HAL
open science

A Rigorous Correctness Proof for Pastry

Noran Azmy, Stephan Merz, Christoph Weidenbach

To cite this version:

Noran Azmy, Stephan Merz, Christoph Weidenbach. A Rigorous Correctness Proof for Pastry. Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016, 2016, Linz, Austria. pp.86-101, 10.1007/978-3-319-33600-8_5 . hal-01322342

HAL Id: hal-01322342

https://inria.hal.science/hal-01322342v1

Submitted on 27 May 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Rigorous Correctness Proof for Pastry

Noran Azmy[1,2,3], Stephan Merz[2,3], and Christoph Weidenbach[1]

[1] Max Planck Institute for Informatics
[2] Inria, Villers-lès-Nancy, France
[3] CNRS, Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-lès-Nancy, France

Abstract. Peer-to-peer protocols for maintaining distributed hash tables, such as Pastry or Chord, have become popular for a class of Internet applications. While such protocols promise certain properties concerning correctness and performance, verification attempts using formal methods invariably discover border cases that violate some of those guarantees. Tianxiang Lu reported correctness problems in published versions of Pastry and also developed a model, which he called LuPastry, for which he provided a partial proof of correct delivery assuming no node departures, mechanized in the TLA$^+$ Proof System. Lu's proof is based on certain assumptions that were left unproven. We found counter-examples to several of these assumptions. In this paper, we present a revised model and rigorous proof of correct delivery, which we call LuPastry$^+$. Aside from being the first complete proof, LuPastry$^+$ also improves upon Lu's work by reformulating parts of the specification in such a way that the reasoning complexity is confined to a small part of the proof.

1 Introduction

In a peer-to-peer network, individual nodes – called *peers* – communicate directly with each other and act as both suppliers and users of a given service. One such service that can be implemented by a peer-to-peer protocol is a *distributed hash table* (DHT): a decentralized distributed system that provides a lookup service similar to a hash table, but where the responsibility for storing key-value pairs is divided among the different nodes on the network. Nodes can efficiently retrieve the value associated with a given key by sending a lookup message for that key. Correct routing guarantees that lookup messages arrive at the node responsible for storing the key-value pair. Pastry [7] and Chord [8] are well-known examples of protocols implementing DHTs.

Using Alloy to formally model and verify Chord, Zave [10] showed that the join protocol is correct provided that no node leaves the network, but that the full version of the protocol may not maintain the claimed invariants. In [11], she presented a version of Chord with a partially-automated proof of correctness. Lu discovered similar correctness problems for Pastry using the TLA$^+$ proof assistant [6]. Some other work has been done in this area that does not rely on mechanized verification, *i.e.*, model checking or theorem proving. Borgström et al. [3] used CCS to verify correctness of the DKS look-up protocol, assuming

that the network remains stable (i.e., nodes neither join nor leave). Bakhshi et al. [2] used process algebra to formally verify the stabilization process of Chord.

Like Lu, we are interested in the formal verification of Pastry using the TLA$^+$ proof system, w.r.t. the safety property *correct delivery* [6]: *At any point in time, there is at most one node that answers a lookup request for a key, and this node must be the closest live node to that key.*

TLA$^+$ [5] is a formal specification language that mainly targets concurrent and distributed systems. It is based on untyped Zermelo-Fraenkel set theory for specifying data structures, and the Temporal Logic of Actions, a variant of linear temporal logic, for describing system behavior. Systems are specified as state machines over a tuple of state variables by defining a state predicate *Init* and a transition predicate *Next* that constrain the possible initial states and the next-state relation. Transition predicates (also called *actions*) are first-order formulas that contain unprimed and primed state variables for denoting the values of the variables in the state before and after the transition. Validation of TLA$^+$ specifications is mechanized by TLC [9], an explicit-state model checker for finite instances of TLA$^+$ specifications, and formal verification by TLAPS, the TLA$^+$ Proof System [4]. TLAPS is based on a hierarchical proof language; the user writes a TLA$^+$ proof in the form of a hierarchy of *proof steps*, each of which is interpreted by the *proof manager*, which generates corresponding proof obligations and passes them to automatic back-end provers, including Zenon, Isabelle/TLA$^+$, and SMT solvers. Larger steps that cannot be proven directly by any of the back-end provers can be broken further into sub-steps. Because the language is untyped, part of the proof effort consists in proving a typing invariant that expresses the shapes of functions and operators.

Using model checking and theorem proving, Lu discovered several problems in the original Pastry protocol and presented a variant of the protocol, called LuPastry, for which he verified correct delivery under the strong assumption that nodes never fail (*i.e.*, leave the network). Notably, his Pastry variant enforces that a live node may only facilitate the joining of one new node at a time. Lu's proof reduces correct delivery to a set of around 50 claimed invariants, which are proven with the help of TLAPS. As such, LuPastry represents a major effort in the area of computer-aided formal verification of distributed algorithms. Still, the proof relies on many unproven assumptions relating to arithmetic and to protocol-specific data structures. Upon examining these assumptions, we discovered counter-examples to several of them. While we were able to prove stronger variants of many assumptions, this was not possible for others. In fact, we were able to find a counter-example to one of Lu's claimed main invariants, for which the TLA$^+$ proof was only possible because of incorrect assumptions. Fixing these problems led to a redesign of the overall proof.

Our contribution in this paper is LuPastry$^+$: a revised specification and complete proof of correct delivery for LuPastry. While the essentials of the actual protocol are not changed, we improve on the LuPastry specification by fixing some unhandled border cases and introducing abstractions in some operator definitions that make the specification more modular and confine the reasoning

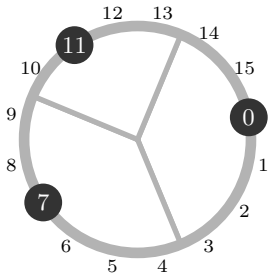


Fig. 1: A Pastry ring of size 16 with three live nodes 0, 7 and 11. The nodes should divide key coverage among them, as indicated by the separators.

complexity to a small part of the proof. The new abstractions typically lead to a significant reduction of the size of higher-level proofs. Moreover, our proof does not rely on unproven assumptions, because all low-level lemmas have been proved using TLAPS.

The paper is organized as follows. Section 2 describes the main aspects of the original LuPastry model, which are also in LuPastry$^+$. We explain our contribution and the structure of the LuPastry$^+$ proof in Section 3. A sketch of the machine-checked proof is given in Section 4. Section 5 summarizes our results and our experience in using TLAPS.

2 The (Lu)Pastry Model

The Pastry network can be visualized as a ring of keys $I \triangleq 0 \mathinner{..} 2^M - 1$ for some positive integer M (see Figure 1). Each live node is assigned a unique key $k \in I$ as an identifier and needs to determine its *coverage*: a contiguous range of keys, including the node's own ID, that the node is responsible for, or *covers*. If a node i covers key k, then i considers itself (1) the proper recipient of all look-up messages addressed to k, and (2) the node responsible for facilitating the joining of any new node with ID k. In the absence of a central server and shared memory, live nodes need to rely on message passing and local information to agree on a proper division of coverage.

Let *ready* nodes be the live nodes that are not in the process of joining the network (they are fully-joined). Ready nodes are of particular interest since only ready nodes may accept look-up messages or facilitate the joining of new nodes. Ideally, the coverage ranges computed by all ready nodes (1) do not overlap, (2) cover the whole range of keys, and (3) are computed based on the smallest absolute distance to the node: if a ready node i covers key k, then k is closer to i in terms of absolute ring distance than it is to any other ready node $j \neq i$, with a rule for breaking ties. These conditions all hold for the ring illustrated in Figure 1. Condition (2) may be temporarily violated when a new node joins but is not yet ready, thus the safety property that we are interested in verifying requires only (1) and (3).

For two nodes x and y, we may be interested in the *clockwise* distance from x to y, or the *absolute* (shortest) distance between x and y.

$$
\begin{array}{l}
\mathit{ClockwiseDistance}(x,\, y) \triangleq \\
\quad \text{IF } y \geq x \text{ THEN } y - x \text{ ELSE } \mathit{RingSize} - x + y \\
\mathit{AbsoluteDistance}(x,\, y) \triangleq \\
\quad \text{LET } d1 \triangleq \mathit{ClockwiseDistance}(x,\, y) \\
\qquad\;\; d2 \triangleq \mathit{ClockwiseDistance}(y,\, x) \\
\quad \text{IN} \quad \text{IF } d1 \leq d2 \text{ THEN } d1 \text{ ELSE } d2
\end{array}
$$

A node i computes its coverage by maintaining a *leaf set*: a set containing what i believes to be its L live neighbor nodes on both the left and right sides, where the positive integer L is a parameter of the specification.[4]

$$
\begin{array}{l}
\mathit{LeafSet} \triangleq \{\, ls \in [\mathit{node} : I,\, \mathit{left} : \text{SUBSET } I,\, \mathit{right} : \text{SUBSET } I] : \\
\quad \wedge\, ls.\mathit{node} \notin ls.\mathit{left} \quad\;\; \wedge\, \mathit{Cardinality}(ls.\mathit{left}) \leq L \\
\quad \wedge\, ls.\mathit{node} \notin ls.\mathit{right} \quad \wedge\, \mathit{Cardinality}(ls.\mathit{right}) \leq L \,\}
\end{array}
$$

The *neighbors* of i are the closest nodes to i in its leaf set ls.

$$
\begin{array}{l}
\mathit{RightNeighbor}(ls) \triangleq \\
\quad \text{IF } ls.\mathit{right} = \{\} \text{ THEN } ls.\mathit{node} \\
\quad \text{ELSE CHOOSE } n \in ls.\mathit{right} : \forall\, p \in ls.\mathit{right} : \\
\qquad \mathit{ClockwiseDistance}(ls.\mathit{node},\, p) \geq \mathit{ClockwiseDistance}(ls.\mathit{node},\, n)
\end{array}
$$

LeftNeighbor is defined analogously. CHOOSE denotes Hilbert's choice operator and will be discussed further in Section 3. Node i considers its coverage range as the interval $[\mathit{LeftCoverage}(ls), \mathit{RightCoverage}(ls)]$, where the key $\mathit{LeftCoverage}(ls)$ is the midpoint between $\mathit{LeftNeighbor}(ls)$ and i, and similarly, $\mathit{RightCoverage}(ls)$ is the midpoint between i and $\mathit{RightNeighbor}(ls)$.

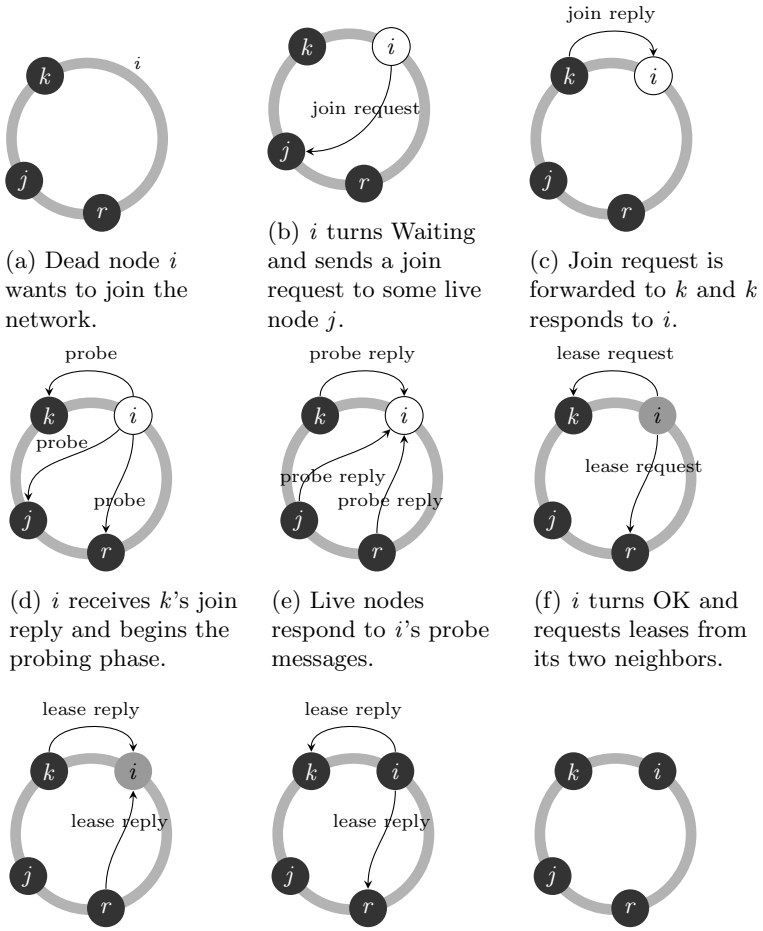
$$
\begin{array}{l}
\mathit{LeftCoverage}(ls) \triangleq \\
\quad \text{IF } \mathit{LeftNeighbor}(ls) = ls.\mathit{node} \text{ THEN } ls.\mathit{node} \\
\quad \text{ELSE } (\mathit{LeftNeighbor}(ls) + \\
\qquad (\mathit{ClockwiseDistance}(\mathit{LeftNeighbor}(ls),\, ls.\mathit{node}) \div 2 + 1))\%\mathit{RingSize} \\
\mathit{RightCoverage}(ls) \triangleq \\
\quad \text{IF } \mathit{RightNeighbor}(ls) = ls.\mathit{node} \\
\quad \text{THEN}(\mathit{RingSize} + ls.\mathit{node} - 1)\%\mathit{RingSize} \\
\quad \text{ELSE } (ls.\mathit{node} + \\
\qquad \mathit{ClockwiseDistance}(ls.\mathit{node},\, \mathit{RightNeighbor}(ls)) \div 2)\%\mathit{RingSize}
\end{array}
$$

In Figure 1, assuming up-to-date leaf sets, node 0's left and right neighbors are 11 and 7, respectively. Therefore, its coverage is the interval $[14, 3]$ (*i.e.*, the set of keys $\{14, 15, 0, 1, 2, 3\}$).

The Pastry model presented here is called LuPastry [6], in which two main restrictions are introduced to the dynamic behavior of the protocol: (1) nodes are assumed to never fail (leave the network), and (2) a ready node may facilitate the joining of at most one new node at a time.

The main dynamic aspect of the protocol is the join process, explained as follows (see also Figure 2). In LuPastry, each node is either *Dead* (not shown),

[4] Nodes also maintain routing tables for the purpose of efficient message routing, but these are irrelevant to our discussion.



(a) Dead node i wants to join the network.

(b) i turns Waiting and sends a join request to some live node j.

(c) Join request is forwarded to k and k responds to i.

(d) i receives k's join reply and begins the probing phase.

(e) Live nodes respond to i's probe messages.

(f) i turns OK and requests leases from its two neighbors.

(g) i's neighbors grant i the requested leases.

(h) i grants leases to its neighbors in its turn, turns Ready.

(i) i is Ready; i and k may accept new join requests.

Fig. 2: The Pastry Join Protocol

Waiting (white), *OK* (gray) or *Ready* (black). Only Ready nodes facilitate the joining of new nodes into the network. A Dead node i that decides to join the network turns to Waiting and sends a *join request* to a Ready node j that it knows about. The request is forwarded to the Ready node k that covers key i. Node k responds to i's request when it is free for handling a new join request, and communicates to i its own leaf set. Node i receives k's reply and, in order to construct its proper leaf set, sends *probe* messages to the nodes in the leaf set received from k. All non-Dead nodes that receive the probe add i to their leaf set if appropriate (*i.e.*, if node i is among the L closest live neighbor nodes), and send a *probe reply* to i with their own leaf set information. This process continues until i has probed all nodes it has heard about and that are close enough to i to be in i's leaf set, then i becomes OK. In order for i to become Ready and eventually serve the IDs closest to i, node i has to exchange *leases* with both its left and right neighbors (one of which must be k). Node i sends out *lease request* messages to both its neighbors. If i's neighbor is Ready or OK, and also considers i to be its neighbor, it grants i the lease in a *lease reply* message. When i has received lease replies from both its neighbors, it switches to Ready, and grants its neighbors leases in turn. When k receives i's lease, it may help other new nodes join the ring.

The global state of the LuPastry network is represented as the tuple *vars* of state variables.[5]

$$vars \triangleq \langle Messages, Status, LeafSets, Probing, Leases, Grants, ToJoin \rangle$$

Messages represents the set of messages currently in transmission. Variables *Status* and *LeafSets* are arrays whose i-th entries are the current status and leaf set of node i. Similarly, $Probing[i]$ is the set of nodes that node i has probed but has not heard back from yet, $Leases[i]$ and $Grants[i]$ are the set of nodes i has acquired leases from, and granted leases to, respectively. Lastly, $ToJoin[i]$ designates the node that is currently joining through i, if any, otherwise $ToJoin[i] = i$.

The TLA$^+$ specifications of the initial state and of the next-state relation are defined by the operators *Init* and *Next* shown in Fig. 3; they use auxiliary operators that represent elementary operations on leaf sets and routing tables, and that define the individual transitions of the Pastry protocol. The constant A is a parameter of the specification designating the nodes that are live (and Ready) initially. Note that, because LuPastry is a distributed system, each action may modify the local variables of at most one node i, the node executing it, besides the set *Messages*. The TLA$^+$ specification of LuPastry is defined as the formula $Spec \triangleq Init \wedge \Box[Next]_{vars}$.

$EmptyLS(i)$ is a leaf set owned by i with no nodes in the right and left sides. $AddToLS(a, ls)$ is the leaf set obtained by adding the set of nodes a to ls. In case of an overflow, the new right (resp., left) leaf set consists of the L closest nodes to i from the right (resp., left); nodes that are farther away are discarded.

[5] For compactness, we omit parts of the specification irrelevant to the discussion. TLA$^+$ functions and operators have been given new names for better readability.

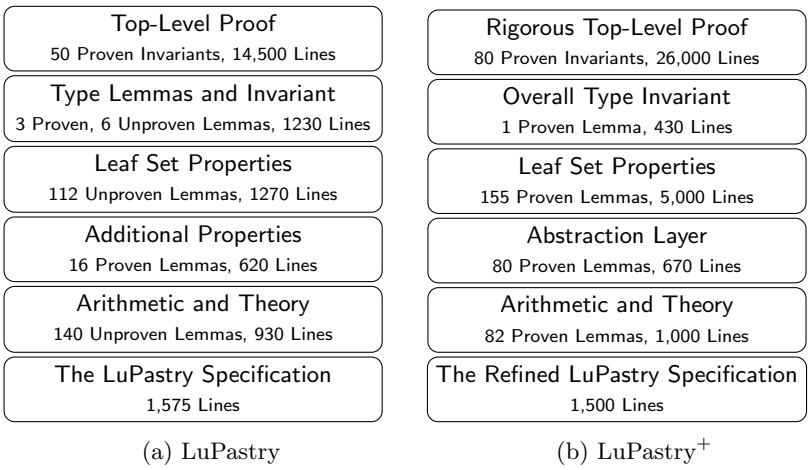
$$
\begin{aligned}
&\mathit{Init} \triangleq \\
&\quad \land\ \mathit{Messages} = \{\} \\
&\quad \land\ \mathit{Status} = [i \in I \mapsto \text{IF } i \in A \text{ THEN } \text{“Ready”} \text{ ELSE } \text{“Dead”}] \\
&\quad \land\ \mathit{ToJoin} = [i \in I \mapsto i] \ \land\ \mathit{Probing} = [i \in I \mapsto \{\}] \\
&\quad \land\ \mathit{Leases} = [i \in I \mapsto \text{IF } i \in A \text{ THEN } A \text{ ELSE } \{i\}] \\
&\quad \land\ \mathit{Grants} = [i \in I \mapsto \text{IF } i \in A \text{ THEN } A \text{ ELSE } \{i\}] \\
&\quad \land\ \mathit{LeafSets} = [i \in I \mapsto \text{IF } i \in A \text{ THEN } \mathit{AddToLS}(A, \mathit{EmptyLS}(i)) \\
&\qquad\qquad\qquad\qquad \text{ELSE } \mathit{EmptyLS}(i)] \\
&\mathit{Next} \triangleq \exists i, j \in I : \\
&\quad \lor\ \mathit{Lookup}(i,j) \qquad\qquad \lor\ \mathit{RouteLookup}(i,j) \\
&\quad \lor\ \mathit{DeliverLookup}(i,j) \qquad \lor\ \mathit{Join}(i,j) \\
&\quad \lor\ \mathit{RouteJoinRequest}(i,j) \qquad \lor\ \mathit{ReceiveJoinRequest}(i) \\
&\quad \lor\ \mathit{ReceiveJoinReply}(i) \qquad \lor\ \mathit{ReceiveProbe}(i) \\
&\quad \lor\ \mathit{ReceiveProbeReply}(i) \qquad \lor\ \mathit{RequestLease}(i) \\
&\quad \lor\ \mathit{ReceiveLeaseRequest}(i) \qquad \lor\ \mathit{ReceiveLeaseReply}(i)
\end{aligned}
$$

Fig. 3: Initial condition and next-state relation specified in TLA$^+$.

Finally, $\mathit{LeafSetContent}(ls) \triangleq \{ls.node\} \cup ls.right \cup ls.left$ is the set of nodes in leaf set ls.

3 The LuPastry$^+$ Model and Proof

Our contribution is illustrated in Figure 4, and can be divided into two parts: (1) changes to the TLA$^+$ specification of LuPastry, and (2) the first complete proof of correctness.



(a) LuPastry (b) LuPastry$^+$

Fig. 4: Structure of the original LuPastry proof versus the rigorous version.

3.1 Changes to the LuPastry Specification

At the bottom layer, we refine LuPastry [6] as follows. In order for the proof to gain in modularity, readability, and simplicity, we introduce additional operators that abstract away from arithmetic calculations and reduce the use of TLA$^+$'s CHOOSE operator, which is difficult for back-end provers to reason about and hence restrains automation. This also makes the specification more concise.

Arithmetic calculations, which mainly involve comparisons between distances between nodes on the ring, appeared so extensively in LuPastry that arithmetic reasoning was frequently needed at the top level of the original proof. For example, typical subformulas $\mathit{ClockwiseDistance}(i, j) \leq \mathit{ClockwiseDistance}(i, k)$ require that the definition of *ClockwiseDistance* be unfolded. Instead, in LuPastry$^+$ we define a predicate $\mathit{ClockwiseArc}(i, j, k)$, which holds if j lies on the clockwise path from i to k.

$$\begin{aligned}&\mathit{ClockwiseArc}(i,\ j,\ k) \triangleq \\ &\quad \mathit{ClockwiseDistance}(i,\ j) \leq \mathit{ClockwiseDistance}(i,\ k)\end{aligned}$$

We then prove once and for all the necessary properties of this relation in TLAPS, using the SMT backend for arithmetical reasoning, so that unfolding of the definition is no longer needed in the top-level proof. The following are some examples of these properties.

$$\begin{aligned}&\text{THEOREM } \mathit{ArcAntiSymmetry} \triangleq \forall\, x,\ y,\ z \in I : \\ &\quad \land\ \mathit{ClockwiseArc}(x,\ y,\ z) \land \mathit{ClockwiseArc}(x,\ z,\ y) \Rightarrow y = z \\ &\quad \land\ \mathit{ClockwiseArc}(x,\ y,\ z) \land \mathit{ClockwiseArc}(y,\ x,\ z) \Rightarrow x = y\end{aligned}$$

$$\begin{aligned}&\text{THEOREM } \mathit{ArcRotation} \triangleq \forall\, x,\ y,\ z \in I : \\ &\quad \land\ x \neq y \land \mathit{ClockwiseArc}(x,\ y,\ z) \Rightarrow \mathit{ClockwiseArc}(y,\ z,\ x) \\ &\quad \land\ y \neq z \land \mathit{ClockwiseArc}(x,\ y,\ z) \Rightarrow \mathit{ClockwiseArc}(z,\ x,\ y)\end{aligned}$$

This abstraction helps automate the proof process since now automatic backends like Zenon or Spass without native support for integer arithmetic are able to solve larger steps.

A related issue is the extensive use of Hilbert's ε-operator for definite choice. The TLA$^+$ expression CHOOSE $x \in S : P(x)$ denotes some fixed but arbitrary element x in set S for which the property P holds, if some such x exists. If P holds for no $x \in S$, as in CHOOSE $x \in \mathit{Nat} : x * 0 = 1$, the result of the CHOOSE expression is not specified.

The LuPastry definition of the operator *RightNeighbor*, shown in the previous section, uses CHOOSE. It is unwieldy to reason about operator *RightNeighbor* by unfolding its definition because we would invariably have to show the existence of a node contained in *ls.right* and whose distance to *ls.node* is minimal among all these nodes. Formally, we need the following lemma:

$$\begin{aligned}&\text{LEMMA } \forall x \in I,\ S \in (\text{SUBSET } I) \setminus \{\{\}\} : \exists y \in S : \forall z \in S : \\ &\qquad \mathit{ClockwiseDistance}(x,\ y) \leq \mathit{ClockwiseDistance}(x,\ z)\end{aligned}$$

In LuPastry$^+$, we perform three changes. First, we redefine *RightNeighbor* to abstract away from the CHOOSE expression. The new term *ClosestFromTheRight*

is itself defined in terms of CHOOSE, but generalizes the previous operator and reuses the operator *ClockwiseArc* introduced above.

$$RightNeighbor(ls) \triangleq ClosestFromTheRight(ls.node,\ ls.right)$$

$$\begin{aligned}&ClosestFromTheRight(x,\ a) \triangleq \\ &\quad \text{IF } a = \{\} \text{ THEN } x \\ &\quad \text{ELSE CHOOSE } y \in a : \forall z \in a : ClockwiseArc(x,\ y,\ z)\end{aligned}$$

Second, since *ClosestFromTheRight* is defined using a CHOOSE expression, we add a lemma that guarantees the existence of a value satisfying the characteristic predicate.

$$\begin{aligned}&\text{LEMMA } choose_ClosestFromTheRight \triangleq \\ &\quad \forall x \in I,\ a \in \text{SUBSET } I : \\ &\qquad a \neq \{\} \Rightarrow \exists y \in a : \forall z \in a : ClockwiseArc(x,\ y,\ z)\end{aligned}$$

Third, we add *type* and *expansion* lemmas that respectively provide type information and the relevant properties of *ClosestFromTheRight*. We introduce similar lemmas for operator *RightNeighbor*.

$$\begin{aligned}&\text{LEMMA } type_ClosestFromTheRight \triangleq \\ &\quad \forall x \in I,\ a \in \text{SUBSET } I : ClosestFromTheRight(x,\ a) \in I\end{aligned}$$

$$\begin{aligned}&\text{LEMMA } def_ClosestFromTheRight \triangleq \\ &\quad \forall x \in I,\ a \in \text{SUBSET } I : \\ &\qquad \wedge\ a = \{\} \Rightarrow ClosestFromTheRight(x,\ a) = x \\ &\qquad \wedge\ a \neq \{\} \Rightarrow ClosestFromTheRight(x,\ a) \in a \\ &\qquad \wedge\ \forall y \in a : ClockwiseArc(x,\ ClosestFromTheRight(x,\ a),\ y)\end{aligned}$$

We illustrate the effect of these abstractions using a simple lemma about adding new nodes to the leaf set data structure, that we prove once with and once without the use of the new operators.

$$\text{LEMMA } \forall ls \in LeafSet,\ a \in \text{SUBSET } I : IsProper(AddToLS(a,\ ls))$$

Basically, the lemma says that the leaf set obtained by adding some new nodes to a leaf set is "proper", where "proper" is defined as follows.

$$\begin{aligned}&IsProper(ls) \triangleq \\ &\quad \wedge\ \forall x \in ls.left \setminus ls.right,\ y \in ls.right : ClockwiseArc(y,\ x,\ ls.node) \\ &\quad \wedge\ \forall x \in ls.right \setminus ls.left,\ y \in ls.left : ClockwiseArc(ls.node,\ x,\ y)\end{aligned}$$

The proof P_1 of this lemma according to the original definition of *IsProper* consists of 23 interactive proof steps that generate 64 proof obligations. With our new abstractions, the new proof P_2 consists of only 12 interactive proof steps (40 proof obligations). This significant difference comes from the fact that the new operators allow back-end provers to succeed directly on some steps in P_2, which have to be broken down into further substeps in the original proof P_1. Already for this simple example we observe a 50% reduction in the number of steps, *i.e.*, user interactions.[6]

[6] Both examples can be found in our proof files in module ProofCorrectness.

Additionally, we fix some corner cases in the original specification. We modify the probing process so that the node does not probe itself. This is clearly unnecessary, and removing it simplifies some parts of the proof. We also add a missing border case to the TLA$^+$ formula $FindNext(i, j)$ that computes the next hop on the route from node i to node j.

3.2 New Proof of Correctness

The upper layers of Figure 4 compare our new proof to Lu's original proof. The original LuPastry proof relied on a large number of unproven assumptions. In attempting to prove these assumptions, we found counter-examples to many of them, such as arithmetic assumptions ignoring border cases. Moreover, several assumptions were not actually used in the proof. For example, Lu's proof relied on 112 unproven assumptions about the leaf set data structure. Upon examining these assumptions, we could prove only 21 directly. We discovered that more than 30 were unused in Lu's proof. The rest of the assumptions were incorrect. Our analysis of Lu's assumptions led us to reformulate those that were needed for the top-level proof. This was possible for all but 6 of the incorrect assumptions. The following assumption, used by Lu as an unproven TLA$^+$ lemma, states that after adding some set of nodes a to a leaf set $ls1$, the right neighbor of the resulting leaf set $ls2$ can only be closer to the leaf set owner i than the original right neighbor of $ls1$.

$$\begin{array}{l}
\textsc{lemma}\ \forall\, ls1,\, ls2 \in LeafSet,\, i \in I,\, a \in \textsc{subset}\ I : \\
\quad i = ls1.node \wedge ls2 = AddToLS(a,\, ls1) \Rightarrow \\
\quad\quad ClockwiseDistance(i,\, RightNeighbor(ls2)) \leq \\
\quad\quad\quad ClockwiseDistance(i,\, RightNeighbor(ls1))
\end{array}$$

This lemma does not hold if the right-hand part of the leaf set is empty (*i.e.*, $ls1.right = \{\}$), because in this case $RightNeighbor(ls1) = i$ and i is closer to itself than to any other node. The lemma was therefore reformulated as follows.

$$\begin{array}{l}
\textsc{lemma}\ \forall\, ls1,\, ls2 \in LeafSet,\, i \in I,\, a \in \textsc{subset}\ I : \\
\quad i = ls1.node \wedge ls1.right \neq \{\} \wedge ls2 = AddToLS(a,\, ls1) \Rightarrow \\
\quad\quad ClockwiseDistance(i,\, RightNeighbor(ls2)) \leq \\
\quad\quad\quad ClockwiseDistance(i,\, RightNeighbor(ls1))
\end{array}$$

Other assumptions had to be eliminated entirely. For example,

$$\begin{array}{l}
\textsc{lemma}\ \forall\, ls \in LeafSet,\, k \in I : \\
\quad LeafSetContent(AddToLS(\{k\},\, ls)) \setminus \{k\} = LeafSetContent(ls)
\end{array}$$

states that the leaf set obtained by adding a node k to some leaf set ls, contains the same nodes in ls, and possibly also k. This is not true: if ls is full, adding a new node k to it will generally result in some other node being removed from the leaf set, invalidating the claimed equality.[7]

[7] See *AddToLSetInvCo* and *AddAndDelete* in LuPastry module *ProofLSetProp*.

Aside from reformulating and proving some assumptions from the original proof, we also added and proved many new facts that were helpful for the proof, resulting in more lemmas in the "Leaf Set Properties" layer.

In the top level of the proof, Lu proves correct delivery by reducing the property to 50 other properties and proves that these properties are invariants of LuPastry, based on the (partly wrong) assumptions made in the lower levels of the proof. Since some assumptions were discovered to fail, the following property *SemJoinLeafSet* is, in fact, not an invariant.

$$
\begin{array}{l}
SemJoinLeafSet \triangleq \\
\quad \forall m \in Messages : m.content.type = \text{"JoinReply"} \Rightarrow \\
\qquad \text{LET } n \triangleq m.content.ls.node \\
\qquad \text{IN } \; \land \; ClockwiseDistance(LeftNeighbor(LeafSets[n]), n) \\
\qquad\qquad\qquad \leq ClockwiseDistance(LeftNeighbor(m.content.ls), n) \\
\qquad\qquad \land \; ClockwiseDistance(n, RightNeighbor(LeafSets[n])) \\
\qquad\qquad\qquad \leq ClockwiseDistance(n, RightNeighbor(m.content.ls))
\end{array}
$$

The predicate asserts that if some node n sends a *JoinReply* message then n's current neighbors are closer to it than its neighbors were at the time when the message was sent. This is not true, however, if n's leaf set was empty at the time the message was sent. As mentioned earlier, in case of an empty leaf set, the left and right neighbors of node n are n itself. Any new neighbors of n will be farther away from n than n itself.

We have written a new, complete correctness proof for correct delivery of the revised protocol specification that does not rely on any unproven assumptions (see Figure 4(b)). At the lowest level of the proof we have 82 lemmas about arithmetic. The abstraction layer provides some 80 lemmas relating to our new operators. The leaf set layer consists of 155 lemmas about the leaf set data structure. On top of this basis are 80 correctness invariants. All lemmas have fully machine-checked proofs.

Because our proof is rigorous, there was a need for a larger number of invariants than in Lu's proof. Also, some of the more involved invariants were split into several invariants in order to facilitate their proof. While our new proof LuPastry$^+$ shares some invariants with the original proof of LuPastry, there is no one-to-one correspondence between the two sets of invariants. In particular, while Lu's original proof depends more on the *lease exchange* phase of the protocol, our own correctness invariants focus more on *probing*.

4 A Proof of Correctness for LuPastry$^+$

Our main TLA$^+$ theorem *LuPastryCorrectness* proves correct delivery, as expressed by the following predicate [6], as an invariant of LuPastry$^+$. The full TLA$^+$ specification and proof are available online [1].[8]

[8] Currently, TLAPS requires that ENABLED be unfolded manually in the machine-checked proof.

$$
\begin{array}{l}
\mathit{CorrectDelivery} \triangleq \\
\quad \forall\, i,\, k \in I : \text{ENABLED } \mathit{DeliverLookup}(i,\, k) \Rightarrow \\
\qquad \land\, \forall\, n \in I : n \neq i \land \mathit{Status}[n] = \text{“Ready”} \\
\qquad\qquad \Rightarrow \mathit{AbsoluteDistance}(i,\, k) \leq \mathit{AbsoluteDistance}(n,\, k) \\
\qquad \land\, \forall\, j \in I \setminus \{i\} : \neg\text{ENABLED } \mathit{DeliverLookup}(j,\, k)
\end{array}
$$

$$
\text{THEOREM } \mathit{LuPastryCorrectness} \triangleq \mathit{Spec} \Rightarrow \Box\, \mathit{CorrectDelivery}
$$

Action *DeliverLookup* is defined as follows.

$$
\begin{array}{l}
\mathit{DeliverLookup}(i,\, j) \triangleq \\
\quad \land\, \mathit{Status}[i] = \text{“Ready”} \;\; \land\, \mathit{Covers}(\mathit{LeafSets}[i],\, j) \\
\quad \land\, \exists\, m \in \mathit{Messages} : \\
\qquad \land\, m.\mathit{content.type} = \text{“Lookup”} \\
\qquad \land\, m.\mathit{destination} = i \\
\qquad \land\, m.\mathit{content.node} = j \\
\qquad \land\, \mathit{Messages}' = (\mathit{Messages} \setminus \{m\}) \\
\quad \land\, \text{UNCHANGED } \langle \mathit{Status},\, \mathit{LeafSets},\, \mathit{Probing},\, \mathit{Leases},\, \mathit{Grants},\, \mathit{ToJoin} \rangle
\end{array}
$$

In what follows, we use shorthand notation instead of the full names of TLA$^+$ functions/operators for compactness. $RN(i) = \mathit{RightNeighbor}(\mathit{LeafSets}[i])$, and $CR(i) = \mathit{ClosestFromTheRight}(i,\, \mathit{ReadyNodes} \setminus \{i\})$ is the closest Ready/OK node to node i from the right. $LN(i)$ and $CL(i)$ are defined analogously. We use $i_1 \rightharpoonup \ldots \rightharpoonup i_n$ to denote a clockwise path on the ring; this is similar to the TLA$^+$ operator *ClockwiseArc*, but extended to an arbitrary number of nodes. The shortest "absolute" path between two nodes i and j may be $i \rightharpoonup j$ or $j \rightharpoonup i$; we denote this shortest path by $i \rightleftharpoons j$. In a ring of 16 nodes, for example, $(3 \rightleftharpoons 5) = (3 \rightharpoonup 5)$, but $(3 \rightleftharpoons 15) = (15 \rightharpoonup 3)$. We write $|p|$ for the length of the path p.

The idea of our proof is intuitive. As pointed out in Section 2, we basically need to prove non-overlapping coverage, *i.e.*, that the coverage of any ready node r_2 starts strictly after the coverage of any other ready node r_1 ends.

$$
\begin{array}{l}
\mathit{NonOverlappingCoverage} \triangleq \forall\, r1,\, r2 \in \mathit{ReadyNodes} : r1 \neq r2 \Rightarrow \\
\quad \mathit{ClockwiseDistance}(r1,\, \mathit{RightCoverage}(\mathit{LeafSets}[r1])) \\
\qquad < \mathit{ClockwiseDistance}(r1,\, \mathit{LeftCoverage}(\mathit{LeafSets}[r2]))
\end{array}
$$

It is easy to prove non-overlapping coverage if we prove the following property, which (adapting notation) was already pointed out by Lu [6] as a main invariant.

$$
\begin{array}{l}
\mathit{CloseNeighbors} \triangleq \forall\, r1,\, r2 \in \mathit{ReadyNodes} : r1 \neq r2 \Rightarrow \\
\quad \land\, \mathit{ClockwiseArc}(r1,\, \mathit{RightNeighbor}(\mathit{LeafSets}[r1]),\, r2) \\
\quad \land\, \mathit{ClockwiseArc}(r2,\, \mathit{LeftNeighbor}(\mathit{LeafSets}[r1]),\, r1)
\end{array}
$$

We prove *CloseNeighbors* by proving a stronger property which we call *stable network*. A node i is *stable* if $CR(i)$ and $CL(i)$ are in i's leaf set (Figure 5a). A Pastry ring is *stable* if all Ready or OK nodes are stable. It is clear that in a stable ring, the properties *CloseNeighbors*, and consequently *NonOverlappingCoverage* hold. For a minimum leaf set size $L = 3$, stable network is an invariant of LuPastry$^+$. Let a *participating node* be a node that is either Ready or OK, or is

the to-join node of a Ready node. Essentially, a participating node is any node that is known to some Ready or OK node. Let i and j be two consecutive Ready or OK nodes on the ring (see Figure 5b). There can be at most two participating nodes k_1, k_2 between i and j: the to-join nodes of i and j. Any other non-Dead node between i and j must be a Waiting node whose join request has not been picked up by i or j (since they are busy facilitating the joins of k_1 and k_2), and so it can not be in the leaf sets of i or j. For a minimum leaf set size $L = 3$, we can ensure that stable i and j remain stable even if new nodes are added to their leaf sets.

We observe that the ring has the following properties, which we have proven in TLA$^+$.

P1 The coverage of a node is computed based on half the distance to its neighbors. A key k covered by a node i lies in either the *right* or *left* coverage regions of i (see Fig. 1). If i and j are leaf set neighbors, *i.e.*, $i = LN(j)$ and $j = RN(i)$, their coverage regions cannot overlap.

P2 If k is in i's right (left) coverage region, $i \rightharpoonup k \rightharpoonup RN(i)$ ($LN(i) \rightharpoonup k \rightharpoonup i$)).

P3 If i is a stable node and $r \neq i$ is some Ready or OK node, then $i \rightharpoonup RN(i) \rightharpoonup r$ and $r \rightharpoonup LN(i) \rightharpoonup i$.

P4 Because we exclude node failure, all protocol actions that modify a node's leaf set do so through the operation *AddToLS*. Therefore, nodes are not purposely removed from a leaf set, but a node j may only be evicted from the leaf set for node i through an *AddToLS* operation that results in an overflow; *i.e.*, if the leaf set of i becomes full and j is replaced by another node that is closer to i.

P5 A new node k joins the network through a Ready node i that initially covers it, and so k will remain closest to i on one side (right or left) until it finishes its join process. Only after k has finished joining and turned Ready can other nodes join the network between k and i. Therefore, any participating node between i and $CR(i)$ is either $ToJoin[i]$ or $ToJoin[CR(i)]$. That is, there can never be three different participating nodes k_1, k_2, k_3 such that $i \rightharpoonup k_1 \rightharpoonup k_2 \rightharpoonup k_3 \rightharpoonup CR(i)$, or dually, $CL(i) \rightharpoonup k_1 \rightharpoonup k_2 \rightharpoonup k_3 \rightharpoonup i$.

P6 If the leaf set size $L \geq 3$, no action can cause $CR(i)$ or $CL(i)$ to be removed from i's leaf set due to an overflow (see Figure 5b).

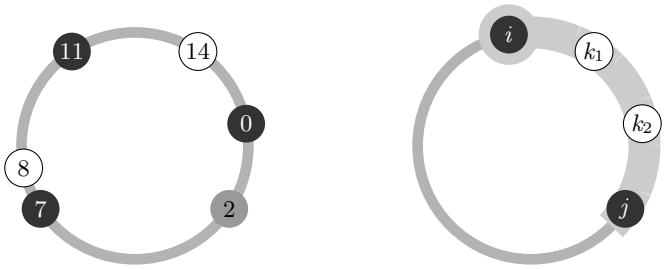
P7 At any point in time, a participating node i is either probing $CR(i)$ (resp., $CL(i)$), or $CR(i)$ (resp., $CL(i)$) is in i's leaf set.

P8 At any point in time, a participating node i is either probing $CR(i)$ (resp., $CL(i)$), or i is in the leaf set of $CR(i)$ (resp., $CL(i)$).

Using these properties, our proof can be outlined in two theorems.

Theorem 1. *In any stable LuPastry$^+$ network, correct delivery holds.*

Proof (Outline). The action $DeliverLookup(i, k)$ is enabled only if i is a Ready node that thinks it covers key k. Assume for the sake of contradiction that $DeliverLookup(i, k)$ and $DeliverLookup(j, k)$ are enabled where $j \neq i$. Nodes i and j are Ready, and both think they cover k. W.l.o.g., $i \rightharpoonup k \rightharpoonup j$; k is in i's



(a) This ring is stable if nodes $0, 2, 7, 11$ are stable. Node 0 is stable if its left leaf set contains 11 and its right leaf set contains 2.

(b) For $L \geq 3$, i's leaf set can always accommodate $k_1 = \mathit{ToJoin}[i]$, $k_2 = \mathit{ToJoin}[j]$, and i's Ready/OK neighbor j.

Fig. 5: Network Stability

right coverage region and j's left coverage region. Therefore, $i \rightharpoonup k \rightharpoonup RN(i) \rightharpoonup j$ and $i \rightharpoonup LN(j) \rightharpoonup k \rightharpoonup j$, by P2, P3. In order for both to hold, it must be that $i = LN(j)$ and $j = RN(i)$. Therefore, the coverage regions of i and j cannot overlap (contradiction, by P1). Therefore, in a stable LuPastry$^+$ network, two Ready nodes i and j cannot both think they cover the same key k, and so at most one of $\mathit{DeliverLookup}(i, k)$ and $\mathit{DeliverLookup}(j, k)$ is enabled.

It remains to show that if $\mathit{DeliverLookup}(i, k)$ is enabled then i is closer to k than any other Ready node r is in terms of absolute distance on the ring. Assume again that k is in i's right coverage region, and so $i \rightharpoonup k \rightharpoonup RN(i) \rightharpoonup r$ (by P3). Because k lies within half the distance from i to $RN(i)$ (by P1), k must lie within half the distance from i to r. Therefore, $|i \rightharpoonup k| = |i \rightleftharpoons k| \leq |k \rightharpoonup r|$. If $|r \rightleftharpoons k| = |k \rightharpoonup r|$, we are done. Alternatively, assume $|r \rightleftharpoons k| = |r \rightharpoonup k|$. Because of the ordering of the nodes on the ring $r \rightharpoonup i \rightharpoonup k$, we have that $|r \rightharpoonup k| = |r \rightharpoonup i| + |i \rightharpoonup k|$. Since path lengths are non-negative, $(i \rightleftharpoons k) \leq (r \rightleftharpoons k)$. □

Theorem 2. *For $L \geq 3$, the network is always stable.*

Proof (Outline). The definition of *Init* implies that the LuPastry$^+$ ring is stable in the initial state: nodes in A are Ready and all other nodes are Dead. The leaf set of each A-node i is composed by adding all other A-nodes to i's empty leaf set. Consequently, the leaf set of i will contain its closest right and left A-neighbors. All A-nodes are stable, and so the network is stable. Now consider a stable Pastry network N. For the induction step, we need to show that N remains stable after executing any sub-action e of *Next*. We use N'_e to refer to the new state of N after the execution of e, and $CR'_e(i)$ and $CL'_e(i)$ the next values of $CR(i)$ (resp., $CL(i)$) for a node i. Note that for a node i that is not Ready or OK, the only action that can change i into a Ready or OK node is $\mathit{ReceiveProbeReply}(i)$: i receives the last probe reply message it was waiting for, its probing set becomes empty, and i becomes OK. We need to show that (1) if e results in some unstable node i in N to become Ready or OK, then

i is stable in N'_e, and (2) all stable nodes in N remain stable in N'_e. (1) Let i be an unstable N-node. Since N is stable, i is not Ready or OK. It must be that $e = \mathit{ReceiveProbeReply}(i)$. Since e can only change the local variables of i, the status of all other nodes remains unchanged; $CR'_e(i) = CR(i)$ and $CL'_e(i) = CL(i)$. By P7 and since i's probing set is empty, $CR(i)$ and $CL(i)$ are in i's leaf set, hence i is stable in N'_e. (2) Let i be a stable node in N. If $CR'_e(i) = CR(i)$ and $CL'_e(i) = CL(i)$, then i remains stable in N'_e by P6. Now suppose $CR'_e(i) \neq CR(i)$ (the proof is similar for $CL'_e(i) \neq CL(i)$). Therefore, $e = \mathit{ReceiveProbeReply}(j)$ and $CR'_e(i) = j$. Now, $i = CL'(j)$. By P8 and since j's probing set is empty, j is in i's leaf set. Therefore, i remains stable. □

The total TLA$^+$ proof consists of more than 30,000 lines. The time taken to run the proof manager on the entire proof is 8 hours and 57 minutes on a single Intel Xeon(R) CPU E5-2680 core running at 2.7GHz with 256GB RAM.

5 Conclusion

This paper presents LuPastry$^+$: the first completely machine-checked proof of correct delivery for the variant of Pastry introduced by Lu [6]. Like Lu's proof, our proof has been mechanized in the TLA$^+$ proof system. Compared to Lu's specification, we introduce some new TLA$^+$ operators that abstract away from arithmetic reasoning and other troublesome TLA$^+$ constructs, and this helps avoid low-level arithmetic reasoning at higher levels of the proof. Most importantly, our proof no longer relies on any unproven assumptions. Because we filled all the holes, our overall proof is longer (more than 30,000 lines overall) than Lu's original proof. Nevertheless, our new operators helped significantly reduce the number of steps in the main proof.

Lu [6] shows that correct delivery does not hold for the original published specification of Pastry, in particular in the case of node failure. Like Lu's original proof, our proof assumes that nodes never fail and focuses only on the join protocol. An interesting future work would be to identify assumptions on node failures that maintain the correctness of the protocol, or of a suitable variant, in the presence of nodes joining and leaving the ring.

Our experience shows that TLA$^+$ is well-suited for modeling concurrent and distributed algorithms such as Pastry. In particular, the set-theoretic nature of the specification language encourages the user to model the algorithm at a suitably high level of abstraction. Moreover, TLA$^+$'s hierarchical proof language lets a user focus on parts of the proof without having to remember details about unrelated parts of the proof. Like Lu, we rely on a large invariant for proving the main safety property of the protocol "in one shot", rather than proceeding by refinement from a high-level model where nodes join atomically, down to a detailed model of the real protocol. TLA$^+$ has a notion of refinement based on trace inclusion, and it would be interesting to develop a refinement-based proof and compare its complexity to that of our proof. The difficulty is that parts of the state, such as contents of leaf sets under construction, become visible when

some node completes its join protocol, revealing information about other nodes joining concurrently that would have to be anticipated in a refinement-based development.

On a technical level, TLAPS includes facilities for checking the status of a proof that can identify which steps are affected by a change in the specification or the proof. While TLAPS can manage a proof of the size reported here, it is barely able to do so. For example, the Java heap size allotted to Eclipse has to be increased to several gigabytes, and status checking currently takes almost as much time as rerunning the proof. We are in contact with the TLAPS developers who are investigating solutions to these bottlenecks. Although users of a mechanical theorem prover always dream of better automation, the main difficulty in the formal verification of algorithms is in fact finding sufficiently strong inductive invariants that underpin the correctness argument, and any assistance in this task would be most welcome.

References

1. LuPastry$^+$: Specification and Proof Files. http://www.mpi-inf.mpg.de/departments/automation-of-logic/people/noran-azmy/.
2. R. Bakhshi and D. Gurov. Verification of Peer-to-Peer Algorithms: A Case Study. *Electronic Notes in Theoretical Computer Science*, 181:35–47, June 2007.
3. J. Borgström, U. Nestmann, L. O. Alima, and D. Gurov. Verifying a Structured Peer-to-Peer Overlay Network: The Static Case. In *Global Computing, IST/FET International Workshop, Revised Selected Papers*, pages 250–265, 2004.
4. D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto. TLA$^+$ Proofs. In D. Giannakopoulou and D. Méry, editors, *18th Intl. Symp. Formal Methods*, volume 7436 of *LNCS*, pages 147–154, Paris, France, 2012. Springer.
5. L. Lamport. *Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, Boston, MA, USA, 2002.
6. T. Lu. Formal Verification of the Pastry Protocol Using TLA$^+$. In X. Li, Z. Liu, and W. Yi, editors, *SETTA 2015*, volume 9409 of *LNCS*, pages 284–299, 2015.
7. A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In R. Guerraoui, editor, *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*, volume 2218 of *LNCS*, pages 329–350, November 2001.
8. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM'01*, pages 149–160. ACM, 2001.
9. Y. Yu, P. Manolios, and L. Lamport. Model checking TLA$^+$ Specifications. In L. Pierre and T. Kropf, editors, *Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *LNCS*, pages 54–66, 1999.
10. P. Zave. Using Lightweight Modeling to Understand Chord. *ACM SIGCOMM Computer Communication Review*, 42(2):49, Mar. 2012.
11. P. Zave. How to Make Chord Correct (Using a Stable Base). CoRR abs/1502.06461, 2015.