Towards Verification of the Pastry Protocol Using TLA+

Tianxiang Lu¹,², Stephan Merz², and Christoph Weidenbach¹

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany  
² INRIA Nancy & LORIA, Nancy, France

Abstract. Pastry is an algorithm that provides a scalable distributed hash table over an underlying P2P network. Several implementations of Pastry are available and have been applied in practice, but no attempt has so far been made to formally describe the algorithm or to verify its properties. Since Pastry combines rather complex data structures, asynchronous communication, concurrency, resilience to churn and fault tolerance, it makes an interesting target for verification. We have modeled Pastry’s core routing algorithms and communication protocol in the specification language TLA⁺. In order to validate the model and to search for bugs we employed the TLA⁺ model checker TLC to analyze several qualitative properties. We obtained non-trivial insights in the behavior of Pastry through the model checking analysis. Furthermore, we started to verify Pastry using the very same model and the interactive theorem prover TLAPS for TLA⁺. A first result is the reduction of global Pastry correctness properties to invariants of the underlying data structures.

Keywords: formal specification, model checking, verification methods, network protocols.

1 Introduction

Pastry [9,3,5] is an overlay network protocol that implements a distributed hash table. The network nodes are assigned logical identifiers from an Id space of naturals in the interval \([0, 2^M - 1]\) for some \(M\). The Id space is considered as a ring, i.e., \(2^M - 1\) is the neighbor of 0. The Ids serve two purposes. First, they are the logical network addresses of nodes. Second, they are the keys of the hash table. An active node is in particular responsible for keys that are numerically close to its network Id, i.e., it provides the primary storage for the hash table entries associated with these keys. Key responsibility is divided equally according to the distance between two neighbor nodes. If a node is responsible for a key we say it covers the key.

The most important sub-protocols of Pastry are join and lookup. The join protocol eventually adds a new node with an unused network Id to the ring.
The lookup protocol delivers the hash table entry for a given key. An important correctness property of Pastry is Correct Key Delivery, requiring that there is always at most one node responsible for a given key. This property is non-trivial to obtain in the presence of spontaneous arrival and departure of nodes. Nodes may simply drop off, and Pastry is meant to be robust against such changes, i.e., churn. For this reason, every node holds two leaf sets of size \( l \) containing its closest neighbors to either side (\( l \) nodes to the left and \( l \) to the right). A node also holds the hash table content of its leaf set neighbors. If a node detects, e.g. by a ping, that one of its direct neighbor nodes dropped off, the node takes actions to recover from this state. So the value of \( l \) is relevant for the amount of “drop off” and fault tolerance of the protocol.

A lookup request must be routed to the node responsible for the key. Routing using the leaf sets of nodes is possible in principle, but results in a linear number of steps before the responsible node receives the message. Therefore, on top of the leaf sets of a node a routing table is implemented that enables routing in a logarithmic number of steps in the size of the ring.

![Pastry Routing Example](image)

**Fig. 1.** Pastry Routing Example

Pastry routes a message by forwarding it to nodes that match progressively longer prefixes with the destination key. In the example of Fig. 1, node 18 received a lookup message for key 95. The key is outside node 18’s coverage and furthermore, it doesn’t lie between the leftmost node and the rightmost node of its leaf sets. Querying its routing table, node 18 finds node 58, whose identifier matches the longest prefix with the destination key and then forwards the message to that node. Node 58 repeats the process and finally, the lookup message is answered by node 65, which is the closest node to the key 95, i.e., it covers