Pace-Maker: Tracking peer availability in large networks

Fabrice Le Fessant, Cigdem Sengul, Anne-Marie Kermarrec

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Pace-Maker: Tracking peer availability in large networks

Fabrice Le Fessant — Cigdem Sengul — Anne-Marie Kermarrec

N° 6594

July 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6594--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

Pace-Maker: Tracking peer availability in large networks

Fabrice Le Fessant*, Cigdem Sengul*, Anne-Marie Kermarrec†

Thème COM — Systèmes communicants
Équipes-Projets Asap

Rapport de recherche n° 6594 — July 2008 — 30 pages

Abstract: Tracking peer availability in a peer to peer network is of utmost importance for many collaborative applications. For instance, such information is invaluable for identifying the most stable peers or group peers with similar uptime characteristics. However, as many applications tend to reward the most stable peers, there is a clear incentive for peers to attempt to lie about their uptime. In this paper, we present a scalable and lightweight protocol that enables nodes to measure the peer availability in the presence of malicious peers. In our protocol, each peer is in charge of maintaining its own availability over time by collecting heartbeats disseminated by a trusted entity using asymmetric cryptographic signatures. Hence, peers gain the ability to challenge other peers about the ability by checking that their real uptime matches the advertized one. Simulation results show that our protocol provides accurate availability measures, even when %XX of the peers in the network are malicious. Furthermore, as the percentage of malicious peers increase beyond %XX, our proposed system degrades gracefully.

Key-words: peer-to-peer, cryptography, availability, monitoring

* INRIA Saclay – Ile de France
† INRIA Rennes - Bretagne Atlantique

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Pace-Maker: mesure de disponibilité d'un pair dans les réseaux large échelle

Résumé : La mesure de disponibilité dans un réseau pair-à-pair peut revêtir une très grande importance pour beaucoup d'applications collaboratives. Ainsi, cette information est inestimable pour identifier les pairs les plus stables, ou les groupes de pairs similaires par leur disponibilité. Cependant, comme de nombreuses applications veulent récompenser les pairs les plus stables, il existe une incitation claire pour les pairs à mentir sur leur disponibilité réelle. Dans ce papier, nous présentons un protocol léger et scalabe qui permet aux nœuds de mesurer la disponibilité d'un pair en présence de pairs égoïstes. Dans notre protocole, chaque pair est chargé de maintenir sa propre disponibilité en collectant des pulsations disséminées par une entité de confiance en utilisant des signatures cryptographiques. Celles-ci permettent à tout pair de vérifier par des challenges les informations de disponibilité transmise par un pair.

Mots-clés : pair-à-pair, cryptographie, disponibilité, monitorage

1 Introduction

A peer-to-peer network is composed of thousands of independent computers, which aggregate their resources over the Internet to run collaborative distributed applications. Such networks are subject to high dynamics: computers (peers) may join and leave arbitrarily or be subject to frequent disconnections. However, it has been observed that peers with high availability in the system are more likely to remain in the system for a longer time [3, 10, 17]. As a consequence, many peer-to-peer networks rely on peer availability to measure the stability of peers, and use this parameter to select peers for specific purposes. For example, the most stable peers can be elected as *super-peers* [8], or as privileged peers to maintain replicas in storage systems [4, 2, 6].

Yet, to the best of our knowledge, current research does not address how availability can be continuously measured in an efficient and trustable way. As we discuss in Section 6, current systems either use expensive and incomplete measurement techniques, or rely on peers to give an honest estimation of their availability in the network. The fact that stable peers are usually rewarded in a peer to peer network provides them with a clear incentive to lie on their availability, in order to be granted a higher status in the network. Subsequently, malicious nodes may get access to more resources than they should be allowed to (free-riders) and are in a position to compromise the network even more. For instance, in a peer-to-peer backup system we are currently building [13], peers that lie about their stability might get undue and undeserved access to storage on the most stable peers in the system.

Motivated by these observations, in this paper, we present a simple lightweight and scalable protocol to measure availability in peer-to-peer networks. The details of our protocol are presented in Section 4. In our protocol, we assume that peers are connected into a redundant mesh [1] used to propagate heartbeats, generated by trusted peers, in the network. We describe our system model in more details in Section 2. In our protocol, peers randomly challenge each other to check out upon the accuracy of the availability claims. Hence, our protocol can be used by any peer to learn the availability of another peer in the system. The availability measure is of medium granularity, for instance, a table representing the hours where a peer has been connected to the network in a few weeks. The protocol prevents malicious peers from lying by increasing their availability in the system. We do not cope with collusions of malicious peers in the present paper.

Our main contribution is allowing each peer to learn the availability of another peer in the system in a completely distributed manner and through local communications (i.e., communication is only necessary with the neighbours) using standard cryptographic mechanisms . Our simulation results in Section 5 show that our system is cheap but yet still is more accurate than currently available systems. Although our system requires the existence of a set of privileged peers, due to its light load, the system remains highly scalable (i.e., the number of privileged peers is limited).

In this paper, we only consider the case of individual malicious nodes, which are trying to gain access to more resources than they are allowed to by lying

[1] Although we describe a simple mesh protocol in this paper for our simulations, many other options exist such as network-based on the peer sampling protocols [18] for example.

about their availability. Such nodes are called free-riders. We have not considered the case where malicious nodes may collude. We discuss our ongoing work on avoiding collusion as part of our future work in Section 7.

2 System Model

2.1 Network Model

In this paper, we consider a large-scale network (in the order of thousands of computers). The nodes (or peers) are connected by an underlying communication network, typically IP. Each network is aware of a small portion of the network, typically know the IP address of a subset of the network. This is typically known as an unstructured network. Although there is no specific assumption about the topology, we will come back later on the type of network we are using in our experimentations.

We assume the existence of a global clock, with which computer clocks are loosely coupled. This is necessary for a peer to know at which periods (the periods are considered system-wise) it was connected to the system. Therefore, this is important that peers have an *approximate agreement* on time.

Peers communicate by sending asynchronous messages. Although there is no bound on communication delays, we assume most messages are received after a short delay of Δ_{std} and are lost after a longer delay of Δ_{lost}. Although this is not a requirement, we expect computers to be *connected* with their neighbours, i.e. communicate through FIFO channels (TCP connections in practice). This helps to enforce sequentiality of some verification operations, although these operations could also be done in a completely disconnected manner.

Pace-Maker relies on the existence of a trusted entity. In this paper, we assume the existence of a set of particular peers, called the *servers* for . Pace-Maker does not require that non-servers peers know the identity of the servers nor of any other peer. However, it is assumed that the network is connected enough to ensure that every peer in the network is reachable from at least one of the servers. This can easily be achieved by an unstructured overlay. In this paper, we assume that the network is built as follows: each peer tries to connect to random peers in the network until its degree in the network reaches a given value (denoted D_{max}). Since servers have a specific role in the protocol, we allow them to have a higher degree than D_{max}. This allows, for instance, to prevent sybil attacks that try to circle them to disrupt the diffusion of heartbeats.

For the sake of simplicity in the rest of the paper, we consider a single-server system. This assumption does not affect our results, since there is no communication needed between the servers. Essentially, the only main requirement is that the server clocks are loosely synchronized with the global clock.

2.2 Cryptographic Model

We assume that peers have access to strong cryptographic primitives, in particular for public-private key operations, which are the following:

- **generate_pair():** generates a new pair of public-private keys. The common usage is that the private key, K_{priv}, is kept secret by the peer, while the public key, K_{pub}, is known to other peers in the system.

- **sign(`data`, $\mathbf{K}_{priv}$):** returns a signature for `data` using the private key K_{priv}.
- **verify(S, `data`, $\mathbf{K}_{pub}$):** verifies that S is a signature for `data` that was created using the private key K_{priv} associated with K_{pub}.
- **hash(`data`):** returns the hash of `data`.

We assume these operations provide a high level of security (i.e., it is almost impossible to break the cryptographic properties of these functions by such as having a collision in the hash function).

We assume that there is a special pair of keys, one public (called KS_{pub}) known by all peers in the system, one secret (called KS_{priv}) known only by the server. Each peer p in the system also owns a pair of keys, noted K^p_{pub} and K^p_{priv}, to sign data. We also define $\mathrm{H}^p =$ **hash($\mathbf{K}^p_{pub}$)**, and use it as unique identifier for p in the network. These keys are used by our system, but they should also be used by the peer-to-peer application to prevent malicious peers from easily changing their identity when they are detected by the availability measurement system. Finally, we assume there exists a way for peers to exchange their public keys, either using dedicated messages, or as a side-effect of communication protocols, such as done by TLS [5].

3 Pace-Maker in a nutshell

Pace-maker is a simple and lightweight protocol to track peer availability in a large-scale system. Each peer is in charge to track its own availability. As there is a natural trend to reward highly available nodes in a collaborative system, such a system must tolerate the presence of selfish nodes. The intuition is as follow: a server is in charge of periodically disseminating *pulses* in the system, say one per hour. Each peer maintain a table storing the identifier of the pulses it heard of. The list of such pulses represents its *presence sheet* in the system. Each peer is in charge of maintaining its own availability but Pace-maker relies on a decentralized verification procedure to check if peers lie on their availability.

A selfish peer might do the following

- not propagate the pulses
- claim it was connected while it was not.

Note that a malicious behavior consisting in keeping propagating pulses implies the presence of colluding peers and is out of the scope of this paper.

In order to tolerate selfish behavior, pulses are generated by the trusted entity, the server [2] as a a pair of public-private keys plus a signature. In order to prevent malicious nodes to get connected periodically only, the period to which pulses are generated is unpredictable. Such pulses are propagated in an epidemic manner in the system once by all the peers in the network to their neighbours. The signature certifies the association of the propagated public key and the current time. Every peer stores this information, as a proof of its presence in the network at the time of a pulse.

[2] Note that Pace-Maker can easily be implemented with a set of servers rather than a single server.

When peers provide their availability to other peers, they are potentially subject to receive a *challenge*, A peer A challenges a peer B to verify that peer B was indeed connected at the times it claims. To this end, A asks B to provide the signature and the public key for a given pulse, and to sign the specific record with the associated private key. A peer claiming that it was available at a specific time, and therefore received teh associated pulse, should be able to prove it by sending the correct information regarding that pulse.

Pace-Maker then provides a peer-to-peer network with a way, for each peer, to know for another peer:

- how long the other peer has been online in the network, and
- how long the two peers have been online *together* in the network, i.e. maybe not connected to each other, but still connected to the network at the same time.

The granularity of the availability measure is in unit time P, which is typically 1 hour. For each peer, we want to provide a history of length N_t, which is assumed to be some weeks or months. To reduce the communication cost, peers send challenges probabilistically depending on the advertised availability.

Dealing with *liars* is out of the scope of this paper. yet a simple way to deal with them is to disseminate their identity in the system to exclude them.

4 Pace-Maker Protocols

Pace-Maker relies on three sub-protocols: *(i)* the dissemination protocol used to disseminate the pulse generated by the server; *(ii)* the inquiry protocol used by a peer to ask for another peer's availability and, *(iii)* the verification protocol used by a peer to challenge another peer.

4.1 Pulse dissemination protocol

The server in Pace-maker is in charge of generating one *pulse* over a given period of time and disseminating it through the system. The dissemination is epidemic and consists for each peer to forward it once to all its neighbours. An example of the pulse dissemination is depicted on Figure 1. The figure shows a redundant mesh network, where there exist multiple paths between every two peers in the network. A redundant mesh ensures that even if one selfish peer does not follow the protocol aHence, the effect of malicious nodes that try to block the propagation of the heartbeat to non-malicious nodes is reduced.

A heartbeat T^i generated at time i is a tuple (i, K^i_{pub}, K^i_{priv}, S^i), where K^i_{pub} and K^i_{priv} is a new fresh public-private key pair. The public-private key pairs can be generated by the server on-demand, or just chosen among a huge set of pre-computed keys, for example, as in a sensor-network. The heartbeat also includes S^i, which is a signature of (i, K^i_{pub}) using the servers' private key KS_{priv}. Figure 2 depicts heartbeat diffusion at the server.

Every peer keeps a collection of N_t of these heartbeats, representing its presence in the network during the period of time (N_t * P). The actions taken by a peer on receipt of heartbeat is shown in Figure 3.

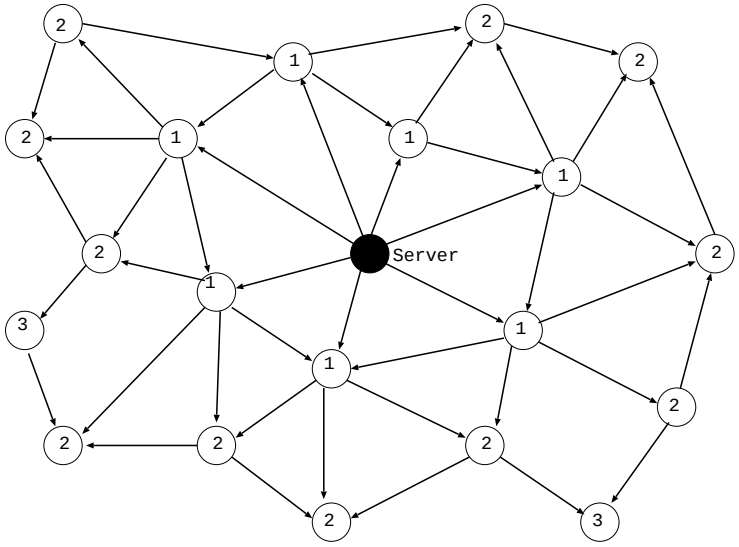

Figure 1: All peers are connected through a redundant mesh to the server. Here, the number in every node indicates the shortest distance to the server in the mesh. Every hour, a new heartbeat is propagated in the mesh by the server.

```
Server at round $i$:
  let ($K^i_{pub}$, $K^i_{priv}$) = generate_pair();
  let S$^i$ = sign( <i, $K^i_{pub}$>, $KS_{priv}$ );
  let T$^i$ = HeartBeat($i$, K$^i_{pub}$, K$^i_{priv}$, S$^i$);
  $\forall q \in$ neighbours$^{server}$, send( $q$, T$^i$ );
```

Figure 2: Hearbeat generation at the server.

```
Node receiving T$^i$= HeartBeat($i$, K$^i_{pub}$, K$^i_{priv}$, S$^i$):
  if
    History[i] = $\emptyset$ and
    verify(S$^i$, <i, $K^i_{pub}$>, $KS_{pub}$ )
  then
    History[i] := T$^i$;
    $\forall q \in$ neighbours$^p$, send( $q$, T$^i$ );
  end if
```

Figure 3: Heartbeat diffusion by a peer.

```
Node $p$ sending to $q$ its availability at round $i$:
  let bits = new bitfield[$T_n$];
  for $x$ in [1..$T_n$]
    if History[$i-T_n+x$] = $\emptyset$ then
      bits[$x$] := 0
    else
      bits[$x$] := 1
    end if
  end for
  let S = sign( < $i$, bits >, K$^p_{priv}$ )
  send( $q$, Availability($i$,bits,S) );
```

Figure 4: Peer availability announcements.

4.2 Inquiry Protocol

Depending on the application, peers need to be able to check the availability of other peers, either regularly or just the first time they connect to each other. This is done by comparing the histories of heartbeats they have collected. For this purpose, each peer sends a message Availability(i,bitfield,S), where bitfield is an array of bits of size N_t, containing, for each period P, 1 if it has the heartbeat, and 0 otherwise. The message also contains a signature of the bit field using the peer private key, K^i_{priv}. This signature can be used to prove later that the message was sent by the peer, in particular, if the peer does not reply to a challenge. The Figure 4 depicts the pseudocode for peer availability announcements.

Using the bit field of the other peer, a peer can:

- compute an approximation of the availability of the peer during the period (N_t * P). Basically, it just computes how many bits are set to one. Note that, in fact, it only proves that the peer was online when the heartbeats where propagated, not during complete hours. However, we show in our simulations that sending the heartbeat at random time in the period P provides a very good approximation of the real availability.
- compute a matching between its own online periods and the other peers' online periods. For that, it just does a logical "and" operation between his own bit field and the other bit field, and count the number of bits remaining.

4.3 Verification Protocol

It is in the interest of some peers to lie about their time online, especially to get more resources than they deserve. We thus provide a verification scheme to allow a peer to verify that the bit field received from another peer is correct: from time to time, the peer selects one bit set to one in the bit field received from another peer. It sends a special request Challenge(i, String), containing i, the time of the bit to be verified, and a nonce, which is a randomly generated short string "String" to make the challenge unique.

On reception of Challenge(i, String), the peer replies Proof(i, String, K^i_{pub}, S^i, S'^i) where K^i_{pub} and S^i are respectively the public key and the signature from heartbeat T^i, and S'^i is the signature of String using the private key K^i_{priv}.

```
Node $p$ receiving Challenge($i$, nonce) from node $q$:
  let HeartBeat($i$, K$^i_{pub}$, K$^i_{priv}$, S$^i$) = History[$i$]
  let reply = sign( < nonce, H$^p$, H$^q$ >, K$^i_{priv}$ )
  send( $q$, Proof($i$, nonce, K$^i_{pub}$, S$^i$, reply) );
```

Figure 5: Peer on receiving a challenge.

```
Node $q$ receiving Proof($i$,nonce,K$^i_{pub}$,S$^i$, reply)
       from node $p$:
  if
     ($i$, nonce, $p$) $\in$ challenges and
     verify(S$^i$, <i, $K^i_{pub}$>, $KS_{pub}$ ) and
     verify(reply, < nonce, H$^p$, H$^q$ >, K$^i_{pub}$ )
  then
     good_reply($p$)
  else
     bad_reply($p$)
  end if
```

Figure 6: Peer on receiving a verification in response to a challenge.

On reception of Proof(i, String, K^i_{pub}, S^i, S'^i), the peer can verify that S'^i has been signed with the correct key K^i_{priv} using the key K^i_{pub} from the message, and can also verify that K^i_{pub} is the public key from the heartbeat T^i using the signature S^i and the known KS_{pub} key.

The measures to be taken when a peer fails to provide a correct reply to a Challenge challenge is out of the scope of this paper, as it mostly depends on the application using our measurement system. However, our protocol is designed so that it is possible to propagate both the Availability message and the Challenge challenge to other peers in the network. Hence, other peers are allowed to use an unreplied Challenge message to challenge the same peer again. To avoid false claims, the peers might use the Availability messages to check that the message was indeed signed by the malicious peer. If the peer was not malicious, and its inability to reply to the challenge was due to a failure, the peer may be expected to clear its availability history as a sign of good will. Figures 5 and 6 depict the pseudo-codes for challenge and verify algorithms.

5 Evaluation

In this section, our goal is to evaluate the efficiency of our availability monitoring protocol. Specifically, we show that our system is:

- *Accurate*: The error between the measured availability for a peer and its real availability is negligible.
- *Scalable*: It is able to work with millions of peers connected together.
- *Secure*: Malicious peers in the system should not be able compromise our measure.

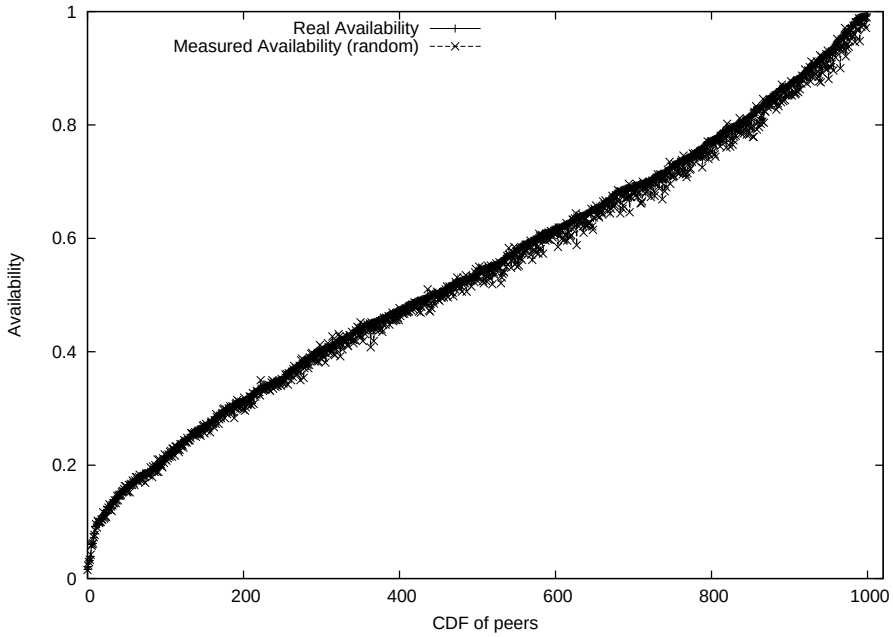

Figure 7: Measured availability compared to real availability. Note that the difference between the real availability and the availability measured using heartbeats is very close.

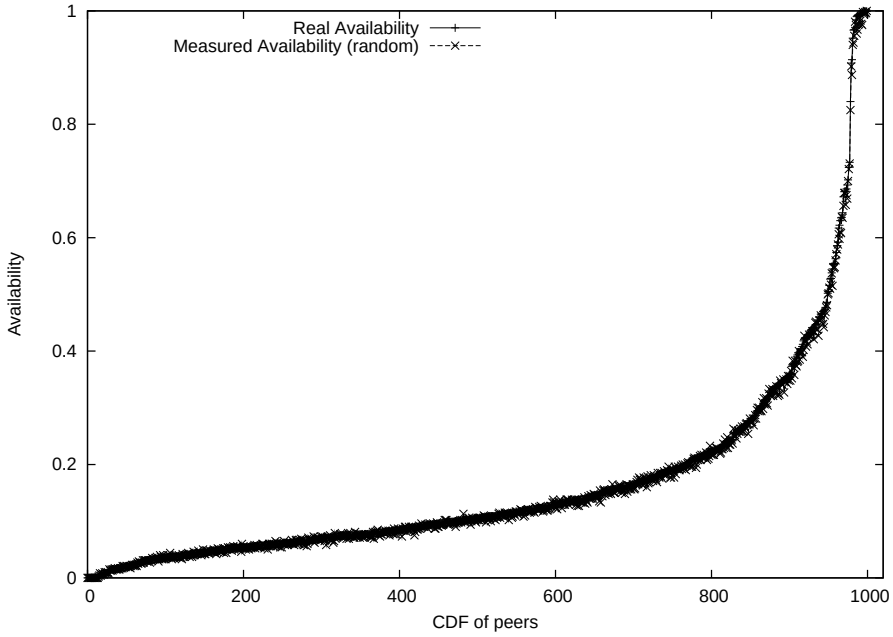

Figure 8: Measured availability compared to real availability. Note that the difference between the real availability and the availability measured using heartbeats is very close.

Table 1: Simulation Parameters

Network size:	1000 peers
Time:	28800 minutes (20 days)
Peer availability:	Unif. or Expon. Dist.
Disconnections per day:	Unif. Dist. in [1;10]
Timezones:	Unif. Dist. in [0;12]
Server Degree:	10 children
Peer Degree:	5 children

- *Efficient*: It is less expensive than other systems providing a similar measure.

5.1 Simulation Protocol

We simulate a network of peers connected in a mesh around a server. The server diffuses heartbeats every hour (P = 1 hour) to measure the availability of peers. Note that the mesh protocol is not part of our protocol: we show that a scalable mesh verifying our assumptions exists (see Section 2), but our protocol can be used with any other mesh verifying the same assumptions, depending on the application.

In the current simulation, the mesh protocol works as follows: the server has a degree of 10 children, and each peer has a degree of 5 children and 5 parents. To connect to the mesh, a peer first queries the server, which replies with a list of its children. The peer then queries the children. Every child either accepts the peer as a child, or sends a random child among its children. The process iterates until the peer is connected to 5 different parents.

We take the following local decisions to improve the system:

- At every round, if a peer has a free child slot, it chooses among all the candidates, who queried it during the round, the one with the best measured availability.
- A peer disconnects its children which are at the same distance from the server.

5.2 Simulation Parameters

Every round in the simulation takes one minute. From a communication point of view, it allows us to assume some timeout on messages that allows detection of peer disconnection (for instance, TCP keepalive is 30 seconds).

The parameters used in our simulations is shown in Table 1.

For each peer, we choose an availability and a number of disconnections per day. We then compute the probabilities being online and offline using a Markov chain. We use these probabilities in the simulation.

We compute the simulations using either a uniform distribution of availability (see Figure 7) or an exponential distribution (see Figure 8). We use the uniform distribution to show the accuracy of the measure for all values of availability, while we use the exponential distribution as more representative of real peer-to-peer systems [1].

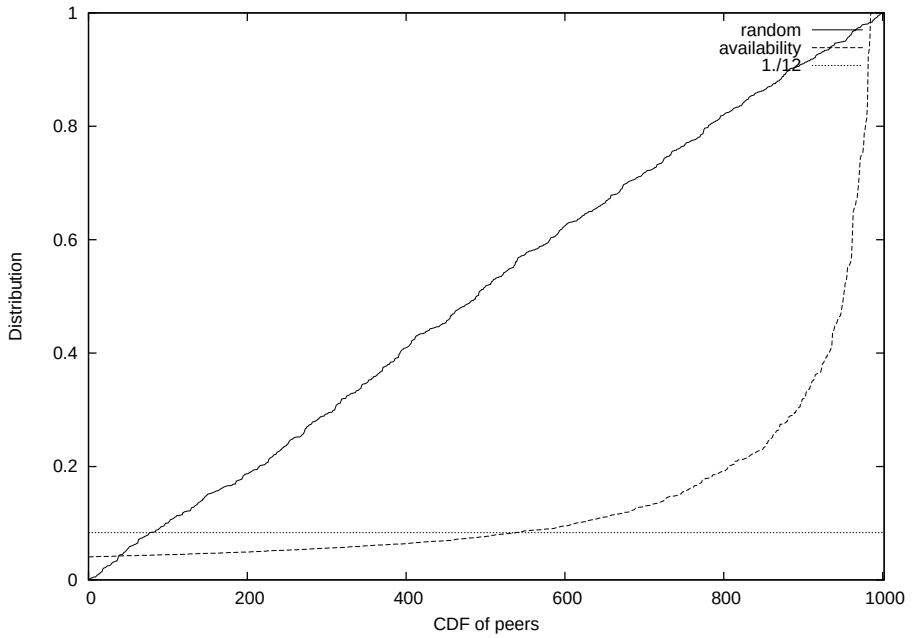

Figure 9: The exponential distribution of availability used in our simulations:$max(0.02, min(1, e^{1-log(2+65x)}))$ The median availability is close to 2 hours.

Timezones are used to obtain a diurnal pattern: during the day, peers have twice their normal probability of coming online and half their normal probability of going offline.

See Figures 10 and 11 for the resulting distributions of session lengths and online peers.

5.3 Accuracy of the Protocol

For each peer in the network, we computed the difference between its real availability and the availability measured by our system, i.e. the availability that it is able to prove to other peers. Figures 7 and 8 plot the availability for our uniform and exponential settings, and show that our measure is globally close to the real value. Figures 12 and 13 plot the absolute error: in the worst case, the random distribution, for 70% of the peers, the error is less than 1% of availability, while for the other 30 %, it is less than 5% of availability. Such an error is acceptable in the applications where we plan to use our system, since it does not change the *class* of availability of a peer.

It is also interesting to compare our measure with what would be obtained using other systems of measure. Thus, we simulated a system where every peer in the network would be monitored every minute using pings by a small set of *observers*. We compared our measure with the sum of the measures obtained by the observers. Although Figure 14 shows a very small error with a small number of observers for the ping-based system, Figure 15 shows that, in a more realistic

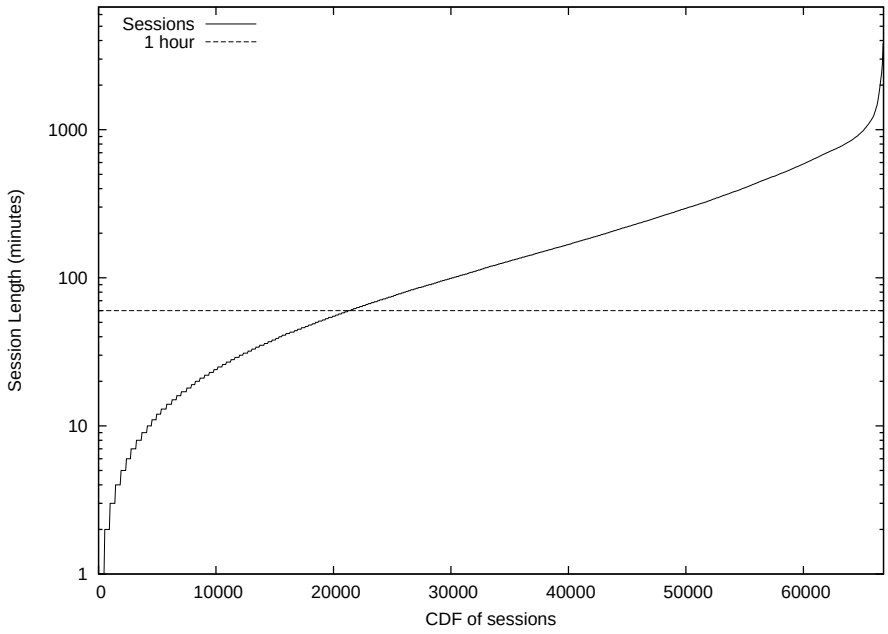

Figure 10: CDF of Sessions lengths. The median session length is around two hours.

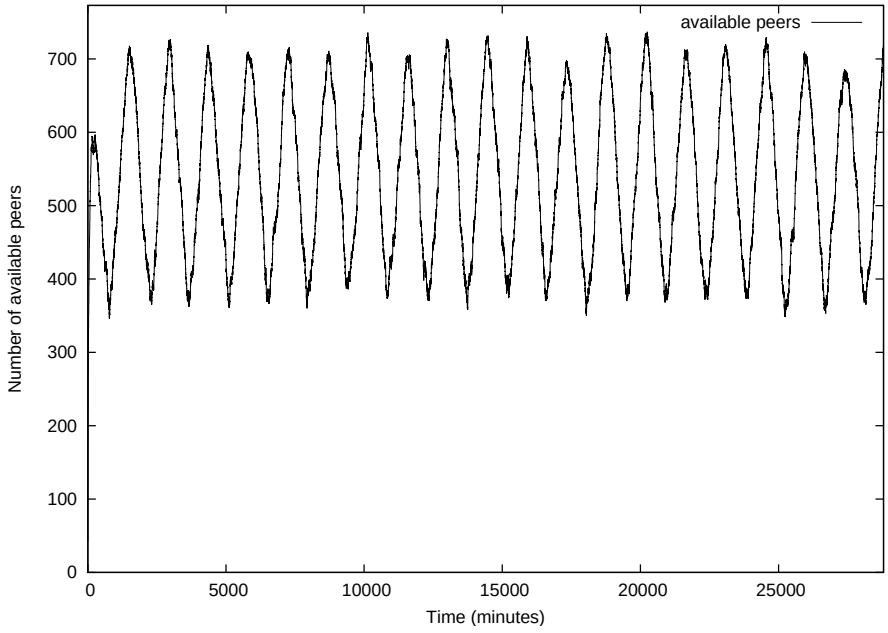

Figure 11: Number of peers online over time. Since the timezones of peers are only on 12 hours, the number of peers follow a diurnal pattern, which would not be the case if the distribution was over 24 hours.

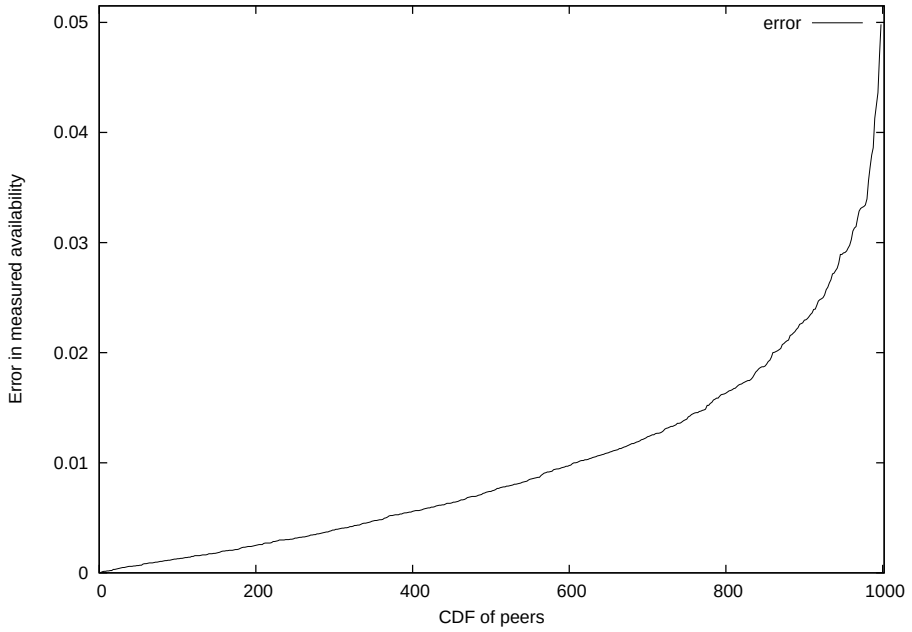

Figure 12: Error on measured availability. Error remains under 1% for 80% of the peers in the network.

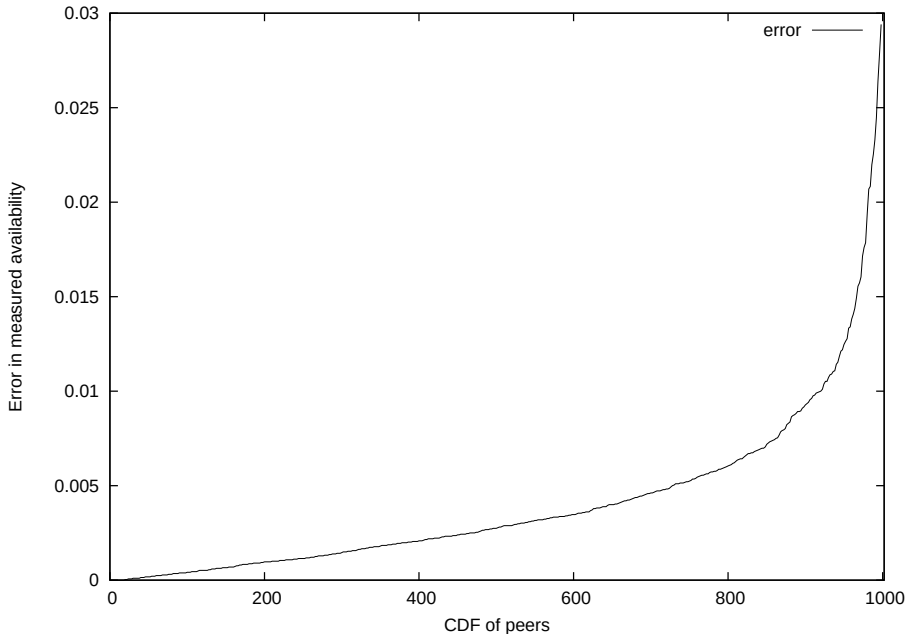

Figure 13: Error on measured availability. Error remains under 1% for 80% of the peers in the network.

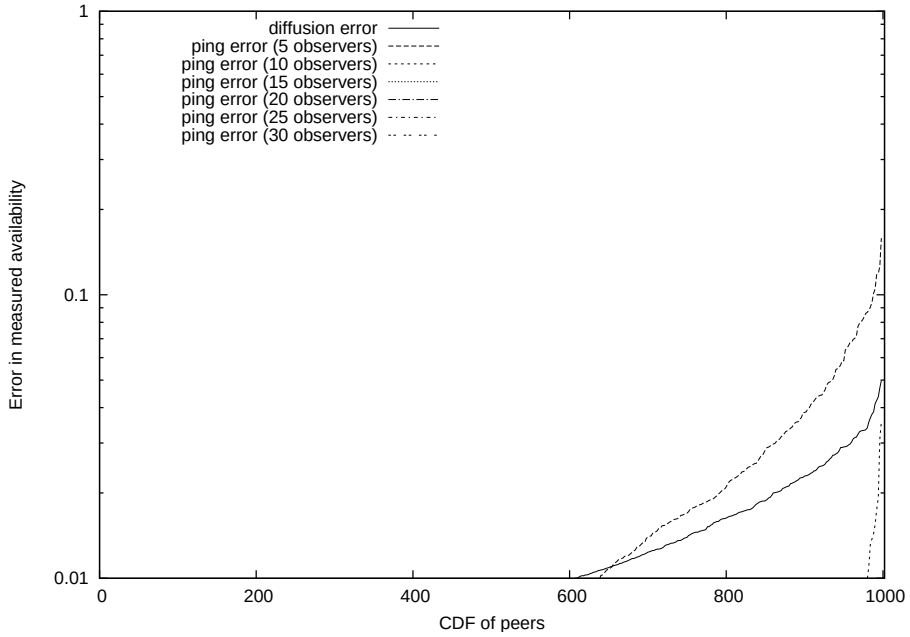

Figure 14: A comparison between our measure (diffusion) and the measure obtained using pings. Every peer is observed by a limited set of random observers, pinging every minute when they are online, and the measure is centralized. Our measure is equivalent to between 5 and 10 observers for a uniform distribution of availability.

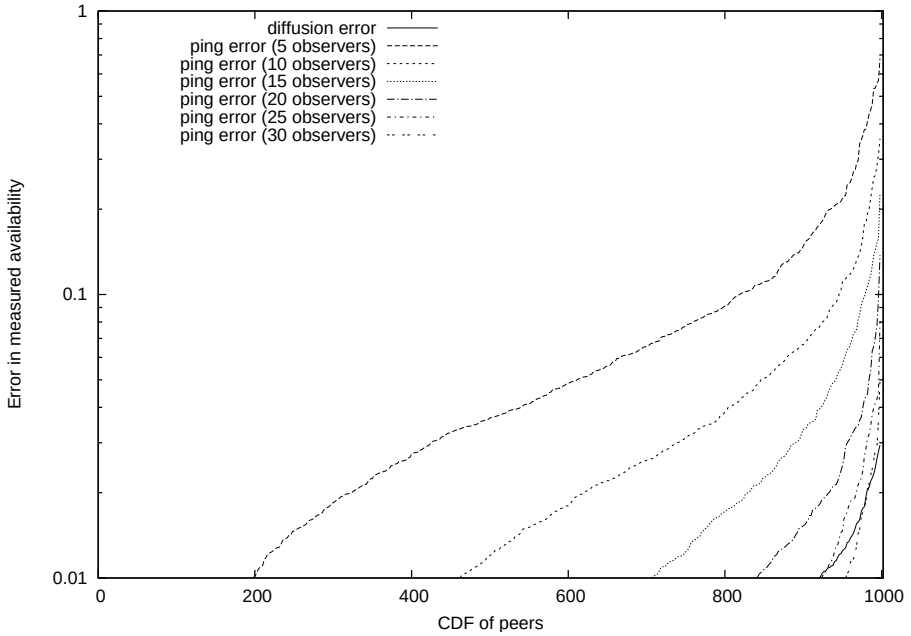

Figure 15: A comparison between our measure and the measure obtained using pings (2). With an exponential distribution of availability (closer to observed peer-to-peer systems), our measure performs better than a system with 20 observers. Such a system would cost 20×60 messages per hour of availability, while our system costs the degree of the mesh (5 in our simulations) per hour.

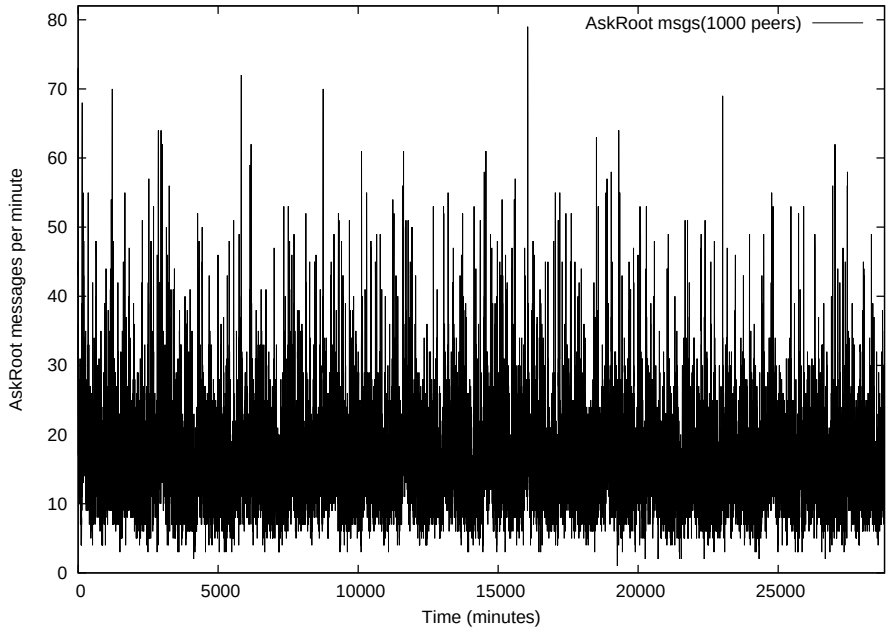

Figure 16: The number of AskRoot messages received by the server per minute in our simulations. The mean rate is 1/50 of the number of nodes, i.e. 100,000 nodes consume 33 messages per second. Even in our simplified mesh, a few servers can easily handle a few millions of peers.

exponential distribution, the ping-based system must use a lot of observers to reach our accuracy. Indeed, in such a system, no measure is done when all the observers are down, which might be often the case.

5.4 Scalability of the Mesh

The scalability of our system mostly relies on the existence of a scalable mesh, to diffuse the heartbeats in the network. Although this mesh is not part of our protocol, we just show in this section that the one we used in our simulations is such a scalable mesh, i.e. it scales well with the number of peers. Peer-to-peer application programers should design their own mesh depending on other services they want to provide.

In our mesh, every peer connecting to the network requests a list of peers from the server. From there, it iterates on these peers to find a set of parents in the network. Figure 16 plots the number of messages received by the server by minute. The mean rate is N/3000 messages per second, where N is the size of the network. 100,000 peers would thus generate a traffic of less than 10 kB/s on the server.

Figure 17 plots the maximal distance to the server. As expected, it grows logarithmically with the number of peers in the network. Figure 18 shows the delay for a new peer in the network to find its first parent, i.e. to be able to

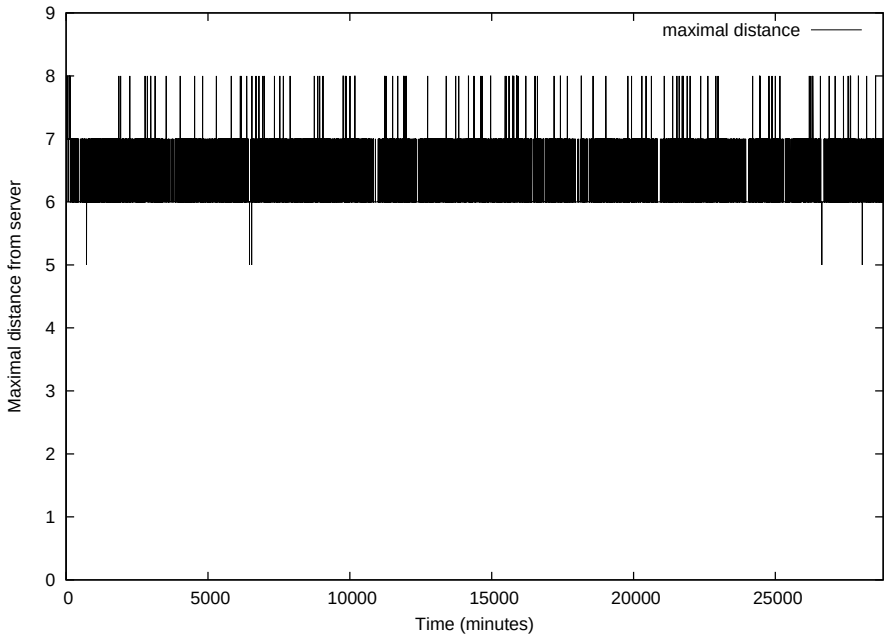

Figure 17: Maximal distance to the server over time. Note that the diameter of the network is not too high.

receive a heartbeat. It shows that the delay is usually neglectible, since we are only interested in measuring availability for long sessions.

Finally, Figure 19 plots the mean availability of peers depending on their distance to the server in our mesh. It is an interesting side-effect of our way of building the mesh, peers close to the server have a higher availability than peers on the external rings. This property of our mesh comes from the fact that, among all the peers applying to become a child of a node during a round, the node always selects the peer with the highest availability. It was not the case without this selection.

5.5 Malicious Peers

In the following, we try to evaluate how our system handles the following malicious behaviors:

- Selfish peers: these peers don't propagate the heartbeat message in the mesh, so that other peers have a smaller measured availability.
- Liars: these peers lie about their availability.
- Opportunist peers: these peers only connect to the mesh to get the heartbeats and immediately disconnect.

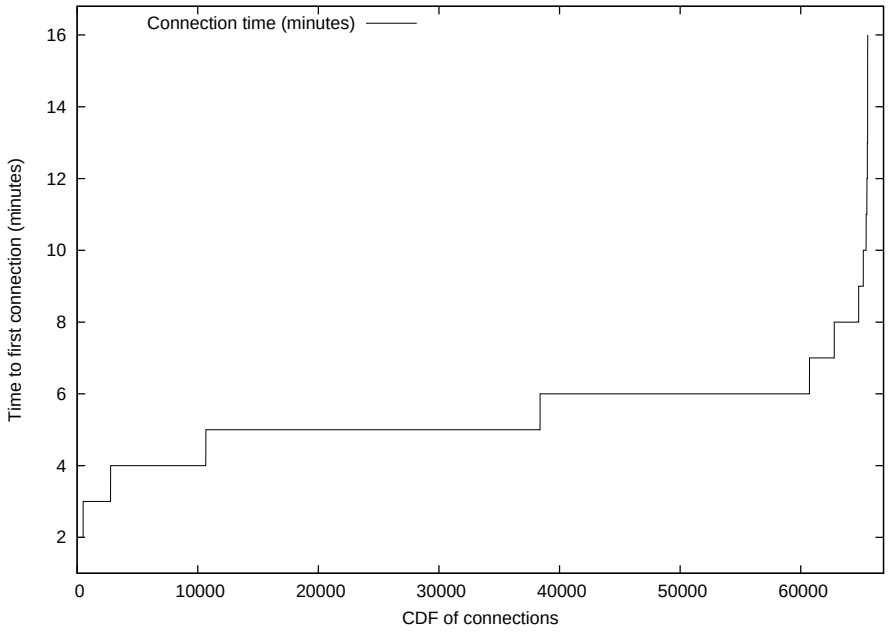

Figure 18: Time spent between the beginning of a peer session and the connection to its first parent. Only in some rare cases, it is above 7 minutes. Since in our system, we focus on long session times (median session length is two hours), this delay is neglectible.

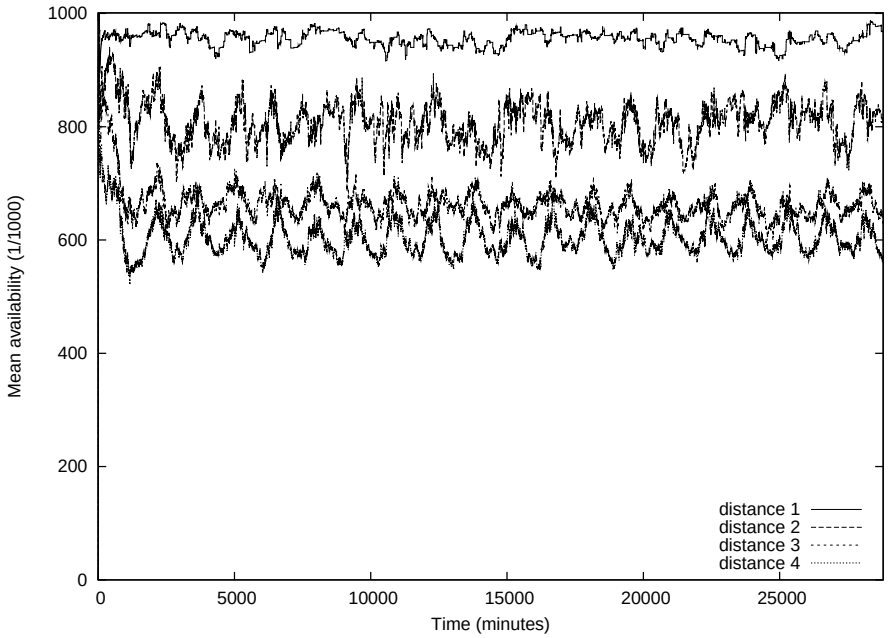

Figure 19: Mean Availability of Connected Peers depending on their distance to the server. It shows that peers at distance 1 from the server have a much better availability, and so on...

5.5.1 Selfish Peers

We define selfish peers as peers that follow the protocol but try to prevent other peers from benefiting from the system. Hence, the corresponding selfish behavior in our particular case is not to propagate the heartbeats to their neighbours. Consequently, selfish peers improve their ratings in the system not by increasing their measured availability but rather by decreasing the measured availability of other peers.

We simulated this behavior by randomly selecting peers in our simulations as selfish peers. The results, depicted in Figure 20, show that the accuracy of the measure is not impacted too much until the percentage of selfish peers hit 50 %. We conclude that:

- When there is a small amount of selfish peers, the impact is negligible on the measured availability of other peers. Hence, they don't really gain any benefits from their behavior.
- When there is a large amount of selfish peers, their impact is more significant, but their measured availability is also as diminished as for non-selfish peers.

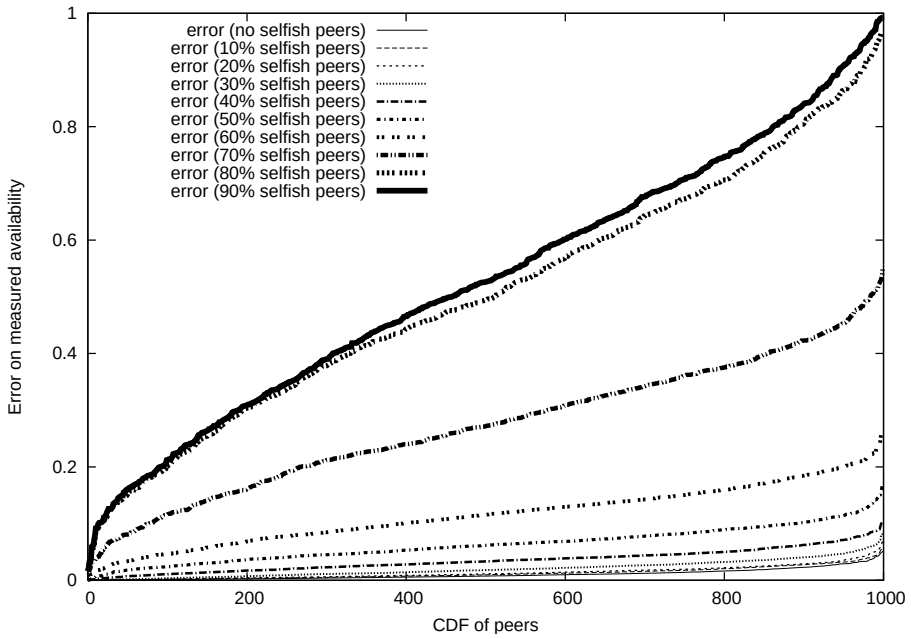

Figure 20: Error on measured availability depending on the number of selfish peers in the system. Although selfish peers don't propagate the heartbeats to their neighbours, the error is under 10% with 50% of selfish peers in the network, and almost not impacted with 30% of selfish peers.

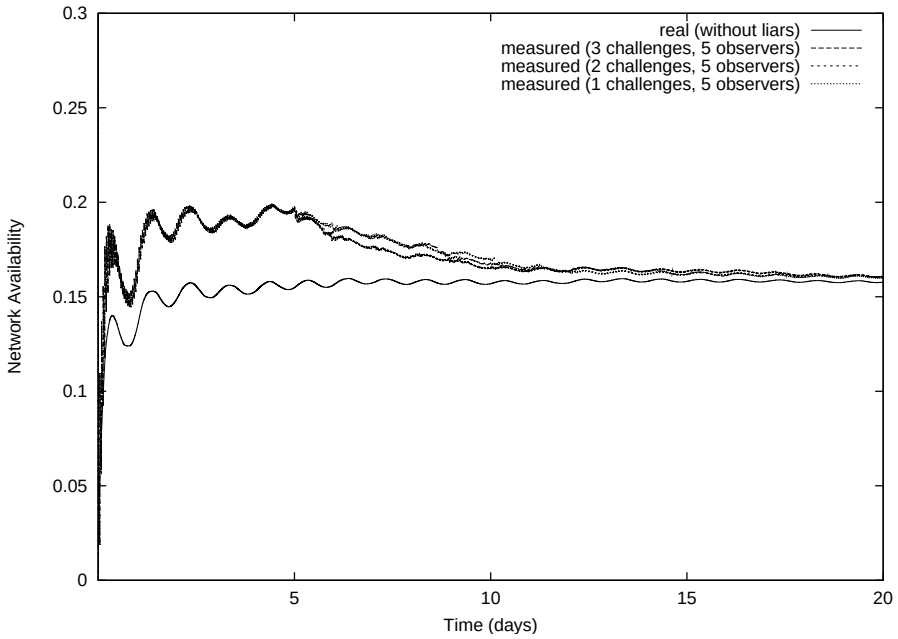

Figure 21: Detection of liars using our system. For every peer, a set of five random observers send a given number of challenge every day. Liars are removed from the system when a challenge fails. We display the network availability, real, and measured by our system.

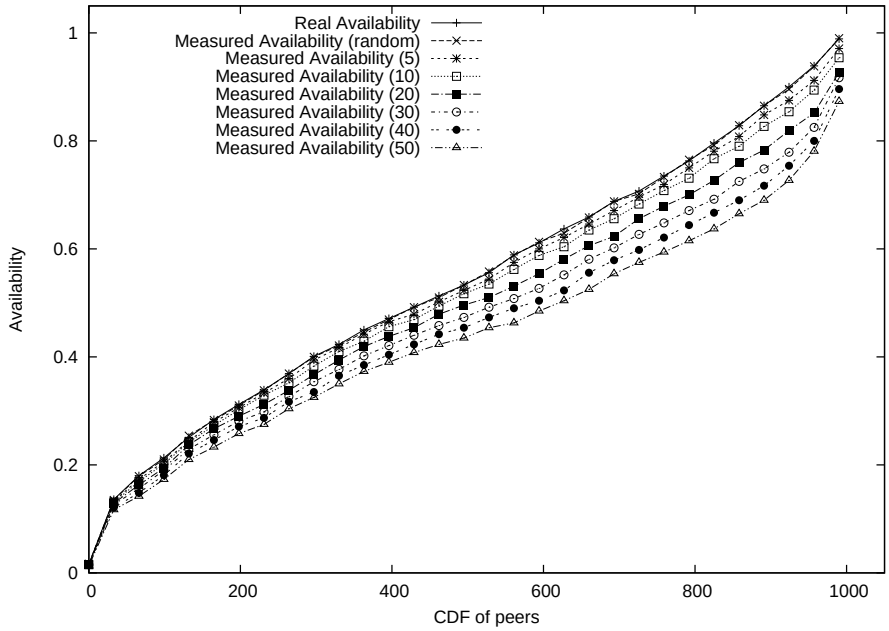

Figure 22: Availabilities measured using either random checks or by requiring some connection length. It shows that sending the heartbeat randomly (at a random minute during the heartbeat hour) performs much better than requiring a minimal connection time (between 5 minutes and 50 minutes) from a child to propagate the heartbeat.

5.5.2 Liars

We define *Liars* as peers trying to get a better status in the network by lying on their availability. In our system, this is done by setting bits in their availability bitfields for whom they have not received the heartbeats.

Our protocol does not specify when challenges should be send to peers, as it is application dependent, and how the system should react when a peer fails to reply to a challenge. Thus, for the purpose of showing that our scheme can be efficiently used to detect liars, we simulated a network where every peer is monitored by a set of five observers. These observers are responsible for sending challenges every day, and, if a peer fails to reply to a challenge, for diffusing the failed challenge in the network, so that other peers will immediatly black-list liars. Figure 21 shows the efficiency of this detection system on the global network availability: we wait for five days to reach the system stable mode before starting the detection system; ten days later, most liars have been detected and the measured global network availability is close to the real one.

5.5.3 Opportunist Peers

We define *Opportunist* peers as peers connecting to the network only to get heartbeats to increase their perceived availability. In our system, such peers would connect at fixed times, depending on the schedule of heartbeats diffusions.

To avoid such behaviors, we proposed two different policies:

- **Random diffusion:** Within each period P, the server should start the heartbeat diffusion at a random time, so that opportunist peers cannot forcast when to connect to the network.
- **Trusted diffusion:** When a peer receives a heartbeat, it should only propagate the heartbeat to children which have been connected for a long period.

Figure 22 plots the impact of these policies on the accuracy of measured availability. It shows that random diffusion performs much better as soon as the required connection period for children becomes to long. As a consequence, we used random diffusion in all the other simulations that were done in this paper.

5.6 Bandwidth Cost

Finally, it is important to evaluate the cost of our protocol, in particular in bandwidth, usually the rarest resource in peer-to-peer networks.

Using RSA, signatures and keys are usually 256 bytes long. Although we will not consider this case in the following, elliptic curves for example already give a good level of security with around 15 bytes.

A reasonable choice for P is 1 hour, and for N_t, we can choose a period of one year for example, i.e. 8760 hours.

Using these values, we can estimate the sizes of the different messages:

- Heartbeat : 800 bytes (mainly 2 keys and 1 signature)
- Availability : 1400 bytes (bitfield[N_t] + 1 signature)
- Challenge : 70 bytes (mostly a string of say 64 bytes)
- Proof : 900 bytes (string + 1 key + 2 signatures)

The Heartbeat is sent once per period P to all the neighbours. Other messages would probably only be sent once a day between two peers working together. Consequently, the bandwidth cost of our system is negligible.

6 Discussion and Related Work

6.1 Discussion

We can now compare our system with other availability measurement systems. To the best of our knowledge, such systems can be put in three different sets:

- Systems using hello messages (pings): such systems are expensive in the number of messages that they generate. To get the same level of accuracy as in our system, they need to send M messages per hour, where M is the number of peers each node monitors, while our system only need D messages, where D is the out-degree of the mesh. Moreover, the result is not complete: peers can only monitor availability while they are online and for peers who IP addresses don't change too often, so matching availabilities is much less efficient.

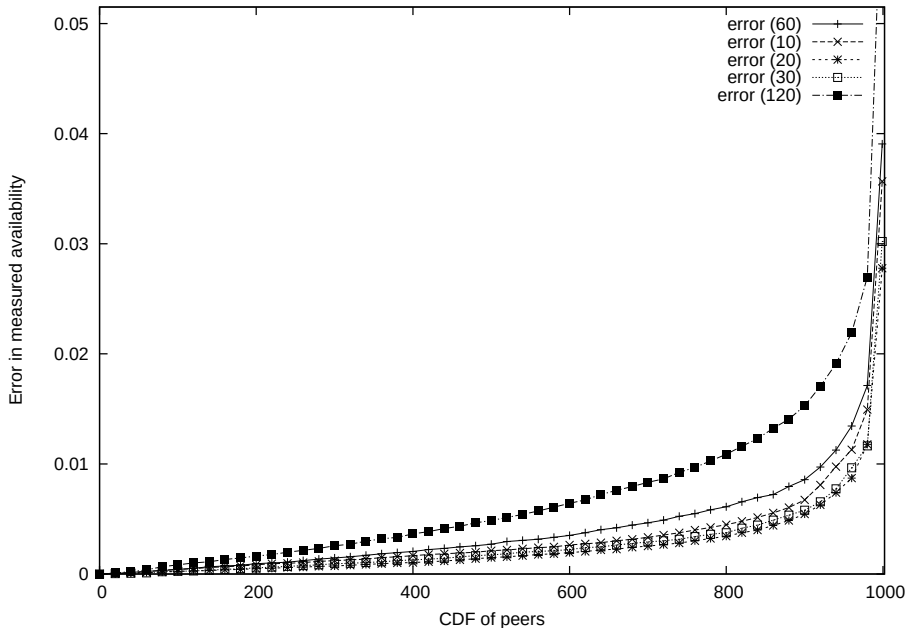

Figure 23: The measured availability depends on the period P. Here, we plot different errors depending on the number of minutes between heartbeats in the exponential distribution.

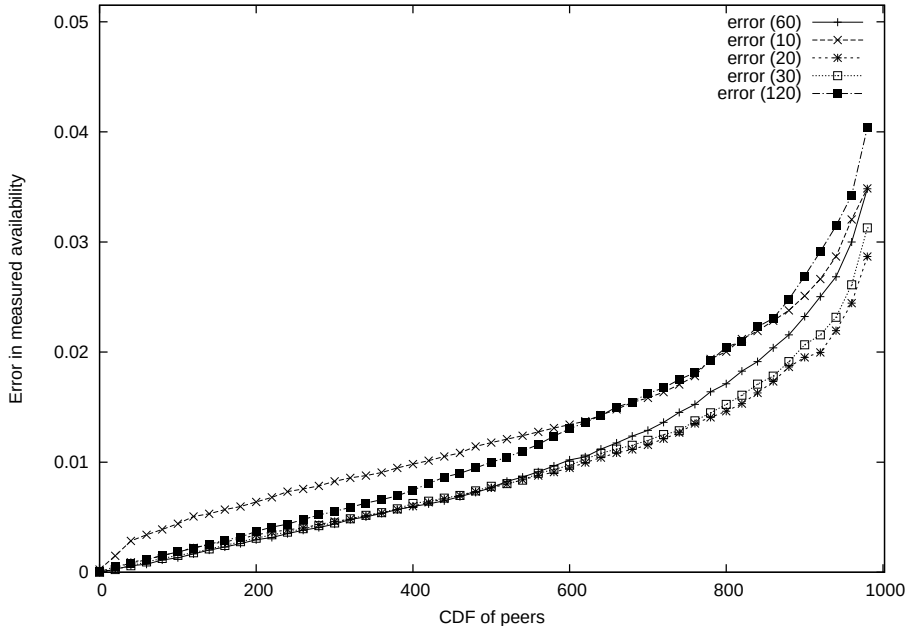

Figure 24: Same as Figure 23 with a uniform distribution of availabilities.

- Systems using regular pings extended with a trust system. These systems try to complete the monitoring for offline periods by sharing monitoring information between nodes, weighted by a trust system. The trust protocol is often expensive, and not resilient to most attacks, unless it uses more cryptographic challenges than in our system.
- Systems where nodes share their periods of availability with other peers. These systems are close to our system, except that they cannot prevent nodes from lying about their availability.

To the best of our knowledge, only one availability monitoring system[12] has been designed especially for peer-to-peer networks. Although it prevents collusions of peers, i.e. resulting usually into over-estimation of availability of a peer, it does not prevent peers from lying by under-estimating the availability of the peers they are supposed to monitor. Moreover, it is unclear how the system really behaves as it assumes that IP:port addresses are uniformely distributed, which is actually not the case in practice and might result in large variations in the sizes of monitoring sets.

6.2 Studies on Availability in Peer-to-Peer Networks

[3] does a crawl of a Gnutella network to measure peers availability. They conclude that peers with higher availability tend to be more stable than other peers. They design a protocol that builds a more stable network by selecting peers with higher availability. However, they don't give any method to prevent peers from lying about their true availability.

[1] does a study of availability in Overnet.

6.3 Use of Availability in Peer-to-Peer Systems

Many papers on organization of peer-to-peer networks for different purposes heavily rely on stability or availability of peers. Indeed, many systems are built using a *stable core* of the system or *super-peers*, selected for their high availability in the system. However, none of them cope with malicious peers, which might lie on their real availability to get a better status in the system.

For example, [14, 15] build a gradient topology, where the most stable peers are at the center of the network, whereas less stable peers stay on the border. They use availability information to construct a network by gossiping, and forward requests along the gradient towards the *core* of the system. Malicious peers can lie on their availability to be included in the *core*, and then refuse to serve requests to break the system.

[8] uses availability to select *super-peers* in the network to build a top-level Chord Distributed Hash Table over other less stable DHT. If malicious peers lie on their availability, they might be included in the top-level DHT, and either use it to break the system or decrease their load.

[9] gives a thorough study of replacement policies to decrease the effect of churn in a peer-to-peer system: in particular, they show that the random policy behaves very well, but not as well as replacements based on maximizing availability or session lengths.

[7] introduces a new definition of availability, *presence-based* availability, where the time spent in the network is weighed by the number of nodes in

the network at the same time. There is an assumption that the number of peers in the network fluctuates a lot, and their evaluation is indeed done on real traces, but biased by timezone patterns.

[6] assumes the existence of an availability monitoring service to proactively repair fragments in a peer-to-peer storage system, depending on an estimation of the failure rate. A fixed threshold is also used to trigger reactive repairs. However, in case the availability is compromised, this would prevent repairs.

[19] discusses the difference between host availability and data availability observed in Maze traces. It proposes a simple availability monitor that complements machine availability history with data availability obtained by probabilistically requesting data.

[16] studies object replica maintenance under temporary and permanent failures, for different peer-to-peer systems. It shows that data tends to accumulate on nodes with long availability and unlimited capacity. With limited capacity, the performance degrades when these nodes are full, as short availability nodes trigger many repairs.

6.4 Avoiding collusions

While waiting for the next algorithm, some mechanisms can be used to prevent collusions:

- Every peer should remember the last availability received from another peer. This would prevent a peer from changing its availability for a time that it has already sent to other peers.

7 Conclusion

In this paper, we have presented a simple but efficient way of monitoring availability in peer-to-peer systems. Our protocol uses a set of servers to propagate cryptographic heartbeat messages in a mesh of peers, allowing them to measure and check the history of availability of other peers easily at any time. Our protocol is resistant to simple liars, which lie about their availability to gain access to more resources in the system.

However, currently, our protocol is not resilient to collusions of peers, which help each other to lie about their availability by illegitimately diffusing old heartbeat messages. We are currently working on the design of a new version of our protocol, which is more resilient to collusions of peers. However, as this new version requires the use of merkle trees [11], the complexity and the cost of the system is expected to increase. Our future work includes the performance evaluation (i.e., a cost-benefit analysis) of our new protocol with increasing number of collusions in the system.

References

[1] Ranjita Bhagwan, Stefan Savage, and Geoffrey Voelker. Understanding availability. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[2] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: system support for automated availability management. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[3] F.E. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in P2P protocols. In *8th International Workshop on Web Content Caching and Distribution, Hawthorne, NY, USA*, September-October 2003.

[4] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[5] T. Dierks and E. Rescorla. RFC 4346: The transport layer security (tls) protocol version 1.1. IETF, April 2006.

[6] Alessandro Duminuco, Ernst W Biersack, and Taoufik En Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *CoNEXT'07, 3rd International Conference on emerging Networking EXperiments and Technologies, December 10-13, 2007, New York, USA*, Dec 2007.

[7] Richard J. Dunn, John Zahorjan, Steven D. Gribble, and Henry M. Levy. Presence-based availability and p2p systems. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 209–216, Washington, DC, USA, 2005. IEEE Computer Society.

[8] L. Garcés-Erice, E.W. Biersack, P.A. Felber, K.W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *Proc. of ACM/IFIP Intl. Conf. on Parallel and Distributed Computing (Euro-Par 2003)*, 2003.

[9] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–158, New York, NY, USA, 2006. ACM.

[10] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. 2002. `http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf`.

[11] Ralph Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.

[12] Ramses Morales and Indranil Gupta. Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 55, Washington, DC, USA, 2007. IEEE Computer Society.

[13] http://www.palabre.net/, 2008.

[14] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Using aggregation for adaptive super-peer discovery on the gradient topology. In *Proceedings of the 2nd International Workshop on Self-Managed Networks (SelfMan 2006)*, volume 3996 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2006.

[15] Jan Sacha, Jim Dowling, Raymond Cunningham, and René Meier. Discovery of stable peers in a self-organising peer-to-peer gradient topology. In Frank Eliassen and Alberto Montresor, editors, *6th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS)*, volume 4025, pages 70–83, Bologna, jun 2006.

[16] Kiran Tati and Geoffrey M. Voelker. On object maintenance in peer-to-peer systems. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February 2006.

[17] Jing Tian and Yafei Dai. Understanding the dynamic of peer-to-peer systems. In Proc. IPTPS 2007.

[18] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.

[19] Zhi Yang, Jing Tian, and Yafei Dai. Towards a more accurate availability evaluation in peer-to-peer storage systems. *Networking, Architecture, and Storages, 2006. IWNAS '06. International Workshop on*, pages 8 pp.–, 1-3 Aug. 2006.

Contents

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399