HTTP(P2P): A transaction based (HTTP) peer-to-peer protocol for the dissemination of web-objects in congested networks.

Robert L.R. Mattson Somnath Ghosh
La Trobe University
rlmattson@ieee.org somnath@cs.latrobe.edu.au

Abstract

Highly popular websites can suffer classical congestion collapse because of the ability of outgoing 'bursty' data to congest the server side link. Using the well known and recognized network simulator ns we look at the effect that this congestion collapse has on user perceived performance (UPP) of a simulated website.

We developed a protocol overlay for HTTP/1.1 and a complementary distribution infrastructure to alleviate the effects of this congestion. Our Protocol HTTP(P2P), shows promise for increasing the maximum load of websites during times of high and very high website load and link contention.

In our simulated environment, HTTP(P2P) significantly decreases the time required for users to download the web page they request. The protocol also distributes the redirected traffic over the network to reduce secondary transient network congestion.

1. Introduction

The Schumacer-Levey effect [1] (more commonly known as the slashdot effect or flash crowds) has the ability to cause wide scale denial of service for specific websites. Massive spikes in demand often result in congestion that is perceived by end users as delay or a denial of their requested service. Thus, we reflect on the wide scale inaccessibility of most major news sites as an effect of the 2001 September 11 terrorist attack. Thousands of users attached to the Internet across the world requested information from a small number of common sources resulting in an excessive demand for outgoing bandwidth at websites creating an outgoing traffic bottleneck for data.

This effect is caused by the over subscription of clients motivated by demand to servers and associated serving resources. In real terms this over subscription of resources results in websites that can handle daily request loads and minor peaks in demand, though when subjected to astronomical demand these sites are not able to provide the throughput required to service such high demand. In part, this is because most Internet services are based on the client/server model wherein a client is subjugated to an entity performing a serving role.

The notion of using peer-to-peer protocols to exchange information is being actively explored. We argue that other than drastically over provisioning server resources the use of peer to peer technology it is the only viable and scalcable solution to manage and mitigate the effect of flash crowds. Peer-to-Peer networks Napster, The Gnutella protocol, CoopNet[2], FreeNet[3], Rubenstien[4] and more recently the BitTorrent[5] architecture are information systems that have facilitated the exchange of an insurmountable amount of data among autonomous peers throughout the internet.

In section II we explore the phenomenon of server side congestion and its cause, also highlighting traditional mechanisms used to alleviate the congestion. We take advantage of peer-to-peer philosophy as applied in the modern Internet and propose the introduction of additional functionality to the existing HTTP specification, version 1.1[6] in Section III. Section IV describes the positive effect of our protocol in the simulated environment. We conclude with discussion of future development on HTTP(P2P) in section V. The metrics provided in this paper are given in time saved at the desktop and bandwidth offset at the server side link.

Our simulations in this paper were initially validated with a custom made simulator written in C++. The results published in this paper were obtained and validated by using the ns network simulator version 2.1b9a (http://www.isi.edu/nsnam/ns/).

0-7803-8430-X/04/$20.00 ©2004 IEEE

2. Problem Analysis

Economists refer to the 'impact of one person's actions on the well being of a bystander' [7] as an externality. The compounded effect of link exclusion by bandwidth utilisation is what we consider as a 'negative externality' [1] i.e. My Internet traffic delays the Internet traffic of my neighbour. At times of very high demand on scarce link bandwidth the effect of *congestion collapse* described in [8] can occur.

Further, routers subjected to high demand can become overloaded with data they cannot relay and must drop incoming packets. Respective sender and receiver TCP stacks interpret this loss of data as congestion and reduce the rate of data transmission. TCPs guaranteed delivery ethos forces the retransmission of data. Retransmission can further contribute to the congestion collapse at the link and exacerbate the cumulative delay imposed on traffic using the link. By using a peer to peer methodology we can disseminate web objects with less probability of traffic convergence causing bottlenecks.

2.1. Simulated Congestion.

We simulate this congestion to prove its effect on the average load time of a web page and propose a protocol to alleviate the congestive collapse in section 3.

The simulation topology was configured as: a single serving node (origin server), principally responsible for responding to web-object requests from browsers over an individual server-side link. Central to the simulated network was a single router, connected directly to the client nodes and the server node. The router used the default ns drop tail (FIFO) packet queuing policy. Links were configured as bi-directional and full-duplex with a propagation delay of 10ms. Server link speed was 1.544Mb/sec; client links were configured at 0.0672Mb/sec.

In our simulations all clients have an exclusive link directly to the router. No node is more than two hops away from another node. This network topology is essentially analogues to the barbell topology highlighted in [9].

Browser agents were configured to create and maintain a maximum of four simultaneous connections per server [10]. Connections are pipelined and persistent for the duration of the simulation.

All browsers initiate their requests to the web server over a pre-defined period of 10 seconds simulated time to generate a flash crowd. The precise time at which the browser requests the root document (/index.html) is determined by equation 1. For a given Peer (*N*) at address (*Address*) for an artificially created flash crowd of *StormDuration* seconds, the peer (*N*) will request the index page at time *t*. Subsequent documents are requested or added to the browser's request queue as the root document is incrementally parsed as new packets arrive.

$$t = \left(\frac{Address}{N} \right) \times StormDuration \qquad - (1)$$

In reality, the requests made to a server for web objects do not arrive at regularly spaced intervals as given by equation 1. The arrival pattern generated by equation 1, however, is expected to show the effect of closely spaced arrivals correctly. Moreover, the non-random arrival pattern created by equation 1 is repeatable, and results generated from such inputs are verifiable. This way we can generate non-random, verifiable simulator results.

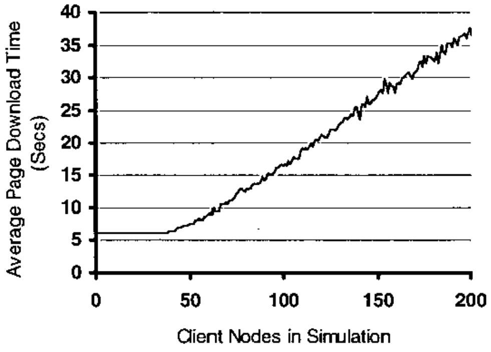

Fig. 1. The average delay experienced by nodes requesting the web page.

We developed a benchmark of the average delay experienced by browsers downloading our web page as described in Figure 1. The simulation reaches 'critical mass' when 45 nodes request the web page in our period of 10 seconds. At this point user perceived performance of the server depreciates significantly. The simulation web page semantics are described in Table 1.

TABLE 1
SIMULATION WEB PAGE STRUCTURE

Symbol	Object Size	Object mark-up offset
/index.html	22,500 Bytes	—
/1.gif	1,150 Bytes	3,500 Bytes
/2.gif	400 Bytes	2,500 Bytes
/3.gif	3,460 Bytes	1,000 Bytes
/4.gif	2,000 Bytes	1,200 Bytes
/5.png	12,000 Bytes	12,000 Bytes
/6.jpg	2,000 Bytes	15,00 Bytes

When the following elementary inequality (2) holds true for N nodes each with a bandwidth of B_{client} actively using the server-side link with a bandwidth of $B_{server,}$ server-client communication will not experience congestion. When this inequality does not hold true, traffic will experience delay. To this end, the time taken to transfer user application level data will be increased.

$$B_{client} \times N \leq B_{server} \qquad -(2)$$

2.2. Current Solutions.

Several technologies and schools of thought exist that deal with managing and mitigating the problem of excessive demand: logically and geographically distributed Server farms [11, 12] and clusters [12] whose processing power is paired with Round Robin Domain Name Services (RR-DNS). TCP routers and a combination of both RR-DNS and TCP routing [11] for load balancing.

Internet caching is such a successful technique of reducing network congestion and latency [13, 14] that it has become a billion dollar industry [15]. Development in the field of caching concentrates on cache placement [16, 17], object replacement algorithms [13] and cache co-ordination schemes [17]. As caches are in integral part of reducing web congestion a cache is usually a client to another serving cache, particularly in en-route and hierarchical caching methodologies. HTTP(P2P) will enable client caches to collaborate should a serving cache become congested.

CoopNet[2] is a peer to peer architecture proposed as a solution to flash crowds. CoopNet assumes that peers are willing to form cooperative groups for a period greater than the time required to receive and transmit a web object. This notion is inconsistent with work by Golle[18] that shows only a very small portion of people share files willingly among peers, unless there are incentives. CoopNet does not inherently advocate incentives beyond a user's altruistic willingness to serve data for an undefined period of time. Rubenstein[4] proposes an interesting use of a persistent gnutella like infrastructure where nodes freely request, transmit and retrieve information from browser-level caches in conjunction with traditional Client/Server HTTP. Some users may be discontent to carry the short fall for organisations unwilling to provide sufficient serving capacity.

Economists have tried to reduce the effect of negative bandwidth externalities through the introduction of pricing mechanisms. These mechanisms attempt to hit the inconsiderate consumer in the hip pocket as a disincentive for users that may choose to unfairly consume bandwidth.

Mechanisms that service providers can impose to moderate excessive bandwidth consumption are: flat rate pricing, usage sensitive pricing, transaction based pricing [19], differentiated QoS at the ISP-user connection [1] and router-based smart-markets [1, 20]. The monetary compensation is, however to be directed to the network provider, not the inconvenienced user. We argue that such pricing schemes create an incentive for congested networks that would be more profitable than decongested networks.

Akamai maintains a system known as 'EdgeSuite' that dynamically redirects object requests via optimised DNS lookups to a proxy server that is as geographically close to the user as possible. This system is highly efficient at diffusing web requests among caches to prevent server side bottlenecks. As a proprietary system it costs money to use and will most likely be inaccessible to all but those organisations that can pay Akamai's premium.

Though, even with these mechanisms flash crowds can still prevent users from obtaining information. We need to design a tightly controlled mechanism that will not abuse a user's willingness to forward information at their expense. The HTTP(P2P) protocol is a centralised mechanism that is effective at reducing the impact of flash crowds. The protocol is as fair as possible, only mandating that users accommodate others for the use of the server side link.

3. Protocol Description.

The protocol described here is based on the expectation that when the server side Internet link is congested it should be possible to improve the load time of a web page for an end-user by redirecting requests to an available or less contested data source. When we think of time as a cost, our protocol would

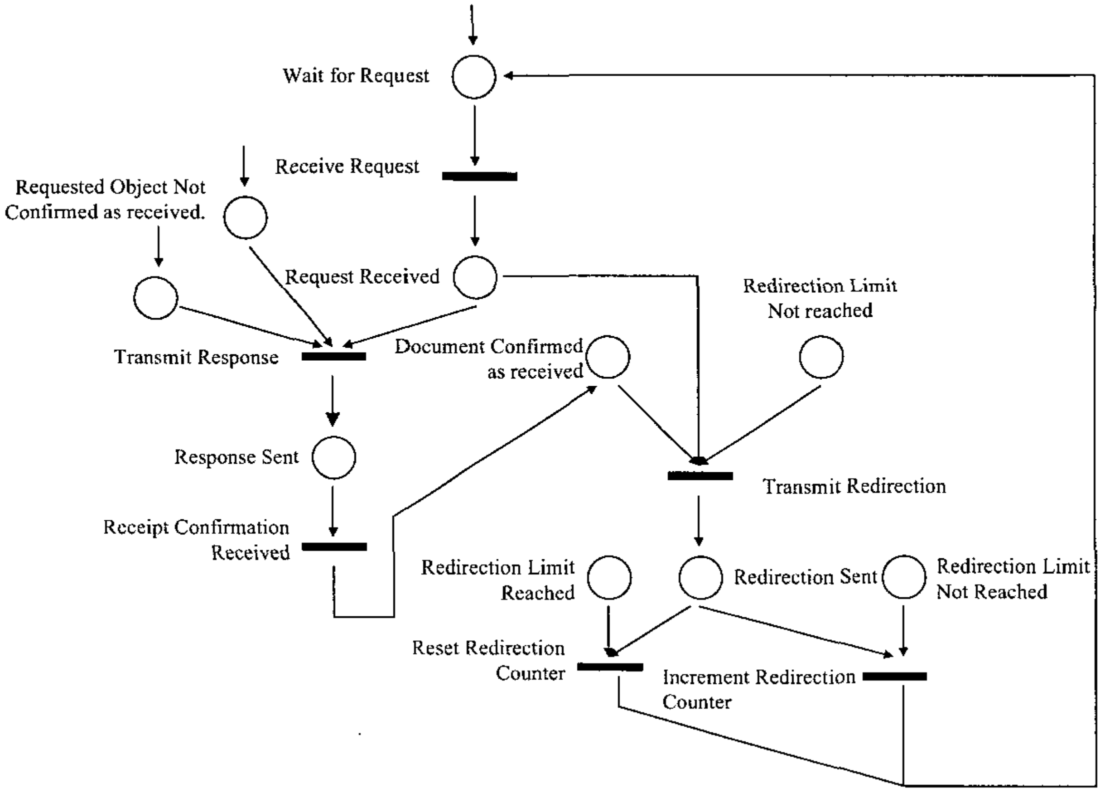
Wait for Request
Receive Request
Requested Object Not Confirmed as received.
Request Received
Transmit Response
Document Confirmed as received
Redirection Limit Not reached
Response Sent
Transmit Redirection
Receipt Confirmation Received
Redirection Limit Reached
Redirection Sent
Redirection Limit Not Reached
Reset Redirection Counter
Increment Redirection Counter

Fig. 2. Implemented origin server logic.

reduce the load time of a page when the cost of the request, redirection, re-request and download is less than the cost to request and download the page from a congested origin server.

Our protocol is a mechanism that allows web servers to distribute client requests to peers. To introduce this mechanism we need support on two levels; Networking services or infrastructure and the protocol messages that the peers in the infrastructure use to communicate. We distinguish between clients that are dependant on web servers and peers that include an implementation of HTTP(P2P).

Our protocol introduces a level of fault tolerance to the information distribution infrastructure that may already be in place. Peer clients would be geographically and logically sparse and the probability of peer-to-peer traffic converging to cause transient network overload is almost eliminated.

This protocol has economic and social justification for its use during flash crowds. As a user I can directly compensate the collective user group of a website for the delay my traffic causes by incurring the cost of disseminating the information that I retrieved to my peers. Peer capacity will increase the potential website throughput available by the cumulative outgoing bandwidth of all users of that site. Such an increase in capacity would be absolutely necessary when disseminating emergency information; the user group of a website would potentially be able to service its own demand.

3.1. Infrastructure.

Peers need a basic implementation of a web server that is capable of responding only to rudimentary HTTP 'GET' requests. These peer side servers would need to be able to interpret the most commonly used browser cache databases (Mozilla, Opera, Netscape and IE). For our simulation we implemented a common object database for which each serving object maintained an independent instance.

We derived the expected process flow of an origin server and implemented states that are required to accommodate for the HTTP(P2P) protocol. Our web server logic is concisely described in the Petri-net, Figure 2. For our serving process an origin server cannot be content with just distributing objects, we introduce a variable to track the peer that has confirmed receipt of objects and a variable that counts the number of times a request has been redirected to that client. These variables determine the behavior of the redirection machine and thus the output of the origin server. We exclude details not directly involved in the implementation of our protocol.

We consider the scenario where a set of requests { a , b , c , d } arrive for atomic web object 'A' at our origin-server. These requests are received and placed in the HTTP listen queue to be processed serially as threads become available[21].

Our state machine (Fig 2) exists for each atomic web object. When the first request 'a' arrives for object 'A' the state machine is initialised and the object is sent to the requesting peer. Upon receiving confirmation of receipt from the requesting peer the state machine is now ready to transmit a redirection to the next request 'b' and the redirection counter is incremented. Once a redirection has been sent the redirection counter is queried, based on a predefined limit 'L' the next query may be fulfilled or subjected to a redirection.

For the case where L = 1, once request 'a' is fulfilled request 'b' is redirected to obtain the cached object from the peer holding a replication of the object 'A'. This removes b's request from the request queue without the expense of dealing with object transmission. Likewise request 'd' is redirected to the peer that made request 'c'. Where L = 2, requests b and c are redirected to the host that made request a. d is fulfilled normally with copy a of the web object. Likewise when n = 3 requests b, c and d obtain the object from the host that made request a.

Our server controlled redirection component prevents the redirection of peers to expired content. Should the situation occur where an object is refreshed or replaced, we reset the state machine and send a new copy of the object. We thus negate the possibility of distributing expired content.

3.2. HTTP Messages.

Redirect-Willing: The purpose of this header is to inform the web server that the client making the object request is willing to distribute the object on the servers behalf. Possible data values are { 0, 1 }. Omission of this header may indicate client non-compliance, or an unwillingness to participate. In times of congestion the web server may simply refuse to service the request because of its contribution to existing congestion. In future, this header may indicate the number of times that the peer is willing to serve the object.

Redirection: Analogous to the 'Location:' header ([6] sec 14.30). We use a different header to differentiate between legitimate use of the location header and redirection with respect to the HTTP(P2P) protocol. The value of this header indicates the location of the object in a peer cache. Its type is URL.

Received <URL> HTTP/1.1: This message header is used by a peer to inform the origin server the object specified in the URL has been successfully received. This is followed by the mandatory 'Host' [6] header and any other optional fields as needed. We could rely on incoming TCP level ACK packets to confirm object receipt. However, these packets can be subject to delayed acknowledgement and using them will blur the layered protocol model. We prefer to rely on explicit application level acknowledgement.

SHA1: This data segment of this header is a 160-bit Secure Hash Algorithm (SHA) sign of the object to be retrieved from the peer. This can effectively guarantee the validity of the object for a few seconds to a few hours.

4. Protocol Performance.

We conducted a simulation experiment identical to the one used to establish our congestion benchmark, this time we attached HTTP agents to each peer and activated our protocol on the origin server.

For a simulation of 200 nodes in our network we can disseminate the web page to all nodes in the network in 30.4 seconds when using the HTTP(P2P) protocol. This compares to 37.67 seconds without the protocol, an improvement of 7.27 seconds. The worst case for page download with HTTP is 68.05 seconds compared to 50.46 seconds with our protocol; we reduced the time required by 17.59 seconds. Our protocol reduces the minimum time required to download the page from 12.26 seconds to 11.97. All nodes in the simulation had confirmed receiving the web page 56.51 seconds after the beginning of the request storm with our protocol, an improvement of 13.09 seconds from 69.6 seconds.

TABLE 2
REDIRECTION CAPACITY FOR 200 PEERS IN 10 SECONDS

FileName	% of requests redirected	Saved Bandwidth
/1.gif	36.5	83 950 Bytes
/2.gif	37	29 600 Bytes
/3.gif	37.5	259 500 Bytes
/4.gif	37.5	150 000 Bytes
/5.png	30	720 000 Bytes
/6.jpg	37.5	150 000 Bytes
/index.html	9	405 000 Bytes
	Total Saved Bandwidth	1 798 050 Bytes

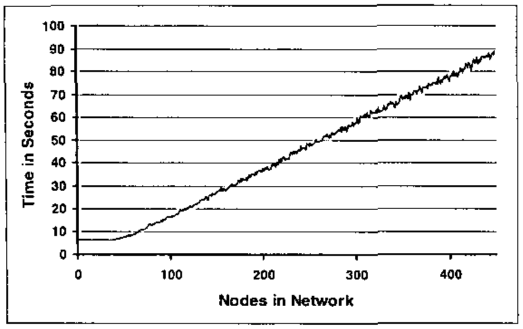

Fig. 3. Benchmark of average download time.

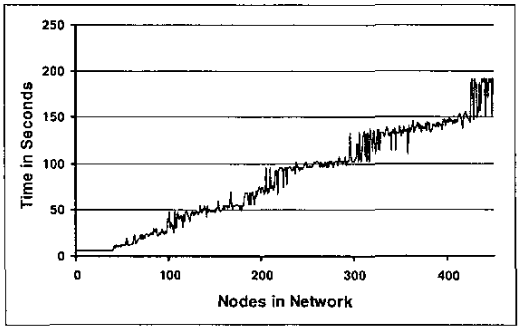

Fig. 4. Benchmark for Maximum download time.

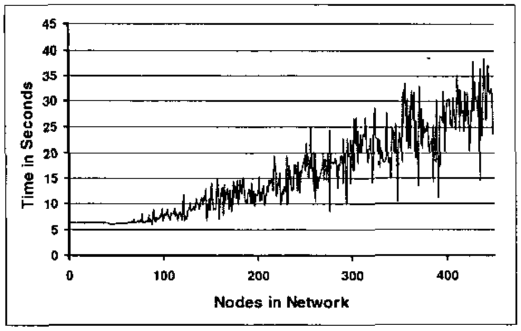

Fig. 5. Benchmark minimum download time.

As expected, less bandwidth was consumed at the server side link as the protocol offset the transmission of web-objects. The capability of the protocol to redirect requests on a file by basis is shown in Table 2. The protocol offset the transmission of 1.798 megabytes of information from the server side link to peers, not including TCP and IP overhead. Bearing in mind that the total transmission link capacity for the duration of the simulation (56.51 seconds) at the physical level is 3.474 M/Bytes, we effectively increased that to 5.272 M/Bytes; approximating to a 16% increase in serving throughput.

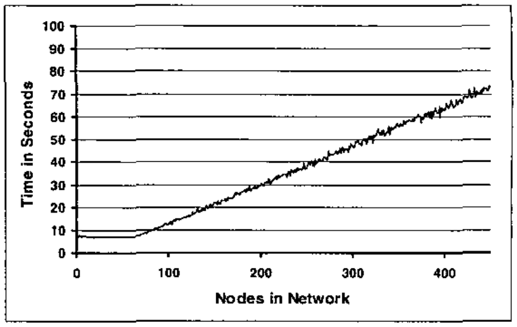

Fig. 6. Average download time with HTTP(P2P) implemented.

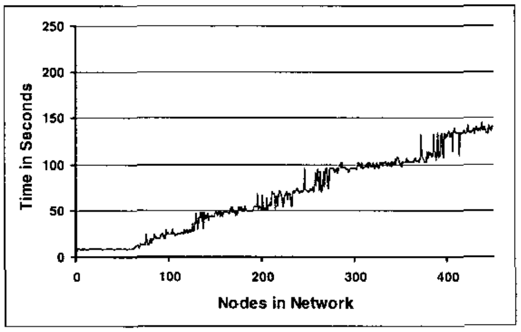

Fig. 7. Maximum download time with HTTP(P2P) implemented.

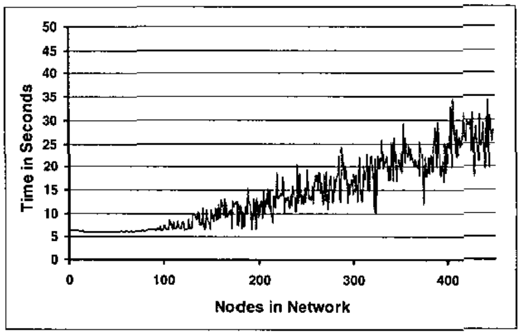

Fig. 8. Minimum download time with HTTP(P2P) implemented.

The benefits of HTTP(P2P) become more significant as the number of nodes are increased. We increased the number of nodes in the network from a moderate to high load and the benefit of the HTTP(P2P) becomes more significant. For 400 peers surging the origin server over 10 seconds, the average time decreases from 75.5 to 62.75, worst case from 147.51 to 129.425 and best case decreased from 28.79 to 20.77 seconds. Total time to service all nodes was 130.125, down from 151.2 seconds.

There is a very high degree of variation when we examine the minimum time required to retrieve the web page (Fig. 5 & 8). This is most likely due to the random nature of packet loss. A precise cause has not been determined. It is however likely that this is the case because introduction of our protocol tends to reduce the variance of times recorded as we avoid the server side link where possible.

We can obtain better performance of the protocol under a higher network load by increasing the maximum number of times that a peer serves information to other peers (varying the value of 'L'). When we subject peers to a maximum of two and three redirections the initial overhead of the protocol increases because the comparative benefit of redirecting requests is not established. However, when the server does become congested it is apparent that the more requests we redirect, the faster peers complete download.

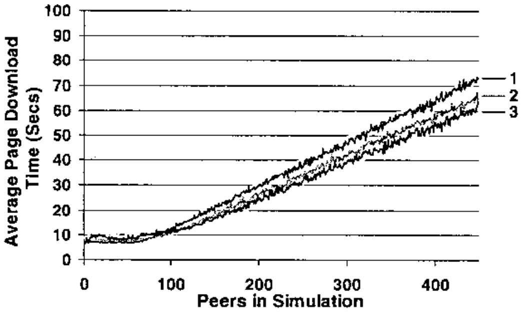

Fig. 9. Average page download times for 1, 2 and 3 redirections per download confirmation.

Using the inequality (2) we can now propose the use of an intelligent mechanism to alternate between the most efficient implementation of the possible protocols available. Thus we would have optimum throughput for a any given load.

5. Future Work.

Our protocol introduces a significant increase (16% in our simulated study) of website throughput over the flash crowd duration. We intend to introduce further mechanisms that will be able to increase the benefit of using the protocol.

We will establish the protocol as a valid solution over a variety of topologies. The question of optimum peer selection then becomes apparent, it would be ideal to select a peer based on logical proximity (i.e. the closest node based on common routing metrics RTT, bandwidth, latency etc...) rather than geographical co-location. Adapting methodology similar to the established use of Border Gateway Routing protocol (BGP) tables to optimise peer selection and subsequent redirection[2] will also help. We believe that using round trip time (via half-open TCP handshake, not ICMP) and hop-count metrics may provide a more optimum peer selection scheme. We would expect this to reduce the average and minimum time required to download the web page.

Currently we use a sliding window style mechanism for tracking the location of replicated objects. This information is discarded quite quickly. It would be optimum to retain peer address information for a slightly longer period of time to further dissipate redirections and object transfer traffic over the network. This would most likely result in lower average download time, though it may increase the probability of making a redirection to a dead peer and possibly over obligate peers to transmit objects.

We intend to investigate the feasibility of implementing a tighter security layer. This layer would provide greater assurances the object being retrieved by redirection is the same as what would have been retrieved from the origin server. Such measures would introduce overhead that may slightly increase the average download time for the object.

To increase the download speed of objects from peers it may be possible to adapt a BitTorrent[5] like approach. Peers would be given the addresses of multiple nodes containing replicated versions of the current web object. A peer would then have the ability to download the object from multiple locations.

6. References.

[1] T. Henderson, J. Crowcroft, and S. Bhatti, "Congestion pricing. Paying your way in communication networks," *IEEE Internet Computing*, vol. 5, pp. 85-9, 2001.

[2] V. Padmanabhan, N. and K. Sripanidkulchai, "The Case for Cooperative Networking," presented at Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA, 2002.

[3] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, "Protecting free expression online with Freenet," *IEEE Internet Computing*, vol. 6, pp. 40-9, 2002.

[4] D. Rubenstein and S. Sahu, "An Analysis of a Simple P2P Protocol for Flash Crowd Document Retrieval," in *Technical report*. New York: Columbia University, 2001, pp. 1-20.

[5] B. Cohen, "Incentives Build Robustness in BitTorrent," 2003.

[6] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*: Internet Engineering Task Force, 1999.

[7] G. Gans, S. King, and N. Mankiw, *Principles of Microeconomics*. Marrickville, NSW: Harcourt Brace, 1999.

[8] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking, vol.7, no.4*, pp. 458-72, 1999.

[9] K. Anagnostakis, M. Greenwald, and R. Ryger, "On the Sensitivity of Network Simulation to Topology," *Proceedings of the 10th IEEE/ACM Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems.*, 2002.

[10] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of HTTP/1.1, CSS1, and PNG," presented at ACM. Computer Communication Review, vol.27, no.4, Oct. 1997, pp.155-66. USA., 1997.

[11] A. Iyengar, J. Challenger, D. Dias, and P. Dantzig, "High performance Web site design techniques," *IEEE Internet Computing*, vol. 4, pp. 17-26, 2000.

[12] R. C. Burns, R. M. Rees, and D. D. E. Long, "Efficient data distribution in a Web server farm," *IEEE Internet Computing*, vol. 5, pp. 56-65, 2001.

[13] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 158-70, 2000.

[14] P. Danzig, "NetCache architecture and deployment," presented at 3rd International Caching Workshop, Manchester, UK. 15-17 June, 1998.

[15] C. Kenyon, "The evolution of Web-caching markets," *Computer*, vol. 34, pp. 128-30, 2001.

[16] T. Xueyan and S. Chanson, "Coordinated En-Route Web Caching," *IEEE Transactions on Computers.*, vol. 51, pp. 595-607, 2002.

[17] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of Web caching architectures: hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 404-18, 2001.

[18] P. Golle, K. Leyton-Brown, I. Ilya Mironov, and M. Lillibridge, "Incentives for Sharing in Peer-to-Peer Networks," presented at Third ACM Conference on Electronic Commerce (EC '01), Tampa, FL, USA, 2001.

[19] L. W. McKnight and J. P. Bailey, "An Introduction to Internet Economics," in *Internet Economics*, L. W. McKnight and J. P. Bailey, Eds. USA: Massachusetts Institute of Technology, 1997, pp. 525.

[20] J. MacKie-Mason and H. Varian, "Pricing the Internet," in *Public Access to the Internet*, Kahin. B. Keller. J, Ed. Cambridge, Massachusetts: MIT Press, 1995, pp. 269-314.

[21] R. D. Van der Mei, R. Hariharan, and P. K. Reeser, "Web server performance modeling," *Telecommunication Systems: Modeling, Analysis, Design & Management*, vol. 16, pp. 361-78, 2001.