# Active Protocols for Agile Censor-Resistant Networks

*Robert Ricci*  *Jay Lepreau*
School of Computing, University of Utah
Salt Lake City, UT 84112, USA
{ricci,lepreau}@cs.utah.edu  http://www.cs.utah.edu/flux/

## Abstract

*In this paper we argue that content distribution in the face of censorship is a compelling and feasible application of active networking. In the face of a determined and powerful adversary, every fixed protocol can become known and subsequently monitored, blocked, or its member nodes identified and attacked. Frequent and diverse protocol change is key to allowing information to continue to flow. Typically, decentralized and locally-customized protocol evolution is also an important aspect in providing censor-resistance.*

*A programmable overlay network can provide this type of manually-initiated protocol diversification. We have prototyped such an extension to Freenet, a peer-to-peer storage and retrieval system whose goals include censor resistance and anonymity for information publishers and consumers.*

## 1. Introduction

The ability to communicate effectively—even in the face of censorship attempts by a hostile party, such as a repressive government—is important to maintaining the values held by many societies. As The New York Times reported [16] in January 2001, a corrupt head of state was toppled from power, "due in no small part" to 100,000 people responding to a "blizzard" of wireless text messages summoning them to demonstrations. But what if the government had deployed a powerful jamming signal, or simply taken over the cell phone company?

The fundamental rationale for active networking [19]—allowing the network itself to be programmable or extensible by less than completely trusted users—is to ease the deployment of new protocols throughout the network infrastructure. Active networking's flexibility is its only real virtue, since any one protocol can be more efficient, more robust, and have fewer troubling side effects if it is part of the fixed network infrastructure. Thus it has proven difficult to demonstrate even a single compelling application of active networking, as it is really the *space* of applications that is compelling.

Many active networking publications discuss the benefits of facilitating deployment of new protocols over existing networking infrastructure. Overlay peer-to-peer networks such as Freenet [6, 7] that aim to provide a censorship-resistant document publishing mechanism seem a good fit for such a protocol update system: once in place, it is possible that hostile governments, ISPs, or network administrators might attempt to monitor or block the links that make up such a network, effectively censoring the network itself. Indeed, as reported recently by Reuters [17], ISPs, which are typically the last and most vulnerable network hop for users, are coming under pressure from corporations to terminate the subscriptions of customers who participate in, among other things, censor–resistant networks.

One way to make monitoring and blocking more difficult is to hamper an attacker's ability to identify such connections by diversifying the hop-by-hop protocols by which two peer nodes can communicate. Ideally, rather than expanding the set of protocols spoken from one to a small finite set (which could be done by including multiple protocols in software releases), the size of the protocol set should be theoretically unbounded, to prevent attackers from learning every member of the set—or even knowing what percentage of the set they have learned. The protocol set should exhibit a reasonable rate of change over time, to present an adversary with a "moving target."

The keys to making this strategy successful are to allow the deployment of new hop-by-hop protocols at any time, even after the system is in wide use, and to allow any user of the system manually to write and introduce new protocols. The system should be able to evolve rapidly to react to changes in its environment. There should be no central source of new protocols that could become vul-

This research was supported by DARPA and monitored by AFRL under grant F30602-99-1-0503; it was evaluated on emulab.net, the Utah Network Emulation Testbed, primarily supported by NSF grant ANI-00-82493 and Cisco Systems. The views contained in this document are those of the authors and should not be interpreted as representing official policies of any agency of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon.

nerable. Any member of the network should be able to decide upon a new protocol to use, and “teach” its neighbors to use the new protocol. Thus, any node is able to take action quickly if it deems a protocol change to be desirable. In principle, these active protocols may function at any communication layer. Such adaptive, evasive protocols may—and probably will—be inefficient, but above some threshold, that is not a significant concern. Like other contemporary projects, we explicitly choose to exploit the ever-growing supply of network bandwidth and processing power for benefits other than speed.

If the publishing system’s core is implemented in Java [1] or a similar typesafe language, or if the core can interface with such typesafe code, then such evolution can be implemented by using mobile code that implements a new node-to-node protocol. When a node wishes to change the protocol with which it communicates to one of its peers, either because of a suspected attack or as a matter of course, it can send the peer the implementation of the new protocol. We call this passed bytecode a *protocol object*, and a hop-by-hop protocol that can be replaced with protocol objects an *agile protocol*.

## 2. Related Work

Outside of active networks, mobile code has often been used to support heterogeneous environments and platforms (pervasive computing [13] and Sun’s Jini), data transcoding and proxies [12], moving computation to data (mobile agents), or towards more abundant computational or I/O resources (e.g., applets in Web browsers).

In the wireless realm there is a long history of electronic response to jamming, either accidental or purposeful, often using spread-spectrum techniques. Software radios [3] and other more traditional approaches provide adaptive physical-layer protocols. All of these wireless efforts emphasize improved performance by seeking less-used parts of the spectrum, or by using spectrum in a more sophisticated manner. Ad-hoc networks, whether mobile or not, apply adaptive protocols in a more extensive manner [15], and although they must sometimes consider issues of trust, have so far also focused on efficiency and performance.

“Radioactive networks” [4], in which active networking is proposed as a way to extend a software radio infrastructure, comes closest to the ideas in this paper. Their goals are primarily the traditional goals of adaptive wireless protocols: better performance through better use of spectrum and energy. However, they do mention security as a potential benefit, and suggest a software radio system that can vary its spreading codes to avoid jamming.

It is interesting that the first known active network, Linkoping Technical Institute’s Softnet [24], also involved radio, though in a different manner. Softnet, in 1983, implemented a programmable packet radio network, building upon a special Forth environment. It allowed users to extend the network with their own services and protocols above the physical layer.

In terms of censor-resistance, a user-programmable collection of wireless nodes would have strengths that a wired network does not possess. In the latter, typical users are almost entirely vulnerable to their sole ISP. Wireless nodes, particularly if they have software-defined waveforms and a multitude of accessible peer nodes, provide a large set of diverse paths to the broader network.

In our content-distribution application area, there are an increasing number of censor-resistant publishing efforts, some of which are outlined in the next section. To our knowledge, none of them uses active code.

## 3. Censor-Resistant Publishing Networks

Freenet is a censor-resistant publishing network that has received much attention lately, and has been the basis for our work on agile protocols so far. It employs a variety of techniques aimed at creating a censorship-free, anonymous environment. Central to Freenet’s strategy is the distribution of data across a large number of independently-administered nodes. Freenet is decentralized; requests are propagated through a series of peer-to-peer links. When a file is transferred, the file is cached by all nodes on the route between the requesting node and the node where a copy of the file is found (which may not be the original publisher.) Frequently requested files are thus replicated at many points on the network, making removal or censorship of files infeasible. A node *forwarding* a Freenet message from one of its peers is indistinguishable from a node *originating* the message, providing a degree of anonymity for the suppliers and requesters of data. Freenet (and, indeed, any censor–resistant overlay network) must send traffic over an underlying network. Hostile networks can potentially block or monitor its connections, preventing Freenet from fulfilling its goals. Agile protocols, therefore, can potentially provide many benefits.

Other systems incorporate similar ideas in different contexts. Gnutella [11] provides a file sharing and search system that is decentralized across dispersed nodes, but does not maintain endpoint anonymity. Publius [21] offers anonymity for publishers and plausible deniability for servers. The FreeHaven [8] design—so far unimplemented and known to be inefficient—has goals similar to Freenet, but uses more techniques to offer stronger guarantees of anonymity and document persistence.

In all of the above systems, monitoring can reveal information, even if it cannot directly discover the contents of a message, or identify its endpoints [5]. Large quantities of cover traffic, many participating nodes, and widespread routine use by others of end-to-end encryption are required for many of the publishing networks to function effectively. Recognizing that a given data stream belongs to one of these networks is not necessarily difficult, and can give an attacker information on the usage, behaviors or identities of network users. In addition, once such communications are recognized, they can be selectively blocked. Using agile protocols for hop-by-hop communication can make this task much more difficult for an attacker.

## 4. Agile Protocols

### The Case for Agile Protocols

An agile protocol, as we define it, is a protocol that can be changed frequently and more-or-less arbitrarily while the system is running, using active networking techniques. The most flexible way to do this is through mobile code.

If the goal of agile protocols were simply to *obscure* the connections of censor-resistant networks, then it would probably be sufficient simply to encrypt them. It is desirable, however, to be able to *disguise* these connections as well, by communicating over a protocol that is similar to some well-known application-layer protocol, such as HTTP, SMTP, etc. Furthermore, since the traffic generated by a given protocol object may be recognizable (using a known pattern of HTTP headers, for example,) it is also desirable to have multiple implementations that mimic the same application-layer protocol. Statically including some set of these protocols in each release of the software would be a solution, but does not meet the goals laid out in the introduction. Additionally, this would create problems with peer nodes running different versions of the software, as they are not under unified administrative control. Instead, by allowing nodes to exchange new protocol implementations, we make the set of protocols spoken on the network dynamic, regardless of the version of the software that nodes are running, and thus make an attacker's job very difficult.

New protocol objects could implement steganography [14], proxying through third parties, tunneling through firewalls and proxy servers, and other techniques, to make them difficult to monitor and block. Most importantly, using agile protocols allows us to take advantage of such technologies and others yet to be discovered *as soon* as they are developed. Users can write protocol objects customized to their own network situation (for example, when they are behind a firewall that allows only certain ports through, or wish to tunnel their connections through some unusual proxy service) and easily distribute it to their Freenet peers and other users without having to go through any central or even local authority. In response to a determined adversary, new protocols might be written and deployed daily or hourly, all on a decentralized and demand-driven basis. A "metaprotocol" that leverages the agile framework by itself generating agile protocols is obviously possible, as well. However, since such a protocol still presents but one target to an attacker, manual generation of new protocols remains our model's key feature.

### Degree and Level of Programmability

Allowing any user to introduce new protocols into a censor-resistant network presents clear threats. The censors themselves will certainly have the means and motivation to introduce malicious protocol objects. We cannot, therefore, allow arbitrary programmability, but must restrict the API available to the active code. There is an obvious and permanent tension between constraining the active code and allowing it space to diversify.

In addition, it is useful to draw a distinction between the overall architecture of an overlay network, i.e., its global invariants and central algorithms, and the details of its hop-by-hop communication protocol. It is clearly safer to allow programmability of the latter than the former, although very recent work does offer some hope of protecting the overall network [23]. Wetherall's retrospective on active networking [22] reaches an analogous conclusion: under complete programmability, it is feasible to assure the safety of individual nodes, but not of the overall network. By constraining the active aspects of the network to the hop-by-hop protocol, we side-step the very difficult—if not intractable—global security and correctness problems posed by the classic active networking model.

Similarly, operating at the overlay layer entirely avoids a heretofore unsolved problem posed by active networking as it was originally conceived: performance. By "activating" an overlay network instead of a lower protocol layer such as the network layer, we avoid the stringent performance requirements and constrained execution environment found at low level operating system layers.

### Comparison to Traditional Active Networking

The issues that confront agile protocols in general, and Agile Freenet in particular, differ in some ways from the problems that traditionally have been the target of active networking research.

First, new protocols need not be spoken along an entire data path, only on individual point-to-point links. In fact, it is desirable to have a very diverse set of different protocols spoken, in order to make the system as dynamic as possible. A given file transfer could be accomplished with as many different protocol objects as there are links in the route it takes. This eliminates the necessity for a more complicated "capsule" system like ANTS [22], which must ensure that each hop on a given route has the proper (and identical) code to run a given protocol.

Second, file sharing overlay networks such as Freenet tend to be systems where requested data is extremely likely to be available from many sources. Indeed, these networks are designed to tolerate a significant number of non-functioning or malicious nodes. This lessens the importance of a single point-to-point link, as, even if an individual link goes down or misbehaves, data is likely to be available through some other route. Additionally, such networks involve some user interaction, so some classes of problems can be addressed by the users. Thus, we do not require strong guarantees about the correctness or efficiency of each protocol object, as long as the overall system continues to function.

## 5. Agile Freenet

Our work thus far has concentrated on Freenet, because it clearly displays the properties that interest us, has seen wide deployment, and is implemented in a language (Java) well-suited to mobile code. We believe that the principles that have guided our work on Freenet are equally applicable to other censor-resistant publishing networks.

### 5.1. Basics

**Increased flexibility:** Because the Freenet model assumes unrestricted communication between peers, there are many scenarios in which nodes cannot participate, even though there may be roundabout ways for them to bypass local restrictions on network traffic. The best example of this is nodes that are behind proxy servers, such as Web proxies, which may still be able to tunnel connections through these proxies.[1] Extensions to provide this type of communication could be added directly to Freenet, but this does not give users the ability to write protocols that serve their own needs. Strategies for getting messages through interference are not likely to be perfectly known when the system is first deployed, and the types of interference present are likely to change, requiring new tactics. Using agile protocols allows these tactics to be updated as conditions change and to be refined as their real-world success rates are observed.

**Identification:** Identifying protocol objects can be done simply by computing a cryptographic hash of the protocol bytecode—this eliminates any need for a centralized naming scheme, and allows hosts to distinguish between protocol objects, independent of any identifier they may be handed. Identification in such a manner also prevents one protocol object from masquerading as another (a source of potential attacks), because the identifier for a protocol object is tied inextricably to its bytecode.

**Bootstrapping:** Bootstrapping an agile protocol can be problematic—when a node wishes to contact a new potential peer, it must be assured that there is a common protocol that both it and its peer speak. This problem, however, is a fundamental one, shared by the base Freenet system itself: joining the network requires learning the IP addresses and ports of potential peers, through some out-of-band mechanism. The sole change that Agile Freenet brings is in the *amount* of data required for bootstrapping. Instead of a small set of numbers that can be written on a slip of paper or even memorized, prospective peers will need to exchange a modest amount of bytecode or source code. That code could, for example, be passed on a floppy disk, downloaded through the Web, sent as an email attachment, or be transported as a Java applet. The question is whether this *quantitative* change represents a *qualitative* change in the fundamental bootstrapping issue. We believe that it does not. Evidence to support or refute this belief will need to be gained through experience.

[1] Due to their solely client-initiated connection nature, HTTP proxies present challenges to tunneling in peer-to-peer networks. It should work reasonably well for clients to both poll and hold connections open for varied times, thereby allowing callbacks to be accepted. This is an area for future research, especially in the face of traffic and timing analysis.

### 5.2. Potential Dangers

Agile protocols help to thwart listeners or attackers in some ways, but also have the potential to be a tool for attackers. Here, we outline potential dangers; the next section addresses solutions.

**Compromise of local information:** A malicious protocol object inserted into Agile Freenet could attempt to discover information about nodes that it is spread to, such as the files in their cache and their list of peer nodes. Discovery of such information could lead to the compromise of some of Freenet's core goals, such as anonymity.

**Disclosure to an outside source:** A malicious protocol object could contact a third party and disclose addresses of nodes, keys being searched for, data being transferred, etc.

**Failure:** Protocol objects could fail, either maliciously or through poor programming. This failure could be total or intermittent.

**Selective failure:** This is a more specialized instance of failure, in which a protocol object fails only for certain requests or data transfers, to censor specific content.

**Corruption:** Protocol objects could corrupt the data passing through them to disrupt the integrity of the information stored in the censor-resistant network, or to somehow "tag" transfers for tracing.

**Resource Usage:** A malicious or poorly written protocol object could consume excessive system resources, degrading its performance.

### 5.3. Combating the Dangers

Given the dangers outlined above, it is easy to see that some precautions will be necessary when adopting agile protocols. Here, we outline ways of protecting the core system from malicious or poorly written protocol objects.

**Namespace isolation:** With current Java Virtual Machines, it is possible, through ClassLoader objects, to limit the classes which a given object can resolve. This, or similar mechanisms in other typesafe languages, can be used to restrict access to parts of the core software, system classes, etc., that are not necessary for a protocol object to use, and prevent compromise of local information.

**Network isolation:** Using the namespace isolation technique discussed above, we can deny protocol objects direct access to the network, forcing them to use a restricted network API, which can perform checks to ensure that a protocol object is not making unauthorized network access or contacting a third party.

**Rating of protocol objects:** To combat protocol object failure, each node can maintain a rating system for each protocol object it uses, evaluating each one's effectiveness based on factors such as success rate for searches, dropped connections, and detected corruption. It can then decide not to use protocol objects whose rating is too low. Note that this may make perfectly good protocol objects look bad—the peers they are used with may not have much data cached, may be unstable, etc. This is acceptable—the primary goal is to prevent the spread of "bad" protocol objects.

Rating of a protocol object can be done passively by noting the behavior of traffic that flows through it while in service, without incurring much overhead. Protocol objects can also be actively rated by testing them locally: the node can loop data back to itself, through a matched pair of protocol objects in a test harness. The active method is more straightforward and provides more confidence, as the node can directly compare the data sent to the protocol object with that received from it. It is not foolproof, however, as a protocol object may be able to use timing information to distinguish between the real and test environments. Finally, some sophisticated information hiding techniques that buffer and reorder messages [5] may falsely appear to the test harness as functioning incorrectly.

It is important to note that ratings are heuristic, and do not provide absolute guarantees of correctness. This should be acceptable because, as noted above, censor-resistant networks are already designed to tolerate a certain fraction of malicious nodes. It may be that heuristic rating of network entities is a feature that would be worthwhile to extend to nodes themselves.

**Selective failure:** A protocol object may look "good" to the rating system if it only fails for a restricted set of messages. Selective failure and corruption can be tackled by sending every message twice: first through the loop-back test harness, and if successful, to the peer node. It could be solved by encrypting data before passing it to the protocol object, using a key unknown to the protocol. Similarly, separately-encrypted checksums or digital signatures would verify data integrity. However, securely obtaining a shared key poses some problems due to potential man-in-the middle attacks mounted by the suspect protocol object. To cope, public keys of the peer nodes would be required, or a separate channel must be used for a Diffie-Hellman key exchange. The first requires a deployed public-key infrastructure, while the second can be achieved by using a more-trusted protocol (or at least a different one) to exchange keys.[2]

**Resource management:** The node software can be run in a Java Virtual Machine that supports resource management between different Java "applications," using features such as those available in emerging OS-model JVMs such as KaffeOS [2] or Janos [20]. However, just as in Web browsers running untrusted Java code, simpler mechanisms should suffice for initial deployment. E.g., existing OS mechanisms can limit the JVM to a fixed share of memory and the CPU, and the node user or administrator can be notified if a limit is consistently reached.

[2] Space limitations preclude further details here. We refer readers to the first author's thesis [18].

### 5.4. Experience

We have implemented the basic framework described in this paper by modifying the current Freenet implementation [10] to incorporate protocol objects. Conveniently, this implementation is written in Java, facilitating the code mobility and language safety features discussed in this paper. Since our extensions to Freenet were different than those envisioned by its developers, we found it moderately difficult to extend. However, once the framework was in place, the resulting extensible system was workable.

Our prototype sends the bytecode for protocol objects over the network and loads it into a restricted Java execution environment using standard Java ClassLoader mechanisms; sensitive Freenet and system APIs are hidden. We provide a restricted network API that allows protocol objects to open network sockets to their peer node, but prevents them from determining the network address of the remote node. We implemented four different protocol objects. One implements the standard Freenet protocol, while another mimics HTTP syntax to facilitate tunneling through HTTP. The third implements TCP "port-hopping" on a per-message basis, and the fourth uses a simple spread-spectrum scheme, splitting individual messages across several TCP ports. Nodes can, at the behest of their peers, change protocol objects at any Freenet "message" (file) boundary. This diversity is on a per-peer basis, allowing a node to speak an arbitrarily different protocol, on a different port, at different times, to each of its peers. Protocol objects can be used asymmetrically—messages passing in different directions along the same peer-to-peer link can use different protocols.

The prototype was developed and tested on `emulab.net`, our scalable network emulation testbed [9]. We plan to continue development of our prototype as a test platform for research on agile protocols.

## 6. Conclusion

We believe that censor-resistant content distribution networks provide a compelling application of active networking technology, since many such networks inherently need diverse protocols more than they need any particular protocol. Through frequent, diverse, decentralized, and locally-adapted protocol change, agile protocols seem likely to improve such networks' resistance to monitoring and blocking, without an undue increase in the potential damage from malicious protocols. We have demonstrated, in the Freenet system, that such an extension is feasible, although there are enough potential issues that it is still unclear whether it will be practically useful.

Among the work remaining is to implement protocol ratings, increase the range of protocol variants, evaluate the extent of improvement through simulation and in the laboratory, and deploy and evaluate the system in the live Freenet. It would be valuable to experiment with censor-resistant networks other than Freenet, to help identify any artifacts due to the base system. Finally, we point out that it should be easy to interest students and others in active contests between publishers and censors, aiding evaluation in both artificial and live environments.

### Acknowledgements

We thank Chad Barb for his useful discussion and help. We also thank Alastair Reid, Dave Andersen, Parveen Patel, John Regehr, Pat Tullmann, and the anonymous referees for their comments, which helped improve this paper.

## References

[1] K. Arnold and J. Gosling. *The Java Programming Language*. The Java Series. Addison-Wesley, second edition, 1998.

[2] G. Back, W. C. Hsieh, and J. Lepreau. Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. In *Proc. of the Fourth Symposium on Operating Systems Design and Implementation*, pages 333–346, San Diego, CA, Oct. 2000. USENIX Association.

[3] V. Bose, M. Ismert, M. Welborn, and J. Guttag. Virtual Radios. *IEEE Journal on Selected Areas in Communications*, 17(4), Apr. 1999.

[4] V. Bose, D. Wetherall, and J. Guttag. Next Century Challenges: RadioActive Networks. In *Proc. of the Fifth Annual ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, pages 242–248, Aug. 1999.

[5] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.

[6] I. Clarke. A Distributed Decentralized Information Storage and Retrieval System. Master's thesis, Univ. of Edinburgh, 1999. URL: http://freenet.sourceforge.net/freenet.pdf.

[7] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. Berkeley, California, July 2000. International Computer Science Institute. Revised December 18 2000. Available at http://freenet.sourceforge.net/icsi-revised.ps.

[8] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. Berkeley, California, July 2000. International Computer Science Institute. Revised December 17 2000. Available at http://www.freehaven.net/doc/berk/freehaven-berk.ps.

[9] Flux Research Group, University of Utah. emulab.net: Utah Network Testbed and Emulation Facility. http://-www.emulab.net/.

[10] The Freenet Project. http://freenet.sourceforge.net/.

[11] The Gnutella Project. http://gnutella.wego.com/.

[12] S. D. Gribble et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Computer Networks*, 2001. To appear, special issue on Pervasive Computing. http://ninja.cs.berkeley.edu/dist/papers/ninja.ps.gz .

[13] R. Grimm, T. Anderson, B. Bershad, and D. Wetherall. A System Architecture for Pervasive Computing. In *Proc. of the Ninth ACM SIGOPS European Workshop*, pages 177–182, Kolding, Denmark, Sept. 2000.

[14] N. Johnson and S. Jajodia. Exploring Steganography: Seeing the Unseen. *IEEE Computer*, 31(2):26–34, 1998.

[15] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proc. of the Fifth Annual ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, Aug. 1999.

[16] Text Messaging is a Blizzard That Could Snarl Manila. *New York Times*. January 20, 2001.

[17] Reuters. Web Piracy–Crackdown Spawns Stealth Platforms. Available at http://dailynews.yahoo.com/h/nm/-20010430/wr/media_web_piracy_dc_1.html.

[18] R. Ricci. Agile Protocols: an Application of Active Networking to Censor–Resistant Publishing Networks. Bachelor's thesis, University of Utah, Aug. 2001. 39 pages. http://www.cs.utah.edu/flux/papers/ricci-thesis-base.html.

[19] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *Computer Communication Review*, 26(2), Apr. 1996.

[20] P. Tullmann, M. Hibler, and J. Lepreau. Janos: A Java-Oriented OS for Active Network Nodes. *IEEE Journal on Selected Areas in Communications*, 19(3):501–510, Mar. 2001.

[21] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant, Web Publishing System. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[22] D. J. Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Proc. of the 17th ACM Symposium on Operating Systems Principles*, pages 64–79, Kiawah Island, SC, Dec. 1999.

[23] A. Whitaker and D. Wetherall. Forwarding Without Loops in Icarus. Submitted for publication. http://www.cs.washington.edu/homes/andrew/-infocom02.pdf, July 2001.

[24] J. Zander and R. Forchheimer. Softnet – An approach to high level packet communication. In *Proc. Second ARRL Amateur Radio Computer Networking Conference (AMRAD)*, San Francisco, CA, Mar. 1983.