Distributed Storage for Tor Hidden Service Descriptors

Karsten Loesing
University of Bamberg, Germany
karsten.loesing@wiai.uni-bamberg.de

March 27, 2007

Abstract

Tor provides a mechanism for responder anonymity via hidden services. As a part of these, Tor provides storage on its central directory servers to allow publishing and retrieving rendezvous service descriptors. This proposal suggests to replace the centralized approach by a distributed storage of descriptors to the larger set of onion routers. Benefits include (1) better load balancing which is vital for the further growing of the network, (2) a more scalable way to publish descriptors, and (3) extensibility of hidden services to features like human-readable names or client authentication. As possible drawback can be seen that new threats arise due to decentralization that need to be discussed and handled.

In this project a structure is applied to the network of onion routers, based on concepts known from distributed hash tables and consistent hashing systems. Every participating router is made responsible for a limited set of descriptors. It is not necessary to maintain an own routing table, but possible to rely on the router list managed by the Tor directory. Some amount of replication needs to be added to overcome node failures and untrustworthy routers. This prevents the worst security threats, as descriptors are signed and have a limited time-to-live. Thus, they cannot be forged or replayed. Preliminary measurements show that routers exhibit a very low churn rate which makes them a perfect field for consistent hashing. The purpose of this project is to extend the current Tor sources to a running prototype that contains a distributed storage of rendezvous service descriptor.

1 Motivation

The Tor system [1] provides a mechanism for responder anonymity via location-protected servers, the so-called hidden services. These enable any node which is running a Tor onion proxy to set up a server (e.g. web server, file server) and make it available for clients via the network of onion routers. Tor provides storage on its central directory servers to publish and retrieve contact information—so-called rendezvous service descriptors (RSDs)—from servers to clients.

In this proposal we suggest to replace the centralized storage of RSDs by a decentralized approach. We argue that the task to store and lookup RSDs can equally be performed by the network of Tor onion routers. The Tor developers state that this task could be done "on any robust efficient key-value lookup system with authenticated updates, such as a distributed hash table" [1]. Øverlier and Syverson argue that "authoritative directory servers for hidden services as a core part of the Tor network are not necessary" and that "the primary motivation for their use initially has been one of convenience" [2]. However, we did not find a design for a distributed storage of RSDs in Tor in the literature, yet.

There are several benefits from distributing storage to multiple Tor routers. The most obvious is load balancing: publish and retrieve operations to the directory nodes cause additional network traffic, require computation (e.g. for validating signatures) and consume memory. In a decentralized solution, directory servers would no longer be burdened with these tasks.

Load balancing is important for Tor, because hidden service usage can reasonably be expected to increase with the growing network size, if not faster.

Another benefit comes from the fact that currently, RSDs have to be replicated manually by hidden service providers to all directory servers. While that solution might work well with the initial 3—or currently 5—directory nodes, it does not scale. Due to the design paper, there might be up to 9 directory nodes as the network scales [1], but there is no reason for such a fixed upper bound if the network size increases further on. The benefit of our proposal is that it restricts replication to a constant number independently to the number of directory nodes, thus allowing more Tor directories to be added without interfering with hidden services.

A third benefit could be extensibility of hidden services to provide additional services, e.g. resolve human-readable names to hidden server addresses, perform client authentication for hidden servers, etc. Such extensions would likely require additional information to be stored at some place inside Tor. With regard to the fact that Tor directories might already be the performance bottleneck for the system, they could not perform these additional tasks very well. Therefore, a decentralized storage might be useful, maybe with slight modifications.

On the other hand, a decentralized storage also implies possible robustness problems and security or anonymity threats that have to be discussed. Tor routers are less fail-safe and less trustworthy compared to the authoritative directory servers. A distributed solution needs to cope with these new threats to provide a similar service as the directory servers do.

In the remainder of this proposal, we identify requirements to the existing as well as the proposed storage for RSDs in section 2. We suggest a possible design in section 3 and present some early performance measurements of Tor routers concerning fail-safeness in section 4. In section 5 we discuss possible security and anonymity impacts that result from less trustworthiness and present conceivable counter measures. Section 6 concludes the proposal.

2 Requirements

Before presenting a design for decentralized storage of RSDs, we want to identify both, functional and non-functional requirements. These include the involved data and roles. These requirements are independent from a concrete realization, be it based on the directory servers or on the routers.

Rendezvous service descriptors The details of an RSD are described in the Tor Rendezvous Specification[1]. An RSD contains contact information, e.g. the addresses of introduction points to which a client needs to establish a connection. The size of an RSD depends on the number of contained introduction point and ranges in the order of some hundred bytes. All information inside an RSD is public. RSDs are signed by the hidden service provider, so that everybody can validate that a RSD originates from the provider. An RSD contains a timestamp and has a limited time-to-live, before it needs to be replaced by a new RSD by the hidden service provider. An RSD can be recognized by a pseudo-unique identifier which is the hash value of the public key of the hidden service.

Publication to RSD storage Hidden service providers need to publish their current RSD whenever their contact information, i.e. the set of introduction points, changes. This occurs first when entering the system, at frequent intervals during normal operation, and possibly when leaving the system. However, published RSDs should remain available in the storage for a couple of hours, even if not being updated. For each publication, hidden service providers need to stay anonymous (hide their location) by tunneling their requests through a sender-anonymous circuit. In the special case of a hidden service provider hosting multiple hidden services, additional precautions must be taken to hide correlations between those services.

Retrieval from RSD storage Clients need to retrieve a current RSD of the hidden service they at-

[1] http://tor.eff.org/svn/trunk/doc/rend-spec.txt

tempt to access. Unless they are aware of an up-to-date RSD from a previous access, they need to retrieve one from the storage. As it applies to hidden service providers, clients also need to stay anonymous throughout this request. Further, if a client requests multiple RSDs at the same time, correlations must be hidden, too.

So far, there are no statistics on the average number of RSDs stored in the directory or on publication and retrieval frequencies. However, an anonymity-preserving measurement of such statistics could provide information on expected usage characteristics in a decentralized design. Such a measurement should be easy to implement in the current directory servers.

3 Proposed design

We propose a design that passes the task of storing and looking up RSDs from a small number of directory servers to a large number of onion routers. Therefore, we employ structure to the network of routers and enable hidden service providers and clients to route requests to a particular onion router. Further, we include means to overcome failing and untrustworthy nodes.

Network structure In contrast to the existing directory-based approach, in our design, every router stores only a small subset of all RSDs. Therefore, a hidden service provider or client must be able to determine which router is responsible for a given RSD. An obvious choice for such a task is a distributed hash table (DHT) like Chord [3] or its precursor, a consistent hashing system [4]. We could make profound experiences with Chord when implementing a Java version of it for use in a distributed service discovery system [5]. Chord assigns to every node and to every item an identifier on an identifier ring and makes every node responsible for storing the entries in the identifier range between its predecessor's and its own identifier. We can borrow this principle by deriving a node's identifier from a router's IP address[2] and using the public key hash of an RSD as entry identifier.

Routing table Both, hidden service providers and clients need an up-to-date router list in order to route requests to the correct onion router (either directly or via multiple hops). In fact, an important part of DHT implementations is to maintain current routing tables at each node. In Chord, this is done by exchanging routing information between nodes in order to store a logarithmic number of addresses. However, Tor greatly facilitates the task to maintain routing information: Every onion proxy and router in the Tor network maintains a complete router list in order to be able to construct circuits. This list needs to be as complete as possible to prevent identification by observing which routers are chosen for circuit establishment. We can simply rely on Tor to provide hidden service providers and clients with up-to-date routing tables, so that they can contact almost every router in a single hop. We even go further and require every node to rely only on his own routing table and not trust in any other node's routing information.

Performing requests Hidden service providers and clients perform requests to onion routers in the same way as when retrieving information from the directory servers. They create a multi-hop initiator-anonymous circuit to the onion router and ask it to answer their request. From an outside perspective, this request cannot be distinguished from other circuits. Requestors might even choose to keep the circuit open for some random time after the request is complete to confuse attackers.

Handling routing and node failures Although Tor nodes have almost complete routing tables, their view of the network might slightly differ. Therefore, it may occur that a router receives a request that it

[2]While it might appear more obvious to use the router's public key hash as identifier, IP-based identities provide better protection against free identifier choice of an attacker. However, Tor needs to assure, that only one router is permitted per IP address, which is currently not the case.

is not responsible for. Further, onion routers are less fail-safe than directory nodes. And even if a directory node fails, there are still 4 (at the moment) which can replace it. Thus, we also need some amount of replication to handle routing and node failures. We again adopt a mechanism from Chord, that puts replicas of entries on neighboring nodes (with respect to node identifiers) [3]. Nodes need to maintain replication by periodically requesting entries from other nodes to replicate them. We decided to apply a simple reactive pull-based scheme that only reacts to failing and leaving nodes whenever such a situation occurs. An alternative approach could be for example to proactively transfer entries upon leaving the system. We decided to postpone such extensions to later discussion.

Handling untrustworthy routers Hidden service providers and clients cannot rely on onion routers to correctly store and retrieve RSDs. Again, replication can help reduce such problems by making more than one router responsible for RSD storage and retrieval. In case of storing, a hidden service provider should store an RSD at more than one router, including the primarily responsible node as well as some replicating ones. Further, it should perform regular checks by trying to obtain its own RSD using a distinct circuit. In case of retrieving, if a client receives an empty result for an RSD from the primarily responsible node, it should try to retrieve it from replicating nodes using different circuits, too.

Logging usage statistics We would like to monitor the system in action, just as one can monitor how much traffic is routed by any router in the system. Therefore, every router could collect aggregated data, e.g. the number of published and requested RSDs in a given time, and publish it to Tor directory.

In summary, we propose to use a DHT/consistent hashing system with entry replication in combination with the router list obtained by the directory servers. The latter greatly reduces complexity compared to usual DHT implementations.

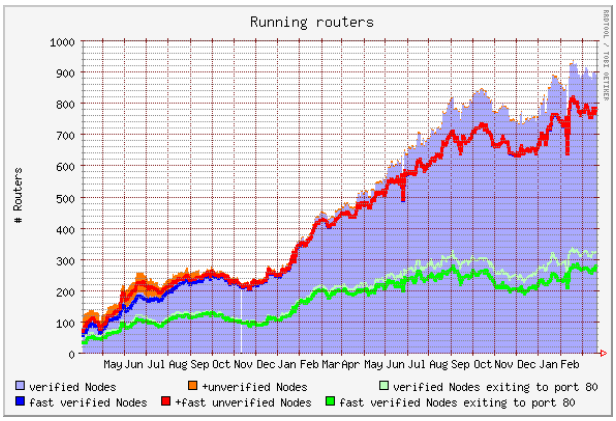

Figure 1: Graph of the number of Tor servers over the last 24 months from March 21th 2007. Source: http://www.noreply.org/tor-running-routers/totalLong.html

4 Feasibility discussion

Feasibility of realizing our design depends in large part on the characteristics of onion routers compared to directory servers. Therefore, we performed a preliminary analysis of the number of routers and their typical uptimes and churn rates.

Average number of routers The average number of routers is an important metric to comprehend the absolute size of the network. However, it does not allow to make statements on the dynamic behavior of joining and leaving nodes. Figure 1 shows the development of the number of routers over a period of 24 months. It shows that there are currently about 900 routers, but when extrapolating the graph, this number is very likely to increase in the future.

Session time Another typical question to be answered is how long routers stay in the system and (related to this metric) to which degree router population changes over time. We used the publicly available Tor directory log history that contains snapshots of router statuses for every hour since December 2005.[3] We picked the logs from February 2007 of

[3] Online available via `rsync asteria.noreply.org::tordir`

the directory server moria2 to derive average session times.

Session time is an important metric that measures the time between joining and leaving the Tor network [6]. The shorter the session time is, the more often descriptors have to be transferred from other nodes. We measured session time by determining the time between the first and last occurrence of a Tor node in a snapshot, i.e. the router may not occur in the previous (next) snapshot with respect to the start (end) of the interval.

In this preliminary measurement, we omitted the situation that a router leaves the network and rejoins it between two snapshots; the reason is that one cannot distinguish such a situation from the ordinary refreshing of a router descriptor. So, if we counted renewal of a descriptor as node failure, this would result in a lot of false positives. For a more precise analysis, we would have to minimize the snapshot interval, e.g. one minute instead of one hour.

From the given logs, we discarded all sessions that have existed at the beginning of the examined interval and those that existed beyond the ending of it. Otherwise, those sessions would have been measured shorter than they really are. However, those sessions that started before and ended after the examined interval of one month have been omitted, too. In a more precise analysis, the examined interval might be extended, e.g. to one year.

In total, we measured 7723 distinct sessions with an average duration of 43.94 hours and standard deviation of 57.27 hours. The high total number of sessions—compared to the average number of routers—results from some routers joining and leaving the network multiple times, thus having multiple sessions throughout the considered interval. There were 353 sessions beginning before and ending after the examined interval. Figure 2 shows a box plot of session times in hours with a logarithmic scale.

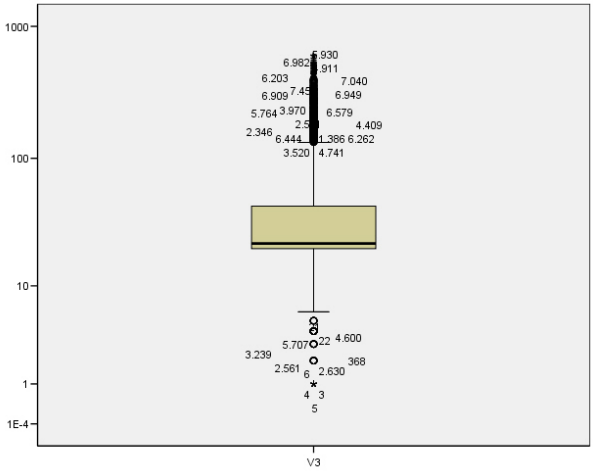

Figure 2: Box plot of session times in hours with a logarithmic scale.

Churn rate Another metric for population change is the churn rate. It determines the fraction of joining and leaving nodes compared to the whole node population in a certain time, e.g. one hour. The more nodes join or leave the network, the more descriptors need to be transferred to other nodes. Thus, churn rate is reciprocal to session time. We measured join and leave rate separately. We define the join rate as the fraction of newly joined nodes in a snapshot compared to all nodes in that snapshot. As leave rate we calculate the fraction of leaving nodes in a snapshot compared to the previous snapshot in which they have been present. The different reference populations of both metrics result from the assumption that both metrics should range from 0–100 %.

As above, we did not consider nodes leaving and rejoining the network within two snapshots. Since this might considerably increase both, join and leave rates, a more precise analysis should be based on shorter intervals between snapshots, e.g. one minute.

In total, we measured 617 join and leave rates resulting from 618 snapshots. The first 54 snapshots in the sample had to be discarded, because they appeared to be erroneous: The first 53 join and leave rates would have been 0 % and the 54th would be 22 times the average. We measured an average join (leave) rate of 1.058 % (1.053 %) with a standard deviation of 0.380 % (0.354 %). Figure 3 shows a box plot of both, join and leave rates.

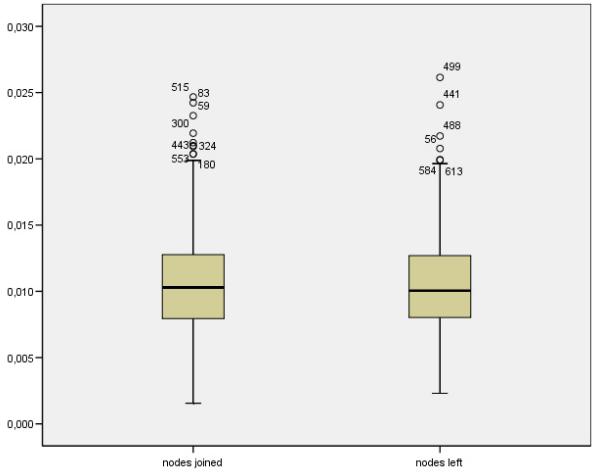

Figure 3: Box plot of join and leave rates, i.e. the number of joining and leaving nodes per hour compared to the whole node population.

From these measurements we conclude, that the population of Tor routers does not change too quickly for applying consistent hashing to it. Even though the sample excludes the session times of one hour or less, the average session time of 43.94 hours makes it very unlikely that such short session times are the norm. DHTs have been designed for situations in which session times range in minutes, rather than in hours [6]. Further, we could design our system in a way, that only stable routers will be included in the distributed storage, thus making high churn even less likely. Therefore, we should figure out if some classes of routers have higher session times and lower churn rates compared to others. Anyhow, we can rely on the fact that Tor itself would face serious other problems, if its routers had churn rates in the magnitude of DHT nodes.

5 Security discussion

In general, onion routers are less trustworthy than directory servers. This comes from the fact that the currently deployed 5 directory servers are run by the Tor developers, which are at least less likely to misuse the trust put in them than arbitrary persons. Onion routers can be set up by anyone with an Internet connection. Though it is possible to register an onion router's name at the Tor people, this constitutes only a limited amount of trust. In the following we want to depict what kind of security threats need to be considered compared to a centralized storage solution.

Logging publication and retrieval requests In either a centralized or a decentralized design, the node which is requested to store or retrieve an RSD learns about that request. Using this information, one can generate request logs for a given hidden service, i.e. the service's uptime and clients' access frequencies and patterns. In the existing design, this information can only be obtained by the directory server administrator. In addition to that, a user can—with a little effort—learn about a hidden service's uptime by regularly polling the directory servers for the hidden service's RSD. In a decentralized design an arbitrary user can learn about the hidden servers for which she is responsible.

While we cannot provide an easy solution to prevent logging, we can make life for an attacker a little harder by sticking her to a fixed node identifier, thus making her responsible for a specific identifier range only. We propose to choose identifiers based on IP addresses, that cannot be changed arbitrarily. On the other hand, hidden service providers who are afraid of this kind of attack might change their identifiers on a regular basis. Further, we can require that routers have to be marked as stable by the Tor directory in order to become part of the distributed storage. At least this raises efforts that are necessary for an adversary to control larger areas of the identifier ring.

After all, it is not clear, if hiding online activity is a primary goal for the current hidden service design. If there was a strong need to do so, it could merely be provided by introducing authentication to hidden services, i.e. by restricting knowledge to a hidden service's RSD to authorized clients only. But this implies changes to several parts of the network, not only to the directory mechanism. It is part of future work, not of the design proposed here.

Dropping valid entries and providing false empty results The facts that RSDs are signed by the hidden service provider and that they have a limited time-to-live makes it impossible for an attacker to forge or replay them. On the other hand, routers could drop valid entries or provide false empty results.

As a counter measure, hidden service providers and clients should never trust a single router to work correctly. The former should publish their RSDs on more than only the primarily responsible router, and the latter should cross-check empty results by requesting the same RSD from a neighboring router.

In order to perform an effective denial of a hidden service, an adversary would need to control all nodes controlling replicas for this RSD. This is rather difficult to accomplish, because of the mapping of IP addresses to identifiers and the requirement to have all stable routers.

In the future, we might consider a system that reports repeated cases of misbehaving nodes to the Tor directory which could easily recheck the misbehavior and possibly remove the router from the list.

Flooding a router with false requests An adversary could perform a denial of service attack by flooding a router with false requests. While publication requests with RSDs containing random data are discarded, correctly signed RSDs have to be stored until their time-to-live expires. A special form of this attack could be performed by an onion router flooding her neighbor with false RSDs when they try to copy replicas.

At the moment, we consider the efforts for performing such an attack to be higher than the caused damage. There are other possible attacks on onion routers that are more effective, e.g. flooding it with false circuit establishment requests. However, possible counter measures could be to require a calculational expensive puzzle that can be efficiently validated for storing RSDs and to restrict the number of copied replicas to a reasonable number.

This security discussion might not be complete due to threats we have not thought about yet. However, the assumptions—unencrypted, but signed contents, no need for anonymity of routers—prevent a lot of security threats right from start.

6 Conclusion

In this proposal we have suggested to replace the centralized storage of RSDs by a decentralized approach. We proposed a design that combines a DHT/consistent hashing approach with entry replication with the router list obtained by the directory servers. We performed some measurements on the dynamics of router population in Tor and found that routers should be feasible to be structured in a consistent hashing network. We also discussed possible security impacts caused by our design.

The next step would be to perform a prototype realization of our proposal. This could be used to measure reliability of routing. Further, the concrete implementation might also raise questions overlooked in the design.

Possible future work might be the extension of hidden services to provide additional services, e.g. resolve human-readable names to hidden server addresses. The difficulty lies in the fact that human-readable names are not provably related to the public key of a hidden service—in contrast to onion addresses. However, a solution would be required to conveniently refer to services or users in Tor.

Another work to be performed in the future would be the inclusion of client authentication to hidden services. We already made use of authenticated hidden services by using an external DHT on top of Tor [7,8], but we envision to include such a protocol in Tor. However, this requires RSDs to be encrypted (like it is done in [2] which contradicts the premises of our proposed design and requires to realize authenticated updates in another way for some thoughts on this). Though authenticated hidden services aim at different types of applications compared to normal hidden services, they promise to be an interesting area for research.

7 Project timetable

The refinement of this proposal towards an implementation as well as the implementation itself is scheduled for summer 2007. We conceive the following timetable to be reasonable in order to finish the work at the end of August 2007.

1. Phase one: Refine concepts, perform preliminary tasks (April 11 – May 27, 6.5 weeks)
   (a) Discuss concepts given in this proposal with Tor developers and refine proposed design
   (b) Setup own test environment to compile Tor from sources
   (c) Implement test management application to start/stop a number of pre-configured Tor processes on a local machine that create a local Tor network
   (d) Identify possible interfaces in the Tor sources where to fit in the new code (by logging and code review)
   (e) Specify exact protocol including message formats; integrate proposed changes with existing specification documents
   (f) Propose code to monitor usage of directory servers for RSDs in the deployed Tor network
2. Phase two: Implement perfect-world prototype (May 28 – July 16, 7 weeks)
   (a) Implement prototype that performs basic routing of requests for a specific identifier to the appropriate router
   (b) Implement prototype in which routers are able to forward inadequate requests to their more appropriate neighbors
   (c) Implement prototype that stores and retrieves RSDs at the one responsible node for the RSD's identifier
3. Phase three: Implement real-world prototype (July 17 – August 31, 6.5 weeks)
   (a) Implement fail-safe prototype by employing replication of entries
   (b) Implement secure prototype that can cope with untrustworthy routers by evading misbehaving routers
   (c) Implement prototype that collects usage statistics and publishes them to the directory
   (d) Write web page that evaluates logged statistics and presents them
   (e) Document collected results of the implementation project in a technical report

8 Acknowledgements

Some questions raised during writing this proposal could not have been answered without input and support from Roger Dingledine and Nick Mathewson. Further, some ideas (e.g. introducing naming to hidden services, restriction of one router per IP address) have been adopted from discussions in the public Tor mailing list. Some ideas related to hidden service authentication originate from discussions with Lasse Øverlier and Paul Syverson. Finally, thanks to Guido Wirtz for fruitful discussion on this topic.

References

[1] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[2] Lasse Øverlier and Paul Syverson. Valet services: Improving hidden servers with a personal touch. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, UK, June 2006. Springer.

[3] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[4] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM Press.

[5] Sven Kaffille, Karsten Loesing, and Guido Wirtz. Distributed Service Discovery with Guarantees in Peer-to-Peer Networks using Distributed Hashtables. In Hamid R. Arabnia, editor, *Proceedings of PDPTA '05*, volume II, pages 578–584, June 2005.

[6] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *Proceedings of the General Track: 2004 USENIX Annual Technical Conference*, June 2004.

[7] Karsten Loesing, Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, and Guido Wirtz. Privacy-aware Presence Management in Instant Messaging Systems. In *20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[8] Karsten Loesing, Maximilian Röglinger, Christian Wilms, and Guido Wirtz. Implementation of an Instant Messaging System with Focus on Protection of User Presence. In *Proceedings of the Second International Conference on Communication System Software and Middleware*, January 2007.