PROGRAMMABLE PORT FORWARDING FOR MOBILE PEERS IN PRIVATE NETWORKS

Peter Tabery, Rüdiger Schollmeier, and Christian Bachmeir
Institute of Communication Networks
Munich University of Technology, Germany
E-Mail: {tabery | schollmeier | bachmeir}@ei.tum.de

ABSTRACT
Peer-to-Peer file sharing networks have gained tremendous popularity in recent years. However, traversing Network Address and Port Translators (NAPT) may still fail in certain topologies. In this paper, we present Programmable Port Forwarding, a lightweight approach for allowing private hosts to fully participate in a Peer-to-Peer network. By extending the NAPT that a private host uses to connect to hosts outside its private realm, we enlarge the applicability of Peer-to-Peer systems in today's networks. Additionally, we show that our proposed solution is able to deal with terminal mobility within the domain of the NAPT server as well.

KEY WORDS
Communication Systems, Peer-to-Peer Networks, Network Address and Port Translator, Mobile Communications

1 Introduction

The shortage of public IPv4 addresses is becoming tighter with the continuing increase of the number of Internet hosts. As IPv6 does not catch on [1], many networks employ private IP addresses. Typically, the terminals with private IP addresses access the public Internet through a Network Address and Port Translator (NAPT) [2]. However, the latter does not support inbound connections. This limitation was tolerable when computers were either clients or servers. But with today's Peer-to-Peer communication schemes, hosts may act as client and server at the same time.

Our approach, Programmable Port Forwarding, adds additional functionality to the NAPT entity in order to achieve transparent support for Peer-to-Peer networks. It avoids any modifications to the remote hosts on the Internet, neither to the peering hosts, nor to other NAPT devices. By basing it on platforms for Programmable Networks [3], high flexibility is achieved and the system may be customized easily.

As more and more computing devices come equipped with wireless LAN or Bluetooth, access points are being installed in places like railway stations and airports. Typically, several ones of them are connected to one IP network. Our mechanism may be extended easily in order to deliver lightweight support for the peers' movement between a domain's IP networks. When the peer changes its access network, a matching translator is instantiated on the peer in order to allow for uninterrupted communication. Then, the IP address change is detected by the gateway and inbound datagrams are directed to the new point of attachment.

Like Mobile IP [4], this approach does not require any changes to the IP routing infrastructure, as changes are limited to several mobility-aware entities. However, by restricting our approach to supporting local mobility for external connections—a typical situation in access networks—we are able to significantly reduce the amount of configuration required.

This paper is organized as follows: In Section 2, private networks and their interconnection with the public Internet are presented shortly. In Section 3, communication in Peer-to-Peer networks is described and problems with private IP addresses are identified. In Section 4, the concept of Programmable Port Forwarding is outlined. After discussing briefly current solutions for mobility support in Section 5, our lightweight approach is presented in Section 6.

2 Private Host Addresses

Private IP addresses [5] have been introduced in order to have a clear separation between public and private hosts, i. e. those that are not supposed to connect to external hosts on the public Internet. However today, even hosts with private IP addresses (private hosts) connect regularly to external hosts utilizing a NAPT [2] or application layer gateway resp. proxy. As shown in Fig. 1, private hosts (*hosts 1 and 2*) are able to establish TCP connections (*denoted by solid lines*) to a public host (*host 3*) using a NAPT, forming a Peer-to-Peer (P2P) overlay network (*dotted lines*) that spans across different private address realms (which we also refer to as a *domain* throughout this paper).

In contrary, external hosts cannot connect to a private host, as datagrams with private IP addresses are not routed on inter-domain links. Although private IP addresses are not globally unique, they are unambiguous within a defined scope, e. g. a particular company's domain. Thus, an external host can nevertheless initiate a communication with a private host, when first addressing a public host within the private realm.

For servers with private addresses, port forwarding is used in order to allow external users to connect. In this case, the external host directs its datagrams for the private

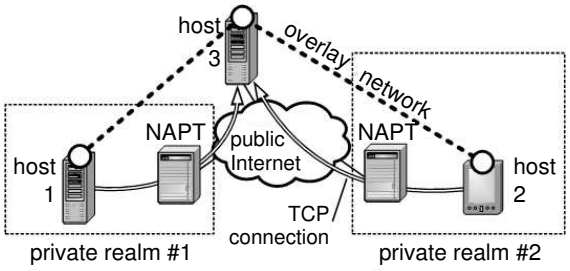

Figure 1. P2P network across private address realms.

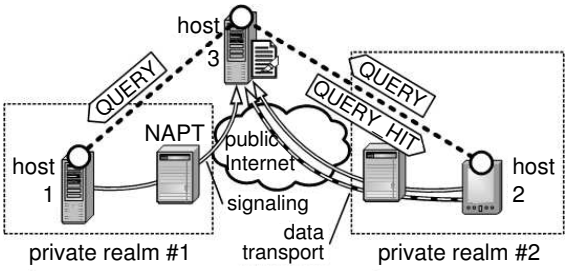

Figure 2. Signaling and data transport in a P2P network.

server to the gateway's public IP address and a particular port. The destination address and port of the datagrams arriving at that port are then translated accordingly. Typically, port forwarding is used for static applications, as it has to be set up manually. A more flexible solution is SOCKS [6], a protocol which allows a private host to open a socket on the gateway dynamically.

3 Peer-to-Peer Networks

In the last few years, since Napster [7] came to life, we can observe a change in the semantic role of the Internet. Peer-to-Peer (P2P) networks make it possible to distribute and find nearly any content on a tremendous number of nodes at the edge of the network. Within this short time, P2P became the dominant usage component of Internet traffic. One year ago P2P already counted up to 51% of the traffic measured in the Abilene backbone [8], with additionally 18% of unidentified traffic, whereas the total traffic accounted to 157.6 TByte transmitted in one week. In research networks with mostly unrestricted Internet connectivity, the amount of bandwidth used by Peer-to-Peer applications even counts up to 60% [9].

Due to reliability reasons, most Peer-to-Peer networks avoid using central index servers like Napster. For locating files, a distributed search mechanism is implemented in the overlay network consisting of globally distributed nodes, which are interconnected by TCP connections. Within the overlay network the nodes offer the content and perform routing tasks. This so called application layer routing enables communication between nodes, which may not be able to connect to each other directly because of NAPT devices.

The traffic mentioned above is caused by a number of P2P protocols and applications, like JXTA [10] offering a P2P framework, Gnutella 0.6 [11] using dynamic hierarchies or hash table based approaches like Chord [12], Pastry [13] or Tapestry [14]. All of them have the same characteristic, that the signaling messages are routed in the overlay network and that the content is transmitted out-band, i. e. via additional TCP connections, as depicted in Fig. 2. Only in Freenet [15] the content is also transmitted on the same connections as the signaling messages.

3.1 Signaling

In the following, we will explain the basic routing functionalities in P2P overlay networks in more detail based on the Gnutella 0.4 protocol [16].

Depending on the bandwidth of the node's network connection, each node in the Gnutella network is connected dynamically to an average of seven nodes [17]. These connections are used to route signaling messages in the network. The messages can be divided into two categories, *query* and *respond* messages. The query messages are used to explore the structure of a terminal's neighborhood (`PING` messages) or to search for certain content, e. g. a particular file in the network (`QUERY` messages).

To avoid a single point of failure, a central look-up table is not implemented in this architecture. Therefore the Gnutella network employs a routing concept, known as viral propagation for the query messages. When searching for content or exploring the network, nodes send out query messages—a `QUERY` or a `PING` message—to all the neighboring nodes, i. e. those directly connected via TCP connections. On receiving this message, the neighbors forward this message to all nodes they are connected to in the virtual overlay network, but not to the one they received the message from. By more or less simply flooding the network, every node is able to explore a certain part of its network neighborhood and to search for data—which is possibly stored on one of the nodes in the network—in a completely decentralized manner without the need for a central entity.

The second type of messages, which are used in the Gnutella network, are the respond messages. These answers may be a `QUERY_HIT` message, if a `QUERY` has been received and the client hosts the demanded content, or a `PONG` message, for making the querying client aware of its presence in response to a `PING` message. As these respond messages are of no interest to the rest of the network, they are routed back to the querying terminal on the same path, that the original query message has been transferred to the receiving node. Flooding the network, as it is employed for query messages, is not adequate for respond messages.

To make backwards routing possible, a unique identifier is attached to every query message, the so called

Gnode ID. This Gnode ID is stored in a routing table together with the IP address of the node, from which it received the query message. The routing table is mostly implemented as a first-in-first-out (FIFO) stack. Thus respond messages can be routed back hop-by-hop to the querying node, as the respond message contains the same Gnode ID as the initial query message.

3.2 Data Transport

Besides the signaling messages, i. e. the respond and query messages, the content a node is querying for must also be distributed through the virtual overlay network. However, to minimize the load on the existing overlay network and especially of its nodes resp. routers, the demanded data is transmitted out-band. As mentioned above, out-band in this context means, that with the help of the information provided in the `QUERY_HIT` message a direct connection between the querying and the responding node is established. This TCP connection is used to transmit the content directly and is closed again as soon as the content has been transmitted.

This works fine, as long as the requesting node can establish a TCP connection to the providing node. This is not the case, if the providing node resides in a private subnetwork behind a NAPT. Only the peer behind the NAPT can establish a TCP connection to peers with public IP addresses, not vice versa. Thus the requestor can not download any content from the providing peer, although it can communicate with this peer via the overlay network. To solve this problem a `PUSH` message is used in Gnutella. This `PUSH` message is transmitted from the querying node to the node providing the content via the overlay. It is routed in the overlay just on the reverse path of the `QUERY_HIT` message. On receiving the `PUSH` message, the providing node establishes a TCP connection to the querying node. Thus the content can also be exchanged between the nodes.

However, if both nodes reside in a private realm, none of them can establish a TCP connection to its counterpart, as none of them has a public IP address. For this case we propose the introduction of our solution based on Programmable Networks, which is outlined in the following section.

4 Programmable Port Forwarding

As pointed out before, Peer-to-Peer (P2P) protocols are typically not NAT-friendly in the sense of [18]. In order to allow for unrestricted communication across Network Address (and Port) Translators, more advanced protocol-specific functionality in the NAPT is needed. We suggest Programmable Port Forwarding as a lightweight approach for providing this additional functionality. In the following, we will consider the Gnutella protocol [11] as it is one of the most widely used ones.

4.1 Scenario

For this section, we assume again a P2P network scenario as depicted in Fig. 2, where two peers are private hosts and connect to the third peer through different NAPT devices. In contrary, however, the requested object is located at host 1 this time:

When host 1 replies to a `QUERY` from host 2 with a `QUERY_HIT` message, the latter will try to reach the private IP address and port stated herein. After realizing that the other peer does not respond—as the private IP address is not routed across the public Internet—it sends a `PUSH` message through the overlay network for requesting a connection setup in the opposite direction, stating its private IP address. This attempt will inevitably fail as well, as both peers are within separate private address realms.

4.2 Static Operation

For solving this problem, we add additional functionality to the NAPT server. Our approach, Programmable Port Forwarding (PPF), enables the direct data transfer between the two private peers by altering the outbound `QUERY_HIT` resp. `PUSH` messages.

First, outbound TCP segments, that are part of the particular overlay network connection, are checked for these messages. In that case, the PPF entity allocates a TCP port on its public interface and adds a mapping to its PPF table:

(*pub.IP*; *pub.port*)↔(*msg.IP_address*; *msg.port*)

Then, this mapping is applied to the message's IP address and port number, whereas the datagram's source IP address and source port number are altered according to the NAPT table. When receiving the `QUERY_HIT` resp. `PUSH` message, the remote private host will initiate a TCP connection with the IP address—which is the NAPT server's address—and port number stated therein. There, all datagrams destined for the particular port are forwarded to the private host's IP address and port, as stated in the PPF table, yielding a successful TCP connection between the two private hosts across two NAPT entities.

If the IP packet's source address and the IP address stated in the `QUERY_HIT` resp. `PUSH` message differ, that message has obviously been relayed already within the Peer-to-Peer overlay network. The PPF entity needs to check whether that private IP address is part of the private realm it serves prior to adding a useless rule to its PPF table. In order to fully rule out the possibility that the message might have come from another private address realm with overlapping address spaces, incoming `QUERY_HIT` and `PUSH` messages have to be evaluated as well. It is subject to further investigation whether this is a serious concern in actual Peer-to-Peer networks topologies.

Although we showed the working of Programmable Port Forwarding exemplarily for the Gnutella protocol [11], the basic principles are valid for the majority of the other Peer-to-Peer network protocols as well. However, every single protocol needs an adjusted functionality on the

NAPT device. Furthermore, even for the same protocol, different users may request diverging levels of support. In a nutshell, our approach leads to a large variety of possible services, where some of them may not even be determined beforehand.

Therefore, we base our approach on platforms for Programmable Networks. The aim of programmable networking consists in simplifying the deployment of new network services leading to networks that explicitly support dynamic service creation, deployment and management in the network infrastructure [3]. Programmable network systems like AMnet [19] support the provisioning of net-centric services to end systems unaware of the additional functionality inside the network. These services may analyze and modify the data passing through e. g. for advanced network monitoring purposes, performance enhancements, and protocol modifications.

As new code for supporting other protocols may be loaded easily on the NAPT gateway whenever a user demands it, the dynamic service creation using Programmable Networks yields the high flexibility needed. This way, intelligence is being shifted from the hosts towards the network itself.

4.3 Security Considerations

In our approach, setting new port forwarding rules in the PPF is controlled by internal hosts only. This is ensured by checking the incoming interface of a packet against the routing table. When assuming that the Peer-to-Peer software of an internal host is not free of errors, an exploit that allows sending arbitrary messages could be used by an external attacker for forwarding any port on the NAPT device. Permanently closing the system ports (i. e. ports < 1024) for the PPF seems a viable remedy, as they are not used by Gnutella anyway.

Hosts or programs in the private realm that are compromitted to a greater extent typically provide a lot more possibilities for attacking other private hosts, making it unnecessary to worry about the port forwarding. Therefore, we think that our approach does not add any considerable security weaknesses to the overall system.

5 Mobility Support

There are several approaches for terminal mobility, i. e. when a mobile node moves from one wireless LAN access point to another one, which is connected to a different network. Mobile IP [4] operates at the network layer whereas approaches like Mobile TCP [20] support mobility on the transport layer.

5.1 Mobile IP

Limiting the modifications needed in the existing network to a few entities at the edge of the network (*home agent*, *foreign agent*, and the *mobile node* itself), while providing fully transparent mobility support, was one of the main design goals of Mobile IP. When away from its home network, the mobile node registers a care-of address with its home agent for packet delivery, typically the IP address of a Mobile IP foreign agent.

In Mobile IP, the mobile node always utilizes one particular, permanently configured home agent, even if another one would have been more favorable. This results in inefficient routing of all of the mobile node's datagrams through the (possibly remote) home agent in order to allow continuing connections across handoffs, as route optimization is not broadly adopted yet.

5.2 Mobile TCP (M-TCP) and MSOCKS

In order to allow a mobile node to resume an ongoing TCP connection after changing its IP address, M-TCP [20] uses *migrate* extensions to standard TCP. M-TCP does not require any changes to the network's infrastructure, but may only be used either when deployed on both communicating end hosts, or if the connection is split by a proxy into an M-TCP and a standard TCP part.

In contrast, MSOCKS [21] is a proxy-based approach, where the mobile node exports a part of its network protocol stack to its MSOCKS proxy. In order to profit from the mobility support provided by the MSOCKS library, the mobile node's *applications* need to be modified accordingly.

Both approaches provide a generally applicable mobility support, that may be used for any TCP-based application, but infer significant protocol and configuration overhead that is unnecessary for simple applications.

6 Programmable Port Forwarding Mobility Extension

Using the existing NAPT in order to provide the level of indirection necessary for supporting the peers' mobility seems like a natural choice for creating a transparent and lightweight mechanism. In the following we assume that, after changing to another network within the current domain, the mobile peer acquires a new (private) IP address, typically using DHCP.

In our solution, for continuously using the established TCP sockets, we add an IP translation table that translates the incoming IP datagrams to the IP address the socket is bound to and the outgoing ones to the IP address currently assigned to the mobile peer, as shown in Fig. 3.

When the first of the mobile node's datagrams arrives from the new network with the topologically correct IP source address, the PPF notes that it is addressed to an already known public IP address and port, but has an unknown IP source address. After verifying, that there is a registered connection with the same TCP source port, sequence and acknowledge counters are then compared to the

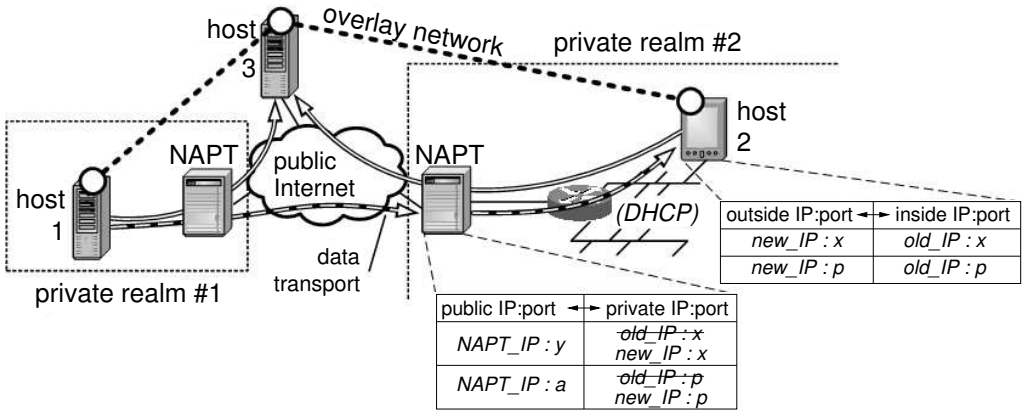

Figure 3. Programmable Port Forwarding and NAPT tables after a mobile peer's handoff.

values of possibly matching transport layer connections. If the counters' values are within an acceptable range, it may be deduced that the current datagram has been sent by the same mobile peer that initiated the connection using a different IP address at another network.

6.1 Acceptable Range Calculation

We argue that the acceptable range for the sequence counter (*SEQ*) and acknowledgment counter (*ACK*) of the *outbound* packet may be deduced from the last *inbound* datagram of a particular connection only, without reducing the accuracy of our approach. Hence, the state information to be stored for each TCP connection comprises the IP addresses, port numbers, sequence and acknowledgment counters, and the advertised window size, effecting 22 octets in total.

When determining the acceptable range for the two counters, we have to keep in mind that, due to dynamic effects and possible packet loss, the sequence and acknowledgment counters of the outbound packet may not refer to the last inbound packet.

For deriving the sequence counter's (*SEQ*) lower bound, we assume a retransmission of the first octet in the sender's window right after the last octet has been acknowledged, supposing that all the inbound acknowledgments have been lost on the wireless link. The upper bound may be elicited by presuming an inbound acknowledge for the first octet of the sending window, followed by the outbound transmission of the last one.

Hence we may state the acceptable range for the sequence counter of the *outbound* packet (denoted SEQ_o) in dependence of the last *inbound* datagram's acknowledgment counter (ACK_i) as follows, where ΔWIN_i is the difference between the maximum TCP window size and the advertised window size:

$$ACK_i - \Delta WIN_i - 1 \leq SEQ_o \leq ACK_i + \Delta WIN_i - 1 \,.$$

For the outbound acknowledgment counter (ACK_o), the acceptable range may derived accordingly as

$$SEQ_i - \Delta WIN_o + 1 \leq ACK_o \leq SEQ_i + \Delta WIN_o + 1 \,.$$

When taking into account that hosts may "shrink the window"—although it is strongly discouraged [22]—the maximum window size has to be considered in both cases instead.

6.2 Probability of a False Match

Obviously, there is a certain probability that two connections are within the same range and the matching algorithm performs an erroneous match. However, this may only happen, when two private hosts connect to the same public host (same destination IP address and port), choosing the same source port. For calculating the maximum probability of a false match under these circumstances, we need to assume that the two hosts connect virtually simultaneously, as the assigned initial TCP sequence number changes every $4\mu s$ with a roll over every 4.55 hours [22]. When assuming that the initial sequence numbers of the two private peers are chosen randomly—which has been confirmed by our experiments—the *maximum probability of a false match* equals to the probability of overlapping windows:

$$2 \cdot WIN_{max}/SEQ_{max} = 2 \cdot 2^{16}/2^{32} = 2^{-15} \approx 3 \cdot 10^{-5} \,.$$

Recent TCP implementations use TCP SYN cookies [23] for protecting servers from TCP SYN attacks. In our investigations, the sequence numbers assigned by the server appeared to increase mostly monotonic with an interval of about 1.1 hour between roll overs. Hence, our worst-case assumption that two hosts may be assigned almost identical sequence numbers—resulting in overlapping windows—is even more unlikely. By combining the values of several TCP connections, the error probability may be decreased further. We may therefore conclude that the probability of a false match is sufficiently low.

The fact that the acceptable range for detecting a client's movement is double the size of the TCP sliding window also doubles the false match probability. Negative impacts due to this inaccuracy may be avoided by transiently *duplicating* the packets to the alleged new IP

address instead. This way, the packet is examined by recipients' TCP instances, where the window size needs to be considered once only. Furthermore this mechanism enhances packet delivery in the case of so-called *ping-pong* handoffs.

6.3 Security Considerations for the Mobility Extension

Several kinds of attacks need to be considered when analyzing the PPF mobility extension. A malicious attacker may attempt to divert the data stream away from the user, hijack that user's TCP connection, or compromise the integrity of the data transmitted. For any of these scenarios, it is required to know the instantaneous status of the respective TCP connections, i. e. IP addresses, ports, and sequence numbers. This information may be obtained easily by eavesdropping in the user's currently visited cell (when assuming that the wired part of the network is properly protected). However, there, an attacker may do any of the things listed above without the support of our architecture.

In case of a handoff, the area where an attacker is able to eavesdrop on the user's communication reaches across two neighboring cells, as the packets are duplicated temporarily. Due to the transient nature of this state, this does not raise additional security issues. As eavesdropping is a general concern in wireless environments, the users are expected to protect the data to be transmitted by cryptographic means. Hence, we conclude that the mobility extension does not compromise the security of the overall system.

7 Conclusion

In this paper, we present Programmable Port Forwarding, a lightweight approach for transparently enabling full connectivity in Peer-to-Peer networks spanning across several private IP address realms. Modifications are only required to the mobile peer and the particular NAPT device, that the private mobile peer utilizes for connecting to the public Internet. Hence, our proposal does not rely on global deployment.

Additionally, we show that Programmable Port Forwarding may be extended easily for supporting terminal mobility. Through limiting the functionality of our mechanism to intra-domain mobility considering external connections of a particular application only, protocol and configuration overhead are kept low, yielding a lightweight solution.

By basing our approach on platforms for Programmable Networks, we close the gap between static network devices, the ever changing application layer protocols, and the users' demands.

This work has been partially supported by the German Federal Ministry of Education and Research within the FlexiNet project (`http://www.flexinet.de/`).

References

[1] Steven J. Vaughan-Nichols. Mobile IPv6 and the future of wireless Internet access. *IEEE Computer*, Feb. 2003.

[2] P. Srisuresh and K. Egevang. Traditional IP network address translator (traditional NAT). RFC 3022, IETF, Jan. 2001.

[3] Andrew T. Campbell, Michael E. Kounavis, and John B. Vicente. *Informatics - 10 Years Back, 10 Years Ahead*, chapter Programmable Networks. Springer Berlin Heidelberg, 2001.

[4] C. Perkins. IP mobility support for IPv4. RFC 3344, IETF, Aug. 2002.

[5] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. RFC 1918, IETF, Feb. 1996.

[6] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS protocol version 5. RFC 1928, IETF, Mar. 1996.

[7] A. J. Howe. Napster and Gnutella: a comparison of two popular Peer-to-Peer protocols, `http://www.csc.uvic.ca/~ahowe/research.html`, Dec. 2000.

[8] Internet2 netflow, weekly reports, week of 20020218, 20020923, 20030505, May 2003. `http://netflow.internet2.edu/weekly`.

[9] G. Foest and R. Paffrath. Peer-to-peer (P2P) and beyond. In *DFN Mitteilungen 58-3*, 2002.

[10] JXTA. `http://www.jxta.org/`.

[11] T. Klingberg and R.Manfredi. *Gnutella 0.6*. Internet Draft (expired), `http://rfc-gnutella.sourceforge.net/draft.txt`, June 2002.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. Technical report, MIT, Mar. 2001.

[13] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, Nov. 2001.

[14] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, U. C. Berkeley, Apr. 2001.

[15] A. Langley. *Peer-to-Peer Harnessing the Power of Disruptive Computing*, chapter Freenet. O'Reilly, Mar. 2001.

[16] Gnutella protocol specification, v0.4, Mar. 2002. `http://www.clip2.com/GnutellaProtocol04.pdf`.

[17] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. Technical report, Univ. of Washington, July 2001.

[18] D. Senie. Network address translator (NAT)-friendly application design guidelines. RFC 3235, IETF, Jan. 2002.

[19] T. Fuhrmann, T. Harbaum, M. Schöller, and M. Zitterbart. AMnet 2.0: An improved architecture for Programmable Networks. In *4th Annual International Working Conference on Active Networks, IWAN 2002, Zurich, Switzerland*, Dec. 2002.

[20] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[21] David A. Maltz and Pravin Bhagwat. MSOCKS: An architecture for transport layer mobility. In *INFOCOM (3)*, pages 1037–1045, 1998.

[22] J. Postel. Transmission control protocol. RFC 793, IETF, Sept. 1981.

[23] D. J. Bernstein. TCP SYN cookies. `http://cr.yp.to/syncookies.html`.