Caching Support for Push-Pull Data Dissemination using Data-Snooping Routers

Ismail Ari
ari@cs.ucsc.edu

Ethan L. Miller
elm@cs.ucsc.edu

Storage Systems Research Center *
University of California Santa Cruz

Abstract

Internet applications such as the HTTP-Web, audio-video streaming and file sharing depend on wide-area data dissemination. Clients of these applications suffer from long delays due to network queuing, bandwidth limitations and adverse effects of bandwidth sharing between different traffic.

Caching reduces delays and saves network bandwidth by holding the fetched data and responding to the subsequent requests locally. Existing distributed caching solutions are application-specific and do not support delivery in the push-pull directions at the same time. Our proposed architecture, called *Storage Embedded Networks*, gives application- and direction-independent caching support by using memory-embedded, *data-snooping* routers. These router caches can act both as a client proxy and a server accelerator. We compare our architecture to the web caches operating in forward proxy mode. We report additional reductions in client response times and server loads over proxies using the same cache sizes.

Keywords: Storage Embedded Networks, web proxy, push-pull, GUOID, network modeling.

1 Introduction

Internet applications depend on wide-area data dissemination. The delays incurred by clients of these applications over the network are still a big problem. For instance, the delays distract students of distance learning from their online lectures and annoy customers of online stores leading to monetary losses. Caches alleviate the adverse effects of network dynamics on wide-area data dissemination by retaining the fetched data and responding to the subsequent requests for the same data locally. We propose the integration of a cache service into the Internet infrastructure through router caching to improve data dissemination.

Recent Internet traffic studies [12] show HTTP-Web to be the dominant source of the Internet traffic. However, they also report the emerging Peer-to-Peer (P2P) file sharing [20] and media streaming applications [15] to be significant traffic contributors; reaching up to 60% of the traffic transferred on some links [11]. All of these existing and emerging applications depend on wide-area data dissemination and are subject to long delays and unpredictable network dynamics due to sharing of the bandwidth between applications. The user access characteristics of these emerging applications [15, 23] are similar to the web access characteristics [6]. The static nature of the content exchanged and the skewed-popularity, or Zipf distribution [24], makes these Internet applications good candidates to benefit from distributed caching.

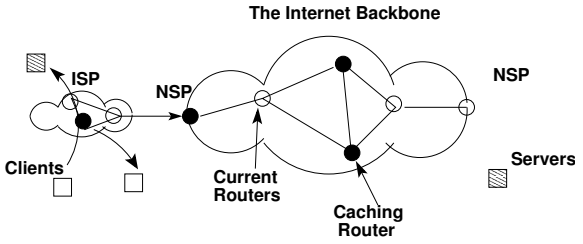


Figure 1: Storage Embedded Network (SEN) architecture gives caching service within networks by using cache-embedded, data-snooping routers. SEN routers reside inside Internet and Network Service Providers (ISP, NSP).

Distributed web proxy caches [8, 3] have successfully been used over the last decade to reduce response times for web clients. However, they are designed to serve only to the web clients and operate mostly in the clients' download or *pull* direction [4]. This is also called *forward proxy* mode. *Push-based* delivery happens when data items are sent by the servers towards the clients without explicit requests from the clients. Today, web servers need to deploy *reverse proxies* or HTTP accelerators for a limited push capability of their content. Content Distribution Networks (CDN) [1] emerged as a new business model by combining the server-side acceleration with client-side caching. Research shows the benefits of supporting both push and pull paradigms for wide-area data dissemination [4, 9]. However, due to the lack of a generic caching support, that covers many applications and both push-pull directions, emerging applications are obliged to reinvent and reimplement similar, but non-interoperable cache services.

Our proposed *Storage Embedded Networks* (SEN) architecture, shown in Figure1, provides an application and direction-independent cache service within the Internet

*This research is supported by the Storage Technologies Department of Hewlett-Packard Laboratories.

This paper was published in the *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS 2004)* , Newport Beach, CA, July 2004.

infrastructure. It uses cache-embedded, snooping routers as its building blocks. Conventional routers do not have this capability. SEN routers are capable of making fast cache-table lookups using globally-unique object identifiers (GUOIDs). Applications use a new protocol called Object Transport Protocol (OTP) to request and retrieve objects using their GUOIDs.

In this paper, we first introduce the operation of existing routers and show how we extend them with a caching capability. Then, we describe the details of SEN architecture, the data-caching routers and the OTP protocol. We use extensive modeling and simulations for comparison of web proxy caches and SEN architecture. We find that when caches can serve requests from all directions as SEN caches do, they utilize the cache space better leading to additional hit rates over forward proxy caches that use the same amounts of cache.

2 Background

Routers are network-level devices that forward packets from their source addresses to their destination addresses. They can be considered the "building blocks" of the Internet infrastructure as they are ubiquitous. Packets traversing the Internet through routers carry the data pertaining to applications and networked protocols. Existing routers commonly use the Internet Protocol (IP) header information to forward packets and ignore the rest of the packet contents. Here, we overview the operation of existing routers to show how we extend these routers with the caching capability in the next section.

2.1 Router Details

Figure 2 shows the simplified hardware model of a router [18]. The *data plane* is the fast path that can forward several million packets per second. The *control plane* [22] is the slow path that handles the route updates and other management routines.

Line cards (Fig. 2) are network interface cards with input and output ports through which packets are received and transmitted, respectively. The core data forwarding function of the line cards is done by a *Forwarding Engine (FE)*. Each FE maintains a *Forwarding Information Base (FIB)* table, which is a partial or full copy of the routing table. FIB is downloaded from the route processor and kept up-to-date. The processor on the FE consults its local FIB to quickly find the next hop information. The *switch fabric* (Fig. 2) has a switching capacity faster than the combined speed of all the input ports. Access into and out of the shared memory in the switch is done by Direct Memory Access (DMA). The *route processor* (Fig. 2) is responsible for maintaining the routing tables. It performs spanning-tree calculations using the route update information disseminated by the routing protocols.

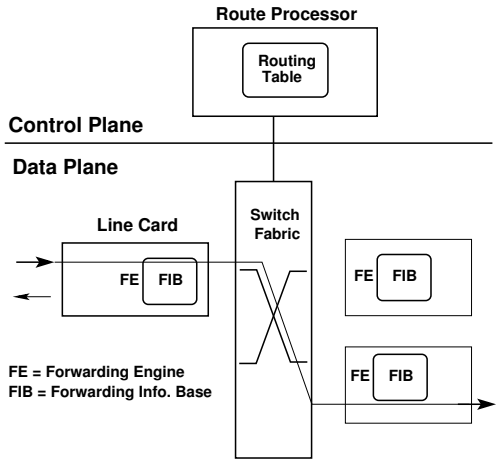


Figure 2: A simplified model of existing routers.

2.2 Packet Forwarding Operation

The packets are received and temporarily stored on the line cards in existing routers. As soon as the packet headers are read from the link-layer (*e.g.* Ethernet) frame, the packet processor forwards the packet header to the Forwarding Engine (FE). The FE is either physically embedded in the line card or is connected to it through the switch. FE reads the destination IP address and finds the matching next-hop entry from the Forwarding Information Base (FIB). The results are quickly (*e.g.* in 2.5 microseconds [2]) returned to the packet processor. An adjacency table holds the link-layer information for the next hops. The processor moves the packet from line card memory to the switch fabric interface. Each packet is moved into the switch fabric frame-by-frame and is stored with a pointer to the output port. A scheduling algorithm such as Weighted Round Robin (WRR) is used to determine which frame should be scheduled to its output interface next. The destination output interface is signaled to take the packet frames out of a known memory location. The frames are sent out to the network via the MAC protocol once they are on the output interface.

2.3 Enabling Technologies

Network Processor Units (NPUs) are software programmable processors designed to process packets at wire speeds. They combine the speed of ASICs with the flexibility of CPUs. NPUs can use packet fields to perform table *lookups*, *pattern matching*, and *data manipulation*. Because of these advantages NPU market has become the fastest growing segment of the microprocessor industry soon after its introduction [14]. The shared internal memory in NPUs can be used to store the program code and possibly small lookup tables. Additional data is stored in the external memories. NPUs enable data caching at wire speeds in SEN routers.

Recently, **fast tree-based lookup techniques** have been implemented in hardware [17, 13]. These techniques are currently being used for route lookups using the 32-bit IPv4 or the 128-bit IPv6 addresses. Caching in routers

requires a similar table lookup for GUOID to determine whether the requested object is cached or not. With IPv4 addresses 20 and 80 million lookups per second are reported using DRAM and SRAM technologies, respectively [17]; for 128-bit IPv6 addresses 2 million lookups per second are projected [13].

3 Storage Embedded Networks

SEN architecture provides caching service within the current Internet infrastructure. It extends current routers with a network card with embedded memory for data caching. Applications use globally unique identifiers (GUOIDs) and a presentation-layer protocol called Object Transport Protocol (OTP) to request and retrieve objects.

3.1 Object Identification

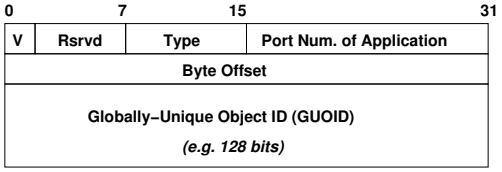
Connection, application and protocol specific numbers are not meaningful beyond their limited domain. To give a generic cache service, a SEN router has to identify objects from different applications using a common format. These identifiers also need to be globally unique to avoid name clashes. Therefore, we use Globally Unique Object Identifiers (GUOID) that we obtain by hashing a file's contents [5]. Emerging Peer-to-Peer (P2P) applications and object-based file systems are already identifying their objects with GUOIDs. Therefore, they don't need any naming change to benefit from SEN caches.

Applications that want to make use of router caching will have to include a GUOID of the embedded objects. Whenever an object is modified, it essentially becomes a new object and is given a new GUOID value. The clients make requests using $<$ *GUOID*, *offset* $>$ pairs. As with the case of web pages, many objects have other embedded static objects. An update operation may only change the GUOID of the top-level object; if so, only that object would need to be retransmitted. Upon reception, clients can hash the contents and compare the result against the expected GUOID of the object to check *data integrity*.

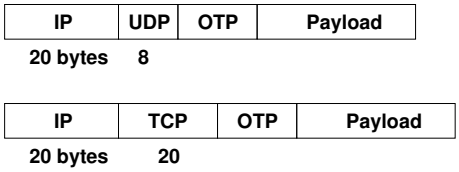
3.2 Object Transport Protocol (OTP)

The data caching service we provide in the routers is not enforced on all by-passing packets. Therefore, existing and emerging applications that see benefit in caching need to indicate their willingness to the routers by setting a bit in the IP-header and including an OTP header.

OTP runs on top of transport protocols and carries objects identified by GUOIDs. OTP is a generalization of the Real Time Protocol (RTP) [21], which is successfully being employed today to carry real-time traffic. RTP introduces object awareness by tagging each packet with a globally unique Synchronization SouRCe (SSRC) identifier, which is the camera-id, and a time-offset for the real-time payload being carried. However, these specific fields make RTP suitable only for real-time traffic.



(a) *OTP header*



(b) *Encapsulation of OTP header*

Figure 3: OTP header and its encapsulation.

The OTP header shown in Figure 3(a) keeps a generic *GUOID* for the object and an *offset* value for the data in packet being transmitted. The length of the payload can be obtained from the transport layer. The *Type* field indicates what kind of action the application wants from OTP. Currently, a `Read` type for a read request and a `Data` type for the returned payloads are defined. Other types such as `Write`, `Publish`, and `Error` are being considered. *Port number* of the original application can be used to communicate the request or reception of the object to the original application. Finally, the *reserved* field could be used to assign priorities to objects to provide differentiated caching services. Figure 3(b) shows the encapsulation of OTP header into the transport (TCP, UDP) and network (IP) headers before it is sent out to network.

3.3 Data-Caching Router

Figure 4 shows the hardware details of our data-caching router. We add a new card that we call *Data-Cache Engine* to the router to enable the caching of objects and process the Object Transport Protocol (OTP) headers. The route lookup in the line cards and the cache lookup in the Data-Cache Engine are done in parallel. The Network Processor Unit (NPU) quickly searches the globally unique object identifiers (GUOID) in the cache table and determines whether it is a hit in the cache or a miss. If hit, the source and destination addresses in the IP header are flipped and the data is returned to the client. Otherwise, the packet processor is informed of the miss result and the packet forwarding proceeds as usual.

The client application reflects its read request by choosing the OTP-READ option in the request type field of OTP header. For example, the HTTP-GET requests used by web browsers to retrieve web pages are mapped to an OTP-READ request. If the router doesn't have the object cached, then the object is retrieved from the server in one or more packets. Each packet carries the bytes starting from the *offset* byte where the previous packet stopped. If the transport is reliable, it handles the packet losses and ordering. After the object is cached by the router, a sub-

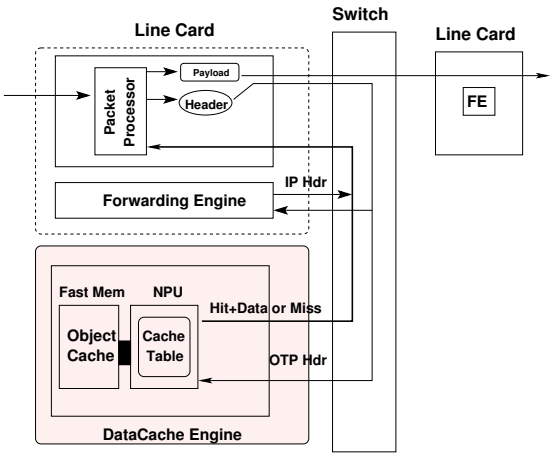


Figure 4: Data-Cache Engine is a specialized line card added to the SEN routers. It makes GUOID lookups to determine whether an object is cached or not.

sequent OTP-READ request may result in a hit. The object is sent the client with OTP-Data field set in the OTP header. If the Maximum Transfer Unit (MTU) between the router and the client is bigger than the MTU between the router and the server, then the router may send more bytes in one jumbo-packet. When the incoming IP packet is marked, denoting that it contains cache-able data, it is forwarded both to the output line card for the next-hop towards the destination and to the Data-Cache Engine. This operation is similar to the IP multicast operation.

SEN architecture is backwards compatible. Therefore, it can be deployed in an incremental fashion into the current Internet infrastructure. Existing routers will forward IP packets with embedded OTP headers as usual and SEN routers will forward unmarked IP headers as existing routers do.

4 Methodology

We use event-driven simulations to compare the proposed Storage Embedded Network (SEN) architecture to the web caches that operate in forward proxy mode. We model a multi-level network topology with nodes representing the clients, servers, caches and routers. The clients and the servers are at the edges of the topology, as shown in Figure 5. We parse web requests from a real proxy trace file and assign them to randomly selected clients. The simulated network nodes forward these requests towards the web servers that contain those objects; calculated by a modulo over the object identifier. Caches are on the paths from clients to the server.

4.1 Network Model

We generate wide-scale network topologies with user directed input for (1) the number of nodes in a Wide Area Network (WAN), (2) the number of Metropolitan Area Network (MAN) nodes per WAN, (3) the number of Local Area Network (LAN) nodes per MAN (4) the number of

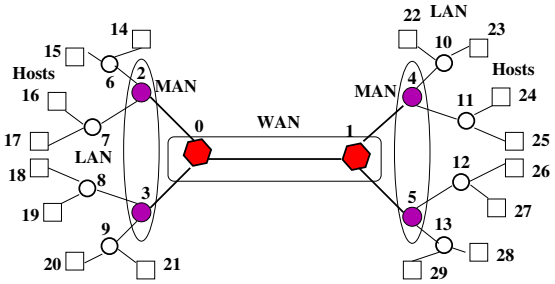


Figure 5: Network topology used for the simulation.

hosts in a LAN, (5) the propagational delays, and (6) the bandwidths for each link. Delays and bandwidths are the same for links of the same type, *e.g.* all the links between the LAN and the MAN levels are the same.

The topology used in the simulations is illustrated in Figure 5. There are 58 directed links. The network delays were based on our **traceroute** collections from our university machines to different universities. The $\langle delay, bandwidth \rangle$ parameters for the links are as follows: WAN-to-WAN $\langle 10ms, 655Mbps \rangle$, WAN-to-MAN $\langle 2ms, 155Mbps \rangle$, MAN-to-LAN links $\langle 1ms, 1.5Mbps \rangle$, and LAN-to-Host $\langle 0.5ms, 10Mbps \rangle$.

4.1.1 Network Queuing Delays

We use a fluid-flow network queuing model [16]. The data that passes through a router is queued in a FIFO queue and serviced at the bandwidth speed of the outgoing link. We use two parameters, *TimeStamp* and *QueueTime*, in each node. *TimeStamp* is the last time when an object was received into the FIFO queue. *QueueTime* is an estimate of how much time the next object will be waiting or *delayed* in this router. *QueueTime* is updated every time a new object is received:

$$\begin{aligned} ServiceTime &= ObjectSize/Bandwidth \\ \Delta T &= (Now - TimeStamp) \\ NewQueueTime &= QueueTime - \Delta T + ServiceTime. \end{aligned} \quad (1)$$

For each queue the *TimeStamp* parameter is initialized to the current simulation time and the *QueueTime* is initialized to zero.

4.1.2 Pending Client Requests

Nodes with caching capabilities can implement a pending-request queue to measure the round trip times (RTT) and to avoid duplicate requests from traversing the same network paths. GUOIDs for the objects that are currently being fetched are enqueued in this pending queue together with the client and timestamp information. When the object is retrieved, all requests that are pending on this object are dequeued and responded. So, in our context *pending* is defined as: holding on to the subsequent requests for an object that has already been requested. We measure the effects of using this strategy for both web proxies and SEN architecture.

4.2 Web Proxy vs. SEN

Web proxy caches [8, 3] are successfully being used today to reduce response times for web clients [6]. These caches define parent-child relationships and structure themselves in a hierarchy thus forming a tree topology from bottom (leaf) to top layers. Clients connect to the leaf caches, make web requests and wait until the proxy returns a response. If the leaf cache cannot fulfill the request, it contacts its parent and so on until the server is contacted and the object is fetched. The object traverses the same path backwards getting cached along the hierarchy.

SEN caches are topologically transparent to the clients and servers. Clients append OTP headers to their request packets and expect improvements from the SEN caches within the Internet infrastructure. The requests are directed towards the server. They are not delayed by additional forwarding to off-the-path parent caches. However, for fairness of comparison we use the same tree topology and same amounts of cache for both architectures.

4.3 Workload

National Laboratory for Applied Network Research (NLANR) operates a global cache hierarchy using Squid proxy caches and has provided important web traces [10]. We use a day long trace collected from a busy web proxy server, SV, in the NLANR Squid cache hierarchy. It contains about 675,342 requests (∼4.20 Gigabytes) to 269,031 unique web objects (∼1.17 Gigabytes). The infinite hit rate is 60% and infinite byte hit rate is 72%. Each client node sees a portion of this workload.

4.4 Cache Replacement

A *cache replacement policy* keeps the cached objects in a priority order and replaces the least valuable objects. An object's access time, access frequency and size are the most commonly used criteria for making replacement decisions. We compare two replacement policies: LRU and GDSF. Least Recently Used (LRU) replaces the least recently accessed object from the cache. Greedy Dual Size with Frequency (GDSF) policy [7] replaces the object with the smallest key $K_i = (C_i \times F_i)/S_i + L$, where C_i is the cost of fetching the object, F_i is the access frequency, S_i is the object size and L is a running age factor. L is set to the key value of the objects that are replaced from the cache and C_i=1.

5 Results and Discussion

We compare web proxy and SEN architectures using the described network topology and web workload. The comparison is based on performance metrics such as hit rates, effects on client response times and effects on the server load. For each metric, the web proxy and SEN nodes are also evaluated separately according to their request pending strategy, *i.e.* with or without pending. Different cache sizes are tested, but the two architectures are always compared at the same cache sizes for fairness.

5.1 Hit Rates

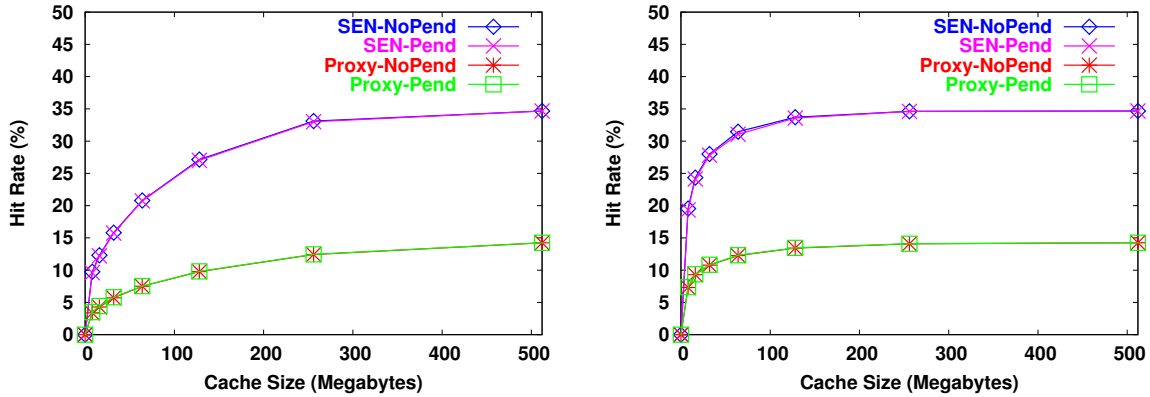
Figure 6 compares the hit rates achieved by the same node in the network (node 6 in Figure 5) when used as a forward web proxy or as a SEN router cache. The hit rates are plotted as a function of increasing cache sizes. The network topology contains caches only at the LAN-level, thus at the edges of the wide-area network. Figure 6(a) shows that the same node achieves up to 20% higher hit rates when used a SEN node rather than as a web proxy node and both with LRU replacement. This is because SEN routers operate both as forward and reverse proxies, simultaneously, thus getting hits in both directions.

Figure 6(b) shows the hit rates for Greedy-Dual Size with Frequency (GDSF) that uses frequency and size criteria for replacement in addition to the recency of accesses. This policy can reach the hit rates achieved by LRU replacement by using only a fraction of the cache sizes used by LRU. This results in the steep increase in hit rates seen in Figure 6(b). The pending strategy does not effect the hit rates, since the decision on whether to pend a request or not comes after a cache miss occurs and after the hit rates are calculated.

Figure 7 compares the hit rates of web proxy and SEN caches with caches at both the LAN and MAN levels. The hit rates reported in this figure are *path hit rates* (PHR), calculated by adding the hit rate of the first level (HR1) to the hit rate of the second level over the traffic missed from the first level: $PHR(1) = HR1 + (1 - HR1) \times HR2$. Note that the path hit rate metric is independent of the direction of the traffic as $PHR(2) = HR2 + (1 - HR2) \times HR1 = PHR(1)$. Due to the increased total cache space the hit rates in Figure 7 are approximately 5% higher in comparison to hit rates in Figure 6. However, the gap between the web proxy and SEN cache hit rates close by 5-10%. The two reasons for this reduction are as follows. First, the web proxies are now serving a relatively larger, unified community (although still only in download or pull direction), thus creating an accelerated path between two LANs within each MAN. Second, with multi-level caches client sharing between communities is exploited to avoid duplications in caches and the saved cache space is used to avoid some capacity misses. Again, the GDSF policy achieves high hit rates with smaller cache sizes than LRU and pending strategy does not affect the hit rates.

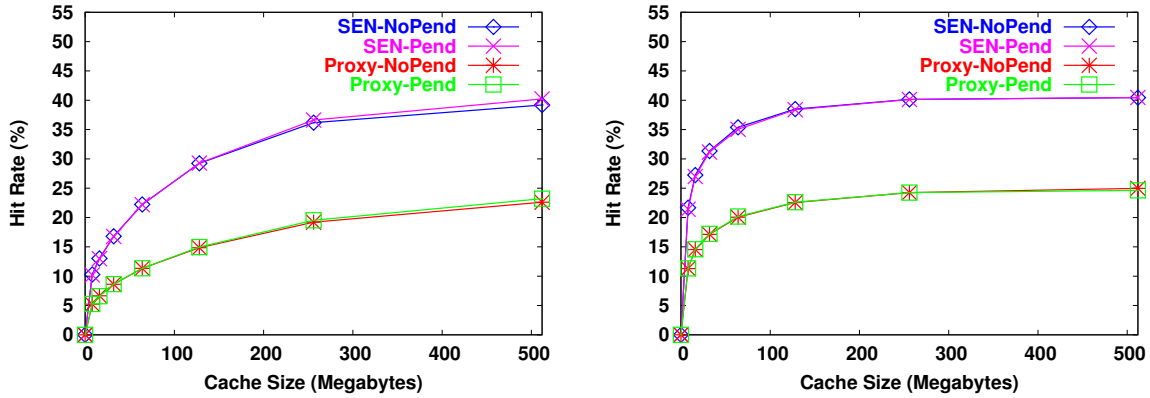
5.2 Client Response Times

Figure 8 compares the average client response times (CRT) of proxy and SEN caches with caches located only at the LAN level. We find that the additional hit rates achieved by SEN caches over proxy caches illustrated in Figure 6 did not immediately result in *additional* reductions in CRT. This is because as a single level cache the SEN router could only accelerate the server content one



(a) LRU replacement. (b) GDSF replacement.

Figure 6: Hit rate comparison of forward web proxies vs. SEN routers; Caches are only at the LAN level.



(a) LRU replacement. (b) GDSF replacement.

Figure 7: Hit rate comparison of forward web proxies vs. SEN routers; Caches are at the LAN and MAN levels.

hop (0.5ms) closer to the clients and not more. We will see the immediate results of increased hit rates on the server load later in this section. Note that, in Figure 8 the CRT for SEN caches are slightly lower than the CRT for proxy caches. The emerging distinction is seen better in Figure 8(b) for GDSF policy that achieves higher hit rates than LRU with smaller cache sizes.

In Figure 9 the success of multi-level SEN caches over multi-level web proxies becomes more eminent. SEN caches reduce the client response times additionally 30-50% over the CRT's achieved by the web proxy caches. *Providing caching on the paths for data traversing in all directions and not limiting the cache service to only the client's pull direction results in better use of the same amount of cache space.*

Another important result seen in both Figure 8 and 9 is the effect of pending policy over client response times. For both SEN and web proxy caches, if the requests for already requested objects can be pended inside the cache, then many repeating requests will be responded at once when the object is fetched for the first request. Pending a request in a router can be more challenging than pending requests in web proxies, since routers are meant to forward packets as fast as possible.

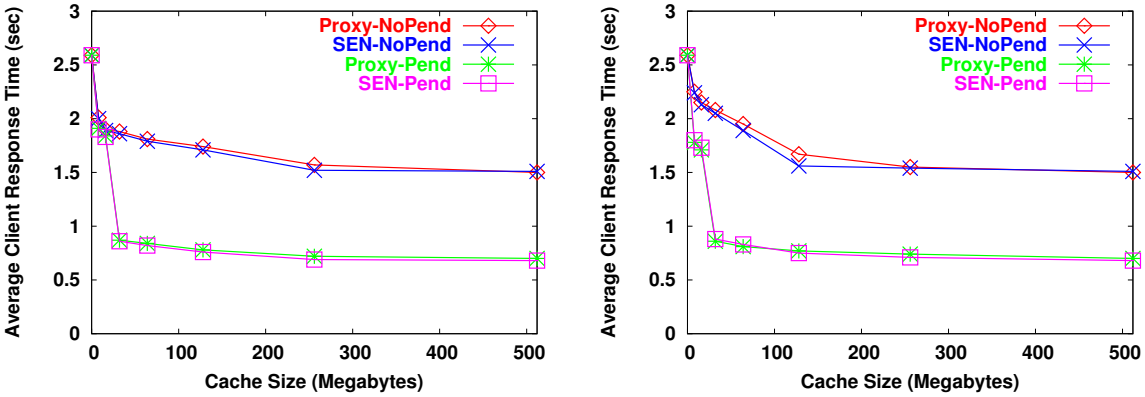
5.3 Server Load

Figure 10 shows the comparison of the effects of web proxy vs. SEN caches on the server load expressed as number of requests handled by the web servers (cumulative). The caches are located only at the LAN level. The web servers experience the additional benefits of SEN caches starting with LAN-level caching. The additional hits achieved by SEN caches result in additional reductions in the load of the server, which is one hop away from the cache. This leads to increased server scalability. The results for two-level caching are similar and are omitted.

6 Related Work

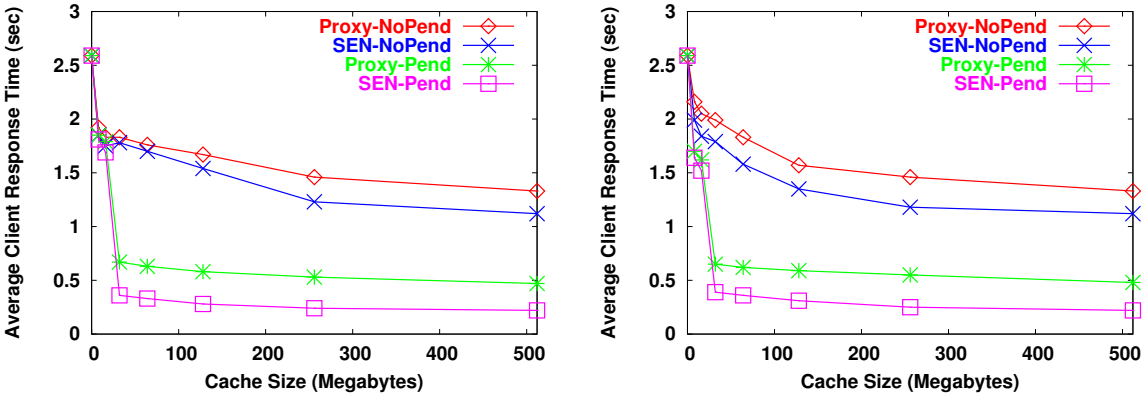
Our proposed globally-distributed cache infrastructure relates to many distributed systems with distributed caches. These include web proxy caches [8, 19], Content Distribution Networks (CDNs) [1], and Peer-to-Peer (P2P) networks [20]. Our technique differentiates itself and complements the mentioned architectures by providing an application- and direction-independent, widely-distributed cache resource within the Internet.

Content Distribution Networks (CDN) such as Aka-



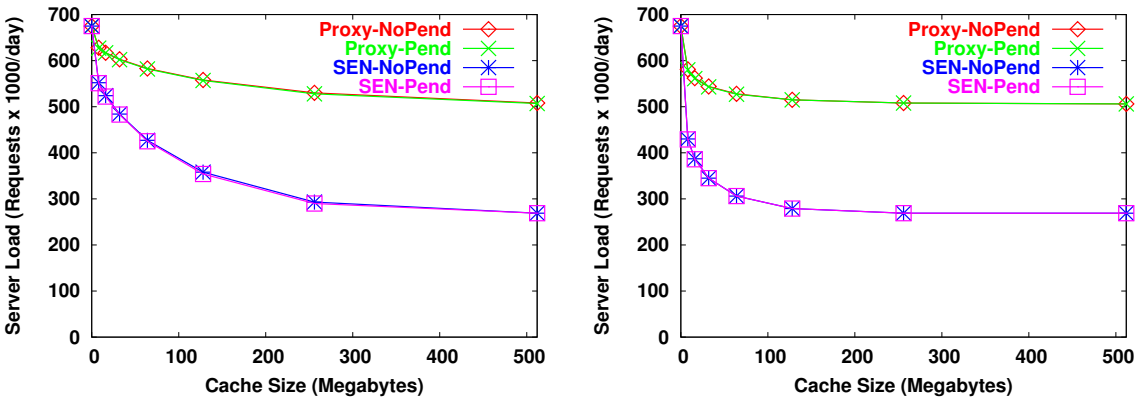
(a) LRU replacement. (b) GDSF replacement.

Figure 8: Client response times (CRT) comparison of web proxy caches vs. SEN routers Caches are only at the LAN level.



(a) LRU replacement. (b) GDSF replacement.

Figure 9: CRT comparison of web proxy caches vs. SEN routers; Caches are at the LAN and MAN levels.



(a) LRU replacement. (b) GDSF replacement.

Figure 10: Server load reduction comparison of web proxy caches vs. SEN routers; Caches are only at the LAN level.

mai [1] "akamaize" the server-side web content and combine push-pull of this content within their private networks. SEN cache architecture uses globally unique object identifiers (GUOIDs) and routers to make its cache service globally accessible to all entities on the Internet. Therefore, SEN architecture can act as a cache infrastructure for CDNs, so that they can push their content along client paths to ensure the delivery of closest copies. SEN architecture can also provide stable caching service between the peers of an overlay network.

7 Conclusions

We described the design of a new caching architecture for the Internet infrastructure, called Storage Embedded Networks (SEN). We compared the SEN architecture to web caches operating in forward proxy mode. We used the same network topology, the same web workload, and same cache sizes for comparisons. We found that being able to service requests over all network paths helps utilize cache spaces better leading to improved performance. The benefits of SEN caches increase as more caches are installed along the network paths: *i.e* additional reductions of client response times and served loads can be achieved over the web proxy caches. The network cost is continuous, but memory cost is a one-time investment. Even with modest hit rates router data caches can pay for themselves within a short period of time.

References

[1] Akamai, http://www.akamai.com.

[2] Cisco Routers Layer 3 Forwarding, http://www.cisco.com.

[3] Squid web proxy cache, http://www.squid-cache.org/.

[4] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 183–194, Tucson, AZ, May 1997.

[5] K. Akala, E. Miller, and J. Hollingsworth. Using content-derived names for package management in Tcl. In *Proceedings of the 6th Annual Tcl/Tk Conference*, pages 171–179, San Diego, CA, Sept. 1998. USENIX.

[6] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96)*, 1996.

[7] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance (WISP '99)*, Atlanta, Georgia, May 1999.

[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Annual Technical Conference*, San Diego, CA, 1996.

[9] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic web data. In *Proceedings of the 10th International World Wide Web Conference*, pages 265–274, Hong Kong, China, May 2001.

[10] S. G. Dykes and K. A. Robbins. A viability analysis of cooperative proxy caching. In *Proceedings of INFOCOM '01*, pages 1205–1214, 2001.

[11] M. Fomenkov, K. Keys, D. Moore, and K. Claffy. Longitudinal study of Internet traffic in 1998-2003. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2003.

[12] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 17(6):6–16, 2003.

[13] T. Harbaum, D. Meier, M. Zitterbart, and D. Brokelmann. Hardware-assist for IPv6 routing table lookup. In *SYBEN'98*, pages 434–443, Zurich, Switzerland, 1998.

[14] G. Memik and W. H. Mangione-Smith. Design and evaluation of a network processor accelerator for layer seven applications. *Submitted to ACM Transactions on Embedded Computing Systems (TECS)*, 2004.

[15] B. K. B. Mohamed M. Hefeeda. On-demand media streaming over the Internet. In *Proceedings of International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, San Juan, Puerto Rico, May 2003.

[16] D. Nicol, M. Goldsby, and M. Johnson. Fluid-based simulation of communication networks using SSF. In *Proceedings of 1999 European Simulation Symposium*, Erlangen-Nuremberg, Germany, Oct. 1999.

[17] D. Pao, C. Liu, A. Wu, L. Yeung, and K. S. Chan. Efficient hardware architecture for fast IP address lookup. New York, NY, June 2002. IEEE.

[18] C. Partridge and P. Carvey. A 50-Gb/s IP router. *IEEE/ACM Transactions on Networking*, 6(3):237–248, June 1998.

[19] P. Rodriguez, C. Spanner, and E. Biersack. Web caching architectures: hierarchical and distributed caching. In *Proceedings of the 4th International WWW Caching Workshop*, 1999.

[20] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, pages 156–170. SPIE/ACM, Jan. 2002.

[21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real time applications. Request For Comments (RFC) 1889, IETF, Jan. 1996.

[22] A. Shaikh and A. Greenberg. Experience in black-box OSPF measurement. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[23] D. A. Tran, K. A. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of INFOCOM '03*, 2003.

[24] G. K. Zipf. Human behaviour and the principle of least effort. *Addison-Wesley*, 1949.