A Case for Caching File Objects Inside Internetworks

Peter B. Danzig
Richard S. Hall
Michael F. Schwartz

CU-CS-642-93 March 1993

University of Colorado at Boulder

Technical Report CU-CS-642-93
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-430

A Case for Caching File Objects Inside Internetworks

Peter B. Danzig
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
danzig@usc.edu

Richard S. Hall, Michael F. Schwartz
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430
{rickhall,schwartz}@cs.colorado.edu

March 1993

Abstract

This paper presents evidence that several, judiciously placed file caches could reduce the volume of traffic on the NSFNET backbone by 21%. In addition, if FTP client and server software automatically compressed data, this savings could increase to 27%. We believe that a hierarchical architecture of whole file caches, modeled after the existing name server's caching architecture, could become a valuable part of any internet.

We derived these conclusions by performing trace driven simulations of various file caching architectures, cache sizes, and replacement policies. We collected the traces of file transfer traffic employed in our simulations on a network that connects the NSFNET backbone to a large, regional network. This particular regional network is responsible for about 5 to 6% of NSFNET traffic.

While this paper's analysis and discussion focus on caching for FTP file transfer, the proposed caching architecture applies to caching objects from other internetwork services.

1 Introduction

The Internet's File Transfer Protocol (FTP [PR85]) is the source of about half of the bytes that traverse the NSFNET backbone [Mer92]. Our previous study showed that many large, essentially read-only files are popularly FTP'd across the Internet many times each day [EHS92]. Through trace-driven simulation and measurements of traffic characteristics, this paper demonstrates that over half of the FTP bytes transmitted over the backbone could be eliminated. Regional networks should see similar savings.

The reason that such a hefty bandwidth savings is possible is simple. FTP has evolved into an embryonic, distributed file system that is open to all computers implementing TCP/IP. Unencumbered by consistency control, locking, security, or caching, FTP's client and server software are tidy and easily ported. Across wide-area networks and a multitude of platforms, FTP is now a ubiquitous global file system. Recently, more advanced facilities have been layered atop FTP, such as primary-copy replication [McL91] and various directory services [ED92, HS93]. Unfortunately, FTP's simplicity also leads to many inefficiencies and inconsistencies. Since FTP will not soon disappear, we ask how can it be improved?

1.1 How to Improve FTP Efficiency

From the perspective of a global file system and a bulk-transfer protocol, FTP lacks *server-independent* file naming, whole-file caching, and automatic file compression. We discuss these three features below.

1.1.1 Server-Independent Naming

As the global FTP file space evolves, popular files are hand-copied or automatically mirrored on many FTP archives. Hand-replication leads to data inconsistencies that frequently force users to filter through many different versions of a file. Excessive mirroring of archived files burdens the user with choosing a "close" and lightly-loaded archive. We illustrate this with two examples.

When MIT released the fifth version of the X-window system (X11R5), they hand-replicated copies of the distribution directory in 20 different FTP archives around the world to help distribute Internet load. People looking for the new distribution discovered the names of these servers through electronic mail, news articles, and FTP directory services like archie. They then hand-selected a server from which to retrieve the distribution. In this example, the lack of server-independent naming means that these identical files have 20 different names, one for the server name + file name at each archive.

The lack of server-independent names, combined with hand-replication, also leads to data inconsistency. Except for the best managed archives, most FTP archives contain out-of-date versions of popular files. For example, archie locates 10 different versions of *tcpdump* archived at 28 different sites, and it locates 20 different versions of *traceroute* stored at 88 different sites. Because the global FTP file system is so desperately inconsistent, users frequently retrieve and sort through several versions of a file before encountering the right one.

For our purposes, the server-independent name of a file should include the hostname and full path name of the primary copy of a file. The actual representation could be the naming convention being developed by the Internet Engineering Task Force (IETF) [Int93].

1.1.2 Caching Architectures

The performance of all distributed systems depends on caching. If they don't cache, they don't perform well. We propose an architecture of file caches, accessed via server-independent file names, that would improve FTP performance, eliminate the inconsistencies that arise from hand-replication, and reduce wide-area FTP traffic.

We propose building a next generation of FTP clients that, given the server-independent name for a file, would retrieve it from file caches rather than from the originating host. The dark ovals in Figure 1 represent file caches residing on secure machines placed near the entry points to regional networks and near the core of backbone networks. The organization of these caches could be similar to the organization of the Domain Name System. Clients

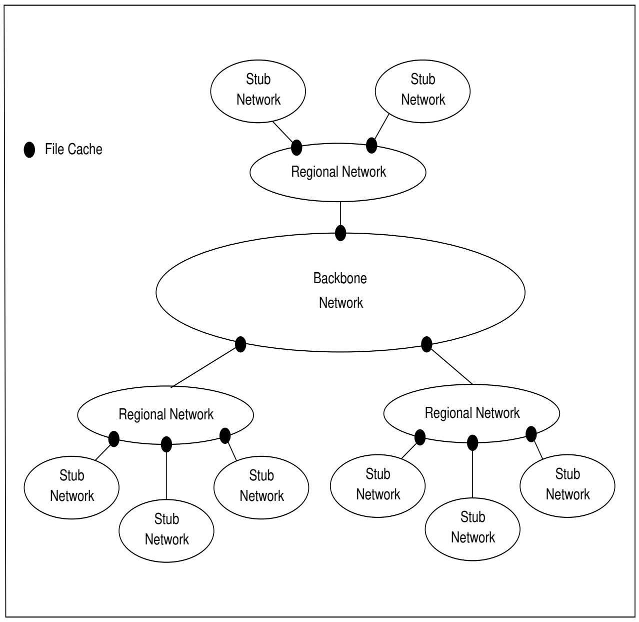

Figure 1: File caches organized hierarchically by network topology.

send their requests to one of their default cache servers. If the request misses the cache, then the cache recursively resolves the request with one of its parent caches or directly from the FTP archive.

Using trace driven simulation, we demonstrate that simple file caching mechanisms implemented on inexpensive workstations could reduce NSFNET backbone loads by 21%. We argue that file caches require changes to neither the definition of FTP nor to its existing servers. File caches and their clients would be layered over the existing FTP service.

1.1.3 Presentation Layer Issues

Our traces show that up to 31% of file transfers are not compressed, and that conservatively speaking, automatic compression would reduce the volume of backbone traffic by up to 6%. Hence, from the perspective of a bulk transfer protocol, we believe that automatic compression should be added to the definition of FTP.

1.2 Why save bandwidth now?

We see two reasons for implementing file caching even as backbone bandwidths blossom.

First, file caches cost a fraction of link upgrades, and they can be employed at regional networks or even at the edge of overloaded, intercontinental links. The volume of Internet FTP traffic will not soon decrease, and caching at one node saves bandwidth everywhere. Moreover, as the Internet becomes commercialized, services other than FTP could exploit these caches to save transmission costs.

Second, although real-time, multimedia traffic will eventually contribute a significant portion of network load, this does not mean that it will dwarf file transfer traffic. Rather, we believe, it means that the average size of FTP transfers will continue to grow, and the fraction of bandwidth due to FTP will stay significant. As justification for this, note that FTP file sizes have grown an order of magnitude from 1989 to 1993 [DJCM91]. Multimedia applications will almost certainly lead to even more demand for widespread distribution of audio and video transcripts, further exacerbating the load caused by file transfers. As Table 6 shows, already 20% of FTP bytes transfer graphics and video traffic.

1.3 Outline of Paper

In Section 2 we describe our traces of FTP file transfers collected from the NSFNET backbone. In Section 3 we report the results of a trace-driven simulation of a single file cache and a synthetic workload driven simulation of a network of file caches. In Section 4 we discuss various practical implementation problems with file caches. We contrast our findings with the work of others in Section 5, and summarize our findings and recommendations in Section 6.

2 Trace Collection

We collected traces of FTP file transfer traffic from the Boulder entry point to the NSFNET backbone. To be precise, we collected data from the 192.43.244 network inside the National Center for Atmospheric Research (NCAR). This is the primary connection from the eastern part of the Westnet regional network (Colorado, New Mexico, and

Wyoming) to the NSFNET backbone (see Figure 2). It also provides primary NSFNET connections for the University Corporation for Atmospheric Research (NCAR's parent organization) and several Mexican networks (via the University Satellite Network), as well as secondary NSFNET connections for the NASA Science Internet and Los Alamos National Laboratory. This entry point contributes between 5% and 7% of the bytes carried by the NSFNET (See file t3-9210.bnss, [Mer92]). It contributed 6.35% of NSFNET bytes during the month the traces were collected.

We captured IP packets on a DECStation 5000, filtered the packets to single out FTP control connections and file transfer connections using a modified NFSwatch program [CM91], discarded information that might compromise the privacy of individuals, and wrote a trace record for each transferred file. Table 1 shows a typical trace record. We recorded only IP network numbers rather than full IP addresses, to preserve individual privacy. A file's *IP source address* is the network address of the machine that provided the file; the *IP destination address* is the network address of the machine that read the file. Note that these two definitions are independent of whether the FTP client issued a *put* or *get* command.

File Name	Masked IP Source Address	Masked IP Dest Address	Time Stamp	File Size	File Signature
sigcomm.ps.Z	128.138.0.0	18.0.0.0	10/8/92 03:45:15	12,345	abc... ...xyz

Table 1: Fields of a trace record.

The *signature* field consists of between twenty and thirty-two bytes uniformly sampled from a file.[1] We used the file size plus signature to identify that files on different hosts are probably identical. We say "probably identical" because, to preserve privacy and keep the trace storage requirements reasonable, we only stored a file's length and signature, rather than its entire contents. If two files' lengths and signatures matched we said they were the same file. Even if they had the same name, if their lengths or signatures differed we said the files were different. We use this knowledge in section 3 when evaluating the impact of file caching on NSFNET load.

[1] We attempted to collect thirty-two bytes, but accepted as few as twenty bytes to make signature collection more resilient to packet loss.

Quantity	Value
Trace dates	9/29/92 - 10/8/92
Trace duration	8.5 days
IP Packets captured	$4.79 \cdot 10^8$
FTP packets	$1.65 \cdot 10^8$
Peak IP packets/second	2,691
Interface drop rate	0.32 %
FTP connections (port 21)	85,323
Avg connection time	209 seconds
Avg transfers per connection	1.81
Actionless connections	42.9%
"dir"-only connections	7.7%
Traced file transfers	134,453
File sizes guessed	25,973
Dropped file transfers	20,267
Fraction PUTs	17.0%
Fraction GETs	83.0%

Table 2: Summary of traces.

2.1 Summary of Traces

Table 2 summarizes key features of the traces. We captured 134,453 separate file transfers in 8 days of tracing; we failed to capture another 20,267 transfers for reasons discussed below. Since 85,323 different FTP control connections transferred these 134,453 + 20,267 files, on average, FTP clients transferred 1.81 files per connection. 42.9% of all connections resulted in no actions, probably indicating mistyped passwords. Another 7.7% of all connections listed directories but did not transfer files, indicating users searching for information or automated directory retrievals being performed by systems like archie.

Table 3 below records other details about the files transferred. "Files" refers to unique files, while "transfers" includes multiple transmissions of individual files. From this table we see that the mean transfer size is significantly larger than the median transfer size. Small files are transferred

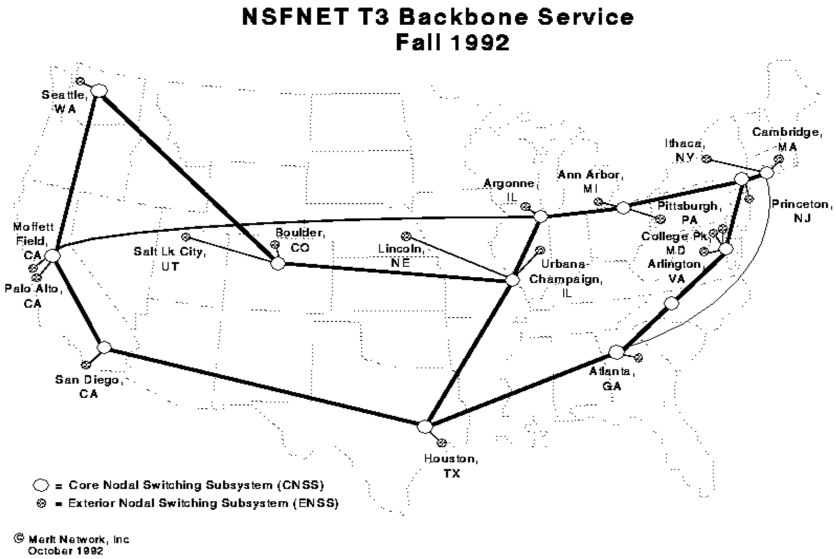

Figure 2: NSFNET backbone topology at time of data collection.

Mean file size (bytes)	164,147
Mean transfer size (bytes)	167,765
Median file size (bytes)	36,196
Median transfer size (bytes)	59,612
Mean file size for dupl. transfers	157,339
Median file size for dupl. transfers	53,687
Total bytes transferred in trace	25.6 GB
Files transferred $\geq$ once/day	3%
Bytes due to these files	32%

Table 3: Summary of transfers.

more often than large files, but some very large files do get transferred (in agreement with our and other studies of Internet traffic [DJC+92]). The table also shows that the relatively few files that are frequently transferred account for a large percentage of the volume.

2.1.1 Estimating Packet Loss Rate

We estimated the trace's packet loss rate by counting the number of missing signature bytes for file transfers whose signatures came from 32 different packets. The signature bytes of transfers equal to or larger than 32 network Maximum Transfer Units (MTUs) come from different packets. Since previous studies [DJC+92, WLC92] have shown that most FTP data connections employ a 512 byte TCP segment size, we approximated that the signature bytes of transfers greater than 512*32 bytes long came from different packets. For each sufficiently long transfer, we found the highest numbered, successfully recorded signature byte. Since any signature byte lower than the highest valid byte must have been transmitted, any missing signature bytes lower than this byte must have been dropped. Ignoring the effects of retransmissions due to flow control, we estimated a loss rate of 0.32 percent.

Reason for Loss	
Unknown but short transfer size	36%
Stated file size wrong or transfer aborted	32%
Transfer too short (< 20 bytes)	31%
Packet Loss	< 1 %
Mean dropped file size	151,236
Median dropped file size	329

Table 4: Summary of lost transfers.

Extension	Compression Format
*.z	UNIX
.arj *.lzh *.zip *.zoo	PC
.*hqx	Macintosh
.gif* *.jpeg* *.jpg	Image
Bytes transferred	25.6 GB
Uncompressed bytes	8.7 GB
Fraction uncompressed	31%
Fraction wasted traffic	6.2%

Table 5: File naming conventions used to detect compression.

2.1.2 Lost Transfers

Table 4 focuses on the 20,267 transfers that we detected but failed to capture because we could not construct a complete signature. These losses occurred for four reasons. First, when an FTP server failed to transmit the size of the file before commencing the data transfer, we computed the signature assuming the file was 10,000 bytes long. Thus, we could not compute a signature for sizeless transfers shorter than (20/32)*10,000 bytes. The factor of 20/32 comes from the fact that we considered a signature valid if we managed to collect 20 of the 32 possible bytes. Second, some FTP servers stated incorrect transfer lengths, or the transfer was aborted by one side or the other. Third, we discarded all transfers of 20 bytes or less because our software insisted on collecting a minimum signature length of 20 bytes. Finally, packet loss prevented us from collecting some signatures, but this was rare.

Although we discarded a substantial number of transfers, most were *small* files (note that the mean dropped file size was much larger than the median in Table 4). Hence these discarded files do not significantly affect the benefits of file caching discussed below.

2.2 FTP's Missing Presentation Layer

As typically implemented, FTP acts as a session layer protocol – any presentation layer transformation, such as encryption, compression, and data representation mappings, must be performed outside of FTP. We identify two situations where network traffic could be saved by placing these transformations, and the decisions about when to apply them, inside FTP.

First, rather than depending on users to do it, FTP could compress data on-the-fly. Unfortunately, we could not directly measure the benefits of compression because, to maintain privacy, we discarded FTP data. Fortunately, filenames frequently convey their data format, and, in this manner, we estimate that only 69% of FTP bytes were transmitted compressed, and many of these bytes were in a 7-bit format that could be further compressed. Table 3 lists the compressed formats we recognized. Assuming FTP implemented Lempel-Ziv compression [Wel84], the most common compression algorithm, and conservatively estimating that the average compressed file is 60% the size of the original, then automatic compression would eliminate 40% of 31% of the FTP bytes transmitted, or 12.4% of FTP bytes. Again, assuming that half of NSFNET bandwidth is FTP transfers, NSFNET backbone traffic would be reduced by 6.2%.

Second, most FTP clients and servers attempt to translate data into and out of an 8 bit ASCII "network standard" representation as a default. To transfer a file without this conversion, the user must precede the file transfer request with a command telling FTP to turn off data conversion. While some clients and servers automatically recognize such *binary* transfers and disable conversion, others do not. A common mistake is to transfer binary data without first disabling conversion.

When this happens, the transfer is garbled and is usually retransmitted.

To estimate the amount of bandwidth wasted by this problem, we counted the number of file transfers for which files with the same name and length but two different signatures were transmitted between the same source and destination network within 60 minutes of each other. We found that 1,370 (2.2%) of the 63,109 files transferred in our trace experienced this problem, causing a total of 278 MB to be transmitted. Because this is 1.1% of the total bytes transmitted, and again assuming that half of NSFNET bandwidth is FTP transfers, this problem accounts for only about .5% of NSFNET backbone traffic.

3 File Caching

This section evaluates the reduction in NSFNET traffic achievable by various file caching architectures, cache sizes, and replacement policies. We obtained these measures by driving simulations with our NCAR traces. We focus on the NSFNET backbone because we know its topology (see Figure 2) and global traffic statistics [Mer92]. We excluded regional and local networks from the measures presented here by substituting NSFNET entry points, or External Nodal Switching Subsystem (ENSS), for each IP address found in the traces. This eliminated our measure's sensitivity to particular regional and local topologies.

While we ignored regional and local topology in our measures, we believe that demonstrating bandwidth savings on the backbone illustrates the magnitude of the possible savings on these networks. We could have applied this same entry point substitution technique to model the impact of caching on stub networks, regional networks, or intercontinental links.

We measured bandwidth savings in units of *byte-hops*. For each traced file transfer, we calculated the actual backbone route over which the data traveled, computed the route's hop count, and multiplied the hop count by the file size. This measure reflects the resources consumed by a given file transfer. We summed this number for all of the transfers to compute the total savings. In the simulations below, we report both the cache hit rate and the percentage drop in backbone traffic after file caching. We used actual NSFNET routes in the simulation so that routes would correspond to realistic networks.

We accounted for cold start caching effects by driving each cache with the first 40 hours of traces before accumulating hit rate and bandwidth reduction measurements.

In the simulation models we did not include the cost of locating cached copies of data, because as files continue to increase in size, location costs become comparatively insignificant. A lookup could be done with a small number of RPCs, similar to how Internet host names are resolved by a small number of RPCs in the Domain Naming System.

Throughout this section, when we speak of placing a cache at an NSFNET switch, we refer to tapping a cache into the adjacent network. Clearly, it would be unwise to burden backbone routers themselves with the additional computational expense of maintaining file caches.

3.1 External Nodes Caches

We begin by evaluating the reduction in NSFNET backbone traffic achievable by placing file caches at the network's ENSS's. With these caches, if a user in Atlanta retrieved a file from the University of Colorado, the file would end up in the Atlanta ENSS's file cache.

Because we only consider the effects of caching on the backbone, caches can only save byte-hops for the portion of file transfers that traverse the backbone. For example, it would not make sense to cache a University of Colorado-sourced file at the NCAR ENSS if the file were destined for Atlanta, because our model treats the network as having no hops between the University of Colorado and NCAR. Therefore, the policy for an ENSS cache should be to cache only those files whose destinations are on the local side of the cache. Because of this policy, it also makes no sense to simulate caches at any other ENSS besides the local one using our trace stream, since those caches will never cache files from the local reference stream. Instead, we compute potential cache savings at a single ENSS.

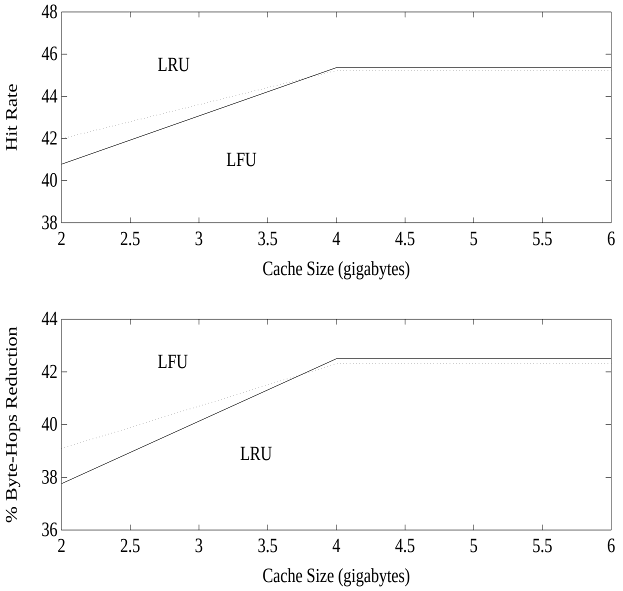

Figure 3: Bandwidth reduction for locally generated traffic from external node caching.

Figure 3 shows the fraction of locally destined bytes that hit the cache and the byte-hop reduction as a function of cache size for a single cache placed at the NCAR ENSS. We simulated 2 GB, 4 GB, and infinite cache sizes. As can be seen, a 4 GB cache achieves nearly optimal savings.

Note that while 4 GB is a significant proportion of the total traffic from our traces (Table 3), a steady state hit rate was reached after only 2.4 GB had been passed through the cache. This number represents the working set size of (Westnet) popular FTP files.

Duplicate transmissions tend to occur within a small time window of one another. For example, as Figure 4 illustrates, the probability of seeing the same duplicate-transmitted file within 48 hours is nearly 90%. For this reason, Least Recently Used (LRU) and Least Frequently Used (LFU) replacement policies are nearly indistinguishable. LFU's slightly better performance for smaller caches indicated in Figure 3 can be explained by the fact that approximately half of the references are unrepeated. Hence, any file transmission that has been repeated once is more likely to be repeated many times than a completely new reference. As the cache gets large, the difference between poli-

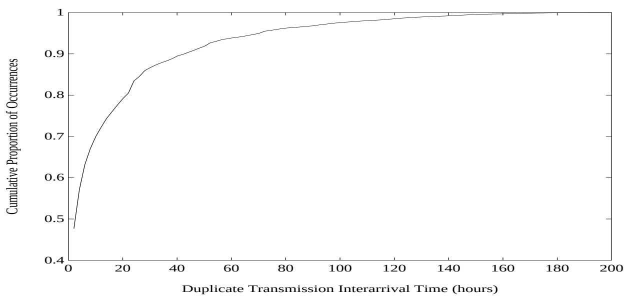

Figure 4: Cumulative interarrival time distribution for duplicate transmissions.

cies becomes insignificant. Because of this, we only simulated one policy (LFU) for the remainder of our experiments.

Our measurements indicate that most files are transferred to three or fewer destination networks, but a small set of highly popular files were duplicate transmitted to hundreds of destination networks. This argues for using multiple caches. Assuming that our traces are representative, if we placed a file cache at each ENSS, then Figure 3 reflects the drop in total NSFNET FTP traffic that would be achieved. Because FTP accounts for about 50% of all NSFNET backbone traffic, this means caching could reduce NSFNET load by 21%.

Below, we evaluate an alternative architecture that employs fewer caches: placing caches inside the backbone network. In the NSFNET, these are called Core Nodal Switching Subsystems or CNSS's.

3.2 Core Node Caches

Because core nodes experience significantly more traffic than peripheral nodes do, it may be possible to place caches at just the most highly traversed core nodes and approach the savings of many more caches at peripheral nodes. Note that unlike the caching policy at ENSS's, transfers for *all* sources and destinations are eligible for caching at CNSS caches.

It is not obvious how well core caching would work. Core caches will experience a broader collection of file references, so the working set may be much larger. Moreover, fewer hops would be saved by these caches, since several hops beyond the periphery are needed to reach these caches.

Because we collected data at only one point in the NSFNET, we evaluated the effect of CNSS caches using a synthetic workload constructed as follows. We began with the subset of transfers with destinations on the local side of the data collection point (NCAR). We excluded transfers with remote destinations (and local sources) because these only represent Westnet's sliver of the global FTP file data. In contrast, those traces with local destinations represent the retrieval patterns of a large population (Westnet) against files found around the global Internet.

Next, we created an artificial workload model consisting of a random generator of file references. These references were composed of a set of globally popular files (those locally destined files transmitted multiple times in the NCAR traces) and a set of globally unique files (those transmitted once in our traces). References to these unique files were created so that they always miss the cache. Note that this parameterization assumes that the ratio of popular to unique files is the same at each ENSS, and that each ENSS requests the same globally popular set of files in the same relative proportions as seen in the NCAR traces. Each popular file is generated with the probability encountered in the trace.

The simulation proceeds in lock step. At every step, each ENSS calls the generator and retrieves the specified file. We modeled the variation in traffic level generated by each ENSS by scaling the number of file transfers by the relative counts of traffic reported by Merit, Inc. (see file t3-9210.bnss, [Mer92]).

We chose where to place CNSS caches by ordering the CNSS's according to which node would prevent the most downstream byte-hops for the given synthetic workload. This corresponds to how one would engineer a caching network by first measuring FTP packet counts at each CNSS over a long period of time. Note that a "perfect" ranking algorithm would require running simulations for one CNSS at a time, and chosing the one that improved caching the most, then for 2 CNSS's at a time, etc. We chose instead to use the following approximate ranking algorithm, which could differ from the perfect algorithm because of interference between the caches:

Let current graph = backbone route graph;
For i = 1 to NumCaches do

Determine the CNSS for which $\sum_{\forall transfers}$ [bytes $\cdot$ (hops remaining to destination)] is maximal, using the current graph;

Assign this CNSS rank i;

Remove this CNSS from the current graph and deduct its outgoing flows to the adjacent nodes;

end

Figure 5 shows the effects of placing caches at the top 1 through 8 CNSS's, with a range of cache sizes. The hit rates and global byte-hops savings for an individual CNSS cache are less than the corresponding savings of a single ENSS cache, because it takes at least one additional hop to reach a CNSS. However, achieving a *global* savings equivalent to the savings in local traffic from one ENSS cache would require placing caches at all ENSS caches. Therefore, the lower savings at the CNSS caches should be interpreted in the context of lower overall costs for purchasing fewer caches. In particular, our traces detected 35 different ENSS's (each of which would need a cache), yet placing caches at just 8 CNSS's would accomplish 77% as much good, at one quarter the cost.

We caution against drawing strong conclusions about exact hit rates or cache placement choices based on this synthetic workload. Rather, our point is to indicate that even with a significant amount of non-duplicated traffic (the synthetic workload passed unique files totaling 74 GB of data through the CNSS caches), core node caching

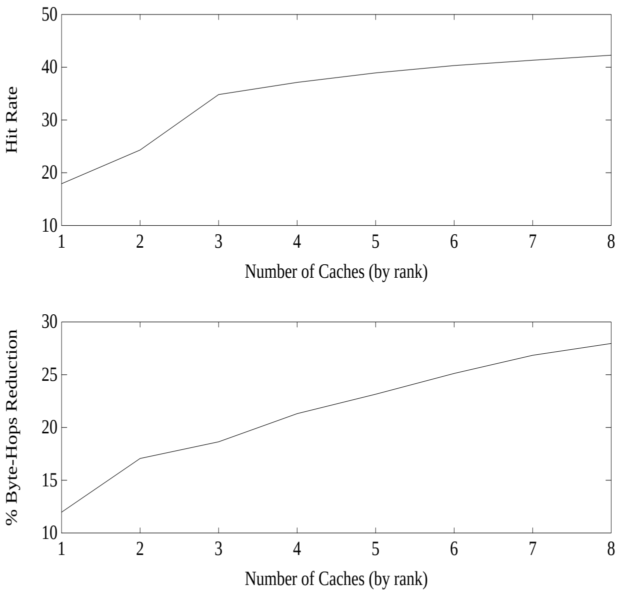

Figure 5: Bandwidth reduction due to core node caching.

can reach a steady state working set with moderate sized caches, and significantly reduce backbone traffic.

The next section describes a hierarchical architecture in which caches fault data across the network from other caches, rather than retrieving it from the orginal source. We did not simulate this architecture because FTP files that are transmitted more than once tend to be transmitted many times (see Figure 6). Faulting from cache to cache would only save transmission costs the first time the file is retrieved. After that point, the files would be in the cache, for the vast majority of the savings. Hence, we are not sure that the complexity of cache-to-cache coordination is justified in the case of FTP files.

4 Practical Matters

We now consider how to make object caching as practically appealing as it is theoretically appealing. This section addresses issues of cache machine performance, cache consistency, how to locate caches, and the privacy and security of fetching objects from caches. We intentionally refer to *objects* rather than FTP files, because services

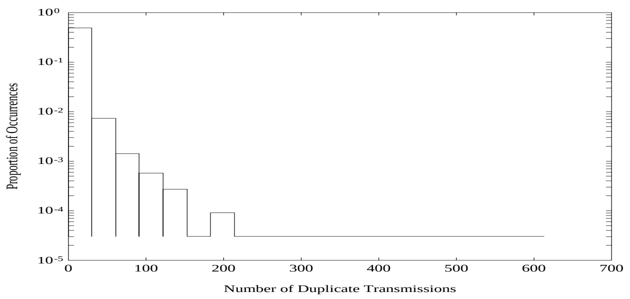

Figure 6: Distribution of repeat transfer counts for duplicate file transmissions .

other than FTP (such as WAIS [KM91]) could employ these caches via universal resource locators [Int93].

4.1 Cache Machine Load

We believe that well designed object caches can keep up with demand rather than becoming performance bottlenecks. They can optimize their performance by exploiting FTP's sequential access. Upon access, they can prefetch objects from disk to memory, and employ a healthy file system block size to make prefetches efficient. Flow control and network round trip time will combine to eliminate disk performance as a major performance factor for caching. Hence, we believe that object cache performance will depend on raw processor speed. As several researchers have demonstrated 100-megabit TCP/IP bandwidths on current processors, we believe that a single cache processor at an ENSS can be designed to meet current demand, and scale to meet future demand.

4.2 Consistency

We argued that caching improves object consistency by reducing the need to replicate archived objects, but how should we keep caches consistent? We suggest using a hybrid approach of time-to-live caching, modeled after the Domain Name System (DNS), and version checking. Upon faulting an object into a cache, the cache assigns it a time-to-live. If the cache faulted the object from another cache, it copies the other cache's time-to-live. If a referenced, cache-resident object's time-to-live is expired, the cache must first connect to the object's source host and either fetch a fresh copy of the object or confirm that it has not been modified. A user's client should, optionally, be able to retrieve the object directly from its source.

4.3 Topology

Caches must choose whether to retrieve an object directly from the object's source or via another cache. For concreteness of discussion, we assume that caches are placed at most regional networks where they meet the NSFNET backbone and at most stub networks where they meet their regional. We propose that clients find their stub network cache through the Domain Name System and apply the simple rule that, if the source is not on the same network as the client, they issue the request through the stub cache.

Stub caches, like clients themselves, must choose whether to retrieve an object directly or from a regional or stub cache. One possible solution would be to query the DNS for the stub cache of the object's source and then query this cache for its regional cache. Given this information, many different cache location policies could be implemented.

4.4 Privacy and Security

This paper has implicitly assumed that object caches, like networks and name servers, are managed by trusted agencies. However, people concerned that caching could make their private objects visible to the rest of the network simply need not retrieve their objects through the caches. Moreover, digital signatures could be used to seal data, to guard against cached copies being modified without their approval.

5 Related Work

This paper deals both with the traffic characterization of the Internet and, to some extent, with caching for distributed file systems. It is closely related to a study by Maffeis [Maf93] and to the Alex file system [Cat92].

Maffeis [Maf93] analyzed the log files of 25 FTP archives. He reported the degree of read-sharing and touched on the update rate of files in FTP archives. He reported that certain files, primarily "ls-lR" and "README" files, are frequently updated and that the number of files in FTP archives tend to grow by 3% a month. However, he did not give sufficient detail to help evaluate cache consistency policies. He suggested that object caching would improve availability and performance, but did not suggest an architecture or evaluate the impact of caching on network load.

A type of file cache for anonymous FTP already exists. Alex [Cat92] is a recently developed system that wraps a Network File System (NFS) interface around the space of anonymous FTP archives. One uses Alex by *mounting* the Alex server as an NFS volume. The Alex server, in turn, translates name requests to FTP operations, caching recently retrieved files. The Alex server disk cache fills up with popular items. Note that Alex is not a distributed architecture, and its dependence on NFS makes it less portable than other new Internet services. Although its impact on network traffic has not been evaluated, this could be substantial if Alex servers were placed at ENSS's and were in common use.

The Australian archive server *archie.au* implements a file cache based on the Prospero file system [Neu92]. Australian users retrieve files through this server to amortize bandwidth on the Australian long-haul links. Unfortunately, if people outside of Australia access this archive, files not in the cache can be transferred across the link twice: once to fill the cache and once to deliver it to the requester.

5.1 File System Caches

Studies of caching for general purpose, distributed file systems have evaluated various file caching architectures, although all of these studies were driven by traces of local-area network, file system traffic rather than wide-area network, FTP archive traffic.

Both the Sprite and Andrew operating system projects [NWO88, HKM+88] optimized their file systems for caching of *read*-shared files, such as program binaries. They both reported little *write* sharing between workstations, although they took different approaches to write-shared files.

Muntz and Honeyman [MH92] and Blaze and Alonso [AB92] simulated multi-level caching architectures driven by traces taken from over a hundred workstations running the Andrew File System at DEC's Systems Research Center. While Muntz and Honeyman found "disappointingly low" hit rates, Blaze and Alonso reported that caching could reduce file server traffic by a factor of two or more, and thought that a hierarchical set of caches could reduce load by an order of magnitude. Blaze and Alonso performed this study in light of the consistency advantages of caching rather than replicating data across hundreds of thousands of computers. It is encouraging to note that their study, based on local-area traces, and our study, based on wide-area traces, reach similar conclusions.

5.2 Internet Traffic Characterization

A number of recent studies have monitored wide-area networks to characterize IP, TCP, and application traffic. Measurement studies of IP packet statistics have documented the growth in traffic volume and troubles with existing protocols [BCCP93]. Another series of studies focused on models of Internet traffic [DJC$^+$92, Hei90, WLC92, Pax91]. Mukjerjee [Muk92] conducted "ping" experiments to study the dynamics of network congestion and round trip times. We traced Domain Name System traffic and showed that 3% of NSFNET bytes are due to defective implementations of the DNS [DOK92]. NSF measures and reports monthly traffic statistics of the NSFNET backbone and maintains an extensive FTP archive of packet counts, link loads, and packet delay measurements [Mer92].

6 Conclusions

This paper presented evidence that a hierarchical file caching mechanism layered atop FTP would eliminate 21% of the traffic that currently traverses the NSFNET backbone and our regional networks. Simultaneously, these caches would lead to improved consistency of FTP archives. We also estimated that automatic compression could reduce backbone traffic volume by another 6%.[2]

While we collected data from only a single regional network and grant that additional data could make the predicted savings due to file caching go up or down a little, we believe our simulations. A separate analysis of two weeks of University of Colorado FTP traffic shows a similar degree of savings [EHS92].

While it could be argued that the volume of file exchange traffic will pale compared to upcoming audio and video traffic, we note that people will undoubtedly FTP recordings of video and audio conferences. This means that the size of shared files will continue to grow as the volume of multimedia traffic increases. Hence, bulk transfer of shared objects will remain an important contributor to network utilization.

More generally, we believe file transfer will continue to be an important support mechanism even as new network services are introduced. As evidence, we note that although many new Internet services have been introduced and traffic has been growing exponentially in the past few years, the proportion of packets and bytes caused by FTP have fluctuated by only a few percent, averaging about 25% and 48[Mer92]. Table 6 shows that FTP is already used extensively for distributing data for new types of services, such as video and audio.

Finally, file caching is cost effective. For the price of some inexpensive caching machines we could eliminate a fifth of NSFNET traffic, or a similar amount of traffic on a regional network. For example, one could purchase a caching machine for $5,500, as compared with paying $1,500 *monthly* for an additional T1 regional link.

For these reasons we believe an architecture of anonymous object caches, accessed by "universal resource locators" [Int93], would be useful to services other than FTP. We hope to deploy a prototype of such a caching architecture.

[2] Adding compression to NNTP[KL86] and SMTP[Pos82] could reduce backbone traffic by another 6%.

Appendix: Traffic By File Type

Table 6 lists common file types by percentage of bandwidth used, based on an analysis of file names. This analysis is interesting, as it indicates the types of applications and users that make up the bulk of FTP traffic.

We constructed this table by first stripping off file naming suffixes (such as ".Z") that concern presentation transformations (such as compression, ASCII encoding, etc.). We then separated the file names into conceptual categories, based on approximately 250 different common naming conventions (such as .jpeg to indicate the Joint Photographic Experts Group graphical image format). We did so iteratively, until no large transfers for files that we could identify remained.

Acknowledgements

We would like to thank the National Science Foundation, Merit, Inc., and Advanced Network and Services, Inc., for giving us permission to collect the data needed for this study. We thank Paul Hyder and Don Morris of the National Center for Atmospheric Research for assisting us in connecting to the NCAR network. We thank Carol Ward of Westnet for explaining the nature of the connectivity of our access point. We thank Tom Dunkenberger of Sprintlink for providing estimates of network link costs. We thank Mark Knopper of Merit, Inc., for his feedback on an earlier version of this paper.

We thank Phil Karn of Qualcomm, Inc., for suggesting the idea of an on-demand distribution and caching mechanism for Internet file transfers.

Danzig was supported for this work in part by the Air Force Office of Scientific Research, under Award Number F49620-93-1-0082. Hall and Schwartz were supported for this work in part by the National Science Foundation under grant numbers NCR-9105372 and NCR-9204853, and by a grant from Sun Microsystems' Collaborative Research Program.

Figure 2 is reprinted by permission of Merit, Inc.

David Ewing helped implement the software used to collect and analyze the trace data.

References

[AB92] R. Alonso and M. Blaze. Dynamic hierarchical caching for large-scale distributed file systems. *Proceedings of the Twelvth International Conference on Distributed Computing Systems*, June 1992.

[BCCP93] Hans-Werner Braun, Bilal Chinoy, Kimberly C. Claffy, and George C. Polyzos. Analysis and modeling of wide-area networks: Annual status report. Technical report, Applied Networking Research, San Diego Supercomputer Center, February 13, 1993.

[Cat92] V. Cate. Alex - a global filesystem. *Proceedings of the Usenix File Systems Workshop*, pages 1–11, May 1992.

[CM91] David Curry and Jeff Mogul. NFSwatch 3.1 Software. Anonymous FTP from gatekeeper.dec.com: pub/net/ip/nfs/nfswatch3.1.tar.Z, January, 1991.

[DJC+92] Peter B. Danzig, Sugih Jamin, Ramon Caceres, Danny J. Mitzel, and Deborah Estrin. An artificial workload model of TCP/IP internetworks. *Journal of Internetworking: Practice and Experience*, 3(1):1–26, March 1992.

[DJCM91] Peter B. Danzig, Sugih Jamin, Ramon Caceres, and Danny Mitzel. Characteristics of wide-area TCP/IP conversations. *ACM SIGCOMM 91 Conference*, September, 1991.

[DOK92] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An analysis of wide-area name server traffic: A study of the Domain Name System. *ACM SIGCOMM 92 Conference*, pages 281–292, August 1992.

[ED92] Alan Emtage and Peter Deutsch. archie: An electronic directory service for the Internet. *Proceedings of the*

Percent by Bandwidth Consumed	Average File Size [kbytes]	Example File Naming Convention (case insensitive)	Probable Meaning of files
20.13	591	.jpeg, .mpeg, .gif, ...	Graphics, video, and other image data
19.82	611	.zoo, .zip, .ljh, ...	IBM PC files
7.52	963	.dat, .d, .db, ...	Binary data
5.57	4,130	.o, .sun4, .sparc, ...	UNIX executable code
5.10	419	.c, .h, .for, ...	Source code
2.73	324	.hqx, .sit, .sit_bin, ...	Macintosh files
2.23	143	.asc, .txt, .doc, ...	ASCII text
1.03	75	readme, index, .list, ...	Descriptions of directory contents
0.78	197	.ps, .postscript, .dvi, ...	Formatted output
0.63	553	.au, .snd, .sound, ...	Audio data
0.54	96	.ms, .tex, .tbl, ...	Word Processing files
0.09	674	.next, next., _next, ...	NeXT files
0.01	164	.vms, vms., .vax, ...	Vax files
33.82	-	-	Unable to determine meaning

Table 6: FTP traffic breakdown by file type.

Winter 1992 Usenix Conference, January 1992.

[EHS92] David J. Ewing, Richard S. Hall, and Michael F. Schwartz. A measurement study of Internet file transfer traffic. Technical Report CU-CS 571-92, University of Colorado, Boulder, January, 1992.

[Hei90] Steven A. Heimlich. Traffic characterization of the NSFNET national backbone. *Proceedings Winter USENIX Conference*, pages 207–227, January 1990.

[HKM+88] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, January, 1988.

[HS93] D. Hardy and M. F. Schwartz. Essence: A resource discovery system based on semantic file indexing. *Proceedings of the USENIX Winter Conference*, pages 361–374, January 1993.

[Int93] Internet Engineering Task Force. Uniform resource locators. Anonymous FTP from cnri.reston.va.us: /ietf.mail.archives/uri/, January, 1993.

[KL86] Brian Kontor and Phil Lapsley. Network news transfer protocol. RFC 977, 1986.

[KM91] B. Kahle and A. Medlar. An information system for corporate users: Wide Area In formation Servers. *ConneXions - The Interoperability Report*, 5(11):2–9, November 1991.

[Maf93] S. Maffeis. File access patterns in public FTP archives and a index for locality of reference. *ACM SIGMETRICS Performance Evaluation Review*, 20(3):22–35, March 1993.

[McL91] Lee McLoughlin. FTP mirroring software. Anonymous FTP from src.doc.ic.ac.uk: package/mirror.shar, August 1991.

[Mer92] Merit/NSFnet Information Services. *NSFNET Statistics*, October 1992. Anonymous FTP from nis.nsf.net:statistics/nsfnet/*.

[MH92] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems - or - your cache ain't nuthin' but trash. *Proceedings of the USENIX Winter Conference*, pages 305–313, January 1992.

[Muk92] Amarnath Mukherjee. On the dynamics and significance of low frequency components of internet load. Technical Report MIS-CIS-92-83, University of Pennsylvania, December, 1992.

[Neu92] B. Clifford Neuman. Prospero: A tool for organizing Internet resources. *Electronic Networking: Research, Applications and Policy*, 2(1):30–37, Spring 1992.

[NWO88] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the sprite network file system. *ACM Transactions on Computer Systems*, 6(1):134–154, January, 1988.

[Pax91] Vern Paxson. Measurements and models of wide-area TCP conversations. Technical Report TR LBL-30840, Lawrence Berkeley Labs, 1991.

[Pos82] J. B. Postel. RFC 821: Simple Mail Transfer Protocol. Technical report, University of Southern California Information Sciences Institute, August 1982.

[PR85] J. Postel and J. Reynolds. RFC 959: File transfer protocol (FTP). Technical report, University of Southern California Information Sciences Institute, October 1985.

[Wel84] T. A. Welch. A technique for high performance data compression. *IEEE Computer Magazine*, 17(6):8–19, June 1984.

[WLC92] Ian Wakeman, Dave Lewis, and Jon Crowcroft. Traffic analysis of trans-atlantic traffic. *Proceedings of INET 92*, pages 417–430, June 1992.