Delft University of Technology
Parallel and Distributed Systems Report Series

BuddyCast: An Operational Peer-to-Peer Epidemic Protocol Stack

J.A. Pouwelse, J. Yang, M. Meulpolder, D.H.J. Epema, H.J. Sips
J.A.Pouwelse@tudelft.nl

report number PDS-2008-005

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.twi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://pds.twi.tudelft.nl/

Abstract

Over the last few years, much research has been performed on epidemic (or gossiping) protocols, but much of it is of a theoretical nature, which leads to the question how such protocols would perform in practice. To address this issue, we present the first large-scale Internet-deployed epidemic protocol, or rather, protocol stack, called *BuddyCast*. BuddyCast is the core of our Tribler P2P file sharing software, which has been downloaded over 150,000 times, and has gone through a number of versions as a result of our experience with its actual deployment in the Internet.

We formulate three challenges for Internet-deployed epidemic protocols: they have to address availability, connectivity, and security. Measurements of BuddyCast show that it does address the first two of these challenges, among other things by maintaining a *live overlay*. As a first step towards addressing the security challenge, we demonstrate that the BarterCast protocol in the BuddyCast protocol stack is effective at spreading information on bartering activity, which enables the calculation of reputations. With BuddyCast we have for the first time moved epidemic protocols from the realm of trusted ideal environments into the complicated environment of the Internet.

Contents

1 Introduction **4**

2 Challenges for the Internet deployment of epidemic protocols **4**

3 The BuddyCast protocol stack **5**

4 Availability and connectivity **6**

5 Security: The BarterCast protocol **9**

6 Related work **11**

7 Conclusions and future work **11**

List of Figures

1 The initial BuddyCast message format. 5
2 The BuddyCast epidemic protocol stack. 5
3 Information reliability in BuddyCast messages. 6
4 Connection attempts after going offline. 7
5 Information reliability after improvements. 8
6 The collecting of torrents in BuddyCast 8
7 Screenshot of the BarterBrowser. 10

List of Tables

1 Introduction

Distributing dynamic information across a large number of computers is a central problem in distributed systems design. Epidemic protocols offer a mechanism for information distribution without relying on central servers. Their simplicity, scalability, and good performance characteristics have made them an active research topic over the last few years [2, 3, 5, 7, 12].

The basic principle of epidemic protocols is simple and elegant. Peers in a network connect to each other, they engage in peerwise information exchange, and process the information received. Based on this receive-and-forward primitive, a self-organizing system for information exchange can be created with high availability and fault-tolerance. When designing epidemic protocols, three questions have to be answered. The first question is that of peer selection. We will argue that merely connecting to random other peers is not good enough (if that would be possible in the first place). The second question is that of selecting the information to gossip, which is very application dependent. The third question is that of information retention, i.e., deciding which information received through gossiping to store (semi-)permanently (the *infected forever* model) and which to forget quickly (the *infect and die* model).

Almost all of the research into epidemic protocols to date has been of an exploratory and theoretical nature, and no case study has been published of an epidemic protocol with a large-scale deployment on the PCs of average users. To fill this gap, we have designed, implemented, and deployed an epidemic protocol called *BuddyCast*. The initial version of BuddyCast of 2005 was deployed from Jan 2006 onwards to thousands of users, and forms the core of our Peer-to-Peer (P2P) file sharing system *Tribler* [9]. The number of Tribler client downloads has recently crossed the 150,000 mark. The cardinal features of BuddyCast are (i) discovery of peers, (ii) discovery of content, and (iii) formation of a *semantic overlay* [12, 13]. These features together with our Bittorrent-compatible download engine form a complete P2P client. In the past years we have continously measured the live network and subsequently improved BuddyCast.

The contributions of this report are as follows. First, we present the challenges of the Internet deployment of the epidemic protocols as discussed in this report. Second, we present the initial version of our BuddyCast protocol. Third, we show measurements and discuss inefficiencies, security vulnerabilities, and design flaws (or rather, lessons learned) in this initial BuddyCast protocol. Fourth, we describe the improvements of BuddyCast addressing resilience against peer failure, bandwidth efficiency, information propagation speed, dealing with NAT/Firewalls, and hardening against malicious peers. The source code of BuddyCast and the network traces are publicly available at `http://www.Tribler.org`.

2 Challenges for the Internet deployment of epidemic protocols

The overall goal of an Internet-deployed epidemic protocol is to transform a set of peers which are potentially malicious, often off-line, and frequently behind a Firewall or Network Address Translation (NAT) box into a secure, reliable, and fast network for information dissemination. This means that the deployment on the Internet presents serious challenges to an epidemic protocol in terms of availability, connectivity, and security.

availability The availability challenge is related to peer selection. Peers are often online for only a few minutes or a few hours and are not willing to execute graceful log-off procedures [10]. The challenge is thus selecting peers which have a high probability of being online, while still providing proper load balancing.

connectivity PC users are often "hidden" behind a NAT/Firewall, thus creating a connectivity challenge. When two peers are hidden behind a NAT/Firewall, they cannot establish a connection and communicate. Dealing with this limited connectivity is another challenge for peer selection.

security A major security challenge is malicious peers which coerce a large number of other peers into sending a message to a target host. This is a classic form of denial-of-service (DoS) which has now been studied in the decentralised context of P2P [3, 4]. Weak P2P networks have been exploited to create attacks with 100,000 coerced machines.[1] Preventing *information poisoning* is another big security challenge. Any

[1] `http://news.netcraft.com/archives/2007/05/23/p2p_networks_hijacked_for_ddos_attacks.html`

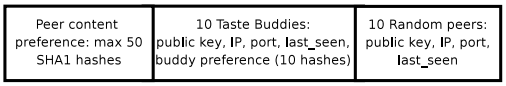

Figure 1: The initial BuddyCast message format.

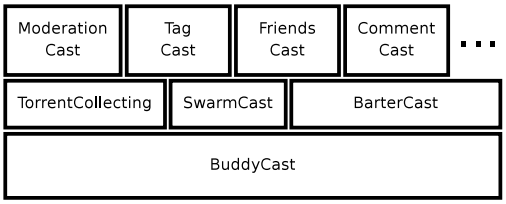

Figure 2: The BuddyCast epidemic protocol stack.

peer can launch an attack by sending out garbage.

3 The BuddyCast protocol stack

In this section, we first review the original BuddyCast epidemic protocol [9]. Subsequently we present the more sophisticated BuddyCast protocol stack.

In Tribler, Buddycast supports the three following features: peer discovery, content discovery, and formation of a semantic overlay [13]. Figure 1 shows the layout of the initial BuddyCast message. Every BuddyCast message contains the top 50 content preferences of a sending peer, information about 10 other peers which have a similar taste called *taste buddies* (identified by applying a similarity function to preference lists), and information about 10 random peers. The *last_seen* field indicates when a particular peer was last seen online. Information about discovered content and peers, and their preferences contained in incoming messages are stored in a local database called the *MegaCache* in just a few MBytes (following the infect-forever model). The taste buddies form the important semantic overlay of BuddyCast.

Peers send a BuddyCast message periodically every 15 seconds (a *round*), or in response to a received BuddyCast message (so in effect, BuddyCast is an epidemic *exchange* protocol). The peer selection algorithm of BuddyCast has a probability of p of selecting a random peer (*exploring* the network for new peers and content) and of $1 - p$ for selecting a taste buddy (*exploiting* the knowledge of taste buddies). Currently, we use $p = 0.5$. To prevent BuddyCast from contacting the same peers too often, a peer is contacted at most once per *cycle* of 4 hours.

The initial BuddyCast algorithm is very similar to the "anti-entropy" algorithm proposed for replicated database maintenance [1]. Subsequent improvements increased this similarity further. BuddyCast has grown from a simple epidemic protocol into the *substrate* of a complete epidemic protocol stack. Figure 2 shows the BuddyCast protocol stack with the BuddyCast substrate. The substrate selects the peers to synchronise with and the higher layer protocols do the synchronisation of MegaCache differences.

For example, the function *TorrentCollecting* is a separate process in the initial BuddyCast implementation which traverses the network in search of the complete metadata associated with a certain SHA1 hash. We realised that using a single peer selection function was more efficient and moved TorrentCollecting on top of BuddyCast. The *SwarmCast* protocol discovers peers in a Bittorrent swarm belonging to certain content, using peers that have shown interest in this content as the starting point [11]. The four functions in the highest layer of the protocol stack are work in progress. For instance, we are building a decentralized reputation system, a protocol for exchanging metadata and tags, and a protocol for decentralized social networking with friends and friends-of-friends.

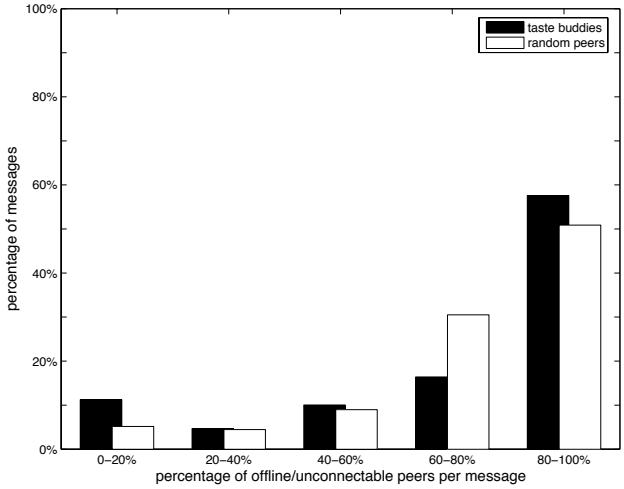

Figure 3: Information reliability in BuddyCast messages.

4 Availability and connectivity

In this section, we will assess how the deployed BuddyCast epidemic protocol stack addresses the availability and connectivity challenges of Internet deployment stated in Section 2. In particular, we will show that we have significantly enhanced the performance of the intial BuddyCast algorithm by removing randomness and adding intelligence, by keeping open TCP/IP connections, by adding NAT/Firewall detection, and by varying the BuddyCast round time.

Prior work has largely ignored the Internet availability and connectivity challenges, and we find that no usable performance indicators have been proposed to quantify the performance of a peer selection algorithm. We have previously measured the efficiency of an Internet-deployed swarm discovery protocol using the Kademlia DHT and found that the delivery of 84.4% of the outgoing DHT messages fails [11]. As a suitable performance metric of a peer selection algorithm we propose the *peer selection efficiency*: the percentage of successfully delivered outgoing messages. This metric accurately reflects the ability of a such an algorithm to find peers which are both available and connectable.

We have performed a number of experiments to determine the peer selection efficiency of BuddyCast. Key to good peer selection is not only a good algorithm, but also reliable input information to such an algorithm. To determine the *information reliability* of BuddyCast, we have investigated the availability and connectability of the peers contained in BuddyCast messages at the moment of receiving such messages. To this end, we have continuously run a Tribler client for over 500 hours and probed all the peers in each received BuddyCast message to see how many peers in these messages are actually online and connectable. At the time of these experiments, there were about 15,000 Tribler clients in the world who had actually downloaded files and the experimental client received 5049 messages in total. The results are presented in Figure 3, which shows how many of the taste buddies and random peers in a message were offline or unconnectable (see Figure 1). As can be seen, most peers in a BuddyCast message are offline, while random peers have higher chance to be online. Almost 60 % of the messages contain taste buddy information which is very unreliable (81-100% offline/unconnectable). A peer selection algorithm which is based on such unreliable information is likely to have poor efficiency.

We have performed another experiment for assessing the peer selection algorithm of BuddyCast in which we determine the number of connection attempts made to a peer after it has gone offline. Ideally, it is detected within minutes, after a few failed connection attempts, that a peer has gone offline. For this experiment, we created special software which runs as a normal peer, but can still detect incoming socket connection attempts after going offline. We started a single peer with this software, which was online for a

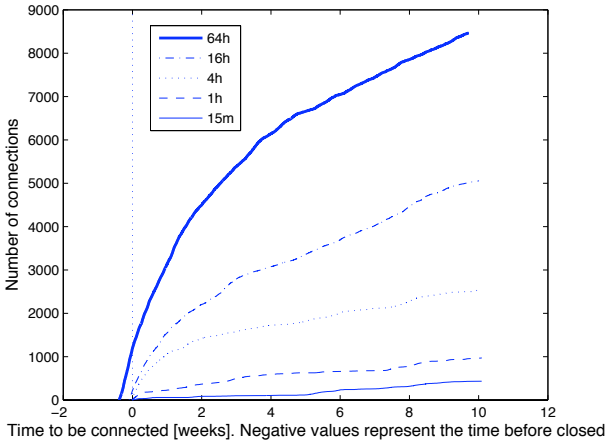

Figure 4: Connection attempts after going offline.

period of time ranging from 15 minutes to 64 hours before going offline. Figure 4 shows the results of this offline detection experiment. Clearly, excessive *offline-peer hammering* occurs, especially when a peer has had a long uptime, and our offline peer is still contacted frequently even after 10 weeks. The algorithms presented in prior work also suffer from this serious defect [5, 7, 12].

We now briefly present three improvements designed to boost the peer selection efficiency and to reduce offline-peer hammering. Further details are available online.[2]

First, in order to exclude unconnectable peers from BuddyCast messages, we have created a mechanism to detect connectivity problems. Using a new *DialBackMessage*, peers can detect their own connectivity and report it in outgoing messages, signaling to others not to include them in their BuddyCast messages.

Secondly, to exclude offline peers from BuddyCast messages, BuddyCast maintains a *live overlay*. For this purpose, every peer maintains open TCP connections to 10 taste buddies and to 10 random peers, thus continuously verifying their online status. This means that when a peer establishes a connection to one of the (connectable) peers in a BuddyCast message it has just received, in effect it opens a connection based on first-hand information about the target peer being online. Connections in the live overlay are dropped when a peer stops being connectable, when a peer was included as a random peer for a number of times, or when a peer with a superior similarity is encountered. We essentialy use a hill-climbing approach to build a robust, live semantic overlay by preserving connections to the 10 most similar online peers.

Thirdly, we have replaced the original probabilistic peer selection algorithm with an intelligent deterministic variant. A new *connection candidate cache* is used to store the 100 discovered freshest peers that have not yet been contacted in the current cycle. We connect to either the freshest peer or the most similar peer in this cache, subject to the random peer selection probability p. Peers running the original algorithm are contacted only if no peers with the newer algorithm are left.

Apart from the above three major improvements, we made other modifications to make BuddyCast more efficient. We introduced a dynamic round time ranging from 1 second to 1 minute to improve the initial bootstrap performance. Furthermore, we accelerated TorrentCollecting by including a list of recently discovered .torrent files in a message, enhancing the discovery of rare information.

We have run experiments that show that the improvements have a strong positive effect on the information reliability, as can be seen in Figure 5. In 80% of the messages, almost none of the random peers and 70% of taste buddies are offline or unconnectable. We conducted another experiment to observe the overall effects of these protocol stack substrate improvements. Figure 6 shows the large difference between the information

[2] `http://www.Tribler.org/BuddyCast`

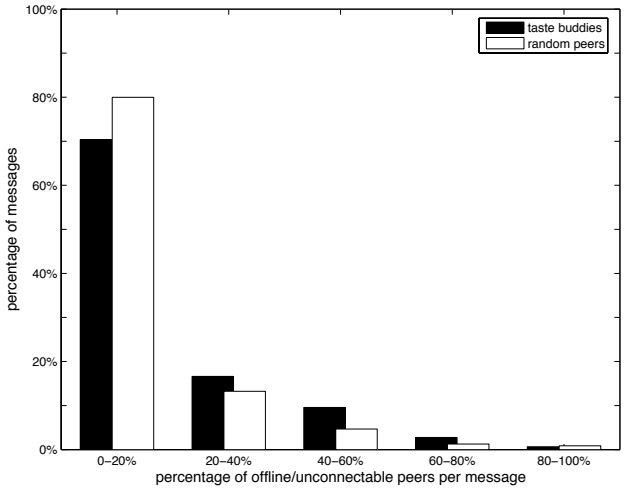

Figure 5: Information reliability after improvements.

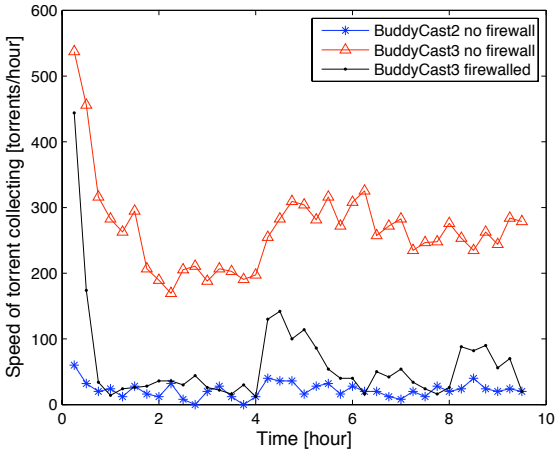

Figure 6: The collecting of torrents in BuddyCast

discovery speed of the original and the improved algorithm. In these experiments we started a Tribler peer and measured the TorrentCollecting speed. Due to the added dynamic round time, a connectable peer starts collecting 540 torrents per hour, a significant improvement from the original which does not go beyond 10 torrents per hour. The 4-hour cycle time is also clearly visible, as peers are re-visited after this time. A lack of connectivity clearly hampers performance, hence an incentive is provided to address NAT/Firewall issues.

Finally, we measured the peer selection efficiency. The BuddyCast efficiency was originally 15.8%, but after all of our improvements it increased to 79.6%.

5 Security: The BarterCast protocol

We now present the first step for a new approach in security research for epidemic protocols using *Socially-Inspired Reputations* (SIR).

The challenge of security is to identify malicious peers, to prevent DoS attacks, and to prevent information poisoning. Our approach is based on the *BarterCast* epidemic protocol that continously spreads information about the positive behavior of peers. Improving Internet security has proven to be a continuous and difficult challenge. Our contribution consists of an operational mechanism to gather information on peer behavior to enable the identification of malicious peers. Using this information effectively and preventing attacks is outside the scope of this report and left for future work.

The auction site `EBay.com` shows that a reputation management system can be used to expose malicious behavior. Early work on reputation management in distributed systems was aiming at global consistency, see for instance EigenTrust [8]. To thwart phishing attacks, recent work such as NetTrust[3] uses a centralised approach which is "grounded in human behavior".

After centuries of human evolution a highly evolved mechanism for judging the *believability* of gossip has emerged. The experiments with human subjects of Hess and Hagen around gossip (e.g., "Cathy was hitting on the company president") and believability have produced an initial understanding of the underlying mechanism [6]. For instance, they show that the believability score increased by 2.2 points on the 1-9 Likert scale when the number of sources increased from 1 to 4 gossipers. They also show that source independence increases believability.

SIR presumes that *distributed reputation systems must be inspired by human mechanisms to judge believability of gossip*. BarterCast is our vehicle to demonstrate SIR. BarterCast uses the peer selection algorithm of the BuddyCast substrate, as shown in Figure 2, to exchange information on positive actions of other peers. By storing all this "gossip" in the MegaCache, each peer can use the opinion of others to estimate trustworthiness levels. BarterCast uses a single message which is exchanged after a BuddyCast message. In our Tribler P2P client we "gossip" on the donation of upload bandwidth. Each peer keeps track of which peers have given him the most bandwidth and uses this to form an outgoing BarterCast message. This message contains a complete *barter* record, with both the number of uploaded and of downloaded MBytes, for 10 peers. Consequently, every BuddyCast cycle a peer tries to receive fresh information on barter activities from 960 gossipers.

After we deployed BarterCast, we started a Tribler peer and allowed it to run continuously for a month, while we downloaded and uploaded a number of files. At the end of this period, the MegaCache contained information about the upload and download behavior of 2351 peers in total, 118 of which directly bartered with this peer. Note that the direct barter records constitute the only *validated* information. The accumulated data represents detailed knowledge about more than 160 GB of data transfer between Tribler peers. In order to make all this gossip on bartering visible we created a real-time *BarterBrowser*. A screenshot of the BarterBrowser with live statistics is shown in Figure 7. The large peer at the center depicts the local peer, other peers are positioned around it, connected by edges induced by barter gossip. Only the gossip information with the shortest "barter path" (up to four steps away) to the local peer is shown. By selecting a peer, additional details are displayed such as additional statistics, nickname, and avatar. The peers in the gossiped network have an average degree of 2.34 and a maximum degree of 53.

[3] `http://www.ljean.com/NetTrust/`

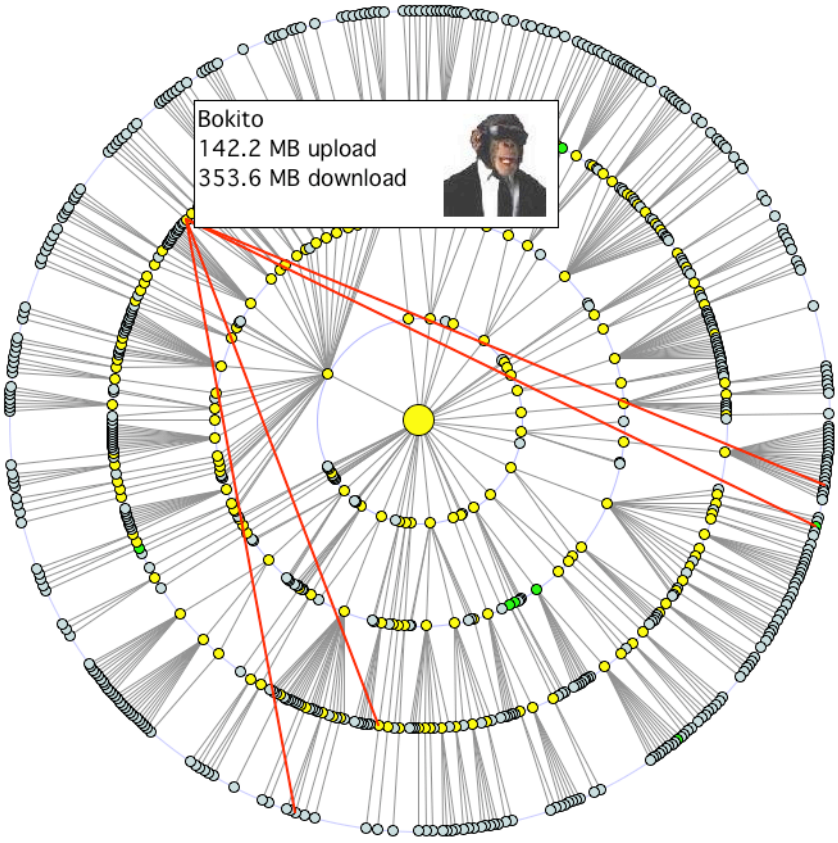

Figure 7: Screenshot of the BarterBrowser.

With BarterCast we believe we have successfully demonstrated the viability of distributing information for calculating reputations. Our experiments show an exponentially growing amount of gossiped information, which provides an immense amount of data for our SIR approach. Key to SIR is that we should employ human inspired mechanisms to judge the believability of this gossip, for instance, by relying on multiple independent sources.

As future work we plan to combine the social networking of FriendsCast (see Section 2) with BarterCast, in order to obtain the trusted first-hand observations of our friends and friends-of-friends. This would further increase our ability to identify malicious peers, prevent DoS attacks, and prevent information poisoning.

6 Related work

The research on epidemic protocols can be divided into three areas: databases, "light protocols", and Internet deployment. Two decades after the first description of epidemic protocols in the context of databases [1], the use of such protocols for database replication is still not very common. The work on protocols in the second area is focused on minimizing statefulness and complexity in trusted environments [7]. A key achievement here was going beyond simulations with the 2002 experiment with 150 sensor nodes equipped with 8 KByte programming memory [2]. Randomness is the foundation for choices in this area, as intelligence adds complexity.

Internet deployment is the most recent area. Light-protocol ideas (randomness, 50% message loss, no security) are combined in [5] with a proposal to address availability and connectivity issues. An early proposal to address security is based on the assumption that information from independent sources is trustworthy [3]. The security challenge is elegantly quantified in [4] by creating a heavy real-world DoS attack with 100 million malicious packets.

7 Conclusions and future work

In this report we have presented the design of and experiments with BuddyCast, the first Internet deployment of what will be a complete epidemic protocol stack. BuddyCast addresses the challenges of availability and connectivity due to peers being offline or behind a NAT/Firewall, and it is intended to address the challenge of security by employing the BarterCast protocol that spreads bartering information. We have introduced a new performance metric called the *peer selection efficiency*, which attains a value of 79.6% for BuddyCast, thus showing that BuddyCast is both effective and efficient.

BuddyCast departs from the "light" approach to epidemic protocol design, for instance by keeping 20 open TCP/IP connections, by adding complexity in peer selection (intelligence vs. randomness), by storing state (with a full SQL-light database instead of a 10-entry cache), and by validating gossiped information.

We have learnt two lessons during our trial-and-error Internet deployment. First, while previous epidemic protocols employ randomness, our experiments show its serious defect of offline-peer hammering. Instead, we advocate maintaining a *live* semantic overlay which naturally creates locality and robustness. Secondly, the efficiency of an epidemic protocol increases with statefulness and complexity; context retained from previous sessions jumpstarts peers and information discovery.

Finally, we conjecture that *Social-Inspired Reputations* are key to calculating accurate reputations and to addressing the security challenge. Security research can benefit greatly from exploiting techniques used by humans while dealing with actual social gossip.

References

[1] A. D. et al. Epidemic algorithms for replicated database maintenance. In *6th ACM symp. on Principles of Distributed Computing*, Vancouver, Canada, Aug 1987.

[2] D. G. et al. An emperical study of epidemic algorithms in large scale multihop wireless networks. Technical Report IRB-TR-02-003, Intel Research, 2002.

[3] D. M. et al. Gossip with malicious parties. Technical report, Leibniz Center, Hebrew University, Jerusalem, 2003.

[4] E. A. et al. Misusing unstructured p2p systems to perform dos attacks. In *Proc. of ACNS 2006*, pages 130 – 145, 2006.

[5] N. D. et al. Arrg: Real-world gossiping. In *Proc. of the 16th International Symposium on High Performance Distributed Computing*, pages 147 – 158, 2007.

[6] N. Hess and E. Hagen. Psychological adaptations for assessing gossip believability. *Human Nature*, 2006.

[7] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, VU, 2002.

[8] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *12th WWW conference*, New York, 2003.

[9] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 19, 2007.

[10] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *4th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, volume 3640 of *LNCS*, pages 205–216. Springer-Verlag, Feb 2005.

[11] J. Roozenburg. Secure Decentralized Swarm Discovery in Tribler. Master's thesis, Delft University of Technology, 2006.

[12] S. Voulgaris and M. van Steen. Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *Euro-Par 2005*, volume 3638 of *LNCS*, pages 1143–1152. Springer-Verlag, Aug 2005.

[13] J. Wang, J. Pouwelse, J. Fokker, and M. J. Reinders. Personalization of a peer-to-peer television system. In *Proc. of European Conference on Interactive Television (EuroITV 2006)*, 2006.