join at a rate λ_a according to a Poisson process, and have an exponentially distributed lifetime with the mean N/λ_a; thus, the number of processes in the system remains roughly the same. In addition, we assume that the system receives lookups at a rate approximately 10 times larger than the join and failure rate, r.

Figure 9(a) plots the lookup failure rate versus the arrival rate of new processes in the system (i.e., rate of join) over $10,000$ lookups. During each simulation there are approximately 1000 new processes that join the system, and 1000 processes that fail. We consider two cases: (i) $c = 2$, $b = 5$, and (ii) $c = 4$, $b = 9$. As expected, the rate of lookup failure increases as the join rate increases. However, increasing the level of redundancy (i.e., parameters b and c) makes a significant difference. While in case (i) we did not record any lookup failure for join rates less or equal to 0.1, in case (ii) we did not see any lookup failure for a join rate five time larger, i.e., 0.5. Furthermore, for a join rate of 2.0 the rate of lookup failure in the first case is about 18 times larger than in the second case.

It is interesting to compare the simulation results with our upper bound on the mean time T_f between two violations of the deterministic constraints. Consider the first case where $c = 2$ and $b = 5$. Using Eq. (1), for a join rate of 0.5 we obtain $T_f = 14$ ms.[2] This is a very small value given the fact that a lookup operation is generated every 50 ms (i.e., there are roughly 10 lookups for every join operation). One explanation for this large discrepancy is that a single constraint violation will hurt only a small fraction of lookups, if at all. Indeed, the lookups that do not use the region of network where the constraints are violated will not be affected.

[2] Here we use $c = 2$, $b = 5$, $\overline{\lambda} = 1/1000$ (there is one join and one failure every 0.5 sec on average and $N = 1000$), and $T_j = 11$ sec.

Figure 9(b) plots the average number of stages (path length) of a lookup versus the rate of join for (i) $c = 2, b = 5$, and (ii) $c = 4, b = 9$, respectively. There are two points worth noting. First, the average path length is significantly smaller than in Chord; in Chord, the expected path length is $\log N/2$, which in our case translates to 5 hops. This is because in MultiChord every process maintains a much larger set of fingers than in Chord. This increases the chance that a MultiChord process will know fingers closer to the target than an equivalent Chord process, which ultimately will reduce the number of lookup stages. Second, as the join rate increases, the lookup path length decreases slightly. To understand this recall that in steady state the average life time of a node is N/λ_a where λ_a is the join rate. However, it takes a process at least T_g time to join the system. Thus a node will be inactive for at least $f = T_g\lambda_a/N$ of its life time, which means that at least fN processes in the system would be inactive on an average. As the join rate increases, the fraction f of inactive nodes increases, which will lead to a corresponding reduction in the number of active nodes in the system. A secondary reason is that as the join rate increases so does the failure rate. Since we do not report failed lookups, and since the failed lookups tend to have more stages, the reported path length is an underestimation.

5 Conclusions and Future Work

In this paper we present MultiChord, a namespace management algorithm based on Chord [7]. MultiChord uses redundancy and lightweight mechanisms to accommodate limited changes in time and space. We analyze MultiChord and show that lookups are guaranteed to be successful and furthermore that the lookup latency is bounded.

It would be interesting to analyze the behavior of the algorithm in situations that are less well-behaved than what we have described in this paper. In particular, we plan to consider what happens if the rate of change exceeds our assumed bound for some part of the execution, but at some point "stabilizes" to obey the rate bound. In such cases, we believe that our algorithm will eventually stabilize to a nearly-ideal state. It remains to determine if this is so and determine bounds on how long this might take.

References

[1] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of SPAA 2002*, July 2002.

[2] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of PODC 2002*, July 2002.

[3] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publisher, Inc., 1996.

[4] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of PODC 2002*, July 2002.

[5] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS 2002*, July 2002.

[6] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware 2001*, 2001.

[7] 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. ACM SIGCOMM'01*, pages 149–160, San Diego, 2001.

6 Appendix A: Analysis

In this appendix we prove the results which were summarized in Section 3.

Let β by a finite sequence of external actions of $MultiChord$, according to the external signature just defined. Then we define the *global ring after* β, *global-ring*(β), to be the set of XIds x such that a join-ack$_{XtoP(x)}$ event occurs in β and no fail$_{XtoP(x)}$ occurs in β. That is, the global ring after β consists of those processes that have completed joining the system and have not failed. We extend this same definition to finite executions of untimed or timed automata that have the given external signature.

If β is a finite timed sequence of actions in the $MultiChord$ external signature, then we define the *augmented ring* after β, *aug-ring*(β), to be *global-ring*$(\beta) \cup X$, where X is the set of XIds x such that fail$_{XToP(x)}$ occurs in β at a time $\geq \ \ell time(\beta) - T_e$. That is, *aug-ring*$(\alpha)$ augments *global-ring*(β) by adding in the logical identifiers of recently failed processes. Again, we extend this definition to finite executions of timed automata that have the given external signature.

6.1 Service Guarantees

We describe safety and latency guarantees. We do not present any liveness guarantees here, replacing them with latency guarantees.

6.1.1 Safety

The following condition is simple a well-formedness condition, expressing basic conditions such as "the service responds only to invocations that were actually made".

- *Well-formedness:* For each i, at most one join-ack$_i$ occurs in β. Any join-ack$_i$ in β is preceded by a join$(*)_i$. Any lookup-ack$_i$ is preceded by a lookup$(*)_i$ with no intervening lookup-ack$(*)_i$. If fail$_i$ occurs in β, then no following outputs occur.

We have not formulated any interesting safety guarantees related to joining. For client lookup, we require the following property, parameterized by $e \in R^{\geq 0}$:

- *e-Lookup-correctness:* Suppose that a lookup-ack$(H)_i$ event occurs in β at time t, in response to a prior lookup$(x)_i$. Let β' be the portion of β ending with the given lookup-ack$(H)_i$ event. Then there exists a ring R such that:

  1. $R \subseteq$ *aug-ring*(β'),
  2. *global-ring*$(\beta') - \{PToX(j) :$ join-ack$_j$ occurs at a time $\geq t - e\} \subseteq R$, and
  3. $H = ppredset(x, c, R)$.

6.1.2 Latency

As noted above, we replace liveness claims by latency bounds:

- *e-Join-latency:* Suppose that a join$(J)_i$ event occurs in β, at time t.

  1. If $J = \emptyset$ then a corresponding join-ack$_i$ occurs at time t.
  2. If there exists $j \in J$ such that join-ack$_j$ occurs before the join$(J)_i$, and neither i nor j fails in β, then a corresponding join-ack$_i$ occurs by time $t + e$.

- *e-Lookup-latency:* If a lookup$(x)_i$ event occurs in β at time t and no fail$_i$ occurs in β, then a corresponding lookup-ack$(*)_i$ occurs by time $t + e$.

6.2 Assumptions for Analysis

In this section we formalize the algorithm constraints and the assumptions about the environment, which we discussed in Section 3

6.2.1 Restrictions on the algorithm

Constraints on values of the constants b, c, d, T_g, T_e, and T_j:

- $T_j > T_g + 2d$
- $T_e > 5(T_g + 2d)$

Scheduling assumptions:

- The locally controlled actions that are enabled are performed without any intervening time-passage.

6.2.2 Restrictions on the environment

Constants:
For the purpose of analysis, we introduce two constants, $joinbd$ and $failbd$. We assume:

- $c > 7joinbd + 4failbd$
- $b \geq 2c + 5joinbd$
- $b \geq 2c + 3joinbd + failbd$

Restrictions on timing and failures:

- No message loss.
- No time passes while a locally-controlled action is enabled.
- *Bounded local joins:* An execution α satisfies *bounded local joins* provided that for any finite prefix α' of α, the following holds.
  Let $x, y \in XId$ where $|global\text{-}ring(\alpha') \cap [x, y]| \leq b + 1$. Then the number of $\mathsf{join\text{-}ack}_k$ events that occur in α' at times $\geq \ell time(\alpha') - (T_g + 2d)$, where $PToX(k) \in [x, y]$, is $\leq joinbd$.
  That is, at any point in the execution α, the number of recent join-ack events among processes in an "arc" of the ring containing at most $b + 1$ processes is at most $joinbd$.

  This assumption is not ideal because it is expressed in terms of the number of join-ack events, which are under the control of the algorithm (rather than the environment). We could justify this assumption in terms of a more primitive assumption that bounds the rate of join events, which are controlled by the environment. To do this, we might need to modify the algorithm so that it schedules the join-acks so that (in the normal case) they occur a fixed amount of time after the joins. Alternatively, a probabilistic justification might be possible.

- *Bounded local failures:* An execution α satisfies *bounded local failures* provided that for any finite prefix α' of α, the following holds.
  Let $x, y \in XId$ where $|global\text{-}ring(\alpha') \cap [x, y]| \leq b + 1$. Then the number of fail_k events that occur in α' at times $\geq \ell time(\alpha') - T_e$, where $PToX(k) \in [x, y]$, is $\leq failbd$.
  That is, at any point in α, the number of recent fail events among processes in an arc of the ring containing at most $b + 1$ processes is at most $failbd$.

We also need a special assumption to ensure that there are "enough" processes in the ring.

- *Enough-processes* An execution α satisfies *enough-processes* provided that it has a finite prefix α' such that:
  1. At least $2b + 1$ join-ack events occur in α'.
  2. No fail event occurs in α'.
  3. In any state of α after α', the total number of live processes is always $\geq 2b + 1$.

  We call the shortest such prefix α' the *initialization prefix*.

A *good* execution is one that observes all the timing and failure restrictions given in this section.

6.3 Basic Lemmas

The first lemma says that $exptime$s of fingers are always $\geq now$.

Lemma 6.1. *The following is true in any state that is reachable in a good execution:*
If $f \in fingers_i$ then $f.exptime \geq now$.

The next lemma says that every physical identifier i that appears in another process' $fingers$ set, or in a message in transit, must correspond to a process whose $status$ is $active$.

Lemma 6.2. *The following is true in any state that is reachable in a good execution:*
Suppose that $f \in Finger$, $f.phys = i$, and any of the following holds:

1. *$f \in fingers_j$ for some $j \neq i$.*
2. *$f \in m.block$ for some $m \in BlockMsg$ that is in transit.*
3. *$f \in m.preds$ for some $m \in LookupResp$ in transit.*
4. *$f \in m.block$ for some $m \in LookupComp$ in transit.*

Then $status_i = \mathsf{active}$.

The next lemma says that, if a process fails at a time t, then no expiration time for that process that is greater than $t + T_e$ ever appears anywhere in the state.

Lemma 6.3. *Suppose that α is a finite execution, and fail_i occurs at time t in α. Suppose that $f \in Finger$ and $f.phys = i$. Suppose that, in $\ell state(\alpha)$, any of the following holds:*

1. *$f \in fingers_j$ for some $j \neq i$.*
2. *$f \in m.block$ for some $m \in BlockMsg$ that is in transit.*
3. *$f \in m.preds$ for some $m \in LookupResp$ in transit.*
4. *$f \in m.block$ for some $m \in LookupComp$ in transit.*

Then $f.exptime \leq t + T_e$.

As a corollary to some of the previous lemmas, the following lemma says that a process that has failed more than T_e time ago does not appear in anyone's $fingers$ set.

Lemma 6.4. *Suppose that α is a finite execution, and fail_i occurs strictly before time $\ell time(\alpha) - T_e$ in α. Suppose that $f \in Finger$ and $f.phys = i$. Then in $\ell state(\alpha)$, f does not appear in $fingers_j$ for any $j \neq i$.*

Proof. By contradiction. Suppose that in $\ell state(\alpha)$, $f \in fingers_j$ for a particular $j \neq i$. Then by Lemma 6.3, in $\ell state(\alpha)$, $f.exptime \leq t + T_e$, where t is the time at which fail_i occurs. Lemma 6.1 implies that, in $\ell state(\alpha)$, $f.exptime \geq now$, that is, $f.exptime \geq \ell time(\alpha)$. These two inequalities together imply that $\ell time(\alpha) \leq t + T_e$. This contradicts the hypothesis that fail_i occurs strictly before time $\ell time(\alpha) - T_e$. □

6.4 Maintaining Neighbor Sets

In this section, we prove that the neighbor sets are properly maintained. We divide the work into three steps: First, we consider what happens when there are no failures and only a bounded number of joins. Second, we consider the general case, with unlimited failures and joins.

The results we prove express knowledge guarantees for live processes. Specifically, we show that all live processes always know about all neighbors that joined more than time $2T_g + 5d$ ago. Moreover, after a process has been live for sufficiently long, it knows about all neighbors that joined more than time d ago.

Breaking the proof up in some such way seems necessary in order to make the proof tractable. Each stage introduces its own new difficulties: the first stage already includes many of the issues involving the timing of the flow of information during and soon after the joining protocol. The second stage introduces issues of local knowledge—each process maintains information about its local neighborhood only. The third stage introduces the complications of failures, which mean that a process cannot rely on responses from any particular other process.

We expect this decomposition to be useful in constructing the general proof, because the ideas of the first stages should be useful in the later stages. Also, the result for the first stage should be directly usable in proving the more general results, in describing properties of the initial set-up phase.

6.4.1 Basic lemmas

The following lemma says that every block message contains a high expiration time for the sender.

The mention of a deadline for a message in transit refers to a detailed state-machine model for a timed channel, in which a deadline is explicitly kept for each message. This deadline is described in terms of absolute time.

Lemma 6.5. *Let α be a good finite execution. If a* block *message is in transit from i with deadline ℓ then it contains a finger for i with exptime $\geq \ell + T_e - d$.*

6.4.2 No failures, limited joins

In the case we consider in this subsection, no processes fail and at most $2c + 1$ join-acks occur. With this limited number of join-acks, every process is in every other process' c-block, so we do not have to worry about issues of local knowledge.

The following lemma says that everyone "always" has a finger for i_0, with a "sufficiently high" expiration time. The precise statement of this is rather complicated, because many different cases are covered.

Lemma 6.6. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c+1$* join-ack *events. Let i_0 denote the process that performs the first* join-ack *in α. Let $i \in PId$.*
Then in $\ell state(\alpha)$:

1. *If $status_i =$* joining *and α contains a* receive(lookup-comp)$_{*,i}$ *event for target $PToX(i)$, then there exists $f \in fingers_i$ with $f.phys = i_0$ such that:*
   (a) *One of the following holds:*
      i. *$f.exptime > ping\text{-}time_i + 2d$.*
      ii. *There is a* ping *message in $out\text{-}queue_i$ addressed to i_0 and $f.exptime > now + 2d$.*
      iii. *There is a* ping *message in transit from i to i_0 with deadline ℓ and $f.exptime > \ell + d$.*
      iv. *There is a* block *message in $out\text{-}queue_{i_0}$ addressed to i, and $f.exptime > now + d$.*
      v. *There is a* block *message in transit from i_0 to i with deadline ℓ, and $f.exptime > \ell$.*
   (b) *$f.exptime > now$.*
2. *If $status_i =$* joining *and α contains a* receive(block)$_{i_0,i}$ *event, then there exists $f \in fingers_i$ with $f.phys = i_0$ such that:*
   (a) *One of the following holds:*
      i. *$f.exptime > ping\text{-}time_i + 2T_g + 7d$.*
      ii. *There is a* ping *message in $out\text{-}queue_i$ addressed to i_0 and $f.exptime > now + 2T_g + 7d$.*
      iii. *There is a* ping *message in transit from i to i_0 with deadline ℓ and $f.exptime > \ell + 2T_g + 6d$.*
      iv. *There is a* block *message in $out\text{-}queue_{i_0}$ addressed to i, and $f.exptime > now + 2T_g + 6d$.*
      v. *There is a* block *message in transit from i_0 to i with deadline ℓ, and $f.exptime > \ell + 2T_g + 5d$.*
   (b) *$f.exptime > now + 2T_g + 5d$.*
3. *If $status_i =$* live *then there exists $f \in fingers_i$ with $f.phys = i_0$ such that:*
   (a) *One of the following holds:*

i. $f.exptime > nbr\text{-}refresh\text{-}time_i + 2T_g + 5d$.

ii. *There is a* block *message in* $out\text{-}queue_i$ *addressed to* i_0*, and* $f.exptime > now + 2T_g + 5d$.

iii. *There is a* block *message in transit from* i *to* i_0 *with deadline* ℓ *and* $f.exptime > \ell + 2T_g + 4d$.

iv. *There is a finger for* i *in* $fingers_{i_0}$ *with* $exptime > nbr\text{-}refresh\text{-}time_{i_0}$*, and* $f.exptime > nbr\text{-}refresh\text{-}time_{i_0} + T_g + 4d$.

v. *There is a* block *message in* $out\text{-}queue_{i_0}$ *addressed to* i*, and* $f.exptime > now + T_g + 4d$.

vi. *There is a* block *message in transit from* i_0 *to* i *with deadline* ℓ*, and* $f.exptime > \ell + T_g + 3d$.

(b) $f.exptime > now + T_g + 3d$.

4. *If a* block *or* lookup-comp *message is in an* $out\text{-}queue$ *then it contains a finger for* i_0 *with* $exptime > now + T_g + 3d$.

5. *If a* block *or* lookup-comp *message is in transit with deadline* ℓ *then it contains a finger for* i_0 *with* $exptime > \ell + T_g + 2d$.

Proof. We proceed by induction on the number of steps in α following the join-ack$_{i_0}$.
Base: 0 steps.
Then the last step of α is join-ack$_{i_0}$. All the conditions are easy to check.
Inductive step: The only actions that could falsify any of the claims are receive(lookup), send(lookup-comp), receive(lookup-comp), join-ping, send(ping)$_{*,i_0}$, receive(ping)$_{*,i_0}$, send(block), receive(block), join-ack, neighbor-refresh, ν, and garbage-collect.
We consider cases.

1. receive(lookup)$_{*,i}$.

   This has the potential to falsify Property 4, in the case where a lookup-comp message is placed in $out\text{-}queue_i$. By inductive hypothesis, Property 3(b), in the pre-state of the final transition, there exists $f \in fingers_i$ such that $f.phys = i_0$ and $f.exptime > now + T_g + 3d$. Therefore, if a lookup-comp message is placed in $out\text{-}queue_i$ as a result of this transition, it contains a finger for i_0 with $exptime > now + T_g + 3d$. This shows Property 4.

2. send(lookup-comp)$_{i,j}$

   This could falsify Property 5. In the pre-state of the final transition, a lookup-comp message is in $out\text{-}queue_i$. Therefore, by inductive hypothesis, Property 4, this message contains a finger for i_0 with $exptime > now + T_g + 3d$. Since $\ell \leq now + d$, we have $exptime > \ell + T_g + 2d$, as needed for Property 5.

3. receive(lookup-comp)$_{*,i}$.

   This could falsify Property 1. Before the step, a lookup-comp message is in transit to i with deadline $\geq now$. By inductive hypothesis, Property 5, this message contains a finger for i_0 with $exptime > now + T_g + 2d$. So after the step, $fingers_i$ contains a finger f for i_0 with $f.exptime > now + T_g + 2d$. Since $ping\text{-}time_i \leq now + T_g$, we have that $f.exptime > ping\text{-}time_i + 2d$. This shows both parts of Property 1.

4. join-ping$_i$.

   This could falsify Property 1(a) or 2(a). For Property 1(a), suppose that $status_i =$ joining and α contains a receive(lookup-comp)$_{*,i}$ event for target $PToX(i)$. The interesting case is where 1a(i) is true just before the step, that is, $fingers_i$ contains a finger f for i_0 with $f.exptime > ping\text{-}time_i + 2d$. Since $ping\text{-}time_i \geq now$, this implies that $f.exptime > now + 2d$. This inequality is true after the step as well.

   We claim that the step results in a ping message addressed to i_0 being placed in $out\text{-}queue_i$; this means that 1a(ii) is satisfied in the post-state, as needed. Since we have assumed that Property 1(a)i is true in the pre-state, we know that $fingers_i$ contains a finger for i_0 in the pre-state. Since a receive(lookup-comp) occurs in α for target $PToX(i)$, we know that there exists r such that $r.id \in join.comp_i$ and $r.target = PToX(i)$. Therefore, the join-ping deposits ping messages addressed to its entire c-block, according to its local ring. This includes i_0, as needed.

   For Property 2(a), the argument is similar to that for Property 1(a). This time, suppose that $status_i =$ joining and α contains a receive(block)$_{*,i}$ event. The interesting case is where 2a(i) is true just before the step, that is,

$\mathit{fingers}_i$ contains a finger f for i_0 with $f.\mathit{exptime} > \mathit{ping\text{-}time}_i + 2T_g + 7d$. Since $\mathit{ping\text{-}time}_i \geq \mathit{now}$, this implies that $f.\mathit{exptime} > \mathit{now} + 2T_g + 7d$. This inequality is true after the step as well.

We claim that the step results in a ping message addressed to i_0 being placed in $\mathit{out\text{-}queue}_i$; this means that 2(a)ii is satisfied in the post-state, as needed. Since we have assumed that Property 2(a)i is true in the pre-state, we know that $\mathit{fingers}_i$ contains a finger for i_0 in the pre-state. Since a receive(lookup-comp) occurs in α for target $PToX(i)$, we know that there exists r such that $r.id \in \mathit{join.comp}_i$ and $r.\mathit{target} = PToX(i)$. Therefore, the join-ping deposits ping messages addressed to its entire c-block, according to its local ring. This includes i_0, as needed.

5. send(ping)$_{i,i_0}$.

   This could falsify Property 1(a) or 2(a). For Property 1(a), suppose that $\mathit{status}_i =$ joining and α contains a receive(lookup-comp)$_{*,i}$ event for target $PToX(i)$. The interesting case is where 1a(ii) is true just before the step, that is, $\mathit{fingers}_i$ contains a finger f for i_0 with $f.\mathit{exptime} > \mathit{now} + 2d$ and there is a ping message in $\mathit{out\text{-}queue}_i$ addressed to i_0. After the step, there is a ping message in transit from i to i_0 with deadline $\mathit{now} + d$. Taking $\ell = \mathit{now} + d$, we see that 1c is true after the step.

   For Property 2(a), the argument is similar: 2(a)ii before the step implies 2(a)iii after the step.

6. receive(ping)$_{i,i_0}$.

   This could falsify Property 1(a) or 2(a). For Property 1(a), suppose that $\mathit{status}_i =$ joining and α contains a receive(lookup-comp)$_{*,i}$ event for target $PToX(i)$. The interesting case is where 1a(iii) is true just before the step, that is, $\mathit{fingers}_i$ contains a finger f for i_0 with $f.\mathit{exptime} > \ell + d$ and there is a ping message in transit from i to i_0 with deadline ℓ. Since $\ell \geq \mathit{now}$, we have that $f.\mathit{exptime} > \mathit{now} + d$. After the step, there is a block message in $\mathit{out\text{-}queue}_{i_0}$ addressed to i. Therefore, 1a(iv) is true just after the step.

   For Property 2(a), the argument is similar: 2a(iii) before the step implies 2a(iv) after the step.

7. send(block)$_{j,k}$.

   This could falsify Property 1(a), 2(a), 3(a), or 5. For Property 1(a), the interesting case is where $j = i_0$, $k = i$, and 1(a)iv is true before the step, that is, $\mathit{fingers}_i$ contains a finger f for i_0 with $f.\mathit{exptime} > \mathit{now} + d$. Since $\mathit{now} + d \geq \ell$, we have that $f.\mathit{exptime} > \ell$, so that 1a(v) holds after the step.

   For Property 2(a), the interesting case is where $j = i_0$, $k = i$, and 2(a)iv holds before the step. Then, arguing as in the previous case, 2(a)v holds after the step.

   For Property 3(a), there are two interesting cases. The first is where $j = i$, $k = i_0$, and 3(a)ii holds before the step; in this case 3(a)iii holds after the step. The second case is where $j = i_0$, $k = i$, and 3(a)v holds before the step; in this case 3(a)vi holds after the step.

   For Property 5, we use Property 4 in the pre-state to show Property 5 in the post-state.

8. receive(block)$_{j,k}$

   This could falsify Property 1(a), 2(a), or 3(a).

   For Property 1(a), the interesting case is where $j = i_0$, $k = i$, and Property 1(a)v holds before the step. Then by Lemma 6.5, the received message contains a finger for i_0 with $\mathit{exptime} \geq \mathit{now} + T_e - d$. By assumptions on the constants, the right-hand side is $> 4T_g + 7d$, so $\mathit{exptime} \geq \mathit{now} + 4T_g + 7d$. Therefore, after the step, $\mathit{fingers}_i$ contains a finger for i_0 with $\mathit{exptime} > \mathit{now} + 4T_g + 7d > \mathit{ping\text{-}time}_i + 2d$. Thus, 1(a)i is satisfied after the step.

   For Property 2(a), the interesting case is where $j = i_0$, $k = i$, and Property 2(a)v holds before the step. Arguing as in the previous case, we see that after the step, $\mathit{fingers}_i$ contains a finger for i_0 with $\mathit{exptime} > 4T_g + 7d \geq \mathit{ping\text{-}time}_i + 2T_g + 7d$. Thus, 2(a)i is satisfied after the step.

   For Property 3(a), there are two interesting cases. The first is where $j = i$, $k = i_0$, and 3(a)iii is satisfied before the step; then we claim that 3(a)iv holds after the step. The argument for this uses Lemma 6.5, applied to i. The second case is where $j = i_0$, $k = i$, and 3a(vi) is satisfied before the step; in this case, 3(a)i holds after the step.

9. join-ack$_i$.

   This could falsify Property 3(a). By inductive hypothesis, Property 2(b), in the pre-state, $\mathit{fingers}_i$ contains a finger for i_0 with $\mathit{exptime} > \mathit{now} + 2T_g + 5d$. Since $\mathit{nbr\text{-}refresh\text{-}time}_i = \mathit{now}$ right after the step, 3(a)i holds after the step.

10. neighbor-refresh$_i$.

    This could falsify Property 3(a). The interesting case is where Property 3a(i) holds in the pre-state. The step puts a block message in $\mathit{out\text{-}queue}_i$ addressed to i_0. Then 3a(ii) holds in the post-state.

11. $\nu(t)$

    This could falsify Property 1, 2, 3, or 4. For Property 1, there are two interesting cases. The first is where 1(a)iv holds in the pre-state. But then time cannot pass, by our timing assumption (no time passes while an $\mathit{out\text{-}queue}$ is nonempty). The second possibility is that we might falsify 1(b). However, note that 1(b) follows from 1(a). Similar arguments hold for Properties 2, 3, and 4.

12. garbage-collect.

    Since in every case, the finger whose existence is claimed has $\mathit{exptime} > \mathit{now}$, it cannot be garbage-collected. Therefore, garbage-collect cannot falsify any of the claims.

□

Next, we describe knowledge that i_0 acquires about the other processes.

Lemma 6.7. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c+1$* join-ack *events. Let i_0 denote the process that performs the first* join-ack *in α. Let $i \in PId$ be such that* join-ack$_i$ *occurs in α at time t.*
Then in $\ell state(\alpha)$, one of the following holds:

1. *$t = \mathit{now}$ and a* block *message addressed to i_0 is in $\mathit{out\text{-}queue}_i$.*
2. *A* block *message is in transit from i to i_0 with deadline $t + d$.*
3. *$\mathit{fingers}_{i_0}$ contains a finger f for i such that one of the following holds:*
   (a) $f.\mathit{exptime} \geq \mathit{nbr\text{-}refresh\text{-}time}_i + T_e - T_g$.
   (b) A block *message addressed to i_0 is in $\mathit{out\text{-}queue}_i$ and $f.\mathit{exptime} \geq \mathit{now} + T_e - T_g$.*
   (c) A block *message is in transit from i to i_0 with deadline ℓ and $f.\mathit{exptime} \geq \ell + T_e - (T_g + d)$.*

Proof. By induction on the number of steps in α following the join-ack$_i$.
Base: 0 steps.
Then the last step of α is join-ack$_i$. Then we claim that Property 1 holds in the post-state. This follows because in the pre-state, i has a finger for i_0, by Lemma 6.6, part 3(b).
Inductive step: The only actions that could falsify the claim are send(block)$_i$, receive(block)$_{i_0}$, neighbor-refresh$_i$, time-passage, and garbage-collect$_{i_0}$.

1. send(block)$_i$

   This could falsify Property 1 or 3(b). However, if it does so, it makes Property 2 or 3(c) (respectively) true.

2. receive(block)$_{i_0}$

   Lemma 6.5 implies that after the step, $\mathit{fingers}_{i_0}$ contains a finger for i with $\mathit{exptime} \geq \ell + T_e - d$, where ℓ is the deadline component of the received message. Since $\ell = \mathit{nbr\text{-}refresh\text{-}time}_i - T_g + d$, (the sending time plus d), this implies that this finger has $\mathit{exptime} \geq \mathit{nbr\text{-}refresh\text{-}time}_i - T_g + d + T_e - d$, that is, $\mathit{exptime} \geq \mathit{nbr\text{-}refresh\text{-}time}_i + T_e - T_g$, which shows that 3(a) is satisfied after the step.

3. neighbor-refresh$_i$

   This could falsify Property 3(a); however, if it does so then Property 3(b) holds after the step.

4. $\nu(i)$

   This could falsify Property 1 or 3(b). However, if 1 or 3(b) holds in the pre-state, then time cannot pass, by our timing assumptions, because an $out\text{-}queue$ is nonempty.

5. garbage-collect$_{i_0}$.

   Because $T_e > T_g + d$, the expiration times of the claimed fingers are all strictly greater than 0. Therefore, this cannot falsify any of the statements.

□

The following corollary summarizes the conclusions of Lemma 6.7, saying that i_0 has a finger for any other process i that has joined at least time d ago, with a high expiration time. Also, any block message that is sent by i_0 sufficiently long after i joins contains a finger for i with a high expiration time.

Corollary 6.8. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most* $2c + 1$ join-ack *events. Let i_0 denote the process that performs the first* join-ack *in α. Let $i \in PId$ be such that* join-ack$_i$ *occurs in α at time t.*
Then in $\ell state(\alpha)$, the following hold:

1. *If $t + d < now$ then fingers$_{i_0}$ contains a finger f for i such that $f.exptime \geq now + T_e - (T_g + d)$.*
2. *If $t + 2d < \ell$ and a* block *message is in transit from i_0 with deadline ℓ, then the message contains a finger for i such that $f.exptime \geq \ell + T_e - (T_g + 2d)$.*

The next lemma gives guarantees about what an arbitrary process i knows about another arbitrary process j. This represents "second-order" information, because i may need to learn this information indirectly, through i_0.

Lemma 6.9. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c+1$* join-ack *events. Let i_0 denote the process that performs the first* join-ack *in α. Let $s = \ell state(\alpha)$. Then:*

1. *Suppose that $s.status_i =$* joining *and α contains a* receive(block)$_{i_0,i}$ *event. Suppose that* join-ack$_j$ *occurs in α at a time $< \ell time(\alpha) - (T_g + 3d)$.*
   Then $s.fingers_i$ contains a finger f for j such that $f.exptime \geq s.now + T_e - (2T_g + 3d)$.
2. *Suppose that $s.status_i =$* active *and* join-ack$_i$ *occurs in α at a time $\geq \ell time(\alpha) - (T_g + 2d)$. Suppose that* join-ack$_j$ *occurs in α at a time $< \ell time(\alpha) - (2T_g + 5d)$. Then $s.fingers_i$ contains a finger f for j such that $f.exptime \geq s.now + T_e - (3T_g + 5d)$.*
3. *Suppose that $s.status_i =$* active *and* join-ack$_i$ *occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that* join-ack$_j$ *occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Then $s.fingers_i$ contains a finger f for j such that $f.exptime \geq s.now + T_e - (2T_g + 2d)$.*

The proofs are based on conveying information through i_0. These proofs are not inductive; rather, they rest directly on previously-proved lemmas.

Proof. 1. Assume that $s.status_i =$ joining and α contains a receive(block)$_{i_0,i}$ event. Also suppose that join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 3d)$.

Lemma 6.6, Part 1(b), implies that whenever i sends a ping message during its joining protocol, it has a finger for i_0. Thus, by the limitation on the number of join-ack events, i_0 is included in the set of destinations of the ping message.

We claim that, in α, process i receives a block message from i_0 sent by i_0 in response to a ping message sent by i at a time $\geq s.now - (T_g + 2d)$. For if not, then the latest block message received by i from i_0 is a response to a ping sent by i at a time $< s.now - (T_g + 2d)$. But then it must be that another ping message is sent by i at a time $< s.now - 2d$, and this receives a response by the end of α, a contradiction.

Since the time of the join-ack$_j$ is $< s.now - (T_g + 3d)$, it must be $< s'.now - d$, where s' is the state just before i_0 sends this block message. Therefore, by Corollary 6.8, Part 1, in state s', *fingers*$_{i_0}$ contains a finger for j with $exptime \geq s'.now + T_e - (T_g + d)$. Therefore, in state s, which is at most time $T_g + 2d$ later, *fingers*$_i$ contains a finger for j with $exptime \geq s.now + T_e - (2T_g + 3d)$, as needed.

2. Suppose that $s.status_i =$ active and join-ack$_i$ occurs in α at a time $\geq \ell time(\alpha) - (T_g + 2d)$. Also suppose that join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (2T_g + 5d)$.

   By the inductive hypothesis, Part 1, we know that, just before the join-ack$_i$, $\mathit{fingers}_i$ contains a finger for j with $exptime \geq now + T_e - (2T_g + 3d)$. Therefore, in state s, which is at most time $T_g + 2d$ later, $\mathit{fingers}_i$ contains a finger for j with $exptime \geq s.now + T_e - (3T_g + 5d)$, as needed.

3. Suppose that $s.status_i =$ active and join-ack$_i$ occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$.

   Corollary 6.8, Part 1, implies that in any state s' of α with $s'.now > t' + d$, $\mathit{fingers}_{i_0}$ contains a finger for i with $exptime \geq s'.now + T_e - (T_g + d)$. Since $T_e > T_g + d$ and because of the limitation on the number of join-ack events, i is included in the set of destinations of every block message sent by i_0 in such a state s'.

   We claim that, in α, process i receives a block message from i_0 sent by i_0 at a time $\geq s.now - (T_g + d)$. For if not, then the latest block message received by i from i_0 is sent by i_0 at a time $< s.now - (T_g + d)$. But then it must be that another block message is sent by i_0 (as part of a neighbor-refresh$_{i_0}$) at a time $< s.now - d$, and this arrives at i by the end of α, a contradiction.

   Now fix s' to be the state just before i_0 sends this block message; thus, $s'.now \geq s.now - (T_g + d)$. Putting this inequality together with the assumption that the join-ack$_j$ occurs at a time $< s.now - (T_g + 2d)$, we may conclude that the join-ack$_j$ occurs at a time $< s'.now - d$. Therefore, by Corollary 6.8, Part 1, in state s', $\mathit{fingers}_{i_0}$ contains a finger for j with $exptime \geq s'.now + T_e - (T_g + d)$. Therefore, in state s, which is at most time $T_g + d$ later, $\mathit{fingers}_i$ contains a finger for j with $exptime > s.now + T_e - (2T_g + 2d)$, as needed.

$\square$

The following lemma describes information that i is guaranteed to have after receiving a lookup-comp message. It represents "third-order" information, because the lookup-comp message could be conveying "second-order" information from its sender.

Lemma 6.10. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c+1$* join-ack *events. Let $i, j \in PId$. Suppose that $status_i =$* joining *and α contains a* receive(lookup-comp)$_{*,i}$ *event for target $PToX(i)$. Suppose that* join-ack$_j$ *occurs in α, at a time $< \ell time(\alpha) - (3T_g + 8d)$.*
Then in $\ell state(\alpha)$, $\mathit{fingers}_i$ contains a finger for j with $exptime > now$.

Proof. (Sketch:) If the time when process i receives the lookup-comp message is $< \ell time(\alpha) - (T_g + 2d)$, then i also receives a block message from i_0 before the end of α. In this case the result follows from Lemma 6.9, Part 1.

On the other hand, if the time when process i receives the lookup-comp message is $\geq \ell time(\alpha) - (T_g + 2d)$, then the result follows from Lemma 6.9, part 2, applied to the sender of the message. In applying this lemma, we add time $T_g + 3d$ (d for the message delay and $T_g + 2d$ for the time that might have elapsed from the receive(lookup-comp)) to the age of the known processes and subtract this from the expiration time of the finger. This uses the fact that $T_e > 4T_g + 9d$. $\square$

The next series of results bound how long it takes for a process i to become an "authority", like i_0. That is, it knows about all processes that have joined more than time d ago. The first case is where anoether process j joins sufficiently long after i so that j knows about i at the point where it joins.

Lemma 6.11. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c+1$* join-ack *events. Suppose that* join-ack$_i$ *and* join-ack$_j$ *occur in α at times t and t', respectively, and where $t + T_g + 3d < t' < now - d$.*
Then in $\ell state(\alpha)$, $\mathit{fingers}_i$ contains a finger for j with $exptime \geq now + T_e - (T_g + d)$.

Proof. We first claim that, at any point in α after the join-ack$_j$, $\mathit{fingers}_j$ contains a finger for i with $exptime > now$. Lemma 6.9, Part 1, implies that, in the state immediately before the *join-ack$_j$*, $\mathit{fingers}_j$ contains a finger for i with $exptime \geq now + T_e - (2T_g + 4d)$. Thereafter in α, through time $t' + T_g + 2d$, $\mathit{fingers}_j$ contains a finger for i with $exptime > now + T_e - (3T_g + 6d)$. Also, at any time after $t' + T_g + 2d$ in α, Lemma 6.9, Part 2 implies that $\mathit{fingers}_j$ contains a finger for i with $exptime \geq now + T_e - (3T_g + 6d)$. Combining these two facts, we conclude that, at any time after the join-ack$_j$, $\mathit{fingers}_j$ contains a finger for i with $exptime \geq now + T_e - (3T_g + 6d) > now$.

Immediately after the join-ack$_j$, and at intervals of T_g thereafter, process j performs a neighbor-refresh$_j$, in which it sends a block message containing a finger for itself with $exptime = T_e$. By the argument in the previous paragraph, i is included in the destination set of each such block message. At the end of α, some such message must have arrived at i which was sent by j at a time $\geq \ell time(\alpha) - (T_g + d)$. Therefore, in $\ell state(\alpha)$, *fingers*$_i$ contains a finger for j with $exptime \geq now + T_e - (T_g + 2d)$, as needed. □

The second case is where i and j both join long enough before the end of the execution.

Lemma 6.12. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c + 1$* join-ack *events. Suppose that* join-ack$_i$ *and* join-ack$_j$ *occur in α at times t and t', respectively, where $t, t' < \ell time(\alpha) - (2T_g + 3d)$.*
Then in $\ell state(\alpha)$, fingers$_i$ contains a finger for j with $exptime > now + T_e - (T_g + d)$.

Proof. By Corollary 6.8, Part 1, by time strictly less than $\ell time(\alpha) - (2T_g + 2d)$, *fingers*$_{i_0}$ contains fingers for both i and j, each with $exptime \geq now + T_e - (T_g + 2d)$. Then by time strictly less than $\ell time(\alpha) - (T_g + d)$, j receives a block message from i_0 telling j about i, resulting in *fingers*$_j$ containing a finger for i, with $exptime \geq now + T_e - (2T_g + 3d)$. And then by time strictly less than $\ell time(\alpha)$, i receives a block message directly from j telling i about j, and producing the needed finger. □

The following corollary says that if process i has joined more than time $3T_g + 6d$ ago, it is an "authority", in the sense that it knows about all processes j that has joined more than time d ago.

Corollary 6.13. *Let α be a good finite execution that contains no* fail *events, and contains at least one and at most $2c + 1$* join-ack *events. Suppose that* join-ack$_i$ *and* join-ack$_j$ *occur in α at times t and t', respectively, where $t < \ell time(\alpha) - (3T_g + 6d)$ and $t' < \ell time(\alpha) - d$.*
Then in $\ell state(\alpha)$, fingers$_i$ contains a finger for j with $exptime > now + T_e - (T_g + d)$.

Proof. This follows from the two previous lemmas. □

6.4.3 Joins and failures

Now we use the ideas in the previous section to talk about what happens when we have unlimited joins and also failures. Now, instead of relying on i_0 as an "authority", processes rely on neighbors that happen to have been around long enough. Because of the failures, we now consider the augmented ring as well as the actual global ring.

From now on, I am being slightly sloppy by writing just i instead of $PToX(i)$ in many places. This is done for the sake of readability. I hope it does not cause any confusion. The first lemma relates various neighborhoods in the same ring.

Lemma 6.14. *Let R be any ring, $i, j, k \in PId$.*

1. *If $j \in block(i, e_1, R)$ and $k \in block(i, e_2, R)$, then $j \in block(k, e_1 + e_2, R)$.*
2. *If $j \in succset(i, e_1, R)$, $k \in succset(i, e_2, R)$, and $k \notin succset(i, e_3, R)$, then $j \in block(k, \max(e_1 - e_3, e_2), R)$.*

Proof. Straightforward. □

The following lemma asserts the existence of neighbors that have joined a long time ago.

Lemma 6.15. *Assume that $c > e_1 + e_2 + e_3 joinbd$. Let α be a good finite execution, $R = global\text{-}ring(\alpha)$. Let $i \in PId$. Suppose that $|R| \geq c + 1$. Then:*

1. *There exists $k \in PId$ such that*
   (a) $k \in succset(i, c - e_1, R)$.
   (b) $k \notin succset(i, e_2, R)$.
   (c) join-ack$_k$ *occurs at a time $< \ell time(\alpha) - e_3(T_g + 2d)$*
   (d) fail$_k$ *does not occur in α.*

2. *There exists $k \in PId$ such that*

   (a) $k \in predset(i, c - e_1, R)$.

   (b) $k \notin predset(i, e_2, R)$.

   (c) $\mathsf{join\text{-}ack}_k$ *occurs at a time* $< \ell time(\alpha) - e_3(T_g + 2d)$

   (d) fail_k *does not occur in* α.

Proof. We prove Part 1; Part 2 is analogous. There are at least $c - (e_1 + e_2)$ processes in the set difference $succset(i, c - e_1, R) - succset(i, e_2, R)$. Of these, at most $e_3 joinbd$ perform a join-ack at times $\geq \ell time(\alpha) - e_3(T_g + 2d)$. Since $c > e_1 + e_2 + e_3 joinbd$, it must be that at least one of these processes, call it k, performs a join-ack at a time $< \ell time(\alpha) - e_3(T_g + 2d)$. This k satisfies all the listed properties. □

The next lemma relates neighborhoods in the global ring to neighborhoods in the augmented ring.

Lemma 6.16. *Let α be a good finite execution, $e \in \mathbb{N}$, $i, j \in PId$.*

1. *If $j \in psuccset(i, e, global\text{-}ring(\alpha))$ then $j \in psuccset(i, e + failbd, aug\text{-}ring(\alpha))$.*
2. *If $j \in ppredset(i, e, global\text{-}ring(\alpha))$ then $j \in ppredset(i, e + failbd, aug\text{-}ring(\alpha))$.*
3. *If $j \in succset(i, e, global\text{-}ring(\alpha))$ then $j \in succset(i, e + failbd, aug\text{-}ring(\alpha))$.*
4. *If $j \in predset(i, e, global\text{-}ring(\alpha))$ then $j \in predset(i, e + failbd, aug\text{-}ring(\alpha))$.*
5. *If $j \in block(i, e, global\text{-}ring(\alpha))$ then $j \in block(i, e + failbd, aug\text{-}ring(\alpha))$.*

Proof. (Sketch) These follow because at most $failbd$ processes in the given region appear in $aug\text{-}ring(\alpha)$ but not in $global\text{-}ring(\alpha)$. □

The next lemma says that neighbors in the augmented ring are also neighbors in the local ring.

Lemma 6.17. *Let α be a good finite execution, $s = \ell state(\alpha)$.*

1. *If $j \in succset(i, e, aug\text{-}ring(\alpha))$ and $fingers_i$ contains a finger for j, then $j \in succset(i, e, s.local\text{-}ring_i)$.*
2. *If $j \in predset(i, e, aug\text{-}ring(\alpha))$ and $fingers_i$ contains a finger for j, then $j \in predset(i, e, s.local\text{-}ring_i)$.*
3. *If $j \in block(i, e, aug\text{-}ring(\alpha))$ and $fingers_i$ contains a finger for j, then $j \in block(i, e, s.local\text{-}ring_i)$.*

Proof. We show Part 1; the rest are similar. If $j \notin succset(i, e, s.local\text{-}ring_i)$, then it must be that there are at least e elements of $s.local\text{-}ring_i)$ in the interval (i, j). But each of these is an element of $aug\text{-}ring(\alpha))$, which contradicts the assumption that $j \in succset(i, e, aug\text{-}ring(\alpha))$. □

The next lemma relates the augmented ring at some point to the global ring at a point not too far in the past.

Lemma 6.18. *Let α be a good finite execution, α' a prefix of α with $\ell time(\alpha') \geq \ell time(\alpha) - T_e$. If $i \in global\text{-}ring(\alpha')$, then $i \in aug\text{-}ring(\alpha)$.*

Proof. By the definition of $aug\text{-}ring$. □

The next lemma says that a neighbor in the augmented ring at a particular time is a neighbor in the global ring at a point not too far in the past.

Lemma 6.19. *Let α be a good finite execution, α' a prefix of α with $\ell time(\alpha') \geq \ell time(\alpha) - T_e$. Let $e \in \mathbb{N}$ and $i, j \in PId$. Suppose $j \in global\text{-}ring(\alpha')$. Then:*

1. *If $j \in psuccset(i, e, aug\text{-}ring(\alpha))$ then $j \in psuccset(i, e, global\text{-}ring(\alpha'))$.*
2. *If $j \in ppredset(i, e, aug\text{-}ring(\alpha))$ then $j \in ppredset(i, e, global\text{-}ring(\alpha'))$.*

3. *If* $j \in \mathit{succset}(i, e, \mathit{aug\text{-}ring}(\alpha))$ *then* $j \in \mathit{predset}(i, e, \mathit{global\text{-}ring}(\alpha'))$.

4. *If* $j \in \mathit{predset}(i, e, \mathit{aug\text{-}ring}(\alpha))$ *then* $j \in \mathit{predset}(i, e, \mathit{global\text{-}ring}(\alpha'))$.

5. *If* $j \in \mathit{block}(i, e, \mathit{aug\text{-}ring}(\alpha))$ *then* $j \in \mathit{block}(i, e, \mathit{global\text{-}ring}(\alpha'))$.

Proof. For Part 1, suppose for the sake of contradiction that $j \notin \mathit{psuccset}(i, e, \mathit{global\text{-}ring}(\alpha'))$. Then $|\mathit{global\text{-}ring}(\alpha') \cap (i, j)| > e$, that is, there are more than e elements of $\mathit{global\text{-}ring}(\alpha')$ in the interval properly between i and j, moving in the clockwise direction. By Lemma 6.18, every such element is also in $\mathit{aug\text{-}ring}(\alpha')$. Therefore, $j \notin \mathit{psuccset}(i, e, \mathit{aug\text{-}ring}(\alpha))$. This is a contradiction.

The proof of Part 2 is analogous. For Part 3, suppose that $j \in \mathit{succset}(i, e, \mathit{aug\text{-}ring}(\alpha))$. If $j \in \mathit{psuccset}(i, e, \mathit{aug\text{-}ring}(\alpha))$ then the conclusion follows from Part 1. The only remaining case is where $j = i$, but this case follows trivially from the fact that $j \in \mathit{global\text{-}ring}(\alpha')$.

Part 4 is analogous. Part 5 follows from Parts 3 and 4. $\square$

The following lemma summarizes facts about the knowledge of a new process at various points during and soon after its joining protocol.

Lemma 6.20. *Let α be a good finite execution, $s = \ell state(\alpha)$. Let i be a process that does not fail in α. Then:*

1. *Suppose that* $s.\mathit{status}_i =$ joining *and a* receive(lookup-comp)$_{*,i}$ *event for target i occurs in α at a time* $\geq \ell time(\alpha) - (T_g + 2d)$. *Suppose that* $j \in \mathit{block}(i, c, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ *occurs in α at a time* $< \ell time(\alpha) - (3T_g + 8d)$, *and* fail$_j$ *does not occur in α.*
Then $\mathit{fingers}_i$ *contains a finger for j with* $\mathit{exptime} > \mathit{now}$.

2. *Suppose that* $\mathit{status}_i =$ joining *and a* receive(lookup-comp)$_{*,i}$ *event for target i occurs in α at a time* $< \ell time(\alpha) - (T_g + 2d)$. *Suppose that* $j \in \mathit{block}(i, b, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ *occurs in α at a time* $< \ell time(\alpha) - (T_g + 3d)$, *and* fail$_j$ *does not occur in α.*
Then $\mathit{fingers}_i$ *contains a finger for j with* $\mathit{exptime} > \mathit{now} + T_e - (2T_g + 3d)$.

3. *Suppose that* $s.\mathit{status}_i =$ active *and a* join-ack$_i$ *occurs in α at a time* $\geq \ell time(\alpha) - (T_g + 2d)$. *Suppose that* $j \in \mathit{block}(i, b, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ *occurs in α at a time* $< \ell time(\alpha) - (2T_g + 5d)$, *and* fail$_j$ *does not occur in α.*
Then $s.\mathit{fingers}_i$ *contains a finger for j with* $\mathit{exptime} \geq s.\mathit{now} + T_e - (3T_g + 6d)$.

4. *Suppose that* $s.\mathit{status}_i =$ active *and a* join-ack$_i$ *occurs in α at a time* $< \ell time(\alpha) - (T_g + 2d)$. *Suppose that* $j \in \mathit{block}(i, b, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ *occurs in α at a time* $< \ell time(\alpha) - (T_g + 2d)$, *and* fail$_j$ *does not occur in α.*
Then $s.\mathit{fingers}_i$ *contains a finger for j with* $\mathit{exptime} \geq s.\mathit{now} + T_e - (2T_g + 2d)$.

5. *Suppose that* $s.\mathit{status}_i =$ active *and a* join-ack$_i$ *occurs in α at a time* $< \ell time(\alpha) - (3T_g + 6d)$. *Suppose that* $j \in \mathit{block}(i, b - \mathit{failbd}, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ *occurs in α at a time* $< \ell time(\alpha) - d$, *and* fail$_j$ *does not occur in α.*
Then $s.\mathit{fingers}_i$ *contains a finger for j with* $\mathit{exptime} \geq s.\mathit{now} + T_e - (T_g + d)$.

Proof. Let R denote $\mathit{global\text{-}ring}(\alpha)$. The proof is by strong induction on the number of steps in α.
Base: The total number of join-ack events in α is at most $2c + 1$.

If there are no join-ack events in α then the statements are all vacuously true. If there are between one and $2c + 1$ join-ack events in α then the five claims follow from Lemma 6.10, Lemma 6.9, Parts 1, 2, and 3, and Corollary 6.13, respectively. (This uses the fact that, in the absence of failures, $\mathit{aug\text{-}ring}$ is the same as $\mathit{global\text{-}ring}$.)
Inductive step: We assume that α contains more than $2c + 1$ join-ack events. We assume that the result is true for all proper prefixes of α and show it for α. We show the five properties in turn.

1. For Part 1, suppose that $s.\mathit{status}_i =$ joining, a receive(lookup-comp)$_{*,i}$ event for target i occurs in α at a time $\geq \ell time(\alpha) - (T_g + 2d)$, and fail$_k$ does not occur in α. Also suppose that $j \in \mathit{block}(i, c, \mathit{aug\text{-}ring}(\alpha))$ and join-ack$_j$ occurs at a time $< \ell time(\alpha) - (3T_g + 8d)$. We must show that $s.\mathit{fingers}_i$ contains a finger for j with positive $\mathit{exptime}$.

Consider the first lookup-comp message for target i that is received by i, and let k be the sender of this message. Let α' be the prefix of α ending just before the receive(response) step in which k sends this message, let $s' = \ell state(\alpha')$ and let $R' = \mathit{global\text{-}ring}(\alpha')$.

By inductive hypothesis, Parts 3 and 4, $s'.\mathit{fingers}_k$ contains a finger for every process in $\mathit{succset}(k, b, \mathit{aug\text{-}ring}(\alpha'))$ whose join-ack event occurs at a time $< \ell time(\alpha') - (2T_g + 5d)$ and that does not fail in α'. Therefore, by Lemma 6.16, $s'.\mathit{fingers}_k$ contains a finger for every process in $\mathit{succset}(k, b - \mathit{failbd}, R')$ whose join-ack event occurs at a time $< \ell time(\alpha') - (2T_g + 5d)$. In particular, $s'.\mathit{fingers}_k$ contains a finger for every process in $\mathit{succset}(k, \min(|R' \cap [k, i)|, b - \mathit{failbd}), R')$ whose join-ack event occurs at a time $< \ell time(\alpha') - (2T_g + 5d)$.

By our assumption on the join rate, at most $3\mathit{joinbd}$ processes in $\mathit{succset}(k, b - \mathit{failbd}, R')$ perform join-ack events at times $\geq \ell time(\alpha') - (2T_g + 5d)$. It follows that $s'.\mathit{fingers}_k$ contains at least $\min(|R' \cap [k, i)|, b - \mathit{failbd}) - 3\mathit{joinbd}$ fingers for processes in $R' \cap [k, i)$.

Now we claim that $k \in \mathit{ppredset}(i, c + 3\mathit{joinbd}, R')$. If not, then $|R' \cap [k, i)| > c + 3\mathit{joinbd}$. Then, since $b > c + 3\mathit{joinbd} + \mathit{failbd}$, we have that $\min(|R' \cap [k, i)|, b - \mathit{failbd}) - 3\mathit{joinbd} > c$, which implies that $s'.\mathit{fingers}_k$ contains strictly more than c fingers for processes in $R' \cap [k, i)$. This implies that $k \notin \mathit{ppredset}(i, c, s'.\mathit{local\text{-}ring}_i)$, However, the definition of the receive(response) transitions implies that $k \in \mathit{ppredset}(i, c, s'.\mathit{local\text{-}ring}_i)$, which yields a contradiction. Therefore, $k \in \mathit{ppredset}(i, c + 3\mathit{joinbd}, R')$, as claimed.

Since $j \in \mathit{block}(i, c, \mathit{aug\text{-}ring}(\alpha))$, Lemma 6.19 implies that $j \in \mathit{block}(i, c, R')$. Since $j \in \mathit{block}(i, c, R')$ and $k \in \mathit{predset}(i, c + 3\mathit{joinbd}, R')$, Lemma 6.14 implies that $j \in \mathit{block}(k, 2c + 3\mathit{joinbd}, R')$. Since (by assumption on constants) $b \geq 2c + 3\mathit{joinbd} + \mathit{failbd}$, we have that $j \in \mathit{block}(k, b - \mathit{failbd}, R')$. Therefore, by Lemma 6.16, $j \in \mathit{block}(k, b, \mathit{aug\text{-}ring}(\alpha'))$.

Now we use the inductive hypothesis, Parts 3 and 4, again, to conclude that $s'.\mathit{fingers}_k$ contains a finger for j with $\mathit{exptime} \geq s'.\mathit{now} + T_e - (3T_g + 6d)$. To apply the inductive hypothesis, we need the fact that join-ack$_j$ occurs at a time $< \ell time(\alpha') - (2T_g + 5d)$; this follows from our assumption that join-ack$_j$ occurs at a time $< \ell time(\alpha) - (3T_g + 8d)$ and the fact that $\ell time(\alpha') \geq \ell time(\alpha) - (T_g + 3d)$.

Since $j \in \mathit{block}(k, b, \mathit{aug\text{-}ring}(\alpha'))$ and $s'.\mathit{fingers}_k$ contains a finger for j, Lemma 6.17 implies that $j \in \mathit{block}(k, b, s'.\mathit{local\text{-}ring}_k)$. Therefore, this finger for j gets included in the block sent by k in the lookup-comp message.

Upon receipt of this message, $\mathit{fingers}_i$ contains a finger for j with $\mathit{exptime} \geq \mathit{now} + T_e - (3T_g + 7d)$. Then at the end of α, at most time $T_g + 2d$ later, $\mathit{fingers}_i$ contains a finger for j with $\mathit{exptime} \geq s.\mathit{now} + T_e - (4T_g + 9d)$. Since $T_e > 4T_g + 9d$, this implies $\mathit{exptime} > s.\mathit{now}$, as needed.

2. For Part 2, suppose that $\mathit{status}_i =$ joining and a receive(lookup-comp)$_{*,i}$ event for target i occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in \mathit{block}(i, b, \mathit{aug\text{-}ring}(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 3d)$, and fail$_j$ does not occur in α. We must show that $\mathit{fingers}_i$ contains a finger for j with $\mathit{exptime} > \mathit{now} + T_e - (2T_g + 3d)$. Without loss of generality, assume that $j \in \mathit{succset}(i, b, \mathit{aug\text{-}ring}(\alpha))$.

   We first claim that there exists $k \in \mathit{PId}$ such that $k \in \mathit{succset}(i, c - 2\mathit{failbd}, R) - \mathit{succset}(i, 2\mathit{failbd}, R)$, join-ack$_k$ occurs at a time $< \ell time(\alpha) - (4T_g + 10d)$, and fail$_k$ does not occur in α. This follows from Lemma 6.15, applied with $e_1 = e_2 = 2\mathit{joinbd}$ and $e_3 = 5$, using the assumption that $c > 5\mathit{joinbd} + 4\mathit{failbd}$.

   Now we claim that $j \in \mathit{block}(k, b - 2\mathit{failbd}, \mathit{aug\text{-}ring}(\alpha))$. We know that $k \in \mathit{succset}(i, c - \mathit{failbd}, \mathit{aug\text{-}ring}(\alpha)))$. Also, since $k \notin \mathit{succset}(i, 2\mathit{failbd}, R)$, we have that $k \notin \mathit{succset}(i, 2\mathit{failbd}, \mathit{aug\text{-}ring}(\alpha))$. Also, by assumption, $j \in \mathit{succset}(i, b, \mathit{aug\text{-}ring}(\alpha))$. Lemma 6.14, Part 2, applied with $e_1 = c - \mathit{failbd}$, $e_2 = b$ and $e_3 = 2\mathit{failbd}$, then implies that $j \in \mathit{block}(k, b - 2\mathit{failbd}, \mathit{aug\text{-}ring}(\alpha))$, as claimed.

   Process i performs a join-ping at some time in the left-closed, right-open interval $[\ell time(\alpha) - (T_g + 2d), \ell time(\alpha) - 2d)$, and i receives responses for all ping messages generated by that join-ping whose destinations do not fail. Let α' be the prefix of α ending just before the join-ping$_i$, $s' = \ell state(\alpha')$, and $R' = \mathit{global\text{-}ring}(\alpha')$.

   We claim that $s'.\mathit{fingers}_i$ contains a finger for k. Since the time of the join-ack$_k$ is $< \ell time(\alpha) - (4T_g + 10d)$, it is also $< \ell time(\alpha') - (3T_g + 8d)$. Since $k \in \mathit{succset}(i, c - 2\mathit{failbd}, R)$, Lemma 6.16 implies that $k \in \mathit{succset}(i, c - \mathit{failbd}, \mathit{aug\text{-}ring}(\alpha)$. Therefore, by Lemma 6.19, $k \in \mathit{succset}(i, c - \mathit{failbd}, R')$. Therefore, by Lemma 6.16, $k \in \mathit{succset}(i, c, \mathit{aug\text{-}ring}(\alpha'))$. Then the inductive hypothesis, Part 1, implies that $s'.\mathit{fingers}_i$ contains a finger for k.

Since $k \in succset(i, c, aug\text{-}ring(\alpha'))$ and $s'.fingers_i$ contains a finger for k, Lemma 6.17 implies that $k \in block(i, c, s'.local\text{-}ring_i)$. Therefore, during the join-ping, i sends a ping message to k. Since k does not fail in α, k responds to the ping message with a block message. Let α'' be the prefix of α ending just before k sends the block message, let $s'' = \ell state(\alpha'')$, and let R'' denote $global\text{-}ring(\alpha'')$.

Since $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$, Lemma 6.19 implies that $j \in block(k, b - 2failbd, R'')$. Then Lemma 6.16 implies that $j \in block(k, b - failbd, aug\text{-}ring(\alpha''))$. Then by inductive hypothesis, Part 5, we know that $s''.fingers_k$ contains a finger for j with $exptime \geq s''.now + T_e - (T_g + d)$.

Since $j \in block(k, b, aug\text{-}ring(\alpha''))$ and $s''.fingers_k$ contains a finger for j, Lemma 6.17 implies that $j \in block(k, b, s''.local\text{-}ring_k)$. Therefore, the finger for j is included in the block sent by k in its block message to i. At most $T_g + 2d$ time elapses from this send until the end of α, which means that $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (2T_g + 3d)$, as needed.

3. For Part 3, suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $t \geq \ell time(\alpha) - (T_g + 2d)$. Suppose also that $j \in block(i, b, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs at a time $< \ell time(\alpha) - (2T_g + 5d)$, and fail$_j$ does not occur in α. We must show that $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (3T_g + 6d)$. Without loss of generality, assume that $j \in succset(i, b, aug\text{-}ring(\alpha))$.

   The argument is similar to that for the previous case, because we argue with respect to pings and block responses near the end of the joining protocol. By Lemma 6.15, there exists $k \in PId$ such that $k \in succset(i, c - 2failbd, R) - succset(i, 2failbd, R)$, join-ack$_k$ occurs at a time $< \ell time(\alpha) - (5T_g + 12d)$, and fail$_k$ does not occur in α. This uses the assumption that $c > 6joinbd + 4failbd$. Then, since $k \in succset(i, c - failbd, aug\text{-}ring(\alpha))$, $k \notin succset(i, 2failbd, aug\text{-}ring(\alpha))$, and $j \in succset(i, b, aug\text{-}ring(\alpha)))$, Lemma 6.14, Part 2, implies that $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$.

   Process i performs a join-ping at some time in the interval $[t - (T_g + 2d), t - 2d)$, and i receives responses for all ping messages generated by that join-ping whose destinations do not fail in α. Let α' be the prefix of α ending just before the join-ping$_i$, $s' = \ell state(\alpha')$, and $R' = global\text{-}ring(\alpha')$.

   We claim that $s'.fingers_i$ contains a finger for k. Since the time of the join-ack$_k$ is $< \ell time(\alpha) - (5T_g + 12d)$, it is also $< \ell time(\alpha') - (3T_g + 8d)$. Since k is in $succset(i, c - 2failbd, R)$, Lemma 6.16 implies that $k \in succset(i, c - failbd, aug\text{-}ring(\alpha)$. Therefore, by Lemma 6.19, $k \in succset(i, c - failbd, R')$. (This uses the assumption that $T_e > 2T_g + 4d$.) Therefore, by Lemma 6.16, $k \in succset(i, c, aug\text{-}ring(\alpha'))$. Then the inductive hypothesis, Parts 1 and 2, imply that $s'.fingers_i$ contains a finger for k.

   Since $k \in succset(i, c, aug\text{-}ring(\alpha'))$ and $s'.fingers_i$ contains a finger for k, Lemma 6.17 implies that $k \in block(i, c, s'.local\text{-}ring_i)$. Therefore, during the join-ping, i sends a ping message to k. Since k does not fail in α, k responds to the ping message with a block message. Let α'' be the prefix of α ending just before k sends the block message, let $s'' = \ell state(\alpha'')$, and let R'' denote $global\text{-}ring(\alpha'')$.

   Since $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$, Lemma 6.19 implies that $j \in block(k, b - 2failbd, R'')$. Then Lemma 6.16 implies that $j \in block(k, b - failbd, aug\text{-}ring(\alpha''))$. Then by inductive hypothesis, Part 5, we know that $s''.fingers_k$ contains a finger for j with $exptime \geq s''.now + T_e - (T_g + d)$. (Here, we need the fact that the time of the join-ack$_k$ is $< \ell time(\alpha'') - (3T_g + 6d)$, and the time of the join-ack$_j$ is $< \ell time(\alpha'') - d$.)

   Since $j \in block(k, b, aug\text{-}ring(\alpha''))$ and $s''.fingers_k$ contains a finger for j, Lemma 6.17 implies that $j \in block(k, b, s''.local\text{-}ring_i)$. Therefore, the finger for j is included in the block sent by k in its block message to i. At most $2T_g + 4d$ time elapses from this send until the end of α, which means that $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (3T_g + 5d)$, which suffices.

4. For Part 4, suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in block(i, b, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$, and fail$_j$ does not occur in α. We must show that $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (2T_g + 2d)$. Without loss of generality, assume that $j \in succset(i, b, aug\text{-}ring(\alpha))$.

   Lemma 6.15 implies that there exists $k \in PId$ such that $k \in succset(i, c - 2failbd, R) - succset(i, 2failbd, R)$, join-ack$_k$ occurs at a time $< \ell time(\alpha) - (4T_g + 10d)$, and fail$_k$ does not occur in α. This uses the assumption that $c > 5joinbd + 4failbd$.

At some time in the interval $[\ell time(\alpha) - (T_g + d), \ell time(\alpha) - T_g)$, k performs a $neighbor\text{-}refresh_k$ whose messages all arrive by the end of α. Let α' be the prefix of α ending just before this $neighbor\text{-}refresh_k$, $s' = \ell state(\alpha')$, and $R' = global\text{-}ring(\alpha')$.

Since $k \in succset(i, c - 2failbd, R)$, Lemma 6.16 implies that $k \in succset(i, c - failbd, aug\text{-}ring(\alpha))$, Also, since $i \in R$, we know that $i \in predset(k, c - 2failbd, R)$ and so, by Lemma 6.16, $i \in predset(k, c - failbd, aug\text{-}ring(\alpha))$. By Lemma 6.19, $i \in predset(k, c - failbd, R')$. Therefore, by Lemma 6.16, $i \in predset(k, c, aug\text{-}ring(\alpha'))$. Then by inductive hypothesis, Part 5, $s'.fingers_k$ contains a finger for i with $exptime \geq \ell time(\alpha') + T_e - (T_g + d)$.

Next, we claim that $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$. We know that $k \in succset(i, c - failbd, aug\text{-}ring(\alpha))$. Also, since $k \notin succset(i, 2failbd, R)$, we know that $k \notin succset(i, 2failbd, aug\text{-}ring(\alpha))$. Then, since $j \in succset(i, b, aug\text{-}ring(\alpha))$, Lemma 6.14 implies that $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$, as claimed.

Therefore, by Lemma 6.19, $j \in block(k, b - 2failbd, R')$. So by Lemma 6.16, $j \in block(k, b - failbd, aug\text{-}ring(\alpha'))$. Then by inductive hypothesis, Part 5, $s'.fingers_k$ contains a finger for j with $exptime \geq \ell time(\alpha') + T_e - (T_g + d)$. Thus, $s'.fingers_k$ contains fingers for i and j, both with $exptime \geq \ell time(\alpha') + T_e - (T_g + d)$.

Since $i \in block(k, b, aug\text{-}ring(\alpha'))$ and $s'.fingers_k$ contains a finger for i, Lemma 6.17 implies that $i \in block(k, b, s'.local\text{-}ring_k)$. Therefore, i is among the targets of the block message sent by k during the $neighbor\text{-}refresh_k$.

Also, since $j \in block(k, b, aug\text{-}ring(\alpha'))$ and $s'.fingers_k$ contains a finger for j, Lemma 6.17 implies that $j \in block(k, b, s'.local\text{-}ring_k)$. Therefore, the finger for j is included in the block sent by k in its block message to i. When the finger is sent, it has $exptime \geq s'.now + T_e - (T_g + d)$. Therefore, at the end of α, which is at most time $T_g + d$ later, $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (2T_g + 2d)$, as needed.

5. For Part 5, suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $< \ell time(\alpha) - (3T_g + 6d)$. Also suppose that $j \in block(i, b - failbd, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - d$, and fail$_j$ does not occur in α. We must show that $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (T_g + d)$. Without loss of generality, assume that $j \in succset(i, b - failbd, aug\text{-}ring(\alpha))$. Let t denote the time of the join-ack$_j$. We consider two cases:

   (a) $t < \ell time(\alpha) - (2T_g + 3d)$.

   Lemma 6.15 implies that there exists $k \in PId$ such that $k \in succset(i, c - 2failbd, R) - succset(i, 2failbd, R)$, join-ack$_k$ occurs at a time $< \ell time(\alpha) - (5T_g + 8d)$, and fail$_k$ occurs in α. Then (as in the argument for Part 2), Lemma 6.14, Part 2, implies that $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$. Therefore, $k \in block(j, b - 2failbd, aug\text{-}ring(\alpha))$.

   Then we claim that k performs a $neighbor\text{-}refresh$ sometime in the interval $[\ell time(\alpha) - (2T_g + 2d), \ell time(\alpha) - (T_g + 2d))$. Let α' be the prefix of α ending just before this $neighbor\text{-}refresh_k$, let $s' = \ell state(\alpha')$, and let $R' = global\text{-}ring(\alpha')$.

   Since $j \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$, Lemma 6.19 implies that $j \in block(k, b - 2failbd, R')$, and so by Lemma 6.16, $j \in block(k, b - failbd, aug\text{-}ring(\alpha'))$. Also, since $k \in block(i, c - 2failbd, R)$, we have that $i \in block(k, c - 2failbd, R)$, so by Lemma 6.16, $i \in block(k, c - failbd, aug\text{-}ring(\alpha))$, so by Lemma 6.19, $i \in block(k, c - failbd, R')$, so again by Lemma 6.16, $i \in block(k, c, aug\text{-}ring(\alpha'))$, so $i \in block(k, b - failbd, aug\text{-}ring(\alpha'))$.

   Then by inductive hypothesis, Part 5, $s'.fingers_k$ contains a finger for each of i and j, both with $exptime \geq s'.now + T_e - (T_g + d)$. Since $j \in block(k, b, aug\text{-}ring(\alpha'))$ and $s'.fingers_k$ contains a finger for j, Lemma 6.17 implies that $j \in block(k, b, s'.local\text{-}ring_k)$. Therefore, j is among the targets of the block message sent by k during the $neighbor\text{-}refresh_k$. Also, since $i \in block(k, b, aug\text{-}ring(\alpha'))$ and $s'.fingers_k$ contains a finger for i, Lemma 6.17 implies that $i \in block(k, b, s'.local\text{-}ring_k)$. Therefore, the finger for i is included in the block sent by k in its block message to j. When the finger is sent, it has $exptime \geq s'.now + T_e - (T_g + d)$.

   This block message arrives at j at a time $< \ell time(\alpha) - (T_g + d)$. Then sometime in the interval $[\ell time(\alpha) - (T_g + d), \ell time(\alpha) - T_g)$, j performs a neighbor-refresh$_j$. Let α'' be the prefix of α ending just before this $neighbor\text{-}refresh_j$, let $s'' = \ell state(\alpha'')$, and let $R'' = global\text{-}ring(\alpha'')$.

   Since $j \in block(i, b - failbd, aug\text{-}ring(\alpha))$, we have, by Lemma 6.19, that $i \in block(j, b - failbd, global\text{-}ring(\alpha''))$. So by Lemma 6.16, $i \in block(j, b, aug\text{-}ring(\alpha''))$. Also, $s''.fingers_j$ contains a finger for i, because the finger for j that arrives in the block message from k has not had time to expire. Then Lemma 6.17 implies that $i \in block(j, b, s''.local\text{-}ring_j)$. Therefore, i is among the targets of the block message sent by j during this neighbor-refresh$_j$.

This block message contains a finger for j, with $exptime = s''.now + T_e$. Therefore, at the end of α, at most time $T_g + d$ later, $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (T_g + d)$, as needed.

(b) $t \geq \ell time(\alpha) - (2T_g + 3d)$.

Then the time between the join-ack$_i$ and join-ack$_j$ is $> T_g + 3d$.

Lemma 6.15 implies that there exists $k \in PId$ such that $k \in predset(j, c - 2failbd, R) - predset(j, 2failbd, R)$, join-ack$_k$ occurs at a time $< \ell time(\alpha) - (6T_g + 13d)$, and fail$_k$ occurs in α. This uses the assumption that $c > 7joinbd + 4failbd$.

Now we claim that $i \in block(k, b - 3failbd, aug\text{-}ring(\alpha))$. Since $k \in predset(j, c - 2failbd, R)$, Lemma 6.16 implies that $k \in predset(j, c - failbd, aug\text{-}ring(\alpha))$. Since $k \notin predset(j, 2failbd, R)$, we have that $k \notin predset(j, 2failbd, aug\text{-}ring(\alpha))$. Since $j \in block(i, b - failbd, aug\text{-}ring(\alpha))$, Lemma 6.14, Part 2, implies that $i \in block(k, b - 3failbd, aug\text{-}ring(\alpha))$, as claimed.

Process j performs a join-ping at some time in the interval $[t - (T_g + 2d), t - T_g)$, and j receives responses for all ping messages generated by that join-ping whose destinations do not fail, strictly before time t. Let α' be the prefix of α ending just before this join-ping$_j$, $s' = \ell state(\alpha')$, and $R' = global\text{-}ring(\alpha')$.

We claim that $s'.fingers_j$ contains a finger for k. Since $k \in block(j, c - failbd, aug\text{-}ring(\alpha))$, Lemma 6.19 implies that $k \in block(j, c - failbd, R')$, and so by Lemma 6.16, $k \in block(j, c, aug\text{-}ring(\alpha'))$. Then by inductive hypothesis, Parts 1 and 2, $s'.fingers_j$ contains a finger for k.

Since $k \in block(j, c, aug\text{-}ring(\alpha'))$ and $s'.fingers_j$ contains a finger for k, Lemma 6.17 implies that $k \in block(j, c, s'.local\text{-}ring_j)$. Therefore, during the join-ping, j sends a ping message to k. Since k does not fail, it responds with a block message. Let α'' be the prefix of α ending just before k sends this block message, let $s'' = \ell state(\alpha'')$, and $R'' = global\text{-}ring(\alpha''))$.

Since $i \in block(k, b - 2failbd, aug\text{-}ring(\alpha))$, Lemma 6.19 implies that $i \in block(k, b - 2failbd, R'')$. Then Lemma 6.16 implies that $i \in block(k, b - failbd, aug\text{-}ring(\alpha''))$. Then by inductive hypothesis, Part 5, we know that $s''.fingers_k$ contains a finger for i with $exptime \geq now + T_e - (T_g + d)$.

Since $i \in block(k, b, aug\text{-}ring(\alpha''))$ and $s''.fingers_k$ contains a finger for i, Lemma 6.17 implies that $i \in block(k, b, s''.local\text{-}ring_k)$. Therefore, the finger for i is included in the block sent by k in its block message to j. This finger is recorded by j, and persists until the end of α.

Immediately after the join-ack$_j$, and at intervals of T_g thereafter, process j performs a neighbor-refresh$_j$, in which it sends a block message containing a finger for itself with $exptime = T_e$.

We claim that i is included in the set of targets of each such block message. This is because $i \in block(j, b - failbd, aug\text{-}ring(\alpha))$, so by Lemma 6.16, $i \in block(j, b, aug\text{-}ring)$ at each point after the join-ack$_j$. Then Lemma 6.17 implies that $i \in block(j, b, local\text{-}ring_j)$ at each point after the join-ack$_j$, which implies that i is included in the set of targets of each such block message.

Some such message must arrive at i that is sent by j at a time $\geq \ell time(\alpha) - (T_g + d)$. Therefore, $s.fingers_i$ contains a finger for j with $exptime \geq s.now + T_e - (T_g + d)$, as needed.

□

6.5 Maintaining the Chords

We state a lemma analogous to the main lemma of the previous section, Lemma 6.20, but for neighbors of each particular chord position x rather than neighbors of the node i itself.

The statements of Part 1, 2, and 3 are entirely analogous to those in Lemma 6.20. However, in Part 4, the fact that i uses chord-pings instead of neighbor-refreshes to keep up-to-date with respect to x after the join-ack$_i$ changes the bound slightly. Part 5, which describes situations where i obtains first-hand knowledge of j directly from j, gets weakened considerably. This is because we have no phenomenon analogous to that of the prior case 5(b), where j informs i directly about its existence immediately after the join. So, the new Part 5 talks only about those j that are so close to the chord position that i pings j directly during its chord-pings. Since i pings only the apparent c-block of x, this involves only those j that are in this tiny neighborhood.

The proof is also different in some interesting ways. Rather than relying on the inductive hypotheses as before, we rely on the earlier lemma about neighborhoods, Lemma 6.20. That is because the relevant information arrives from neighbors of the chord position x.

Lemma 6.21. *Let α be a good finite execution, $s = \ell state(\alpha)$. Let i be a process that does not fail in α. Let $k \in \mathbb{N}, 0 \le k \le n-1$, and $x = PToX(i) + 2^k$. Then:*

1. *Suppose that $s.status_i =$ joining and a receive(lookup-comp)$_{*,i}$ event for target x occurs in α at a time $\ge \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in block(x, c, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (3T_g + 8d)$, and fail$_j$ does not occur in α.*
   Then $fingers_i$ contains a finger for j with $exptime > now$.

2. *Suppose that $status_i =$ joining and a receive(lookup-comp)$_{*,i}$ event for target x occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in block(x, b, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 3d)$, and fail$_j$ does not occur in α.*
   Then $fingers_i$ contains a finger for j with $exptime > now + T_e - (2T_g + 3d)$.

3. *Suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $\ge \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in block(x, b, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (2T_g + 5d)$, and fail$_j$ does not occur in α.*
   Then $s.fingers_i$ contains a finger for j with $exptime \ge s.now + T_e - (3T_g + 6d)$.

4. *Suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $< \ell time(\alpha) - (T_g + 2d)$. Suppose that $j \in block(x, b, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (T_g + 3d)$, and fail$_j$ does not occur in α.*
   Then $s.fingers_i$ contains a finger for j with $exptime \ge s.now + T_e - (2T_g + 3d)$.

5. *Suppose that $s.status_i =$ active and a join-ack$_i$ occurs in α at a time $< \ell time(\alpha) - (2T_g + 4d)$. Suppose that $j \in block(x, c - failbd, aug\text{-}ring(\alpha))$, join-ack$_j$ occurs in α at a time $< \ell time(\alpha) - (2T_g + 5d)$, and fail$_j$ does not occur in α.*
   Then $s.fingers_i$ contains a finger for j with $exptime \ge s.now + T_e - (T_g + 2d)$.

Proof. Parts 1, 2, and 3, are proved similarly to before, but instead of inductive hypotheses, they use the relevant parts of Lemma 6.20.

For Part 4, we rely on the chord-ping mechanism. And again, the relevant parts of Lemma 6.20 rather than inductive hypotheses.

For Part 5, we use Part 4 to conclude that i learns about j by time $\ell time(\alpha) - (T_g + 2d)$, and then rely on the chord-ping mechanism. The key is that in this last chord-ping, i communicates directly with (pings) j.

□

6.6 Correctness of Lookup Results

Theorem 6.22. *Every good execution α satisfies $2T_g + 6d$-lookup-correctness.*

Proof. (Sketch:) Let α' be a prefix of α ending just before a lookup-ack$(H)_i$ event, which is a response to a prior lookup$(x)_i$. Let $s' = \ell state(\alpha')$ and $R' = global\text{-}ring(\alpha')$.

It suffices to produce a ring R such that $R \subseteq aug\text{-}ring(\alpha')$, R contains every XId in R' except possibly for those j such that join-ack$_j$ occurs in α' at a time $\ge \ell time(\alpha') - (2T_g + 6d)$, and $H = ppredset(x, c, R)$.

Define the ring R to be the union $S \cup T$, where:

- S is the set of all $PToX(j) \in R'$ such that join-ack$_j$ occurs at a time $< \ell time(\alpha') - (2T_g + 6d)$.
- T is that set of all $XIds$ in $s'.fingers_i$.

We show that R satisfies the three properties.

The first property is immediate, because all $XIds$ in $s'.fingers_i$ are in $aug\text{-}ring(\alpha')$. The second property is also immediate, because $S \subseteq R$. For the third property, the code for lookup-ack$(H)_i$ implies that $H = ppredset(x, c, s'.local\text{-}ring_i)$. We need to show that $H = ppredset(x, c, R)$. Since $local\text{-}ring_i$ is the set of $XIds$ in $fingers_i$, and that set is a subset of R, it is enough to show that every $j \in ppredset(x, c, R)$ is also in $s'.fingers_i$.

So, fix $j \in ppredset(x, c, R)$. If $j \in T$ then we are done so assume that $j \in S$. Thus, $j \in R'$ and join-ack$_j$ occurs at a time $< t - (2T_g + 6d)$. Since $j \in ppredset(x, c, R)$, we have that $j \in ppredset(x, c + 3joinbd, R')$.

The lookup-ack$(H)_i$ event follows the receipt by i of a lookup-comp message, with no intervening time passage. Let k be the sender of this lookup-comp message. Then k sent this message at some time $\geq \ell time(\alpha') - d$. Let α'' be the prefix of α ending just before k composed this message, $s'' = \ell state(\alpha'')$, and $R'' = global\text{-}ring(\alpha'')$.

We claim that $k \in ppredset(x, c + 3joinbd, R'')$; the argument is like one in Lemma 6.20, Part 1.

Since $j \in ppredset(x, c + 3joinbd, R')$, it follows that $j \in ppredset(x, c + 3joinbd + failbd, R'')$. Since $j \in ppredset(x, c + 3joinbd + failbd, R'')$ and $k \in ppredset(x, c + 3joinbd, R'')$, it follows that $j \in block(k, c + 3joinbd + failbd, R'')$. Since $b \geq c + 3joinbd + 2failbd$, we have that $j \in block(k, b - failbd, R'')$. Therefore, $j \in block(k, b, aug\text{-}ring(\alpha''))$.

By Lemma 6.20, Parts 3 and 4, $s''.fingers_k$ contains a finger for j with $exptime \geq s''.now + T_e - (3T_g + 6d)$. This finger for j gets included in the block sent by k in the lookup-comp message. After i receives this message, $fingers_i$ contains a finger for j with $exptime \geq now + T_e - (3T_g + 7d)$. Then, since $T_e > 3T_g + 7d$, $s'.fingers_i$ contains a finger for j. This is what we needed to show. □

6.7 Latency Bounds

6.7.1 Latency of a request

Theorem 6.23. *Suppose that α is a good execution, α' a finite prefix of α containing at least $2c + 1$* join-ack *events. Suppose that:*

1. *The final step of α' is a* lookup$_i$ *step in which i initiates request r, with target x.*
2. *No other requests (on behalf of joins, client lookups, or stabilizes) are active at any time $\geq \ell time(\alpha') - T_e$.*

Then request r terminates with a receive(lookup-comp) *step, at a time that is $\leq \ell time(\alpha') + 4(\log N + 1)d$.*

Proof. (Sketch:) We first claim that, at any point during the lookup, for any process $\neq i$ in the ring, the known predecessors of the target x are "bunched together" in at most two c-blocks in the actual global ring. One of these is the block of actual predecessors of x in the ring, and the other may be anywhere else.

Claim 6.24. *At any point in α after α', and for any $j \neq i$, all processes in $ppredset(x, c, local\text{-}ring_j)$ that have not failed lie within two c-blocks of consecutive processes in global-ring: $ppredset(x, c, global\text{-}ring)$ and one other c-block.*

Proof. Everyone except i keeps only its neighborhood and chord fingers, as specified by the underlying infrastructure. These have the needed property. (Two blocks can arise if the target x is in the middle of one of j's blocks.) □

Claim 6.25. *At any point in α after α', and before a $actreceive$(*lookup-comp*)$_i$ event, all processes in $ppredset(x, c - 4failbd, local\text{-}ring_i)$ that have not failed lie within two c-blocks of consecutive processes in global-ring: $ppredset(x, c, global\text{-}ring)$ and one other c-block.*

Proof. (Sketch:) This is more complicated than the previous claim, because process i acquires fingers from other nodes' tables in the course of the lookup.

The ways in which process i acquires new fingers are somewhat constrained: by normal neighborhood and chord refreshing, by receiving a lookup-resp message or by receiving a lookup-comp message. We rule out the last case by assumption—we are considering only what happens before the first receive(lookup-comp)$_i$ happens.

Thus, whenever i acquires new fingers, it acquires an entire block of size at least c from some other node, which by the previous claim is included in only two c-blocks in the actual global ring at the time the block was sent, one of these blocks being $ppredset(x, c, global\text{-}ring)$.

Since at most $failbd$ of each of these blocks could have failed before the block was sent, and at most another $failbd$ from each of these blocks could fail after the send and up to the point of reference, it must be that at least $c - 4failbd$ of the newly-arrived fingers do not fail by the point of reference and lie within two c blocks in $global\text{-}ring$, with one of these blocks being $ppredset(x, c, global\text{-}ring)$.

But this doesn't quite tell us that all processes in $ppredset(x, c-4failbd, local\text{-}ring_i)$ that have not failed lie within these two c-blocks of consecutive processes in $global\text{-}ring$. For this, we have to use the fact that the blocks in $fingers_i$ that are closest to x don't "degrade" by having too many processes fail. The reason this doesn't happen is that i keeps moving the algorithm along—pinging "enough" nodes among its closest predecessors for x, and receiving responses from many of them, which provide information about blocks that are still closer to x. □

Now the key claim describes how the "distance" to the destination x is halved every time $4d$, until near the end of the lookup:

Claim 6.26. *Let e be a power of two, $e \leq N$.*
Suppose that, at some point during the lookup, the clockwise distance from $pred(x, c - 4failbd, local\text{-}ring_i)$ to x (in the identifier space) is $\leq e$.
Then by time $4d$ later, at least one of the following holds:

1. *The lookup ends (with the receipt of a* lookup-comp *message).*

2. *$fingers_i$ contains at least $c - 2failbd$ of the members of $ppredset(x, c, global\text{-}ring)$.*

3. *The clockwise distance from $pred(x, c - 4failbd, local\text{-}ring_i)$ to x is $\leq e/2$.*

Proof. (of Claim:) Assume that the lookup doesn't end within time $4d$, that is, Case 1 doesn't hold. Then within time $2d$, process i performs a new join-ping, which results, within an additional time $2d$, in a response from one of the processes corresponding to the XIds in the assumed $ppredset(x, c-4failbd, local\text{-}ring_i)$. (The fact that one responds depends on the fact that not all of these processes can have failed recently or fail during the ping-response exchange. This in turn relies on our assumed bound on failure rate, and the assumption that they are all within two c-blocks in the global ring.)

Let j be such a responding process. If $PToX(j) + 1 = x$, that is, x is the immediate successor of j in the XId space, then j sends a lookup-comp message, contradicting the fact that Case 1 doesn't hold. So, we may assume that x is not the immediate successor of j in the XId space.

Then choose k to be the largest natural number such that $PToX(j) + 2^k \in (PToX(j), x)$, that is, the largest power-of-two successor of j that does not reach x.

The response from j to i contains a set F of fingers representing j's c best predecessors for x at the time j sends its response. There are two cases:

1. F contains only elements in the open interval $(PToX(j) + 2^k, x)$. That is, only elements after the given largest power-of-two successor of j.

   In this case, after i receives the message, the clockwise distance from $pred(x, c - 2failbd, local\text{-}ring_i)$ to x is $\leq e/2$, which suffices to satisfy Case 3.

2. F contains at least one element that is not in the open interval $(PToX(j) + 2^k, x)$.

   Lemma 6.21 implies that, when j sends the lookup-response message, $fingers_j$ contains entries for all elements of $block(j + 2^k, b, augmented\text{-}ring)$ that have not failed. Since the set F contains at least one element that is not in the open interval $(j + 2^k, x)$, we claim that F contains actual predecessors of x in the global ring, specifically, F contains at least $c - failbd$ of the members of $ppredset(x, c, global\text{-}ring)$ at the time j sends the message. (Up to $failbd$ of the fingers in F could have already failed at the time of the send.) Just after i receives the message, $fingers_i$ contains at least $c - 2failbd$ of the members of $ppredset(x, c, global\text{-}ring)$. This yields Case 2.

□

To complete the proof, we use the last claim repeatedly, as long as Case 3 holds. Since we cannot keep halving forever, eventually, either Case 1 or Case 2 arises. If Case 1 arises first, then we are done. On the other hand, if Case 2 arises first, then within only one more ping round, i receives a lookup-comp message, so again we are done.

7 Appendix B: Using Nondeterministic Assumptions

As described in Section 3) and Appendix B our analysis in this paper is based on deterministic assumptions. In general, we assume that there are at most v relevant events that occur in an "arc" of the ring containing at most r processes during a time interval Δ.

These assumptions however are not realistic for many distributed environments. In practice join and failure events are modeled by probability distribution functions (e.g, Poisson) which makes it impossible to put a deterministic bound of the number of such events during an interval of time Δ.

To establish a relationship between the more realistic probabilistic assumptions and the deterministic assumptions next we compute the mean time T_f between two violations of the deterministic bounds under the probabilistic assumptions. In other words, T_f represents the expected time for which a MultiChord will remain in the quasi-ideal state.

For tractability, we assume a system in which processes join according to a Poisson process with arrival rate λ_a and that the process lifetimes are exponentially distributed with a mean of l. Assuming that the MultiChord ring is in steady state we have $l = N/\lambda_a$, i.e., the rate of joins is equal to the rate of failures or leaves. Thus, the rate of changes is $\lambda = 2\lambda_a$.

Next, we bound the probability that the deterministic assumption–that no more than v relevant events occur during a time interval Δ in an arc of the ring of r processes—is violated.

The average number of events that occur in a given arc of the ring consisting of r processes during an interval of time Δ is

$$\mu = \Delta \cdot \lambda(r/N), \tag{2}$$

where $\Delta \cdot \lambda$ represents the average number of events that occur in the entire system during a time interval Δ, and (r/N) represents the fraction of these events that occur during that portion of the ring.

Because events are generated from a Poisson distribution we can apply the Chernoff bound:

$$Pr(X > (1+\delta)\mu) < e^{-\mu\delta^2/4}, \tag{3}$$

where $Pr(X > (1+\delta)\mu)$ represents the probability that no more than $(1+\delta)\mu$ events occur in a given arc of r processes during a time interval Δ. Taking $v = (1+\delta)\mu$, the probability that the deterministic bound is violated in a given arc of r processors during a time interval Δ is

$$Pr(X > v) < e^{-\frac{(v-\mu)^2}{4\mu}}. \tag{4}$$

The probability $p(\Delta, r)$ that the deterministic bound is violated in *any* arc of r processors during an interval Δ is bounded above by

$$p(\Delta, r) < NPr(X > v) < Ne^{-\frac{(v-\mu)^2}{4\mu}}. \tag{5}$$

Then the mean time T_f between two violations of the deterministic bound is

$$T_f = \frac{\Delta}{p(\Delta, r)} > \frac{\Delta}{N}\ e^{\frac{(v-\mu)^2}{4\mu}} \tag{6}$$

Expanding μ yields

$$T_f > \frac{\Delta}{N}\ e^{\frac{(v-\overline{\lambda}\Delta\cdot r)^2}{4\overline{\lambda}\Delta\cdot r}}. \tag{7}$$

where $\overline{\lambda} = \lambda/N$ represents the normalized rate of change.

Next, let us consider how do deterministic constraints presented in Section 3 map to Ineq. (7). In particular, we consider the following constraints:

$$
\begin{aligned}
T_e &\geq 5T_j \qquad (8)\\
c &> 7joinbd + 4failbd\\
b &\geq 2c + 3joinbd + \max(2joinbd, failbd), \qquad (9)
\end{aligned}
$$

where $failbd$ represents the number of failures in an arc of $b+1$ processes during time T_e, and $joinbd$ represents the number of joins in an arc of $b+1$ processes during time T_j.

Because we assume steady state, the number of failures and joins in an arc of $b+1$ processes is roughly the same during a given time interval. This means that $failbd = joinbdT_e/T_j$. If we take $T_e/T_j = 5$, the last two constraints in Ineqs. (8) become:

$$
\begin{aligned}
c &> 27joinbd \qquad (10)\\
b &\geq 61joinbd
\end{aligned}
$$

during an interval of time T_e, and

$$
\begin{aligned}
c &> 5.4joinbd \qquad (11)\\
b &\geq 12.2joinbd
\end{aligned}
$$

during an interval of time T_j.

Since constraints (11) imply constraints (10) next we consider only constraints (11). Let us take $c = 6joinbd$, $b = 13joind$, values which satisfy both these constraints.

Finally, we take $r = b+1$, $\Delta = T_j$, and $v = 2joinbd$ (the factor of 2 is because v accounts for both joins and failures during the interval T_j). With these values, the expected time before the deterministic constraints are violated (see Ineq (7)) becomes

$$
T_f > \frac{T_j}{N}\ e^{\frac{(c/3-\overline{\lambda}T_j(b+1))^2}{4\overline{\lambda}T_j(b+1)}}, \qquad (12)
$$

where $b \geq 13c/6$.