Providing Mutually Anonymous Services in Peer-to-Peer Networks

Byung.R Kim[1], Ki.C Kim[2]

[1] Department of Computer and Science Engineering, Inha Univ., 253, YongHyun-Dong, Nam-Ku, Incheon, 402-751, Korea
doolyn@gmail.com

[2] Department of Information and Communication Engineering, Inha Univ., 253, YongHyun-Dong, Nam-Ku, Incheon, 402-751, Korea
kichang@super.inha.ac.kr

Abstract. P2P networks provide a basic form of anonymity, and the participating nodes exchange information without knowing who is the original sender. Packets are relayed through the adjacent nodes and do not contain identity information about the sender. Since these packets are passed through a dynamically-formed path and since the final destination is not known until the last time, it is impossible to know who has sent it in the beginning and who will be the final recipient. The anonymity, however, breaks down at download/upload time because the IP address of the host from which the data is downloaded (or to which it is uploaded) can be known to the outside. We propose a technique to provide anonymity for both the client and the server node. A random node along the path between the client and the server node is selected as an agent node and works as a proxy: the client will see it as the server and the server looks at it as the client, hence protecting the identity of the client and the server from attacker.

1 Introduction

Peer-to-Peer(P2P) is characterized by resource sharing among computing machines that participate in the network. In Server/Client environment, the user sends query messages to the server, and a popular server suffers a heavy load resulting degraded service. P2P network alleviates this problem by distributing the server data among the participating nodes and providing a mechanism to locate the data dynamically. This way copies of data are scattered around in the network, and the load concentration problem can be handled; however, it became harder and more expensive to find out which nodes have the data.

Currently locating data is done via Flooding techniques as in FreeNet[1] or Gnutella[2] or Distributed Hash Table techniques as in Tapestry[3,4], Can[5], or Chord[6]. Flooding model involves a repeated broadcasting of a query and inevitably causes a heavy traffic. Distributed Hash Table reduces query traffic considerably but still has the problem of keeping the table fresh. Both, however, do not provide anonymity for server locations and could expose servers to DoS or storage flooding attack or anonymity-breaking attacks[7,8,9,10].

In this paper, we propose a technique to provide anonymity both for the client and the server node. An intermediate node is selected as the agent who goes between the client and the server. This agent will pose itself as the server to the client and creates this illusion by replacing the true server IP with its own one in the query hit message packets. It also relays the client's content request to the true server and relays the data back to the client pretending as the true server.

The rest of the paper is as follows. Section 2 summarizes previous researches on providing anonymity in P2P network. Section 3 looks at the probabilistic characteristics in P2P packets which will be used in our algorithm that is explained in Section 3. Section 4 shows the experimental results, and Section 5 concludes the paper.

2 Related Researches

Anonymity problem in P2P networks is studied in several strands of related work. The primary goal for Freenet security is protecting the anonymity of requestors and inserters of files. As Freenet communication is not directed towards specific receivers, receiver anonymity is more accurately viewed as key anonymity, that is, hiding the key which is being requested or inserted. Anonymous point-to-point channels based on Chaum's mix-net scheme[11] have been implemented for email by the Mixmaster remailer[12] and for general TCP/IP traffic by onion routing[13,14] and freedom[15]. Such channels are not in themselves easily suited to one-to-many publication, however, and are best viewed as a complement to Freenet since they do not provide file access and storage. Anonymity for consumers of information in the web context is provided by browser proxy services such as the Anonymizer[16], although they provide no protection for producers of information and do not protect consumers against logs kept by the services themselves. Private information retrieval schemes[17] provide much stronger guarantees for information consumers, but only to the extent of hiding which piece of information was retrieved from a particular server. In many cases, the fact of contacting a particular server in itself can reveal much about the information retrieved, which can only be counteracted by having every server hold all information. Reiter and Rubin's Crowds system[18] uses a similar method of proxing requests for consumers, although Crowds does not itself store information and does not protect information producers. Berthold *et al.* propose Web MIXes[19], a stronger system that uses message padding and reordering and dummy messages to increase security, but again does not protect information producers.

The Rewebber[20] provides a measure of anonymity for producers of web information by means of an encrypted URL service that is essentially the inverse of an anonymizing browser proxy, but has the same difficulty of providing no protection against the operator of the service itself. Publius[21] enhances availability by distributing files as redundant shares among *n* webservers, only *k* of which are needed to reconstruct a file; however, since the identity of the servers themselves is not anonymized, an attacker might remove information by forcing the closure of *n-k*+1 servers. The Eternity proposal[22] seeks to archive information permanently and anonymously, although it lacks specifics on how to efficiently locate stored files, making it more akin to an anonymous backup service. Free Haven[23] is an interesting anonymous publication system that uses a trust network and file trading mechanism to provide greater server accountability while maintaining anonymity.

MUTE[24] forces all intermediate nodes along the path between the client and the server node to work as proxies to protect the identities of the client and the server. Every node in the path including the client and the server thinks its previous node is the client and its next one the server. Therefore the data from the true server will be relayed node by node along the path causing a heavy traffic, especially for large multimedia files. Tarzan[25] is a peer-to-peer anonymous IP network overly. so it works with any internet application. Its peer-to-peer design makes it decentralized, scalable, and easy to manage. But Tarzan provides anonymity to either clients or servers. Mantis[26] is similar to Crowds in that there are helping nodes to propagate the request to the candidate servers anonymously. Since Mantis preserves the anonymity of the server only, the server knows where is the client. The server sends the requested data to the client directly but in UDP hiding its IP. UDP is convenient to hide the server's identity but due to the packet loss inevitable in UDP Mantis needs additional packet retransmission mechanism.

Our technique is similar to Onion routing in that it uses a proxy node to hide the identity of the client and the server. But it differs from it in that the proxy node is selected dynamically for each session and not fixed as in Onion routing. The fixed proxy might be convenient to use and is useful when loaded with additional features such as encryption as in Onion routing. However it is a sort of compromise from P2P point of view: we are introducing something similar to a control center that could become a bottleneck in the network. Temporary proxy doesn't have this problem: it is used once and discarded, and later another node is selected as the proxy. However using temporary proxy has a different problem: it is hard to find the right proxy. It should be in a particular path set up between the client and the server, and it should be in some position where the proxying service can be most efficiently performed. In this paper, we show a technique to find such proxy, explain the rationale behind the decision, and prove its effectiveness through experimentation.

3 Providing anonymity via random agent nodes

To hide the client and the server from each other, our algorithm selects a limited number of random agent nodes and let them relay the actual data between the server the client. The agents are selected dynamically per each session, hence it is hard to trace the location of the server or the client via the agent nodes. Also our algorithm makes sure the number of selected agents is very limited, in most cases one or two, so that the relay overhead can be minimized. The algorithm each node performs as it receives a packet is given in Fig. 1. We explain the algorithm in following sections.

When the incoming packet is a QueryHit packet, each node performs an algorithm to determine whether it is going to be an agent for this query. It generates a random number in [0..PATH_LEN-1]. PATH_LEN is the length of the communication path this query belongs to and can be computed by TTL + Hops - 1 in the received packet. When added, TTL and Hops in the QueryHit packet represents the length of the path between the client and the server; by choosing a random number in [0..PATH_LEN-1], we hope approximately one node in the communication path will establish itself as an agent node. If the random number is 0, it becomes an agent. There is a chance no node in the path will pick 0; in this case there will be no agent node. There is also a chance that multiple nodes pick 0's; then we have multiple agent nodes. Multiple agent nodes will increase the relay-overhead but will strengthen the degree of anonymity as well. Failure to select an agent node will expose the client and the server to each other; but both still don't know for sure that the other part is the actual server or client.

```
If Received_Pakcet is QueryHit
  random_hop = random number between [0, path lenght between client & server]
  if Received_Packet->hops == random_hop
    Replace IPAddress and Port of the Received_Packet with MY IPAddress and Port
    Remember this connection in SessionTable
    Send out the packet
  else
    Pass the packet
else if Received_Packet is HTTP GET
   if HTTP_GET_Request->ServantSessionUUID != MY->global.sessionGUID
     // This means I should work as a proxy for this request.
     for each SessionRecord at SessionTable
       if found Session Record with HTTP_GET_Request->ServantSessionUUID
         Make a PUSH packet to send to the sever node remembered in it.
                Send out the packet through SessionRecord->Connection
       endif
     endfor
  endif
...............
endif
```

Fig. 1. Packet processing at each node.

The agent node will replace the IP Address and Port in the QueryHit packet with its own ones. The ServantSessionUUID remains the same. Now this packet will have the IP Address and Port of the agent instead of the original server, but it still has the ServerSessionUUID of the original server. The client, after receiving this packet, will contact the specified IP Address with a wrong UUID, and this discrepancy will be detected by our agent node allowing it to act as a proxy between the client and the server. For this purpose, the information about the original server is saved in the SessionTable of the agent node as shown in the algorithm. The same algorithm is repeated at following agent nodes when there are more than one agent. For the second agent, the first agent will be treated as the original server, and it will establish itself as a proxy between the client and the first agent node. This argument goes on for the third and fourth agent nodes as they are added in the path. Note we are not adding all intermediate nodes between the client and the server as agent nodes as in MUTE; we are selecting only those who picks 0.

Fig. 2 shows a typical flow of a Query and QueryHit packet. Node 1 broadcasts a Query packet for which Node 7 has the requested data. Node 7 sends a QueryHit packet back to Node 1 via the same path that the Query packet followed. Without anonymity mechanism, the client can contact Node 7 directly since the QueryHit packet contains the IP Adress and Port of Node 7. Node 7, on the other hand, also contacts node 1 directly to answer the request; hence both are exposed to one another.

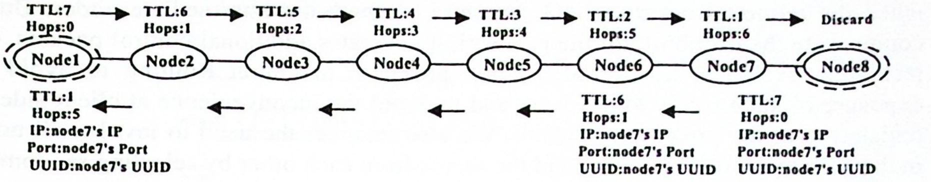
TTL:7 Hops:0
TTL:6 Hops:1
TTL:5 Hops:2
TTL:4 Hops:3
TTL:3 Hops:4
TTL:2 Hops:5
TTL:1 Hops:6
Discard
Node1
Node2
Node3
Node4
Node5
Node6
Node7
Node8
TTL:1 Hops:5 IP:node7's IP Port:node7's Port UUID:node7's UUID
TTL:6 Hops:1 IP:node7's IP Port:node7's Port UUID:node7's UUID
TTL:7 Hops:0 IP:node7's IP Port:node7's Port UUID:node7's UUID

Fig. 2. Flow of Query and Query Hit packets.

In our scheme, some agent nodes will be elected as the QueryHit packet traces back to Node 1. Suppose Node 6 and Node 3 are such agent nodes. Upon deciding to become an agent, Node 6 starts to modify the packet header: the IP Address and Port of Node 7 are replaced with those of Node 6. And the related information is saved in the SessionTable. Node 6 now acts as if it is the server who has sent the QueryHit packet. Node 3 also processes the packet similarly, but in this case, it thinks Node 6 is the server and sends the modified packet to Node 1 as a proxy for Node 6.

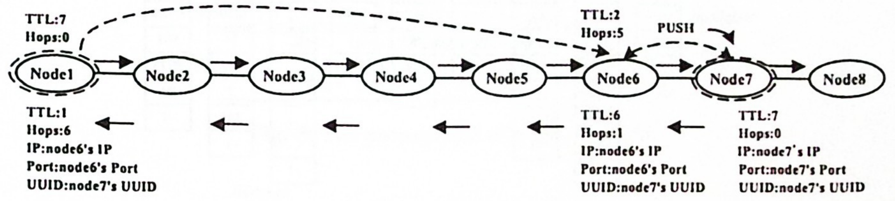
TTL:7 Hops:0
TTL:2 Hops:5
PUSH
Node1
Node2
Node3
Node4
Node5
Node6
Node7
Node8
TTL:1 Hops:6 IP:node6's IP Port:node6's Port UUID:node7's UUID
TTL:6 Hops:1 IP:node6's IP Port:node6's Port UUID:node7's UUID
TTL:7 Hops:0 IP:node7's IP Port:node7's Port UUID:node7's UUID

Fig. 3. Selection of agent nodes and flow of requested data through them.

Node 1, upon receiving QueryHit packet, contacts Node 3, the first agent, and sends HTTP header to it requesting data. The UUID included in the packet, however, is that of Node 6. Node 3 knows that this packet should be delivered to Node 6 by noticing this mis-matching between UUID and the destination IP address. It builds a PUSH packet to request the indicated data from Node 6. Now Node 3 works as an agent between Node 1 and Node 6. The response from Node 6 will be relayed to Node 1 via Node 3. Similar events happen in Node 6 because it is another agent between Node 3 and the final server. Two agents, Node 3 and Node 6, will relay the data from the actual server to the client hiding the identities of both ends successfully

4 Experimentation

Onion Routing sets up an Onion Proxy to separate the client and the server from each other. Onion Proxy prepares an anonymous path between the client and the server consisting of Onion Routers. The data itself is encrypted and decrypted at each router until it reaches the final destination. MUTE utilizes all nodes in the path between the client and the server to provide a strong anonymity. For a large file, the transmission speed is severely affected because of this. Mantis, on the other hand, tries to solve this problem by employing UDP protocol. With UDP, the client doesn't have to set up a connection with the server: the server can send directly to the client when the request from the client reaches it through the intermediate nodes. But to compensate the instability of the protocol, it generates additional control packets. Our technique excludes the use of a static proxy as in Onion Routing to avoid the exposure of such proxy to attackers and to avoid the inconvenience at client side (of registering to the proxy beforehand). We also removes the need to involve all nodes in the path to protect the client and the server from each other by selecting randomly a small number of agents between them.

To prove the effectiveness of our algorithm, we have built a simulated P2P network with Minism[27]. Minism allows us to generate a random topology for a number of nodes and to simulate packet broadcasting. Things such as what percentage of the original packets reaches the final destination or how many packets are discarded during the propagation can be answered.

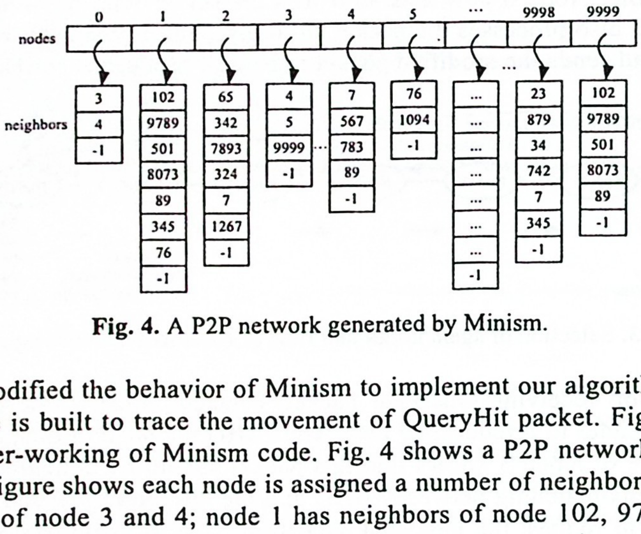

Fig. 4. A P2P network generated by Minism.

We have modified the behavior of Minism to implement our algorithm. Especially a routing table is built to trace the movement of QueryHit packet. Fig. 4 and Fig. 5 shows the inner-working of Minism code. Fig. 4 shows a P2P network generated by Minism. The figure shows each node is assigned a number of neighbor nodes: node 0 has neighbors of node 3 and 4; node 1 has neighbors of node 102, 9789, etc. Fig. 5 shows the propagation of Query packet. The "reached" array shows the nodes the Query packet reached at each stage. In "reached" array, nodes with -1 are ones that are not reached yet; nodes with 1 are those that are visited already; finally a node with (1) is one we are going to visit next. Below "reached" array, we can see the state of the stack that contains the Query packet. To simulate the propagation of a packet, the stack shows at each stage which path each duplicated Query packet should follow (from which node to which node). For example, at stage 1, all nodes are marked with -1 except node 0 since we haven't visited any node yet, and the starting node is node 0. The stack contains the path we should relay the packet: from -1 (no where) to 0 (the starting node). At stage 1, we can see node 0 is already visited (at stage 1); the next node we should visit is node 3 because node 0 has two neighbors - node 3 and 4 - and node 3 is selected as the next one. The stack shows the two path segments we should take (from 0 to 3 and from 0 to 4) for the Query packet. The segment at the top of the stack (from 0 to 3) will be chosen.

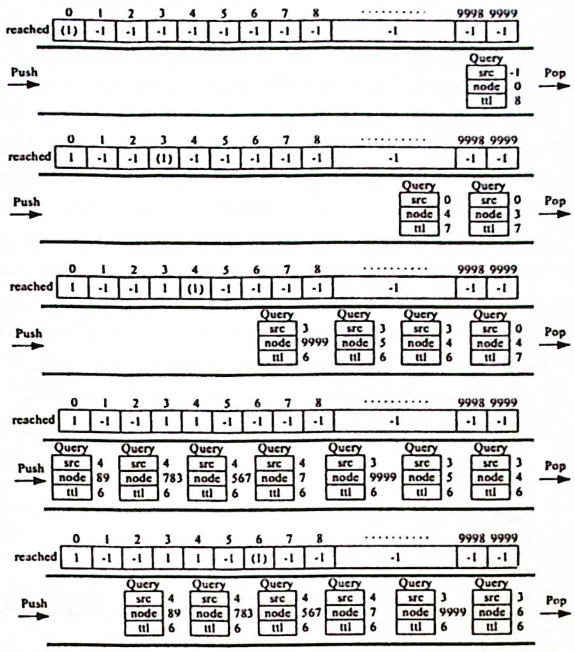

Fig. 5. The propagation of a Query packet.

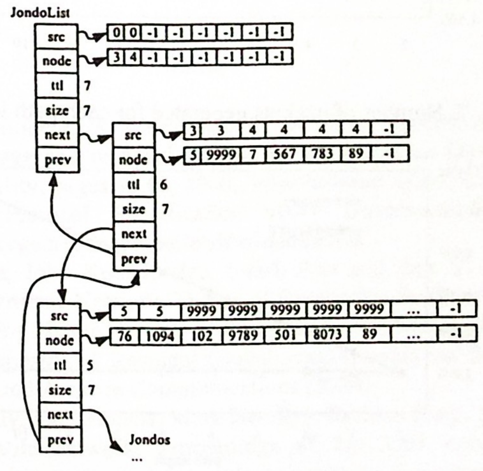

Fig. 6. A routing history table.

To implement our algorithm, we have included routing tables that show at each stage which path the Query packet has followed. Fig. 6 shows such tables. At stage 1, the first table shows route(0,3) and route(0,4) has been taken. At stage 2, it shows that route(3,5), route(3,9999), route(4,7), etc. has been taken. With these tables, we can backtrack the paths starting from the destination all the way back to the original client.

The result is shown in Fig. 7 and 8. Fig. 7 shows the number of packets generated for our simulation. Path length between the client and the server is chosen to vary from 2 to 11. Fig. 8 shows the performance of our algorithm compared to that of MUTE. As expected MUTE increases the traffic linearly as the path length gets longer, while ours stays almost the same. In most cases only one node chooses to be an agent. Since we do not allow any additional controlling packet to relay the information about this selection process, there is a chance that no node will decide to become the agent or all nodes become the agents. The latter case will make our algorithm to mimic that of MUTE, the former to that of no anonymity case. It could be a problem if there are no node selected as an agent: the client and the server is not hidden from each other. But as explained in the previous section, with our scheme the client or the server never knows that it is dealing with the other side directly. The chance is very small and each side cannot attack the other with such a small percent of certainty.

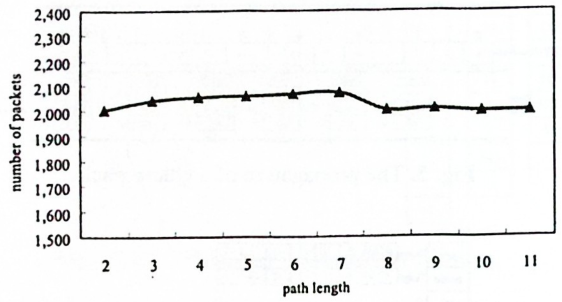

Fig. 7. Number of packets generated for each path length.

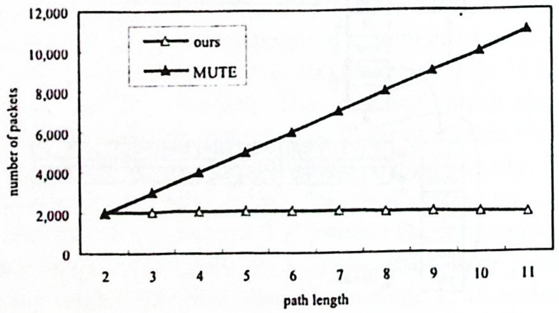

Fig. 8. Comparison between MUTE and our technique in terms of generated traffic to deliver the contents.

5 Conclusions

Currently, many techniques are suggested to provide anonymity for both the client and the server. Some of them guarantee anonymity by setting up a static proxy and forcing all communication to go through it. The clients, however, should know the presence of the proxy beforehand. Other techniques provide anonymity by allowing all intermediate nodes to participate in the data transmission. The data delivery time suffers delay inevitably. We proposed a technique that stood about in the middle of these two approaches. We employed the idea of a proxy but not static one. The proxy is selected dynamically during the traverse of the QueryHit packet. Since the selection process is truly distributed, no one knows exactly how many proxies, or agents, are selected and where they are located. The agents are linked together only by neighbors: each agent knows only its previous and the succeeding one. We have designed the process such that only very limited number of agents are selected. Since these agents are used only for the current session (and will be replace by others in the next session), it is hard to attack them or to predict the location of the client or the server by analyzing their packets. We explained how it works and showed that its performance improved substantially over previous techniques through experimentation.

Acknowledgements

This work was supported by INHA UNIVERSITY Research Grant.

References

1. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, Freenet: A distributed anonymous information storage and retrieval system, In Workshop on Design Issues in Anonymity and Unobservability, pages 46.66, 2000., http://citeseer.nj.nec.com/clarke00freenet.html.
2. The Gnutella Protocol Specification v0.41 Document Revision 1.2., http://rfc-gnutella.sourceforge.net/developer/stable/index.html/
3. Kirsten Hildrum, John Kubiatowicz, Satish Rao and Ben Y. Zhao, Distributed Object Location in a Dynamic Network, Theory of Computing Systems (2004)
4. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz, Tapestry: A Resilient Global-scale Overlay for Service Deployment, IEEE Journal on Selected Areas in Communications (2004)
5. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Schenker, A scalable content-addressable network, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications table of contents.
6. Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking (2003)

7. Neil Daswani, Hector Garcia-Molina, Query-flood DoS attacks in gnutella, Proceedings of the 9th ACM conference on Computer and communications security table of contents (2002)
8. P. Krishna Gummadi, Stefan Saroiu, Steven D. Gribble, A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems, ACM SIGCOMM Computer Communication Review (2002)
9. A. Back, U. M¨oller, and A. Stiglic, Traffic analysis attacks and trade-offs in anonymity providing systems, In I. S. Moskowitz, editor, Information Hiding (IH 2001), pages 245.257. Springer-Verlag, LNCS 2137, 2001.
10. J. F. Raymond, Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems, In Workshop on Design Issues in Anonymity and Unobservability. Springer-Verlag, LNCS 2009, July 2000.
11. D.L. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, Communications of the ACM 24(2), 84-88 (1981).
12. L. Cottrell, Frequently asked questions about Mixmaster remailers, http://www.obscura.com/~loki/remailer/mixmaster-faq.html (2000).
13. Roger Dingledine, Nick Mathewson, Paul Syverson, Tor: The Second-Generation Onion Router, Proceedings of the 13th USENIX Security Symposium (2004)
14. D. Goldschlag, M. Reed, and P. Syverson, Onion routing for anonymous and private Internet connections, Communications of the ACM 42(2), 39-41 (1999).
15. Zero-Knowledge Systems, http://www.zks.net/ (2000).
16. Anonymizer, http://www.anonymizer.com/ (2000).
17. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, Private information retrieval, Journal of the ACM 45(6), 965-982 (1998).
18. M.K. Reiter and A.D. Rubin, Anonymous web transactions with Crowds, Communications of the ACM 42(2), 32-38 (1999).
19. O. Berthold, H. Federrath, and S. Kopsell, Web MIXes: a system for anonymous and unobservable Internet access, in Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA. Springer: New York (2001).
20. The Rewebber, http://www.rewebber.de/ (2000).
21. M. Waldman, A.D. Rubin, and L.F. Cranor, Publius: a robust, tamper-evident, censorship-resistant, web publishing system, in Proceedings of the Ninth USENIX Security Symposium, Denver, CO, USA (2000).
22. R.J. Anderson, The Eternity service, in Proceedings of the 1st International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT '96), Prague, Czech Republic (1996).
23. R. Dingledine, M.J. Freedman, and D. Molnar, The Free Haven project: distributed anonymous storage service, in Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA. Springer: New York (2001).
24. MUTE: Simple, Anonymous File Sharing., http://mute-net.sourceforge.net/
25. Michael J. Freedman, Robert Morris, Tarzan: A Peer-to-Peer Anonymizing Network Layer, in Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), Cambridge, MA, USA (2002)
26. Stephen C. Bono, Christopher A. Soghoian, Fabian Monrose, Mantis: A Lightweight, Server-Anonymity Preserving, Searchable P2P, Information Security Institute of The Johns Hopkins University, Technical Report TR-2004-01-B-ISI-JHU (2004)
27. Gnutella Developer Forum., http://groups.yahoo.com/group/the_gdf/