A Self-repairing Peer-to-Peer System Resilient to Dynamic Adversarial Churn*

Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer

Computer Engineering and Networks Laboratory,
ETH Zurich, 8092 Zurich, Switzerland

Abstract. We present a dynamic distributed hash table where peers may join and leave at any time. Our system tolerates a powerful adversary which has complete visibility of the entire state of the system and can continuously add and remove peers. Our system provides worst-case fault-tolerance, maintaining desirable properties such as a low peer degree and a low network diameter.

1 Introduction

Storing and handling data in an efficient way lie at the heart of any data-driven computing system. Compared to a traditional client/server approach, decentralized peer-to-peer (P2P) systems have the advantage to be more reliable, available, and efficient. P2P systems are based on common desktop machines ("peers"), distributed over a large-scale network such as the Internet. These peers share data (as well as the management of the data) that is conventionally stored on a central server. Usually, peers are under control of individual users who turn their machines on or off at any time. Such peers join and leave the P2P system at high rates ("churn"), a problem that is not existent in orthodox distributed systems. In other words, a P2P system consists of unreliable components only. Nevertheless, the P2P system should provide a reliable and efficient service.

Most P2P systems in the literature are analyzed against an adversary who can crash a functionally bounded number of random peers. After crashing a few peers the system is given sufficient time to recover again. The scheme described in this paper significantly differs from this in two major aspects. First, we assume that joins and leaves occur in a worst-case manner. We think of an adversary which can remove and add a bounded number of peers. The adversary cannot be fooled by any kind of randomness. It can choose which peers to crash and how peers join.[1] Note that we use the term "adversary" to model worst-case behavior. We do not consider Byzantine faults. Second, the adversary does not have to wait until the system is recovered before it crashes the next batch of peers.

* Research (in part) supported by the Hasler Stiftung and the Swiss National Science Foundation.

[1] We assume that a joining peer knows a peer which already belongs to the system. This is known as the *bootstrap* problem.

M. Castro and R. van Renesse (Eds.): IPTPS 2005, LNCS 3640, pp. 13–23, 2005.

Instead, the adversary can constantly crash peers while the system is trying to stay alive. Indeed, our system is *never fully repaired* but *always fully functional*. In particular, our system is resilient against an adversary which continuously attacks the "weakest part" of the system. Such an adversary could for example insert a crawler into the P2P system, learn the topology of the system, and then repeatedly crash selected peers, in an attempt to partition the P2P network. Our system counters such an adversary by continuously moving the remaining or newly joining peers towards the sparse areas.

Clearly, we cannot allow our adversary to have unbounded capabilities. In particular, in any constant time interval, the adversary can at most add and/or remove $O(\log n)$ peers, n being the total number of peers currently in the system. This model covers an adversary which repeatedly takes down machines by a distributed denial of service attack, however only a logarithmic number of machines at each point in time. Our algorithm relies on messages being delivered timely, in at most constant time between any pair of operational peers. In distributed computing such a system is called *synchronous*. Note that if nodes are synchronized locally, our algorithm also runs in an asynchronous environment. In this case, the propagation delay of the slowest message defines the notion of time which is needed for the adversarial model.

The basic structure of our P2P system is a hypercube. Each peer is part of a distinct hypercube node; each hypercube node consists of $\Theta(\log n)$ peers. Peers have connections to other peers of their hypercube node and to peers of the neighboring hypercube nodes. In the case of joins or leaves, some of the peers have to change to another hypercube node such that up to constant factors, all hypercube nodes own the same number of peers at all times. If the total number of peers grows or shrinks above or below a certain threshold, the dimension of the hypercube is increased or decreased by one, respectively.

The balancing of peers among the hypercube nodes can be seen as a dynamic token distribution problem [1] on the hypercube. Each node of a graph (hypercube) has a certain number of tokens, the goal is to distribute the tokens along the edges of the graph such that all nodes end up with the same or almost the same number of tokens. While tokens are moved around, an adversary constantly inserts and deletes tokens. Our P2P system builds on two basic components: i) an algorithm which performs the described dynamic token distribution and ii) an information aggregation algorithm which is used to estimate the number of peers in the system and to adapt the dimension accordingly.

Based on the described structure, we get a fully scalable, efficient P2P system which tolerates $O(\log n)$ worst-case joins and/or crashes per constant time interval. As in other P2P systems, peers have $O(\log n)$ neighbors, and the usual operations (e.g. search) take time $O(\log n)$. In our view a main contribution of the paper, however, is to propose and study a model which allows for dynamic adversarial churn. We believe that our basic algorithms (dynamic token distribution and information aggregation) can be applied to other P2P topologies,