Robust Random Number Generation for Peer-to-Peer Systems

Baruch Awerbuch*[1] and Christian Scheideler**[2]

[1] Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA
[2] Institute for Computer Science, TU Munich, 85748 Garching, Germany

Abstract. We consider the problem of designing an efficient and robust distributed random number generator for peer-to-peer systems that is easy to implement and works even if all communication channels are public. A robust random number generator is crucial for avoiding adversarial join-leave attacks on peer-to-peer overlay networks. We show that our new generator together with a light-weight rule recently proposed in [4] for keeping peers well-distributed can keep various structured overlay networks in a robust state even under a constant fraction of adversarial peers.

1 Introduction

Due to their many applications, peer-to-peer systems have recently received a lot of attention both inside and outside of the research community. Most of the structured peer-to-peer systems are based on two influential papers: a paper by Plaxton et al. on locality-preserving data management in distributed environments [20] and a paper by Karger et al. on consistent hashing and web caching [13]. The consistent hashing approach is a very simple and elegant approach that assigns to each peer a (pseudo-)random point in the $[0, 1)$-interval. Based on this approach, various local-control rules have been proposed to decide how to interconnect the peers so that they form a well-connected network with good routing properties that is easy to maintain (see, e.g., [18] for a general framework).

In open peer-to-peer systems, the presence of adversarial peers cannot be avoided. Hence, not only scalability but also robustness against adversarial behavior is an important issue. The key to scalability and robustness for peer-to-peer networks based on the consistent hashing approach is to keep the honest and adversarial peers well-distributed in the $[0, 1)$-interval. However, just assigning a random or pseudo-random point to each new peer (by using some random number generator or cryptographic hash function) does not suffice to keep the honest and adversarial peers well-spread [2]. People in the peer-to-peer community are aware of this problem [8, 9] and various solutions have been proposed that may help alleviating it in practice [6, 7, 19, 27, 28, 30] but until recently no

* Email: baruch@cs.jhu.edu. Partially supported by NSF grants CCF 0515080, ANIR-0240551, CCR-0311795, and CNS-0617883.
** Email: scheideler@in.tum.de

mechanism was known that can *provably* keep the peers in a well-distributed state without sacrificing the openness of the system.

Various light-weight perturbation rules that can keep the honest and adversarial peers well-distributed have recently been proposed in [4, 10, 26]. These rules do not need to be able to distinguish between the honest and adversarial peers, but a crucial prerequisite for them to work is a robust distributed random number generator. This random number generator has to work correctly in a system without mutual trust relationships and must be robust against arbitrary adversarial behavior to be applicable to peer-to-peer systems. Certainly, designing such a random number generator is not an easy task.

1.1 Robust distributed random number generation

How can we generate random numbers in a peer-to-peer system with adversarial presence? The most naive approach is to let every peer generate its own random numbers. This approach is problematic since in a dynamic peer-to-peer system it is impossible to collect sufficient statistical evidence to accuse a particular peer of generating non-random numbers. Yet, somewhat surprisingly, it is still possible to use this approach to maintain a robust peer-to-peer network, but at the cost of losing scalability [3]. So a different approach is needed.

A more reasonable approach is the following. Suppose that we need a random number generator that generates a number by selecting a binary string uniformly at random out of $\{0,1\}^s$ for some s. Consider the situation that a group P of the peers wants to generate a random number. Each (honest) peer p in P may then select a random number $x_p \in \{0,1\}^s$ and commit to it to all other peers in P using a bit commitment scheme (a particularly secure one-way hash function h for which $h(x)$ does not reveal anything about x) [12, 17]. Once all commitments have been made, the peers will reveal their random numbers, and if they all do, every peer computes $x = \bigoplus_{p \in P} x_p$, where $\oplus$ is the bit-wise XOR operation. The XOR operation has the nice property that as long as at least one x_p is chosen uniformly at random and the other numbers are independent of it, x is distributed uniformly at random in $\{0,1\}^s$. Hence, *if* the scheme succeeds and at least one honest peer participates in it, a random number x will be generated. But the adversarial peers can easily let the scheme fail, and this not only in an oblivious manner but also in an adaptive manner (by just waiting for enough numbers x_p to be revealed before revealing their own numbers). Thus, in order to avoid a significant bias on the successfully generated random numbers, the fraction of adversarial peers in the system would have to be so small that no adversarial peer will be present in most of the groups P that are used for the random number generation. Such an approach was pursued in [2].

To avoid the problems above, we recently proposed a distributed random number generator that is based on verifiable secret sharing [4]. This random number generator can still fail if the peer initiating it does not behave correctly, but it has the advantage that if the peer initiating it is honest, then the random number generation is guaranteed to succeed, and whenever the random number generation succeeds, the number generated will be random.

Yet, using this scheme is not completely satisfying. First of all, an adversary can let it fail in an adaptive manner (i.e., it can let it fail after knowing the final key), which is sufficient to create a significant bias, even though the adversary cannot undermine the randomness of the generated key. It just has to run sufficiently many attempts until a key is generated that falls into a desired range. Furthermore, the scheme is not easy to implement and private channels are needed between the peers. So the question that led to this paper was:

Is it possible to design an elementary and sufficiently unbiased distributed random number generator that even works for public channels and a constant fraction of adversarial peers?

Remarkably, this paper shows that this is possible.

1.2 Related work on random number generation

Surprisingly little has been published about robust random number generators for distributed systems. Random number generators have mostly been studied in the context of pseudo-random number generators (PRNGs) with small seed or cryptographically secure random number generators (CSRNG). The main difference between a PRNG and a CSPRNG is that a CSPRNG should be indistinguishable from random on any examination, whereas a PRNG is normally only required to look random to standard statistical tests. For foundations and surveys on random number generators see, e.g., [11, 16, 23, 32].

There are many protocols for distributed systems with adversarial presence that need random numbers for atomic broadcasting, leader election and almost-everywhere agreement (e.g., [14, 21] for recent results), but in these it is sufficient that every peer chooses its own random numbers.

Unbiased random numbers can be computed via verifiable secret sharing or secure multiparty computation schemes (e.g., [5, 29]), but these are not easy to implement (since they need error correction techniques), and they require private channels.

1.3 Details of our random number generator

The basic idea behind our random number generator is the insight that generating a *single* random number is difficult with public channels but generating a *batch* of random numbers is doable. An *m-random number generator* (or *m*-RNG) is a random number generator that generates a batch of up to m random numbers. We assume that every random number is represented as a binary string in $\{0,1\}^s$ for some fixed s. Given an m-RNG G and any subset $S \subseteq \{0,1\}^s$, let $\mathrm{E}_G(S)$ be the expected number of keys y generated by G with $y \in S$. Ideally, G should satisfy $\mathrm{E}_G(S) = m \cdot |S|/2^s$ for all $S \subseteq \{0,1\}^s$. Let $\mathrm{E}(S) = m \cdot |S|/2^s$. Then we define the *bias* $\beta(G)$ of G as

$$\beta(G) = \max_{S \subseteq \{0,1\}^s} \max \left\{ \frac{\mathrm{E}_G(S)}{\mathrm{E}(S)} \, , \, \frac{\mathrm{E}(S)}{\mathrm{E}_G(S)} \right\}$$

The m-RNG that we present in this paper is called *round-robin random number generator* (or short round-robin RNG). Let P be the group of m peers this protocol is applied to. The basic ideas of the protocol can be summarized as follows:

- When correctly initiated, every peer in P will supervise the generation of one random number in $\{0,1\}^s$. A peer whose random number generation fails can send an accusation to the peers in P in which it can accuse exactly one other peer. Honest peers will run the random number generation one after the other (using a proper timing scheme) so as to maximize the effect of the accusations and thereby minimize the number of times an adversarial peer can cause the failure of a random number generation supervised by an honest peer.
- A single random number is generated by the supervising peer taking over the role of a dealer and the others being a group of players. Both the players and the dealer commit to a key. However, as we will see, the dealer key is a special master key that is committed first and revealed last. In this way, the dealer is the only one that can *adaptively* decide whether to let the random number generation fail or not. However, this is the only way in which the dealer can bias the random number generation. It cannot make its probability distribution non-uniform if at least one honest player is participating in it.

More details are given in Section 2. For this protocol, the following theorem is shown.

Theorem 1. *Suppose that $|P| = m$ and there are $t < m/6$ adversarial peers in P. Then the round-robin RNG generates random keys $y_1, y_2, \ldots, y_k \in \{0,1\}^s$ with $m - 2t \leq k \leq m$ and the property that for all subsets $S \subseteq \{0,1\}^s$ with $\sigma = |S|/2^s$,*

$$\mathrm{E}[|\{i \mid y_i \in S\}|] \in [(m-2t)\sigma,\ m \cdot \sigma]$$

The worst-case message complexity of the protocol is $O(m^2)$.

Hence, the bias of our m-RNG is just $1 + \frac{2t}{m-2t}$, which is a constant. It turns out that this bias is small enough in order to maintain a scalable and robust peer-to-peer network.

1.4 Application to robust peer-to-peer networks

In the area of peer-to-peer systems, work on robustness in the context of overlay network maintenance has mostly focused on how to handle a large fraction of faulty peers (e.g., [1, 24, 31]) or churn, that is, peers frequently enter and leave the system (e.g., [15, 22]). However, none of these approaches can protect a peer-to-peer network against adaptive join-leave attacks. In an adaptive join-leave attack, adversarial peers repeatedly join and leave a network in order to occupy certain areas of the network. To prevent them from doing this, proper join and leave protocols have to be found so that the honest and adversarial peers are kept

well-spread in the $[0,1)$-interval. More precisely, what we would like to aim for is that at any time point with n peers in the system the following two conditions can be met for every interval $I \subseteq [0,1)$ of size at least $(c \log n)/n$ for a constant $c > 0$:

- *Balancing condition:* I contains $\Theta(|I| \cdot n)$ peers.
- *Majority condition:* the honest peers in I are in the majority.

If this is the case, then proper region-based overlay networks and routing rules can be defined to guarantee connectivity and correct routing (e.g., [4]). However, maintaining the two conditions under adaptive adversarial join-leave attacks turns out to be quite tricky. Just assigning a random or pseudo-random point to each new peer (by using some random number generator or cryptographic hash function) does not suffice to preserve the balancing and majority conditions [2]. Fortunately, just recently we found a join operation, called cuckoo rule, that can solve this problem [4].

1.5 The cuckoo rule

In the following, a *region* is an interval of size $1/2^r$ in $[0,1)$ for some integer r that starts at an integer multiple of $1/2^r$. Hence, there are exactly 2^r regions of size $1/2^r$. A *k-region* is a region of size (closest from above to) k/n, and for any point $x \in [0,1)$, the k-region $R_k(x)$ is the unique k-region containing x.

Cuckoo rule: If a new node v wants to join the system, pick a random $x \in [0,1)$. Place v into x and move all nodes in $R_k(x)$ to points in $[0,1)$ chosen uniformly and independently at random (without replacing any further nodes).

Suppose that we have n honest peers and ϵn adversarial peers in the system for some $\epsilon < 1$. For the situation that the adversary adaptively rejoins the system with its peers in a one-by-one fashion, it was shown [4] that as long as $\epsilon < 1 - 1/k$, the k-cuckoo rule satisfies the balancing and majority conditions for a polynomial number of rejoin operations, with high probability. However, for the cuckoo rule to be implementable in a distributed system, a robust distributed random number generator is needed. Furthermore, the cuckoo rule may need up to $O(\log^2 n)$ random bits in the worst case (for $O(\log n)$ peers that need to be replaced).

1.6 The round-robin cuckoo rule.

The problem with $O(\log^2 n)$ bits is solved by proposing a slight adaptation of the cuckoo rule that we call the *de Bruijn cuckoo rule*. The new rule has the benefit that only $O(\log n)$ random bits are needed in the worst case (for two random points in $[0,1)$).

In order to solve the problem with the random number generator, we combine the round-robin RNG with the de Bruijn cuckoo rule to the so-called *round-robin cuckoo rule*. It works in a way that for every successful random number

generation in the round-robin RNG, the de Bruijn cuckoo rule is used. The protocol has the following performance.

Consider adversarial join-leave attacks in a system with n honest peers and ϵn adversarial peers. Let β be the bias of the round-robin RNG. Then it holds:

Theorem 2. *For any constants ϵ, k and β with $\epsilon < (1/\beta^2 - 1/k)$, the round-robin cuckoo rule satisfies the balancing and majority conditions for a polynomial number of rounds, with high probability, for any adversarial strategy within our model.*

We also investigate the case of using arbitrary m-RNGs with bias β, which gives a slightly worse result.

1.7 Structure of the paper

In Section 2, we present the round-robin random number generator, and in Section 3 we show how to use it to counter join-leave attacks in peer-to-peer networks. The paper ends with conclusions.

2 Robust random number generation

In this section we consider the situation that we have a set P of m players denoted $p_1, \ldots, p_m$. We distinguish between honest and adversarial players. The honest players follow the protocol in a correct and timely manner, whereas the adversarial players may behave in an arbitrary way, including arbitrary collusion among the adversarial players. Our goal is to find *elementary* protocols that construct random numbers with a uniform distribution in $\{0,1\}^s$ for some given s, even under adversarial presence.

First, we state some assumptions, and then we present the round-robin random number generator. After its analysis, we discuss some extensions for peer-to-peer systems.

2.1 Assumptions

We assume that only point-to-point communication is available and that all information sent out by a player can be seen by the adversary. Thus, no broadcasting primitive and no private channels are given, which is often the case in other robust distributed protocols like verifiable secret sharing. We just need a mechanism that allows the players to verify the sender of a message. For this, we assume the existence of a non-repudiable signature scheme. A message m signed by player p will be denoted as $(m)_p$.

Honest players are supposed to act not only in a correct but also a timely manner (which is important to maintain dynamic systems such as peer-to-peer networks). We assume that any message sent from one honest player to another honest player needs at most δ time steps to be received and processed by the

recipient for some fixed δ, and we assume that the clock speeds of the honest players are roughly the same. But the clocks do not have to be synchronized (i.e., show the same time) nor do we require the protocols to run in a synchronous mode (i.e., all players must send their messages at exactly the same time). The latter assumption makes it hard to generate unbiased random keys even though there is a notion of time because the adversarial players can always choose to be the last to send out messages, thereby maximizing the control they have on the generation of the random number.

For the random number generation, we need a bit commitment scheme h, i.e., a scheme where $h(x)$ does not reveal anything about x. In practice, a cryptographic hash function might be sufficient for h so that the protocols below can be easily implemented. Furthermore, we assume that all honest players have a perfect random number generator. In practice, pseudo-random number generators that pass a certain collection of statistical tests (such as the diehard tests) might be sufficient here.

2.2 Round-robin random number generator

Suppose that we have a set P of m players, $p_1, \ldots, p_m$, that mutually know each other and the indexing, with any t of them being adversarial for some $t < m/6$. The round-robin random number generator works as follows for some player $p^* \in P$ initiating it.

1. p^* sends a signed request to initiate the random number generation to all players in P.
2. Once player $p_i \in P$ receives p^*s signed initiation request for the first time (from anywhere), it forwards it to all other players in P. Afterwards, it sets $P_i := P \setminus \{p_i\}$ and waits for $i \cdot 8\delta$ time steps. Each time it receives an accusation $(p_k)_{p_j}$ from a player $p_j \in P$ it has not received an accusation from yet, it sets $P_i := P_i \setminus \{p_k\}$. Once the $i \cdot 8\delta$ steps are over, p_i initiates step (3). p_i terminates after $(m+1)8\delta$ steps.
3. If $|P_i| \geq 2m/3$, then p_i chooses a random $x_i \in \{0,1\}^s$ and sends $(h(x_i), P_i)_{p_i}$ to all players in P_i. Otherwise, p_i aborts the protocol (which will not happen if $t < m/6$).
4. Each player $p_j \in P_i$ receiving a message $(h(x_i), P_i)_{p_i}$ for the first time from p_i with $P_i \geq 2m/3$ chooses a random $x_j \in \{0,1\}^s$ and sends the message $(p_i, h(x_j), P_i)_{p_j}$ to p_i. Otherwise, it does nothing.
5. If all players in P_i reply within 2δ time steps, then p_i sends $(\{(p_i, h(x_j), P_i)_{p_j} \mid p_j \in P_i\})_{p_i}$ to all players in P_i. Otherwise, p_i sends an accusation $(p_j)_{p_i}$ for any $p_j \in P_i$ that did not reply correctly or in time to all players in P and stops its attempt of generating a random number.
6. Once $p_j \in P_i$ receives $(\{(p_i, h(x_k), P_i)_{p_k} \mid p_k \in P_i\})_{p_i}$ from p_i, p_j sends $(x_j)_{p_j}$ to p_i.
7. If p_i gets a correct reply back from all players in P_i within 2δ time steps, then it sends $(x_i, \{(x_j)_{p_j} \mid p_j \in P_i\})_{p_i}$ to all players in P_i and computes $y_i = x_i \oplus \bigoplus_{p_j \in P_i} x_j$ where $\oplus$ is the bit-wise XOR operation. Otherwise, p_i

sends an accusation $(p_j)_{p_i}$ to all players in P for any $p_j \in P_i$ that did not reply correctly or in time and stops.

8. Once $p_j \in P_i$ receives $(x_i, \{(x_k)_{p_k} \mid p_k \in P_i\})_{p_i}$, p_j verifies that all keys are correct. Then p_j computes $y_j^{(i)} = x_i \oplus \bigoplus_{q_k \in P_i} x_k$ and sends the message $(y_j^{(i)})_{p_j}$ to p_i.
9. If p_i receives y_i from at least $2m/3$ players in P'' within 2δ time steps, it accepts the computation and otherwise sends an accusation $(p_j)_{p_i}$ to all players in P for any $p_j \in P_i$ that did not reply correctly or in time.

We define the random number generation of p_i to be *successful* if p_i receives the same key from at least $2m/3$ many players in step (9). This is important for p_i since it will need the support of at least $2m/3$ other players for further operations that we will discuss in the next section.

2.3 Analysis of the round-robin RNG

The round-robin RNG has the following performance.

Theorem 3. *Suppose that $|P| = m$ and there are $t < m/6$ adversarial players in P. Then the round-robin RNG generates random keys $y_1, y_2, \ldots, y_k \in \{0,1\}^s$ with $m - 2t \leq k \leq m$ and the property that for all subsets $S \subseteq \{0,1\}^s$ with $\sigma = |S|/2^s$,*

$$\mathrm{E}[|\{i \mid y_i \in S\}|] \in [(m-2t)\sigma, m \cdot \sigma] \,.$$

The worst-case message complexity of the protocol is $O(m^2)$.

In order to prove the theorem, we start with some simple claims.

Some basic facts. Because of the flooding strategy in step (2) and the definition of δ it holds:

Claim. No matter whether p^* is adversarial or not, all honest players start the protocol within δ steps.

Since each honest player p_i needs at most 7δ time steps to complete the protocol from step (3) to (9) and starts after waiting for $i \cdot 8\delta$ steps, the claim above implies the following claim.

Claim. No two honest players execute their random number generation scheme (steps (3) to (9)) at the same time.

Hence, honest player p_i can make use of the accusations of all honest players p_j with $j < i$ in order to keep its own problems with the random number generation as small as possible.

Next, we bound the size of any P_i for an honest player p_i. Recall that honest players are supposed to work in a correct and timely manner. Hence, honest players will never accuse other honest players of any wrongdoing but only adversarial players. Since every adversarial player can issue at most one accusation, there will be at least $m-2t$ honest players left in every set P_i of an honest player p_i throughout the protocol. Hence, we get:

Lemma 1. *If $t < m/6$ then $|P_i| \geq 2m/3$ throughout the protocol for every honest player p_i.*

Moreover, every player p_i can only be successful for one key. This is because all players in P_i have to see commitments to the same P_i for all players in P_i and $|P_i| \geq 2m/3$ before revealing their random keys in step (6). Since $t < m/6$, this means that there must be more than $m/2$ honest players in P_i, which can only be possible for at most one P_i. Hence, we get.

Lemma 2. *If $t < m/6$ then every player can be successful for at most one key.*

Analysis of steps (3) to (9). Next, we focus on the execution of steps (3) to (9) by some fixed peer p_i. First, we consider the case that p_i is honest, and then we consider the case that p_i is adversarial.

Lemma 3. *If p_i is honest and $|P_i| \geq 2m/3$, then no matter how many adversarial players there are in P_i, if the protocol terminates successfully, then the key y_i generated by p_i is distributed uniformly at random in $\{0,1\}^s$ and all honest players in P_i compute the same key as p_i.*

Proof. p_i will not reveal x_i before the keys in P_i have all been revealed. Hence, the probability distribution on $z = \bigoplus_{p_j \in P_i} x_j$ must be independent of x_i. But for any probability distribution on $z = \bigoplus_{p_j \in P_i} x_j$ that is independent of x_i it holds that if x_i is chosen uniformly at random in $\{0,1\}^s$, then also $y_i = x_i \oplus z$ is distributed uniformly at random in $\{0,1\}^s$. Moreover, also the decision of the adversarial players to let the random number generation fail must be independent of x_i and can only be a function of z because x_i will not be revealed before. Hence, it holds for any adversarial strategy and any $y^* \in \{0,1\}^s$ that

$$\Pr[y_i = y^* \mid \text{generation of } y_i \text{ successful}] = \Pr[y_i = y^*] = \frac{1}{2^s}$$

If p_i succeeds with computing y_i, then it informed all players in P_i about the revealed keys, and all honest among them will accept these keys since they match the message sent out by p_i in step (5). Hence, all honest players in P_i compute the same key as p_i. □

Notice that if the adversarial players knew about x_i *before* deciding to let the random number generation fail, they can create a significant bias, even if the other keys were chosen independent of x_i. A simple example for this would be:

Focus on any fixed $y^* \in \{0,1\}^s$. If $y_i = y^*$, then let the attempt fail, and otherwise let it be successful.

It is easy to see that this would make it very unlikely for the round-robin RNG to generate y^* (since it would have to be generated more than t times to be successful at least one). Hence, it is crucial that x_i is only revealed *after* all the other keys have been revealed. Next, we consider the case that p_i is adversarial.

Lemma 4. *If p_i is adversarial, then no matter what p_i and the other adversarial players in P_i do, whenever an honest player p_j reveals its key x_j, $y_i^{(j)}$ has a uniform distribution on $\{0,1\}^s$.*

Proof. An honest player p_j will only reveal x_j once it receives $(\{(p_i, h(x_k), P_i)_{p_k} \mid p_k \in P_i\})_{p_i}$ from p_i and $p_j \in P_i$ (so that $y_j^{(i)}$ is well-defined). In this case, x_j is a random number that is independent of $z = x_i \oplus \bigoplus_{p_k \in P_i \setminus \{p_j\}} x_k$, and since x_j is independent of z and chosen uniformly at random, $y_j^{(i)} = x_j \oplus z$ has a uniform distribution. □

Notice, however, that p_i can commit to different sets P_i to different honest players without being detected, so the keys $y_j^{(i)}$ can differ among the honest players. Nevertheless, if p_i wants to be successful (i.e., collect commitments to the same key from at least $2m/3$ many players), it must let more than $m/2$ honest players p_j succeed with computing the same $y_j^{(i)}$, which has a uniform distribution.

Still, the adversarial players can create a bias on the *successfully* computed keys since after knowing y_i, an adversarial player p_i still has the option to let the key generation be successful or not. Fortunately, this bias cannot be too large, as shown in the following lemma.

Analysis of the entire protocol.

Lemma 5. *If $t < m/6$ then at least $m - 2t$ of the $m - t$ random number generations initiated by the honest players are successful, irrespective of whether p^* is adversarial or not. Furthermore, it holds for all subsets $S \subseteq \{0,1\}^s$ with $\sigma = |S|/2^s$ that $\mathrm{E}[|\{i \mid y_i \in S \text{ for a successful } y_i\}|] \in [(m-2t)\sigma,\ m \cdot \sigma]$*

Proof. According to Lemma 4, every key y that an honest player p commits to must be distributed uniformly at random in $\{0,1\}^s$. However, whereas the adversarial players can adaptively abort the random number generation initiated by adversarial players, it follows from Lemma 3 that they can only do this in an oblivious way for the honest players. We know that the adversarial players can only sabotage the random number generation of at most t honest players. Hence, at least $m-2t$ random number generations of honest players p_i will be successful, and their success does not depend on their values. Thus, the probability for any of these players p_i that $y_i \in S$ is equal to σ and, therefore, the expected number of successful p_i's with $y_i \in S$ is at least $(m-2t)\sigma$.

On the other hand, at most m key generations can be successful, and since every successfully generated key y_i is distributed uniformly at random in $\{0,1\}^s$, the probability for any y_i to be in S is equal to σ. Hence, the expected number of successful p_i's with $y_i \in S$ is at most $m \cdot \sigma$. □

The next lemma follows immediately from the protocol.

Lemma 6. *The message complexity of the round robin-random RNG is $O(m^2)$.*

2.4 Extensions

In our random number generator we assumed that the players in P know each other and the indexing. This assumption can be problematic in peer-to-peer systems since there might be disagreement among the honest players about the set of adversarial players in P. However, if at least all the honest players know each other and there are, for example, less than $m/8$ adversarial players in P, this can easily be fixed. Suppose that every honest player p_i uses a threshold of $3m_i/4$ instead of $2m/3$, where m_i is the number of players that p_i knows in P and $m = |P|$. Then all results above still hold since $3m_i/4 \geq (3/4) \cdot (7m/8) > m/2$.

Another problem is how to fix the indexing issue. When there is disagreement about P, it will not be possible for the honest players to agree on a common indexing scheme. Instead, they can use the following simple trick. Each player p_i picks a random slot out of $c \cdot m_i$ many slots for generating a random number, where c is a fixed constant. Then it is easy to calculate that the number of slots occupied by the honest players is at least $(1 - 1/(2c))m_h$, where m_h is the number of honest players. Hence, the adversarial players could manage now to let up to $t + m_h/(2c)$ random number generations of honest players fail instead of just t, which is still acceptable if c is sufficiently large.

3 Application to robust peer-to-peer networks

In this section we show how to use the round-robin random number generator above to satisfy the balancing and majority conditions for any adversarial join-leave strategy for a polynomial number of rejoin operations, with high probability. We start with a formal model. Then we present the de Bruijn cuckoo rule, and afterwards we combine it with the round-robin RNG to obtain the round-robin cuckoo rule.

3.1 Model

Recall that we want to associate all peers with points in $[0, 1)$. These points can be encoded as binary strings from $\{0, 1\}^s$ (in a sense that $b = (b_1, \ldots, b_s)$ represents $x_b = \sum_{i \geq 1} b_i/2^i$) for a sufficiently large s (in SHA-1, which is used by the Chord system, for example, $s = 160$).

There are n blue (or honest) nodes and ϵn red (or adversarial) nodes for some fixed constant $\epsilon < 1$. There is a rejoin operation that, when applied to node v, lets v first leave the system and then join it again from scratch. The leaving is done by simply removing v from the system and the joining is done with the help of a join operation to be specified by the system. We assume that the sequence of rejoin requests is controlled by an adversary. The adversary can only issue rejoin requests for the red nodes, but it can do this in an arbitrary adaptive manner. That is, at any time it can inspect the entire system and select whatever red node it likes to rejoin the system. The goal is to find an *oblivious* join operation, i.e., an operation that does not distinguish between the blue and red nodes, so that for *any* adversarial strategy above the balancing and majority conditions can be kept for any polynomial number of rejoin requests.

3.2 The de Bruijn cuckoo rule

Recall the original cuckoo rule in Section 1.5. We present a slight but crucial modification to this rule, called the *de Bruijn cuckoo rule*, which only needs two random numbers in $\{0,1\}^s$, irrespective of k. The prefix *de Bruijn* was chosen because the rule can be easily implemented in dynamic de Bruijn graphs (e.g., [18]).

de Bruijn cuckoo rule: If a new peer v wants to join the system, pick random $x, y \in [0,1)$. Place v into x and replace all peers in $R_k(x)$ in the following way. If $|R_k(x)| = 0$, we are done, and if $|R_k(x)| = 1$, then the peer in $R_k(x)$ is moved to position y. Otherwise, let $b = \lceil \log |R_k(x)| \rceil$. Given that y is represented by a binary string $(y_1, \ldots, y_s) \in \{0,1\}^s$, peer $i \geq 0$ in $R_k(x)$ is moved to position $((y_{s-b+1}, \ldots, y_s) \oplus (i)_2) \circ (y_1, \ldots, y_{s-b})$ where $(i)_2$ represents the binary representation of i and $\circ$ the concatenation.

For example, suppose that $y = 0100110$ and $|R_k(x)| = 3$. Then the new positions of the three peers are $(10 \oplus 00) \circ 01001 = 1001001$ for peer 0, $(10 \oplus 01) \circ 01001 = 1101001$ for peer 1, and $(10 \oplus 10) \circ 01001 = 0001001$ for peer 2. This rule of mapping peers to new points has the following property:

Lemma 7. *Every replaced peer is moved to a position that is distributed uniformly at random in* $\{0,1\}^s$.

Proof. Consider peer i in $R_k(x)$ for any fixed i and suppose that y is distributed uniformly at random in $\{0,1\}^s$. Then $(y_{s-b+1}, \ldots, y_s) \oplus (i)_2$ is distributed uniformly at random in $\{0,1\}^b$ and $(y_1, \ldots, y_{s-b})$ is distributed uniformly at random in $\{0,1\}^{s-b}$, resulting in the lemma. □

Moreover, any two peers in a region $R_k(x)$ with p peers have a distance of at least $(1/2)^{\log p - 1} \geq 1/(2p)$ of each other. Hence, when looking at the analysis in [4], it turns out that all results still hold when using a perfect random number generator (though in Lemma 2.6 and Lemma 2.10 the independence property of the new node positions has to be replaced by negative correlation).

Theorem 4. *For any constants ϵ and k with $\epsilon < 1 - 1/k$, the de Bruijn cuckoo rule with parameter k satisfies the balancing and majority conditions for a polynomial number of rounds, with high probability, for any adversarial strategy within our model. The inequality $\epsilon < 1 - 1/k$ is sharp as counterexamples can be constructed otherwise.*

3.3 The round-robin cuckoo rule

Finally, we show how to combine the de Bruijn cuckoo rule and the round-robin random number generator into a simple and efficient join protocol called *round-robin cuckoo rule* that achieves a result similar to Theorem 4.

Recall the definition of a region in Section 1.5. Given a node $v \in [0,1)$, we define its *quorum region* R_v as the unique region of size closest from above to $(\gamma \log n)/n$, for a fixed constant $\gamma > 1$, that contains v.

We demand that whenever a new node u wants to join the system, it has to do so via a node v already in the system. v then initiates the following protocol:

1. v initiates the round-robin RNG in R_v (i.e., v acts as p^*).
2. For each successful node $v_i \in R_v$, v_i initiates the de Bruijn cuckoo rule by sending a message $(y_i, \{(y_j^{(i)})_{p_j} \mid v_j \in P_i\})_{p_i}$ with $1 + 2m/3$ signed keys to all nodes in R_v.
3. Once node $w \in R_v$ receives a correctly signed $(y_i, \{(y_j^{(i)})_{p_j} \mid v_j \in P_i\})_{p_i}$ containing more than $2m/3$ keys, it forwards it to all other nodes in R_v and initiates the de Bruijn cuckoo rule.

In the de Bruijn cuckoo rule, majority decision is done to execute the proper actions (see [4] for more details). Since step (3) ensures the "all or nothing" principle concerning the honest nodes, the de Bruijn cuckoo rule can be guaranteed to be executed in a correct and timely manner. The new node u can choose to assume any one of the new positions of a successfully executed de Bruijn cuckoo rule. It just needs to commit to one to R_v. If the node v just wants to rejoin the system (like in the adversarial strategies considered here), then we identify v with u.

3.4 Perturbation with biased randomness

Recall that we consider adversarial join-leave attacks in a system with n honest nodes and ϵn adversarial nodes. Consider the situation of using any m-RNG with bias β together with the round-robin cuckoo rule. Then it holds:

Theorem 5. *For any constants ϵ, k and β with $\epsilon < (1/\beta^2)(1/\beta^2 - 1/k)$, the round-robin cuckoo rule with an m-RNG with bias β satisfies the balancing and majority conditions for a polynomial number of rounds, with high probability, for any adversarial strategy within our model.*

Proof. (Sketch) Recall the proof in [4]. Lemma 2.4 holds as before. In Lemma 2.5, the age range of a quorum region has to be adjusted to $[((1-\delta)/\beta)(c \log n)(n/k), (1+\delta)\beta(c \log n)(n/k)]$, where the age of a k-region is defined as the number of (not necessarily successful) attempts of executing the de Bruijn cuckoo rule since it has last been hit by a new node. Lemma 2.6 still holds. In Lemma 2.9, the range for the number of evicted blue nodes in a time interval of size T has to be adjusted to $[((1-\delta)/\beta)T \cdot k, (1+\delta)\beta T \cdot k]$, and the range for the red nodes has to be adjusted to $[((1-\delta)/\beta)T \cdot \epsilon k, (1+\delta)\beta T \cdot \epsilon k]$. Technically, what needs to be shown is a reduction (via domination arguments) to sums of uncorrelated random variables so that Chernoff bounds for uncorrelated random variables can be used (e.g., [25], Section 3.3.3). With these bounds one easily obtains the theorem using the rest of the proof in [4]. □

Using the round-robin RNG in the analysis results in the improved bound $\epsilon < 1/\beta^2 - 1/k$ due to the one-sided bias of our RNG.

4 Conclusions

For the overlay network part, we only assumed a sequential execution of rejoin operations (see our model) though we expect that as long as not too many rejoin operations are executed concurrently, there should be only insignificant side effects (see also the comments in [26]).

Further interesting problems are, how to make a peer-to-peer network robust against adaptive join-leave behavior by both honest and adversarial peers, and how to incorporate mechanisms against low-level denial-of-service attacks by the adversarial peers that can shut down honest peers.

References

1. J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 14th ACM Symp. on Discrete Algorithms (SODA)*, pages 384–393, 2003.
2. B. Awerbuch and C. Scheideler. Group Spreading: A protocol for provably secure distributed name service. In *Proc. of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, 2004.
3. B. Awerbuch and C. Scheideler. Robust distributed name service. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.
4. B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. In *Proc. of the 18th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2006. See also http://www14.in.tum.de/personen/scheideler.
5. M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proc. of the 13th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 183–192, 1994.
6. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the 5th Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.
7. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the 2nd Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 1999.
8. S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security*, 2003.
9. J. R. Douceur. The sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
10. A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. In *Proc. of the European Symposium on Algorithms (ESA)*, 2005.
11. O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.
12. S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO 96*, pages 201–215, 1996.
13. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahi. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *29th ACM Symp. on Theory of Computing (STOC)*, pages 654–663, 1997.
14. V. King, J. Saia, V. Sanwalani, and E. Vee. Towards a secure and scalable computation in peer-to-peer networks. In *Proc. of the 47th IEEE Symp. on Foundations of Computer Science (FOCS)*, 2006.

15. F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.
16. M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.
17. M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
18. M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2003.
19. S. Nielson, S. Crosby, and D. Wallach. Kill the messenger: A taxonomy of rational attacks. In *Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.
20. G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 311–320, 1997.
21. H.V. Ramasamy and C. Cachin. Parsimonious asynchronous Byzantine-fault-tolerant atomic broadcast. In *9th Conference on Principles of Distributed Systems (OPODIS)*, pages 88–102, 2005.
22. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *USENIX Annual Technical Conference*, 2004.
23. T. Ritter. RNG implementations: A literature survey. http://www.ciphersbyritter .com/RES/RNGENS.HTM.
24. J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
25. C. Scheideler. *Probabilistic Methods for Coordination Problems*. HNI-Verlagsschriftenreihe 78, University of Paderborn, 2000. See also `http://www14.in.tum.de/personen/scheideler`.
26. C. Scheideler. How to spread adversarial nodes? Rotate! In *Proc. of the 37th ACM Symp. on Theory of Computing (STOC)*, pages 704–713, 2005.
27. A. Singh, M. Castro, A. Rowstron, and P. Druschel. Defending against Eclipse attacks on overlay networks. In *Proc. of the 11th ACM SIGOPS European Workshop*, 2004.
28. E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
29. K. Srinathan and C.P. Rangan. Efficient asynchronous secure multiparty distributed computation. In *Proc. of the 1st Int. Conference on Progress in Cryptology*, pages 117–129, 2000.
30. M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *Proc. of the 20th IEEE Computer Security Applications Conference (ACSAC)*, 2004.
31. 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. of the ACM SIGCOMM '01*, 2001. See also `http://www.pdos.lcs.mit.edu/chord/`.
32. J. Viega. Practical random number generation in software. In *Proc. of the 19th Annual Computer Security Applications Conference*, 2003.