

# The Case for Cooperative Networking

Venkata N. Padmanabhan\*  
Microsoft Research

Kunwadee Sripanidkulchai†  
Carnegie Mellon University

*Abstract—*

**In this paper, we make the case for Cooperative Networking (CoopNet) where end-hosts cooperate to improve network performance perceived by all. In CoopNet, cooperation among peers complements traditional client-server communication rather than replacing it. We focus on the Web flash crowd problem and argue that CoopNet offers an effective solution. We present an evaluation of the CoopNet approach using simulations driven by traffic traces gathered at the MSNBC website during the flash crowd that occurred on September 11, 2001.**

## I. INTRODUCTION

There has been much interest in peer-to-peer computing and communication in recent years. Efforts in this space have included file swapping services (e.g., Napster, Gnutella), serverless file systems (e.g., Farsite [2], PAST [11]), and overlay routing (e.g., Detour [13], RON [1]). Peer-to-peer communication is the dominant mode of communication in these systems and is central to the value provided by the system, be it improved performance, greater robustness, or anonymity.

In this paper, we make the case for Cooperative Networking (CoopNet), where end-hosts cooperate to improve network performance perceived by all. In CoopNet, cooperation among peers complements traditional client-server communication rather than replace it. Specifically, CoopNet addresses the problem cases of client-server communication. It kicks in when needed and gets out of the way when normal client-server communication is working fine. Unlike some of the peer-to-peer systems, CoopNet does not assume that peer nodes remain available and willing to cooperate for an extended length of time. For instance, peer nodes may only be willing to cooperate for a few minutes. Hence, sole dependence on peer-to-peer communication is not an option.

The specific problem case of client-server communication we focus on is *flash crowds* at Web sites. A flash crowd refers to a rapid and dramatic surge in the volume of requests arriving at a server, often resulting in the server being overwhelmed and response times shooting up. For instance, the flash crowds caused by the September 11 terrorist attacks in the U.S. overwhelmed major news sites such as CNN and MSNBC, pushing site availability down close to 0% and response times to over 45 seconds [18].

Flash crowds are typically triggered by events of great interest — whether planned ones such as a sports event or unplanned ones such as an earthquake or a plane crash. However, the trigger need not necessarily be an event of widespread global interest. Depending on the capacity of a server and the size of the files served, even a modest flash crowd can overwhelm the server.

The CoopNet approach to addressing the flash crowd problem is to have clients that have already downloaded content to turn around and serve the content to other clients, thereby relieving the server of this task. This cooperation among clients is only invoked for the duration of the flash crowd. The participation of individual clients could be for an even shorter duration — say just a few minutes. We argue that the CoopNet approach is self-scaling and cost-effective.

The rest of this paper is organized as follows. In Section II, we present our initial design of CoopNet and discuss several research issues. In Section III, we analyze the feasibility of CoopNet using traces gathered at MSNBC [20], one of the busiest news sites in the Web, during the flash crowd that occurred on September 11, 2001. We conclude in Section IV by comparing CoopNet with alternative approaches to addressing the flash crowd problem.

## II. COOPERATIVE NETWORKING (COOPNET)

In this section, we present our initial design of CoopNet. We begin by taking a closer look at the impact of a flash crowd on server performance.

### A. Where is the bottleneck?

A key question is what the most constrained resource is during a flash crowd: CPU, disk or network bandwidth at the server, or bandwidth elsewhere in the network. It is unlikely that disk bandwidth is a bottleneck because the set of popular documents during a flash crowd tends to be small, so few requests would require the server to access the disk. For instance, the MSNBC traces from September 11 show that 141 files (0.37%) accounted for 90% of the accesses and 1086 files (2.87%) accounted for 99% of the accesses. It is quite likely that this relatively small number of files would have fit in the server's main memory buffer cache.

The CPU can be a bottleneck if the server is serving dynamically generated content. For instance, Web pages on MSNBC are by default implemented as active server pages (ASPs), which include code that is executed upon each access. (ASPs are used primarily to enable ad ro-

\*<http://www.research.microsoft.com/~padmanab/>

†<http://www.andrew.cmu.edu/~kunwadee/>. The author was an intern at Microsoft Research through much of this work.

tation and customization of Web pages based on HTTP cookie information.) So when the flash crowd hit in the morning of September 11, the CPU on the server nodes quickly became a bottleneck. For instance, the fraction of server responses with a 500 series HTTP status code (error codes such as “server busy”) was 49.4%. However, MSNBC quickly switched to serving static HTML and the percentage of error status codes dropped to 6.7%. Our conversations with the Web site operators have revealed that network bandwidth became the primary constraint at this stage.

Since Web sites typically turn off features such as customization during a flash crowd and only serve static files, it is not surprising that network bandwidth rather than server CPU is the bottleneck. A modern PC can pump out hundreds of megabits of data per second (if not more) over the network. For instance, [4] reports that a single 450 MHz Pentium II Xeon-based system<sup>1</sup> with a highly tuned Web server implementation could sustain a network throughput of well over 1 Gbps when serving static files 32 KB in size.

On the other hand, the network bandwidth of a Web site is typically much lower. In an experiment conducted recently [12], the bottleneck bandwidth between the University of Washington (UW) and a set of 13,656 Web servers drawn from [21] was estimated using the Nettimer tool [7]. The bottleneck bandwidth (server to UW) was less than 1.5 Mbps (T1 speed) for 65% of the servers and less than 10 Mbps for 90% of the servers<sup>2</sup>. So it is clear that in the vast majority of cases network bandwidth will be the constraint during a flash crowd, not server CPU resources.

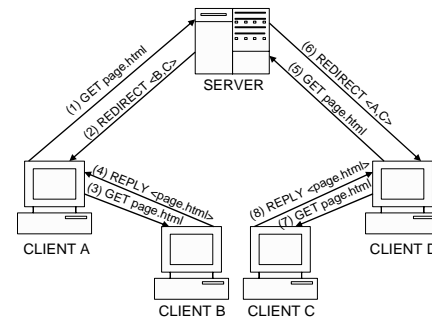
While it is possible that there may be bottleneck links at multiple locations in the network, it is likely that the links close to the server are worst affected by the flash crowd. So our focus is on alleviating the bandwidth bottleneck at the server.

### B. Basic Operation of CoopNet

As mentioned in Section I, the basic idea in CoopNet is to have clients serve content to others clients, thereby alleviating load on the server. Since network bandwidth tends to be the bottleneck rather than server CPU, CoopNet is tailored to drastically reducing bandwidth demands at the server. HTTP requests from clients arrive at the server as usual. During a flash crowd, the server redirects some or all of the requesting clients (depending on how constrained the server’s network bandwidth is) to others clients that have downloaded the URL in the recent past. The clients then resend the request to one or more of these peers. Figure 1 illustrates the operation of CoopNet.

<sup>1</sup>The system had 4 processors, but only one CPU was used for the experiments reported in [4].

<sup>2</sup>Given the good network connectivity of UW, is likely that the bottleneck link in most cases was close to the server. While the bottleneck could have been “in the middle” for some distant servers (e.g., servers overseas), it is still likely to constrain communication between



**Fig. 1. The basic operation of CoopNet. The numbers in parentheses indicate the ordering of the steps. Note that the list of peers returned by the server is updated as new requests arrive.**

Clients indicate their willingness to participate in CoopNet by including a new HTTP pragma field in the request header. We call these the “CoopNet clients” and the rest as the “non-CoopNet clients”. The server remembers the IP addresses of CoopNet clients that have requested each file in the recent past. For each file, it may be sufficient for the server to remember a relatively small number — say a few tens — of client addresses. The server then picks between 5 and 50 addresses at random from this set and includes this in the redirection message. It is quite likely that at least one of these peers is able and willing to serve the requested file. Since the server’s list of addresses is constantly being updated as new requests arrive, the redirection procedure would tend to spread load rather evenly across the set of CoopNet clients.

The redirection response, which is a generalization of HTTP redirection, is quite small in size — 200-300 bytes including all protocol headers and the list of peer IP addresses. In contrast, even the slimmed down version of the MSNBC front page during the flash crowd of September 11 was 18-22 KB in size. Thus request redirection saves the server nearly two orders of magnitude in bandwidth. This alone may often be sufficient to help the server tide over the flash crowd problem. Furthermore, server-based redirection often enables a client to locate the desired content within two hops<sup>3</sup> — one to the server and another to a peer. In contrast, a distributed lookup scheme like Chord [15] or Pastry [11] has a lookup cost of  $O(\log(N))$  hops, where  $N$  is the number of nodes in the system. Thus server-based redirection is advantageous in many cases. In some situations, however, it may be desirable to avoid server-based redirection, as we discuss in Section II-E.

We have built a prototype implementation of CoopNet. The server piece is implemented as an extension to the Microsoft IIS server using the ISAPI interface. The client piece is implemented as a client-side proxy that serves requests both from the local browser and from peers.

the server and the large number of clients in the U.S.

<sup>3</sup>We mean end-to-end hops between hosts, not network hops.

### C. Peer Selection

An important question is how a client, upon receiving a redirection message from the server, decides which peer(s) to download a file from. Clearly, it would be desirable to find nearby peers that are well-connected without resorting to expensive network measurements. We employ a multi-pronged approach to the peer selection problem:

1. We use the scheme proposed in [6] to find peers that are topologically close to the client that issued a request. The basic idea is to use address prefix information derived from BGP routing tables. Two peers are deemed to be topologically close if their IP addresses share a common address prefix. The server uses this algorithm to find topologically close peers to include in its redirection response. There exist ways of doing such prefix matching operations very efficiently without imposing much of a burden on the server (e.g., [16]). If it is unable to find any peers with a matching prefix, the server just responds with a random list of peers. However, as we discuss in Section III-C, the September 11 traces suggest that the server may often be able to find topologically close peers.

2. A match in address prefix does not necessarily mean that two peers are close to one another. For instance, an address prefix may correspond to a large network such as a national or global ISP. Therefore, it may be desirable to have the peers do a quick check to confirm their proximity. Our approach is to have each peer determine its “coordinates” by measuring the network delay to a small number (say 10) of “landmark” hosts. The intuition is that peers that are close to each other would tend to have similar delay coordinates. Similar approaches have been used in a number of contexts recently: network delay estimation [8], geographic location estimation [9], overlay construction [10], and finding nearby hosts [5].

3. For large file transfers, network bandwidth may be a critical metric for peer selection. The last-mile link is often the bottleneck. As in Napster, our approach is to have clients report their bandwidth (suitably quantized — e.g., dialup modem, DSL, T1, etc.) to the server as part of the requests they send. (Clients estimate their last-mile bandwidth by passively monitoring their network traffic in normal course.) The key distinction compared to the Napster approach is that in its redirection messages the server tries to only include peers whose reported bandwidth matches that of the requesting client. The motivation for this is two-fold. First, low-bandwidth clients are anyway constrained by their thin pipes, so they may not gain much from connecting to high-bandwidth peers. Second, clients do not have an incentive to under-report their bandwidth (a problem that afflicts Napster) because that would lead the server to redirect them to peers with a similar low bandwidth.

4. Even after applying the preceding steps, a client may still have a choice of say 2-3 peers to pick from. In such a case, the client could request non-overlapping pieces of data from multiple peers (say using the HTTP byte-range option [3]), determine which connection is the fastest, and

then pick the corresponding peer for the remainder of the data transfer. Clearly, this procedure is likely to be worthwhile only in the case of large file transfers. In Section II-D, we discuss the case of streaming media files where it may be desirable to persist with multiple peers for the entire duration of data transfer.

### D. Streaming Media Content

Streaming media content presents some interesting issues in the context of a flash crowd. First, due to the large size of streaming media files and the relatively high bandwidth needed for streaming, even a small flash crowd can easily overwhelm the server or its network. For instance, a server behind a T1 link would be able to pump out no more than a dozen 128 Kbps streams simultaneously. Second, unlike static Web content, streaming media content is not normally cached at clients. Third, the burden of serving an entire stream to another client may be too much for a client, which is after all not engineered to be a server.

Our approach is to have clients save a local copy of streams during a flash crowd so that it can be streamed to other clients if needed. Where possible, a group of peers transmits non-overlapping portions of a stream (i.e., a set of “sub-streams”) to the requesting client. The client combines these sub-streams on the fly to reconstruct the original stream. Distributed streaming reduces the burden on individual peers and also provides robustness in the face of congestion or packet loss suffered by a subset of the sub-streams.

### E. Avoiding Server-based Redirection

In some cases, it may be desirable to avoid having all requests be redirected by the server. First, in an extreme case, the bandwidth and/or processing needed to send the redirection messages may itself overwhelm the server. Second, it may be that only a small fraction of all clients are willing to participate in CoopNet. So cooperation among the CoopNet clients may not help reduce server load noticeably during the flash crowd. While CoopNet clients may still benefit significantly from their mutual cooperation (since they can download most of the bytes from one other instead of from the congested server), even getting the (small) initial redirection message from the congested server may take a long time (because of packet loss and the resulting TCP timeouts). So the total latency for CoopNet clients may remain large.

For these reasons, it may be desirable for CoopNet clients to check with their peers first before turning to the server. How to do this checking efficiently is an interesting and open research question. We present our initial thoughts here. We term the set of peers among which a client searches for content as its *peer group*. (The peer group could, in principle, include all CoopNet clients.) On the face of it, the problem of searching for content in the peer group is similar to recent work on distributed key searching (e.g., CAN [10], Chord [15], Pastry [11], Tapestry [17]).

However, we believe that these schemes may be too heavy-weight for the flash crowd problem because (a) individual clients may not participate in the peer-to-peer network for very long, necessitating constant updates of the distributed data structures, (b) as we show in Section III-A, much of the benefit of cooperation can be obtained even if the peer group size for each client is relatively small (say 30-50 peers), so there is not really the need for a distributed search mechanism that scales to millions of peers, and (c) the search for content in the peer group need not always be successful since there is always the fallback option of going back to the server.

Our approach exploits the observation that the peer group size for each client is relatively small. It may well be feasible for each member of a peer group to know about all other members. For each URL, there would be a designated “root” node within each peer group that would keep track of all copies of the file within the peer group. The assignment of the root node for a URL can be made using a hash function so that any member of the peer group can locate content in just two steps: first finding the root node by hashing on the URL and then finding a node that has the desired content. Redirection via the server can be used both to discover other clients and form a peer group initially, and also as a fallback option in the event that the desired content is not found within the peer group.

#### F. Security Issues

There are two security-related issues to consider: ensuring the integrity of content and ensuring the privacy of peers (i.e., not revealing to a client’s peers what content it has accessed).

The integrity of the server’s content can easily be ensured by having the server digitally sign the content. A client can obtain the signature either directly from the server (as part of the redirection message) or from a peer. The client can then verify the authenticity of the content it receives from its peers. For the sake of computational efficiency, the server could sign a 160-bit SHA-1 hash of the content rather than the content itself. In any case, since the signature need only be computed once for each version of a file, the burden placed on the server is minimal.

Ensuring privacy is much harder. While there exist proposals for enabling anonymous communication between hosts (e.g., [14]), anonymity comes at the cost of performance. This trade-off may not be appropriate in a flash crowd situation since performance is the key issue. In fact, clients may not care about privacy during a flash crowd because the content served during such times is, in any case, likely to be of widespread interest.

### III. EXPERIMENTAL EVALUATION

In this section, we evaluate the feasibility and potential performance of end-host cooperation during a flash crowd. The goals of the evaluation are to answer the following questions:

- How often can a client retrieve content from its peer group and avoid accessing the server?
- How much additional load do peers incur by participating in CoopNet?
- How often can cooperating peers be found nearby?
- What is the duration of time for which peers are active?

The cooperation protocol used in our simulations is based on the one described in Sections II-B and II-E. A client who is willing to cooperate initially contacts the server to get IP addresses of other CoopNet clients. The server maintains a fixed size list of the CoopNet clients’ IP addresses, and includes the most recent  $n$  clients in its redirection message. In our simulations,  $n$  ranges from 5 to 50 clients. Once the client has a list, it always contacts peers on the list to ask for content. If content cannot be found at these peers, the client returns to the server to request the full content and an updated peer list.

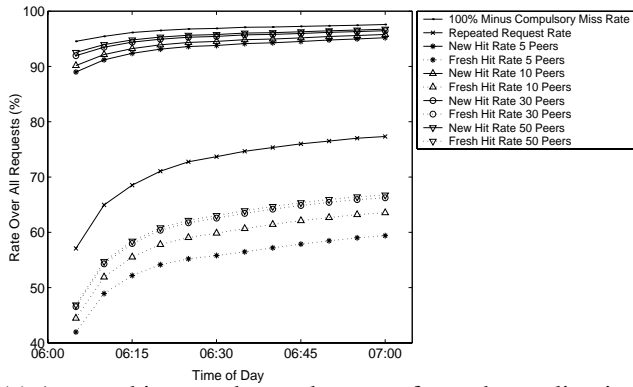
We use traces collected at the MSNBC website during the flash crowd of September 11, 2001 for our analysis. The flash crowd started at around 6am PDT, and lasted for the rest of the day. The peak load was ten times the typical load. Due to computing limitations, we focus our analysis on the first hour of the flash crowd, between 6:00 am to 7:00 am PDT, containing over 40 million requests.

#### A. Finding Content

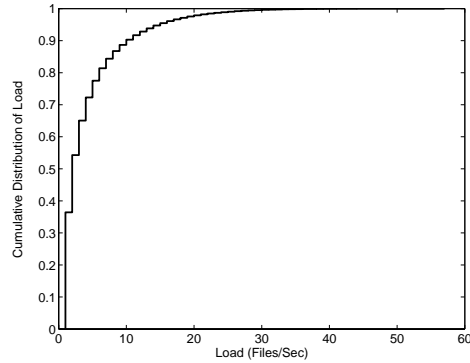
In order for cooperation to be effective, clients need to avoid retrieving content from the loaded server to the extent possible. We define two metrics that capture how often content can be retrieved from one’s peer group. The first metric is new content hit rate, which is the fraction of requests for new files that can be served by hosts in the peer group. The second metric is fresher content hit rate, which is the fraction of time that a fresher copy of a file can be found within the peer group. Fresher content hit rate only applies to the case when clients are looking for updated versions of files that they had downloaded in the past. If these two hit rates are high, that would indicate that CoopNet is providing an effective mechanism for improving client performance.

Figure 2(a) depicts the hit rates observed when the number of CoopNet clients is 200 (i.e., only 200 of the many hundreds of thousands of clients are willing to cooperate). The peer list returned by the server, which determines the peer group used by a client, is drawn from this set of 200 CoopNet clients. The peer list size ranges from 5 to 50 clients. The vertical axis in Figure 2 is the observed rate, and the horizontal axis is observation times at every 5 minutes after the beginning of the trace at 6:00am. Each line represents the rate observed for a particular peer list size.

We present two analyses — optimistic and pessimistic. In the optimistic analysis, we assume that files are not modified between accesses. So an access is either a repeated request (i.e., a request for a URL that a client has previously accessed) or a request for a new (i.e., previously unseen) URL. The solid line in the middle of Figure 2(a) is the rate



(a) Average hit rates observed at peers for each peer list size.



(b) Load at peers during busy periods.

**Fig. 2. Hit rates and load observed at peers.**

of repeated requests. The solid lines at the top show the sum of the repeated request rate and the hit rate for new content. This sum represents the overall hit rate in the optimistic setting. The upper bound for the overall hit rate is the difference between 100% and the compulsory miss rate (which corresponds to the case when content must be retrieved directly from the server because none of the 200 CoopNet clients has a copy of that content). This upper bound is the line at the top of the figure. We observe that for all peer list sizes, the overall hit rate is close to the upper bound, with less than 5% of requests ending up in a miss. We also observe that hit rates increase with time because of cache warming effects similar to those reported for Web proxies.

In the pessimistic analysis, we assume that a file is updated each second it is retrieved from the server. So in the case of a repeated request, a client would actually look for a fresher copy of the content than it has. The rate for finding fresher content from cooperating peers is represented by the dotted lines in Figure 2(a). Clearly, the upper bound for finding fresher content is the repeated request rate. After 5 minutes of cooperation, peers find fresher content 46% of the time out of the maximum achievable 56%, using a peer list size of 30. After an hour of cooperation, peers find fresher content 65% of the time out of the maximum achievable 77%, using a list size of 30. Increasing the list size from 30 to 50 peers does not significantly improve hit rates.

In summary, we find that cooperation among a small group of peers is effective. Clients need to retrieve content from the server only 15% of the time when using a peer list size of 30.

### B. Load on Peers

CoopNet clients contribute resources, such as network bandwidth, to the system. To maintain good performance, it is important not to completely exhaust those resources. Here we examine the network bandwidth overhead incurred by clients serving content.

Over 80% of the time, peers are idle and do not serve content. Figure 2(b) depicts the cumulative distribution of

load, measured as the rate at which peers serve files, during the remaining 20% of time for a peer list size of 30. This distribution is representative of the load observed across all simulations of different peer list sizes. For the most part, peers can sustain the bandwidth requirement for serving content. Over half of the time during busy periods, peers serve at most 2 files in a second. However, in a few cases, load may be unevenly distributed, leading to a flash crowd at peers. The load can be as high as 57 files/second. Although the load is much less than that observed at the server, it may be enough to cause an overload at peers. We are presently investigating load distribution and peak bandwidth requirement for peers.

### C. Finding Nearby Peers

Finding nearby peers can greatly improve the efficiency of peer-to-peer interaction. For example, a peer at CMU can retrieve content more quickly from another peer at CMU than it can from a peer in Europe. In some cases, the peer-to-peer performance could be comparable or better than client-server performance.

We use the following metric to determine network proximity. Peers that are in the same BGP prefix cluster are considered to be “close” to each other. Although this metric does not express closeness of peers that are in different BGP prefix clusters, it provides an approximation to whether or not it is possible to find a nearby peer.

We look at the IP addresses of clients in the trace in the initial 30-minute period. There were 563,284 unique clients, and 69,778 unique BGP prefix clusters. The probability of there being another client in the same prefix cluster during the first 2 minutes of the trace is 80%. The probability grows to 90% for the entire 30-minute period. Therefore, it is likely that peers will cooperate with nearby peers.

### D. Duration of Activity Period for Peers

The duration of time for which peers are active affects how well CoopNet performs. If peers are active at a website for very short periods of time, peer lists must also be updated frequently.

To determine the period of activity, we consider the in-

terarrival time between requests in the initial 30 minutes of the trace<sup>4</sup>. We treat an interarrival period that is longer than a threshold as representing the end of an activity period (and the start of the next). We consider two values of the threshold — 1 minute and 5 minutes. We find that the average activity period is 1.5 minutes and 4.5 minutes, respectively, in the two cases. This indicates that peer lists may become stale on the order of a few minutes and should be updated frequently.

#### IV. COMPARISON WITH ALTERNATIVE APPROACHES

We now discuss two alternative approaches to solving the flash crowd problem: proxy caching and infrastructure-based content distribution networks (CDNs). An advantage that both of these approaches have over CoopNet is that they can be deployed transparently to clients.

Proxy caching can help alleviate server load during a flash crowd by filtering out repeated requests from groups of clients that share a proxy cache. However, the effectiveness of proxy caching is limited for a few reasons. First, for them to be really effective in the context of a flash crowd, proxy caches need to be deployed widely. Since this requires substantial infrastructural investments by a large number of organizations, a widespread deployment of proxy caches is only likely if it results in significant performance improvement during “normal” (i.e., non-flash crowd) times as well. However, cache hit rates have remained quite low, and the growing share of dynamic and customized content will only make matters worse.

A second issue is that even a universal deployment of proxy caches may not be sufficient to alleviate a flash crowd in certain situations. For instance, the small Web site for a high school alumni association may be overwhelmed by the flash crowd caused when a link to the video clip of a recent football game is sent out to all members via email. The clients interested in this content are likely to be dispersed across the Internet, so proxy caches at the local or organizational level may not filter out much of the load.

An alternative approach is to depend on an infrastructure-based CDNs (e.g., Akamai [19]) to distribute content. This may be an effective approach for ensuring high availability and good performance both during a flash crowd and in normal times. However, it is unlikely that a small Web site would be in a position to afford the services of a commercial CDN. Moreover, the absolute volume of traffic at such a site even during a flash crowd may not be large enough to be of interest to a commercial CDN.

In summary, we believe that CoopNet offers advantages compared to both proxy caching and infrastructure-based CDNs. CoopNet offers a low-cost but effective solution to the flash crowd problem, which is likely to be especially attractive to small Web sites with limited resources. That said, we do *not* view CoopNet as a replacement for infrastructure-based CDNs. As noted on Section I, CoopNet’s peer-to-peer content distribution kicks in when

needed during a flash crowd but lies dormant during normal times. In contrast, an infrastructure-based CDN is engineered to provide a wide range of services (e.g., hit metering, high availability, performance guarantees, etc.) during all times. Thus we believe that there is a role for both CoopNet and infrastructure-based solutions.

#### ACKNOWLEDGEMENTS

We would like to thank Stefan Saroiu for making his Web server bandwidth measurements available to us. We would also like to thank Lili Qiu for early discussions on CoopNet and the anonymous reviewers for their insightful comments.

#### REFERENCES

- [1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek and R. Morris. “Resilient Overlay Networks”, *ACM SOSP*, October 2001.
- [2] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. “Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs”, *ACM SIGMETRICS*, June 2000.
- [3] R. Fielding *et al.* “Hypertext Transfer Protocol – HTTP/1.1”, *RFC-2616, IETF*, June 1999.
- [4] P. Joubert, R. King, R. Neves, M. Russinovich, and J. Tracey. “High-Performance Memory-Based Web Servers: Kernel and User-Space Performance”, *Usenix 2001*, June 2001.
- [5] C. Kommareddy, N. Shankar, and B. Bhattacharjee. “Finding Close Friends on the Internet”, *IEEE ICNP*, November 2001.
- [6] B. Krishnamurthy and J. Wang. “On Network-Aware Clustering of Web Clients”, *ACM SIGCOMM*, August 2001.
- [7] K. Lai and M. Baker. “Nettimer: A Tool for Measuring Bottleneck Link Bandwidth”, *USENIX Symposium on Internet Technologies and Systems*, March 2001.
- [8] T. S. E. Ng and H. Zhang. “Towards Global Network Positioning”, *ACM SIGCOMM Internet Measurement Workshop*, November 2001.
- [9] V. N. Padmanabhan and L. Subramanian. “An Investigation of Geographic Mapping Techniques for Internet Hosts”, *ACM SIGCOMM*, August 2001.
- [10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. “A Scalable Content-Addressable Network”, *ACM SIGCOMM*, August 2001.
- [11] A. Rowstron and P. Druschel. “Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility”, *ACM SOSP*, October 2001.
- [12] S. Saroiu. “Bottleneck Bandwidths”, October 2001. <http://www.cs.washington.edu/homes/tzoompy/sprobe/webb.htm>
- [13] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. “The End-to-End Effects of Internet Path Selection”, *ACM SIGCOMM*, August 1999.
- [14] C. Shields and B. N. Levine. “A Protocol for Anonymous Communication Over the Internet”, *ACM Conference on Computer and Communication Security*, November 2000.
- [15] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. “Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications”, *ACM SIGCOMM*, August 2001.
- [16] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. “Scalable High Speed IP Routing Lookups”, *ACM SIGCOMM*, September 1997.
- [17] B. Zhao, J. Kubiawicz, and A. Joseph. “Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing”, *U. C. Berkeley Technical Report UCB//CSD-01-1141*, April 2000.
- [18] “Web acts as hub for info on attacks”, <http://news.cnet.com/news/0-1005-200-7129241.html?tag=rtdnws>, 11 September 2001.
- [19] Akamai. <http://www.akamai.com/>
- [20] MSNBC Web site. <http://www.msnbc.com/>
- [21] List of Web servers. <http://www.icir.org/tbit/daxlist.txt>

<sup>4</sup>Clearly, the limited length of the trace could bias our results.