USENIX Association

Proceedings of the General Track: 2004 USENIX Annual Technical Conference

Boston, MA, USA
June 27–July 2, 2004

Reliability and Security in the CoDeeN Content Distribution Network

Limin Wang*, KyoungSoo Park, Ruoming Pang, Vivek Pai and Larry Peterson
Department of Computer Science
Princeton University

Abstract

With the advent of large-scale, wide-area networking testbeds, researchers can deploy long-running distributed services that interact with other resources on the Web. The CoDeeN Content Distribution Network, deployed on PlanetLab, uses a network of caching Web proxy servers to intelligently distribute and cache requests from a potentially large client population. We have been running this system nearly continuously since June 2003, allowing open access from any client in the world. In that time, it has become the most heavily-used long-running service on PlanetLab, handling over four million accesses per day. In this paper, we discuss the design of our system, focusing on the reliability and security mechanisms that have kept the service in operation.

Our reliability mechanisms assess node health, preventing failing nodes from disrupting the operation of the overall system. Our security mechanisms protect nodes from being exploited and from being implicated in malicious activities, problems that commonly plague other open proxies. We believe that future services, especially peer-to-peer systems, will require similar mechanisms as more services are deployed on non-dedicated distributed systems, and as their interaction with existing protocols and systems increases. Our experiences with CoDeeN and our data on its availability should serve as an important starting point for designers of future systems.

1 Introduction

The recent development of Internet-scale network testbeds, such as PlanetLab, enables researchers to develop and deploy large-scale, wide-area network projects subjected to real traffic conditions. Previously, such systems have either been commercial enterprises (e.g., content distribution networks, or CDNs), or have been community-focused distributed projects (e.g., free file-sharing networks). If we define a design space of latency versus throughput and tightly-controlled versus decentralized management, we can see that existing CDNs and file-sharing services occupy three portions of the space. The remaining portion, latency-sensitive decentralized systems, remains more elusive, without an easily-identifiable representative. In this paper, we describe CoDeeN, an academic Content Distribution Network deployed on PlanetLab, that uses a decentralized design to address a latency-sensitive problem.

To reduce access latency, content distribution networks use geographically distributed server surrogates, which cache content from the origin servers, and request redirectors, which send client requests to the surrogates. Commercial CDNs [2, 23] replicate pages from content providers and direct clients to the surrogates via custom DNS servers often coupled with URL rewriting by the content providers. The infrastructure for these systems is usually reverse-mode proxy caches with custom logic that interprets rewritten URLs. This approach is transparent to the end user, since content providers make the necessary changes to utilize the reverse proxies.

Our academic testbed CDN, CoDeeN, also uses caching proxy servers, but due to its non-commercial nature, engages clients instead of content providers. Clients must currently specify a CoDeeN proxy in their browser settings, which makes the system demand-driven, and allows us to capture more information on client access behavior. Given the high degree of infrastructural overlap, our future work may include support for non-commercial content providers, or even allowing PlanetLab members to automatically send their HTTP traffic to CoDeeN by using transparent proxying.

As shown in Figure 1, a CoDeeN instance consists of a proxy operating in both forward and reverse modes, as well as the redirection logic and monitoring infrastructure. When a client sends requests to a CoDeeN proxy, the node acts as a forward proxy and tries to satisfy the requests locally. Cache misses are handled by the redirector to determine where the request should be sent, which is generally another CoDeeN node acting as the reverse proxy for the origin server. For most requests, the redirector considers request locality, system load, reliability, and proximity when selecting another CoDeeN node. The reliability and security mechanisms can exclude nodes from being candidates, and can also reject requests entirely for various reasons described later.

*current contact: Dept of EECS, Case Western Reserve University

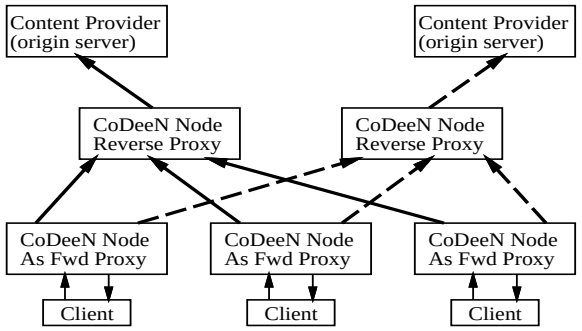


Figure 1: CoDeeN architecture – Clients configure their browsers to use a CoDeeN node, which acts as a forward-mode proxy. Cache misses are deterministically hashed and redirected to another CoDeeN proxy, which acts as a reverse-mode proxy, concentrating requests for a particular URL. In this way, fewer requests are forwarded to the origin site.

Although some previous research has simulated caching in decentralized/peer-to-peer systems [13, 26], we believe that CoDeeN is the first deployed system, and one key insight in this endeavor has been the observation that practical reliability is more difficult to capture than traditional fail-stop models assume. In our experience, running CoDeeN on a small number of PlanetLab nodes was simple, but overall system reliability degraded significantly as nodes were added. CoDeeN now runs on over 100 nodes, and we have found that the status of these proxy nodes are much more dynamic and unpredictable than we had originally expected. Even accounting for the expected problems, such as network disconnections and bandwidth contention, did not improve the situation. In many cases, we found CoDeeN unsuccessfully competing with other PlanetLab projects for system resources, leading to undesirable behavior.

The other challenging aspect of CoDeeN's design, from a management standpoint, is the decision to allow all nodes to act as "open" proxies, accepting requests from any client in the world instead of just those at organizations hosting PlanetLab nodes. This decision makes the system more useful and increases the amount of traffic we receive, but the possibility of abuse also increases the chances that CoDeeN becomes unavailable due to nodes being disconnected. However, we overestimated how long it would take for others to discover our system and underestimated the scope of activities for which people seek open proxies. Within days of CoDeeN becoming stable enough to stay continuously running, the PlanetLab administrators began receiving complaints regarding spam, theft of service, abetting identity theft, etc.

After fixing the discovered security-related problems, CoDeeN has been running nearly continuously since June 2003. In that time, it has received over 300 million requests from over 500,000 unique IP addresses (as of December 2003), while generating only three complaints. Node failure and overload are automatically detected and the monitoring routines provide useful information regarding both CoDeeN and PlanetLab. We believe our techniques have broader application, ranging from peer-to-peer systems to general-purpose monitoring services. Obvious beneficiaries include people deploying open proxies for some form of public good, such as sharing/tolerating load spikes, avoiding censorship, or providing community caching. Since ISPs generally employ transparent proxies, our techniques would allow them to identify customers abusing other systems before receiving complaints from the victims. We believe that any distributed system, especially those that are latency-sensitive or that run on non-dedicated environments, can benefit from our infrastructure for monitoring and avoidance.

The rest of the paper is organized as follows. In Section 2, we discuss system reliability and CoDeeN's monitoring facilities. We discuss the security problems facing CoDeeN in Section 3, followed by our remedies in Section 4. We then show some preliminary findings based on the data we collected and discuss the related work.

2 Reliability and Monitoring

Unlike commercial CDNs, CoDeeN does not operate on dedicated nodes with reliable resources, nor does it employ a centralized Network Operations Center (NOC) to collect and distribute status information. CoDeeN runs on all academic PlanetLab sites in North America, and, as a result, shares resources with other experiments.[1] Such sharing can lead to resource exhaustion (disk space, global file table entries, physical memory) as well as contention (network bandwidth, CPU cycles). In such cases, a CoDeeN instance may be unable to service requests, which would normally lead to overall service degradation or failure. Therefore, to maintain reliable and smooth operations on CoDeeN, each instance monitors system health and provides this data to its local request redirector.

In a latency-sensitive environment such as CoDeeN, avoiding problematic nodes, even if they (eventually) produce a correct result, is preferable to incurring reliability-induced delays. Even a seemingly harmless activity such as a TCP SYN retransmit increases user-perceived latency, reducing the system's overall utility. For CoDeeN to operate smoothly, our distributed redirectors need to continually know the state of other proxies and decide which reverse proxies should be used for request redirection. In practice, what this entails is first finding a healthy subset of the proxies and then letting the redirection strategy decide which one is the best. As a result, CoDeeN includes significant node health monitoring facilities, much of which is not specific to CoDeeN and can be used in other latency-sensitive peer-to-peer environments.

[1] Resource protection in future PlanetLab kernels will mitigate some problems, but this feature may not exist on non-PlanetLab systems.

Two alternatives to active monitoring and avoidance, using retry/failover or multiple simultaneous requests, are not appropriate for this environment. Retrying failed requests requires that failure has already occurred, which implies latency before the retry. We have observed failures where the outbound connection from the reverse proxy makes no progress. In this situation, the forward proxy has no information on whether the request has been sent to the origin server. The problem in this scenario is the same reason why multiple simultaneous requests are not used – the idempotency of an HTTP request can not be determined *a priori*. Some requests, such as queries with a question mark in the URL, are generally assumed to be non-idempotent and uncacheable. However, the CGI mechanism also allows the query portion of the request to be concatenated to the URL as any other URL component. For example, the URL "/directory/program/query" may also be represented as "/directory/program?query". As a result, sending multiple parallel requests and waiting for the fastest answer can cause errors.

The success of distributed monitoring and its effectiveness in avoiding problems depends on the relative difference in time between service failures and monitoring frequency. Our measurements indicate that most failures in CoDeeN are much longer than the monitoring frequency, and that short failures, while numerous, can be avoided by maintaining a recent history of peer nodes. The research challenge here is to devise effective distributed monitoring facilities that help to avoid service disruption and improve system response latency. Our design uses heartbeat messages combined with other tests to estimate which other nodes are healthy and therefore worth using.

2.1 Local Monitoring

Local monitoring gathers information about the CoDeeN instance's state and its host environment, to assess resource contention as well as external service availability. Resource contention arises from competition from other processes on a node, as well as incomplete resource isolation. External services, such as DNS, can become unavailable for reasons not related to PlanetLab.

We believe that the monitoring mechanisms we employ on PlanetLab may be useful in other contexts, particularly for home users joining large peer-to-peer projects. Most PlanetLab nodes tend to host a small number of active experiments/projects at any given time. PlanetLab uses *vservers*, which provide a view-isolated environment with a private root filesystem and security context, but no other resource isolation. While this system falls short of true virtual machines, it is better than what can be expected on other non-dedicated systems, such as multi-tasking home systems. External factors may also be involved in affecting service health. For example, a site's DNS server failure can disrupt the CoDeeN instance, and most of these problems appear to be external to PlanetLab [17].

The local monitor examines the service's primary resources, such as free file descriptors/sockets, CPU cycles, and DNS resolver service. Non-critical information includes system load averages, node and proxy uptimes, traffic rates (classified by origin and request type), and free disk space. Some failure modes were determined by experience – when other experiments consumed all available sockets, not only could the local node not tell that others were unable to contact it, but incoming requests appeared to be indefinitely queued inside the kernel, rather than reporting failure to the requester.

Values available from the operating system/utilities include node uptime, system load averages (both via "/proc"), and system CPU usage (via "vmstat"). Uptime is read at startup and updated inside CoDeeN, while load averages are read every 30 seconds. Processor time spent inside the OS is queried every 30 seconds, and the 3-minute maximum is kept. Using the maximum over 3 minutes reduces fluctuations, and, at 100 nodes, exceeds the gap between successive heartbeats (described below) from any other node. We avoid any node reporting more than 95% system CPU time, since we have found it correlates with kernel/scheduler problems. While some applications do spend much time in the OS, few spend more than 90%, and 95% generally seems failure-induced.

Other values, such as free descriptors and DNS resolver performance, are obtained via simple tests. We create and destroy 50 unconnected sockets every 2 seconds to test the availability of space in the global file table. At our current traffic levels, 50 sockets are generally sufficient to handle two seconds of service on a single node. Any failures over the past 32 attempts are reported, which causes peers to throttle traffic for roughly one minute to any node likely to fail. Similarly, a separate program periodically calls `gethostbyname()` to exercise the node's DNS resolver. To measure comparable values across nodes, and to reduce off-site lookup traffic, only other (cacheable) PlanetLab node names are queried. Lookups requiring more than 5 seconds are deemed failed, since resolvers default to retrying at 5 seconds. We have observed DNS failures caused by misconfigured "/etc/resolv.conf" files, periodic heavyweight processes running on the name servers, and heavy DNS traffic from other sources.

2.2 Peer Monitoring

To monitor the health and status of its peers, each CoDeeN instance employs two mechanisms – a lightweight UDP-based heartbeat and a "heavier" HTTP/TCP-level "fetch" helper. These mechanisms are described below.

2.2.1 UDP Heartbeat

As part of its tests to avoid unhealthy peers, CoDeeN uses UDP heartbeats as a simple gauge of liveness. UDP has low overhead and can be used when socket exhaustion prevents TCP-based communication. Since it is unreliable, only small amounts of non-critical information are

sent using it, and failure to receive acknowledgements (ACKs) is used to infer packet loss.

Each proxy sends a heartbeat message once per second to one of its peers, which then responds with information about its local state. The piggybacked load information includes the peer's average load, system time CPU, file descriptor availability, proxy and node uptimes, average hourly traffic, and DNS timing/failure statistics. Even at our current size of over 100 nodes, this heartbeat traffic is acceptably small. For larger deployments, we can reduce heartbeat frequency, or we may divide the proxies into smaller groups that only exchange aggregate information across groups.

Heartbeat acknowledgments can get delayed or lost, giving some insight into the current network/node state. We consider acknowledgments received within 3 seconds to be acceptable, while any arriving beyond that are considered "late". The typical inter-node RTT on CoDeeN is less than 100ms, so not receiving an ACK in 3 seconds is abnormal. We maintain information about these late ACKs to distinguish between overloaded peers/links and failed peers/links, for which ACKs are never received.

Several policies determine when missing ACKs are deemed problematic. Any node that does not respond to the most recent ACK is avoided, since it may have just recently died. Using a 5% loss rate as a limit, and understanding the short-term nature of network congestion, we avoid any node missing 2 or more ACKs in the past 32, since that implies a 6% loss rate. However, we consider viable any node that responds to the most recent 12 ACKs, since it has roughly a 54% chance of having 12 consecutive successes with a 5% packet loss rate, and the node is likely to be usable.

By coupling the history of ACKs with their piggybacked local status information, each instance in CoDeeN independently assesses the health of other nodes. This information is used by the redirector to determine which nodes are viable candidates for handling forwarded requests. Additionally, the UDP heartbeat facility has a mechanism by which a node can request a summary of the peer's health assessment. This mechanism is not used in normal operation, but is used for our central reporting system to observe overall trends. For example, by querying all CoDeeN nodes, we can determine which nodes are being avoided and which are viable.

2.2.2 HTTP/TCP Heartbeat

While the UDP-based heartbeat is useful for excluding some nodes, it cannot definitively determine node health, since it cannot test some of the paths that may lead to service failures. For example, we have experienced site administrators port filtering TCP connections, which can lead to UDP packets being exchanged without obstruction, but all TCP connections resulting in failure after failed retransmission attempts.

To augment our simple heartbeat, we also employ a tool to fetch pages over HTTP/TCP using a proxy. This tool, conceptually similar to the "wget" program [10], is instrumented to specify what fails when it cannot retrieve a page within the allotted time. Possible causes include socket allocation failure, slow/failed DNS lookup, incomplete connection setup, and failure to retrieve data from the remote system. The DNS resolver timing measurements from this tool are fed into the instance's local monitoring facilities. Since the fetch tool tests the proxying capabilities of the peers, we must also have "known good" web servers to use as origin servers. For this reason, each CoDeeN instance also includes a dummy web server that generates a noncacheable response page for incoming requests.

The local node picks one of its presumed live peers to act as the origin server, and iterates through all of the possible peers as proxies using the fetch tool. After one iteration, it determines which nodes were unable to serve the requested page. Those nodes are tested to see if they can serve a page from their own dummy server. These tests indicate whether a peer has global connectivity or any TCP-level connectivity at all.

Over time, all CoDeeN nodes will act as an origin server and a test proxy for this testing. We keep a history of the failed fetches for each peer, and combine this with the UDP-level heartbeats to determine if a node is viable for redirection. To allow for network delays and the possibility of the origin server becoming unavailable during one sweep, a node is considered bad if its failure count exceeds the other nodes by more than two. At current scale, the overhead for this iteration is tolerable. For much larger deployments, a hierarchical structure can limit the number of nodes actively communicating with each other.

2.3 Aggregate Information

Each CoDeeN proxy stores its local monitoring state as well as its peer summary to disk every 30 seconds, allowing offline behavior analysis as well as anomaly detection. The summary is also published and updated automatically on the CoDeeN central status page [16] every five minutes. These logs provide the raw data that we use in our analysis in Section 5. A sample log entry, truncated to fit in the column, is shown in Figure 2.

Most of the fields are the measurements that have been mentioned earlier, and the columns in the tabular output represent data about the other nodes in CoDeeN. Values in these lines are usually the counts in base-32 format, where 'w' represents 32. The exception is SysMxCPU, which is the percentage value divided by 10 and rounded up. Based on collected information through UDP heartbeat and HTTP tests, each redirector decides the "Liveness" for each CoDeeN node, indicating whether the local node considers that peer node to be viable.

In this particular example, this node is avoiding six of its peers, mostly because they have missed several UDP

```
FdTstHst: 0x0
ProxUptm: 36707
NodeUptm: 111788
LoadAvgs: 0.18 0.24 0.33
ReqsHrly: 5234 3950 0 788 1004 275 2616
DNSFails: 0.00
DNSTimes: 2.48
SysPtCPU: 2 2 1 3 2 4

Liveness: ..X.. ..X.. ..... .X.XX ..... ...X. .....
MissAcks: 10w00 00001 00000 0w066 00010 000v0 00020
LateAcks: 00000 00000 00000 00000 00000 00000 00000
NoFdAcks: 00000 00000 00000 00000 00000 00000 00000
VersProb: 00000 00000 00000 00000 00000 00000 00000
MaxLoads: 41022 11111 11141 20344 11514 14204 11111
SysMxCPU: 81011 11111 11151 10656 11615 15564 11111
WgetProx: 00w00 00100 00010 0w110 00000 000s0 00010
WgetTarg: 11w11 10301 01021 1w220 00111 101t0 11121
```

Figure 2: Sample monitoring log entry

ACKs. The eighth node, highlighted in boldface, is being avoided because it has a WgetTarg count of 3, indicating that it has failed the HTTP fetch test (with itself as the target) three times out of the past 32. More analysis on the statistics for node avoidance is presented in Section 5.

3 Security Problems

To make the system more useful and increase the amount of traffic we receive, we allow all CoDeeN nodes to act as "open" proxies, accepting requests from any client in the world. However, this choice also opens the doors to many security problems. In this section, we discuss some of the problems we encountered during the early development and testing of CoDeeN, and the measures we took to deal with these problems. For the purposes of discussion, we have broadly classified the problems into those dealing with spammers, bandwidth consumption, high request rates, content theft, and anonymity, though we realize that some problems can fall into multiple areas.

3.1 Spammers

The conceptually simplest category of CoDeeN abuser is the spammer, though the mechanisms for spamming using a proxy server are different from traditional spamming. We encountered three different approaches – SMTP tunnels, CGI/formmail POST requests, and IRC spamming. These mechanisms exist without the use of proxies, but gain a level of indirection via proxies, complicating investigation. When faced with complaints, the administrators of the affected system must cooperate with the proxy administrators to find the actual spammer's IP address.

SMTP tunnels – Proxies support TCP-level tunneling via the CONNECT method, mostly to support end-to-end SSL behavior when used as firewalls. After the client specifies the remote machine and port number, the proxy creates a new TCP connection and forwards data in both directions. Our nodes disallow tunneling to port 25 (SMTP) to prevent facilitating open relay abuse, but continually receive such requests. The prevalence and magnitude of such attempts is shown in Figure 3. As a test, we directed these requests to local honey-pot SMTP servers.

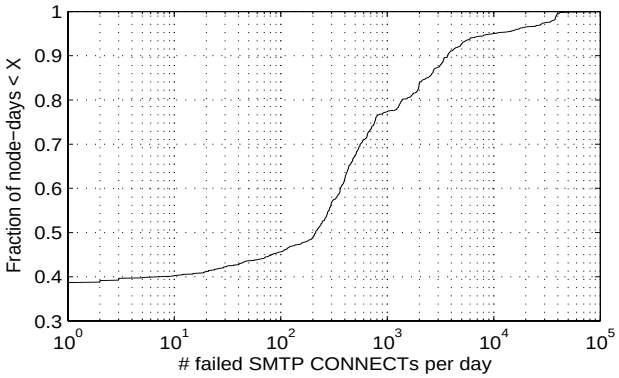


Figure 3: CONNECT activity for 38 nodes – Almost 40% of the samples show no activity, while 20% show over 1000 attempts/day. The maximum seen is over 90K attempts to one node in one day.

In one day, one of our nodes captured over 100K spam e-mails destined to 2,000,000 addresses. Another node saw traffic jump from 3,000 failed attempts per day to 30,000 flows in 5 minutes. This increase led to a self-inflicted denial-of-service when the local system administrator saw the activity spike and disconnected the PlanetLab node.

POST/formmail – Some web sites use a CGI program called *formmail* to allow users to mail web-based feedback forms to the site's operators. Unfortunately, these programs often store the destination e-mail address in the form's hidden input, relying on browsers to send along only the e-mail address specified in the form. Spammers abuse those scripts by generating requests with their victims' e-mail addresses as the targets, causing the exploited site to send spam to the victim.

IRC – Spammers target IRC networks due to their weak authentication and their immediate, captive audience. Most proxies allow CONNECTs to ports above the protected port threshold of 1024, which affects IRC with its default port of 6667. IRC operators have developed their own open proxy blacklist [4], which checks IRC participant IP addresses for open proxies. We were alerted that CoDeeN was being used for IRC spamming, and found many of our nodes blacklisted. While the blacklists eliminate the problem for participating IRC networks, the collateral damage can be significant if other sites begin to refuse non-IRC traffic from blacklisted nodes.

3.2 Bandwidth Hogs

CoDeeN is hosted on PlanetLab nodes, with the hosts absorbing the bandwidth costs. Since most nodes are hosted at high-bandwidth universities, they attract people performing bulk data transfers. Due to lack of locality, such transfers provide no benefit to other CoDeeN users – they cause cache pollution and link congestion.

Webcam Trackers – Sites such as SpotLife.com provide a simple means to use digital cameras as auto-updating web cameras. This *subscription-based* service

allows the general public to broadcast their own "webcams". We noticed heavy bandwidth usage of the SpotLife site, with individual IP addresses generating multiple image requests per second, far above the rate limits in the official SpotLife software. SpotLife claims to bundle their software with over 60% of digital cameras, and a community of high-rate downloaders has formed, to SpotLife's consternation. These users clearly have enough bandwidth to access webcams directly, but use CoDeeN to mask their identity.

Cross-Pacific Downloads – CoDeeN nodes in Washington and California received very high bandwidth consumption with both source and destination located along the Eastern rim of Asia. The multi-megabyte downloads appeared to be for movies, though the reason that these clients chose a round-trip access across the Pacific Ocean is still not clear to us. A direct connection would presumably have much lower latency, but we suspect that these clients were banned from these sites, and required high-bandwidth proxies to access them effectively. Given the high international bandwidth costs in Asia, Western US proxies were probably easier to find.

Steganographers – While large cross-Pacific transfers were easy to detect in access logs, others were less obvious. This class had high aggregate traffic, spread across uniformly-sized, sub-megabyte files marked as GIFs and JPEGs. Large images sizes are not uncommon in broadband-rich countries such as South Korea, but some size variation is expected given the unpredictability of image compression. We downloaded a few of these large files and found that they generated only tiny images on-screen. From the URL names, we assume that these files contain parts of movies stuffed inside image files to hide their actual payload. Although there is existing research on steganography [18], we have not found the appropriate decryption tools to confirm our guess.

3.3 High Request Rates

TCP's flow/congestion controls mitigate the damage that bulk transfers have on other CoDeeN users. In contrast, another class of users generated enough requests that we were concerned that CoDeeN might be implicated in a denial-of-service attack.

Password Crackers – We found an alarming number of clients using CoDeeN to launch dictionary attacks on Yahoo, often via multiple CoDeeN nodes. At one point, we were detecting roughly a dozen new clients per day. Since Yahoo can detect multiple failed attempts to a single account, these users try a single password across many accounts. The attacks appear to be for entertainment, since any victim will be random rather than someone known to the attacker. The problem, again, is that the requests appear to come from CoDeeN, and if Yahoo blocks the IP address, then other PlanetLab services are affected.

Google Crawlers – Like password crackers, we found a number of clients performing Google web/image searches on a series of sorted words. These were clearly mechanical processes working from a dictionary, and their requests were evenly spaced in time. We speculate that these clients are trying to populate their own search engines or perhaps build offline copies of Google.

Click-Counters – Ad servers count impressions for revenue purposes, and rarely do we see such accesses not tied to actual page views. The one exception we have seen is a game site called OutWar.com. Points are obtained when people click on a player's "special link", which delivers a Web page containing ad images. The system apparently counts hits of the player's link instead of ad views, which seems to invite abuse. We have noticed a steady stream of small requests for these links, presumably from players inflating their scores.

3.4 Content Theft

The most worrisome abuse we witnessed on CoDeeN was what we considered the most sophisticated – unauthorized downloading of licensed content.

Licensed Content Theft – Universities purchase *address-authenticated* site licenses for electronic journals, limited to the IP ranges they own. PlanetLab's acceptable use policies disallow accessing these sites, but CoDeeN unintentionally extended this access worldwide. We discovered this problem when a site contacted PlanetLab about suspicious activity. This site had previously experienced a coordinated attack that downloaded 50K articles. Unfortunately, such sites do not handle the *X-Forwarded-For* header that some proxies support to identify the original client IP address. Though this header can be forged, it can be trusted when *denying* access, assuming nobody would forge it to deny themselves access to a site.

Intra-domain Access – Many university Web pages are similarly restricted by IP address, but are scattered within the domain, making them hard to identify. For example, a department's web site may intersperse department-only pages among publicly-accessible pages. Opportunities arise if a node receives a request for a local document, whether that request was received directly or was forwarded by another proxy.

3.5 Anonymity

While some people use proxies for anonymity, some anonymizers accessing CoDeeN caused us some concern. Most added one of more layers of indirection into their activities, complicating abuse tracking.

Request Spreaders – We found that CoDeeN nodes were being advertised on sites that listed open proxies and sold additional software to make testing and using proxies easier. Some sites openly state that open proxies can be used for bulk e-mailing, a euphemism for spam. Many of these sites sell software that spreads requests over a

collection of proxies. Our concern was that this approach could flood a single site from many proxies.

TCP over HTTP – Other request traffic suggested that some sites provided HTTP-to-TCP gateways, named *http2tcp*, presumably to bypass corporate firewalls. Other than a few archived Usenet messages on Google, we have not been able to find more information about this tool.

Non-HTTP Port 80 – While port 80 is normally reserved for HTTP, we also detected CONNECT tunnels via port 80, presumably to communicate between machines without triggering firewalls or intrusion detection systems. However, if someone were creating malformed HTTP requests to attack remote web sites, port 80 tunnels would complicate investigations.

Vulnerability Testing – We found bursts of odd-looking URLs passing through CoDeeN, often having the same URI portion of the URL and different host names. We found lists of such URLs on the Web, designed to remotely test known buffer overflow problems, URL parsing errors, and other security weaknesses. In one instance, these URLs triggered an intrusion detection system, which then identified CoDeeN as the culprit.

4 Protecting CoDeeN

Our guiding principle in developing solutions to address these security problems is to allow users at PlanetLab sites as much access to the Web as they would have without using a proxy, and to allow other users as much "safe" access as possible. To tailor access policies, we classify client IP addresses into three groups – those local to this CoDeeN node, those local to any site hosting a PlanetLab node, and those outside of PlanetLab. Note that our security concerns focus on how we handle possibly malicious client traffic, and not node compromise, which is outside the scope of this paper.

4.1 Rate Limiting

The "outside" clients face the most restrictions on using CoDeeN, limiting request types as well as resource consumption. Only their GET requests are honored, allowing them to download pages and perform simple searches. The POST method, used for forms, is disallowed. Since forms are often used for changing passwords, sending e-mail, and other types of interactions with side-effects, the restriction on POST has the effect of preventing CoDeeN from being implicated in many kinds of damaging Web interactions. For the allowed requests, both request rate and bandwidth are controlled, with measurement performed at multiple scales – the past minute, the past hour, and the past day. Such accounting allows short-term bursts of activity, while keeping the longer-term averages under control. Disallowing POST limits some activities, notably on e-commerce sites that do not use SSL/HTTPS. We are investigating mechanisms to determine which POST actions are reasonably safe, but as more transactions move to secure sites, the motivation for this change diminishes.

To handle overly-aggressive users we needed some mechanism that could quickly be deployed as a stopgap. As a result, we added an explicit blacklist of client IP addresses, which is relatively crude, but effective in handling problematic users. This blacklist was not originally part of the security mechanism, but was developed when dictionary attacks became too frequent. We originally analyzed the access logs and blacklisted clients conducting dictionary attacks, but this approach quickly grew to consume too much administrative attention.

The problem with the dictionary attacks and even the vulnerability tests is that they elude our other tests and can cause problems despite our rate limits. However, both have fairly recognizable characteristics, so we used those properties to build a fairly simple signature detector. Requests with specific signatures are "charged" at a much higher rate than other rate-limited requests. We effectively limit Yahoo login attempts to about 30 per day, frustrating dictionary attacks. We charge vulnerability signatures with a day's worth of traffic, preventing any attempts from being served and banning the user for a day.

Reducing the impact of traffic spreaders is more difficult, but can be handled in various ways. The most lenient approach, allowing any client to use multiple nodes such that the sum does not exceed the request rate, requires much extra communication. A stricter interpretation could specify that no client is allowed to use more than K proxies within a specified time period, and would be more tractable. We opt for a middle ground that provides some protection against abusing multiple proxies.

In CoDeeN, cache misses are handled by two proxies – one acting as the client's forward proxy, and the other as the server's reverse proxy. By recording usage information at both, heavy usage of a single proxy or heavy aggregate use can be detected. We forward client information to the reverse proxies, which can then detect clients using multiple forward proxies. While forwarding queries produces no caching benefit, forwarding them from outside users allows request rate accounting to include this case. So, users attempting to perform Yahoo dictionary attacks (which are query-based) from multiple CoDeeN nodes find that using more nodes does not increase the maximum number of requests allowed. With these changes, login attempts passed to Yahoo have dropped by a factor of 50 even as the number of attackers has tripled.

4.2 Privilege Separation

To address the issue of restricting access to content, we employ privilege separation, which works by observing that when a proxy forwards a request, the request assumes the privilege level of the proxy since it now has the proxy's IP address. Therefore, by carefully controlling which proxies handle requests, appropriate access privileges can be maintained. The ideal solution for protect-

ing licensed content would be to insert an 'X-Forwarded-For' header, but it requires cooperation from the content site – checking whether both the proxy address and forwarded address are authorized. Although this is a simple change, there are some sites that do not handle the header. For such sites, content protection requires CoDeeN to identify what content is licensed and we take an approximate approach. Using Princeton's e-journal subscription list as a starting point, we extracted all host names and pruned them to coalesce similarly-named sites, merging journal1.example.com and journal2.example.com into just example.com. We do not precisely associate subscriptions with universities, since that determination would be constantly-changing and error-prone.

When accessing licensed content, we current only allow requests that preserve privilege. Clients must choose a CoDeeN forward proxy in their own local domain in order to access such content. These *local clients* are assumed to have the same privilege as the CoDeeN forward proxy, so this approach does not create additional exposure risks. These requests are sent directly to the content provider by the forward proxy, since using a reverse proxy would again affect the privilege level. All other client requests for licensed content currently receive error messages. Whether the local client can ultimately access the site is then a decision that the content provider makes using the CoDeeN node's IP address. Though we cannot guarantee the completeness of the subscription list, in practice this approach appears to work well. We have seen requests rejected by this filter, and we have not received any other complaints from content providers. In the future, when dealing with accesses to licensed sites, we may redirect clients from other CoDeeN sites to their local proxies, and direct all "outside" clients to CoDeeN proxies at sites without any subscriptions.

A trickier situation occurs when restricted content is hosted in the same domain as a CoDeeN node, such as when part of a university's Web site is restricted to only those within the university. Protecting these pages from outside exposure cannot use the coarse-grained blacklisting approach suitable for licensed content. Otherwise, entire university sites and departments would become inaccessible. To address this problem, we preserve the privilege of local clients, and de-escalate the privilege of remote clients. We determine if a request to example.edu originates locally at example.edu, and if so, the request is handled directly by the CoDeeN forward proxy. Otherwise, the request is forwarded to a CoDeeN node at another site, and thereby gets its privilege level dropped to that of the remote site through this "bouncing" process. To eliminate the exposure caused by forwarding a request to a site where it is local, we modify our forwarding logic – no request is forwarded to a CoDeeN proxy that has the same domain as the requested content.

Since our security mechanisms depend on comparing host names, we also disallow "outside" accesses to machines identified only by IP addresses. After implementing this approach, we found that some requests using numerical IP addresses were still being accepted. In the HTTP protocol, proxies receive requests that can contain a full URL, with host name, as the first request line. Additional header lines will also identify the host by name. We found some requests were arriving with differing information in the first line and in the Host header. We had not observed that behavior in any Web browser, so we assume such requests were custom-generated, and modified our redirector to reject such abnormal requests.

4.3 Effectiveness of the Solutions

We have received a handful of queries/complaints from system administrators at the local PlanetLab sites, and all but one have been false alarms. Most queries have been caused by system administrators or others using/testing the proxies, surfing through them, and then concluding that they are open proxies.

We have been using CoDeeN daily, and have found that the security restrictions have few effects for local users. Using non-Princeton nodes as our forward proxy, we have found that the restrictions on licensed sites can be overly strict at times. We expect that in the future, when we bounce such requests to completely unprivileged proxies, the special handling for those sites will not be noticeable. These bounced requests will obtain the privilege level of those proxies (i.e., no subscriptions), and will be able to access unrestricted portions of those sites. By changing the configuration information, we have also been able to use CoDeeN as an outside user would see it. Even on our high-speed links, the request rates limits have not impacted our daily browsing.

Restricting outside users from using POST does not appear to cause significant problems in daily use. Searches are commonly handled using the GET method instead of the POST method, and many logins are being handled via HTTPS/SSL, which bypasses the proxy. The most noticeable restrictions on outsiders using POST has been the search function on Amazon.com and some chat rooms. Over two months, local users have generated fewer than 300 POST requests, with the heaviest generator being software update checkers from Apple and Microsoft.

Our security measures have caused some confusion amongst malicious users, and they could not figure out whether or not CoDeeN is a network of real "open" proxies. We routinely observe clients testing proxies and then generating requests at very high rates, sometimes exceeding 50K reqs/hour. However, rarely do CoDeeN nodes see more than 20K valid reqs/hour. Some clients have generated over a million unsuccessful requests in stretches lasting longer than a day.

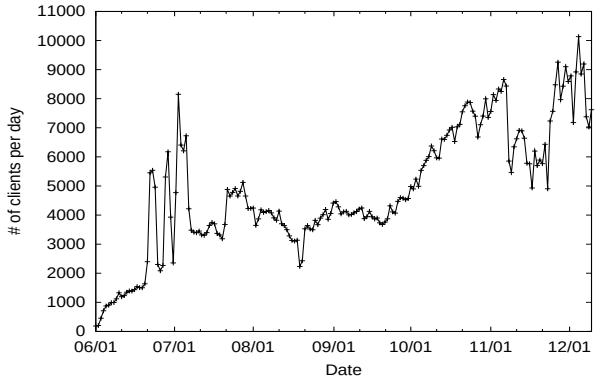


Figure 4: Daily Client Population (Unique IP) on CoDeeN

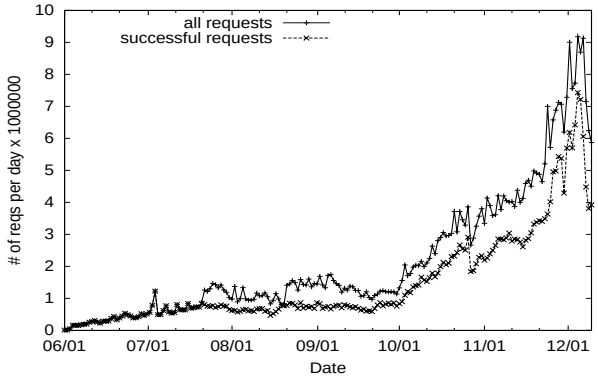


Figure 5: Daily Requests Received on CoDeeN

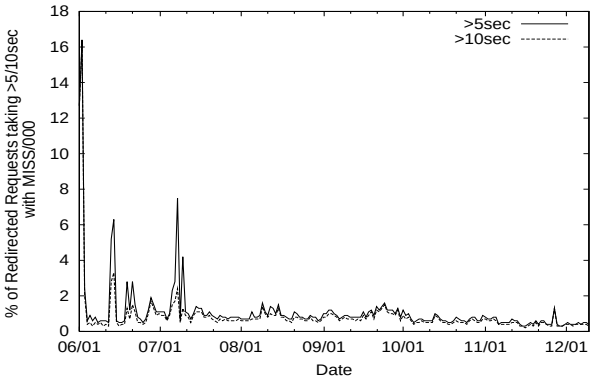


Figure 6: Percentage of Non-serviced Redirected Requests

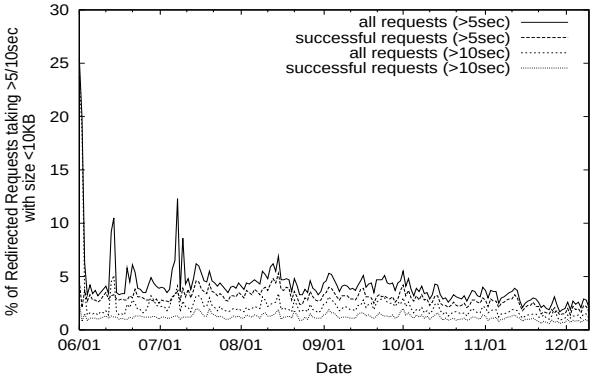


Figure 7: Percentage of Redirected Requests ($<$ 10KB)

5 Results

In this section, we analyze the data we collected during six months of CoDeeN's operation. These results not only show the status of CoDeeN over time, but also provide insights into the monitoring and security measures.

5.1 Traffic

Since starting our public beta test at the end of May, the number of unique IP addresses used to access CoDeeN has passed 500,000, with the daily values shown in Figure 4. Some of these clients appear to be human, while others are programs that interact with proxies. Now, our daily traffic regularly exceeds 7,000 unique IPs.

The daily traffic served by CoDeeN now hovers above more than 4 million requests, and peaks at 7 million, as seen in Figure 5. The total count of daily requests, including those that are rejected, is approaching 7 million per day and peaks at 9 million. We began logging rejected requests in late July, so earlier figures are not available.

5.2 Response Performance

Since reliability has been one of the main thrusts of our current work, the response time behavior of CoDeeN is largely a function of how well the system performs in avoiding bad nodes. In the future, we may work towards optimizing response time by improving the redirector logic, but that has not been our focus to date.

The results of our efforts to detect/avoid bad nodes can be seen in Figure 6, which shows requests that did not receive any service within specific time intervals. When this occurs, the client is likely to stop the connection or visit another page, yielding an easily-identifiable access log entry (MISS/000). These failures can be the result of the origin server being slow or a failure within CoDeeN. The trend shows that both the magnitude and frequency of the failure spikes are decreasing over time. Our most recent change, DNS failure detection, was added in late August, and appears to have yielded positive results.

Since we cannot "normalize" the traffic over CoDeeN, other measurements are noisier, but also instructive. Figure 7 shows the fraction of small/failed responses that take more than a specific amount of time. Here, we only show redirected requests, which means they are not serviced from the forward proxy cache. By focusing on small responses, we can remove the effects of slow clients downloading large files. We see a similar trend where the failure rate decreases over time. The actual overall response

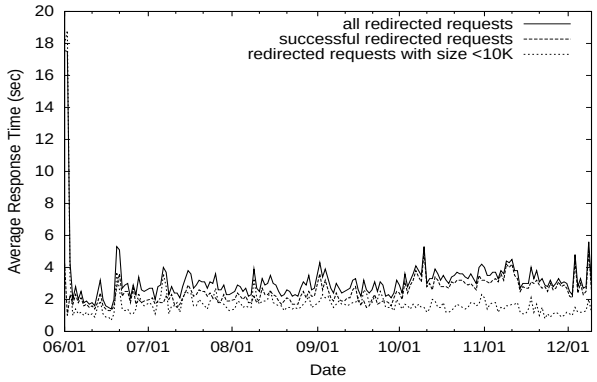


Figure 8: Average Response Time of Redirected Requests

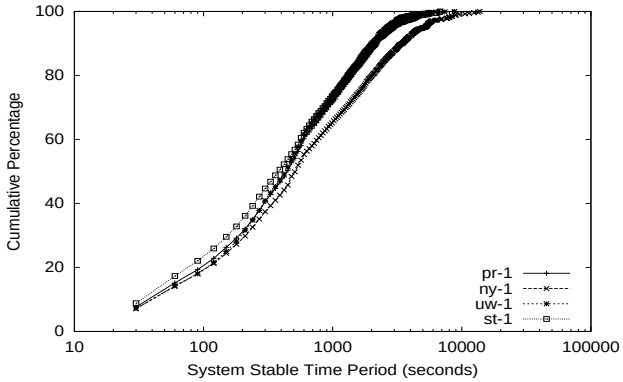


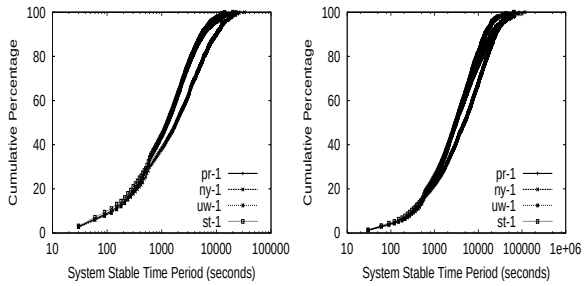
Figure 9: System Stability View from Individual Proxies

times for successful requests, shown in Figure 8, has a less interesting profile. After a problematic beginning, responses have been relatively smooth. As seen from Figure 5, since the beginning of October, we have received a rapidly increasing number of requests on CoDeeN, and consequently, the average response time for all requests slightly increases over time. However, the average response time for small files is steady and keeps decreasing. This result is not surprising, since we have focused on reducing failures rather than reducing success latency.

5.3 Node Stability

The distributed node health monitoring system employed by CoDeeN, described in Section 2.2, provides data about the dynamics of the system and insight into the suitability of our choices regarding monitoring. One would expect that if the system is extremely stable and has few status changes, an active monitoring facility may not be very critical and probably just increases overhead. Conversely, if most failures are short, then avoidance is pointless since the health data is too stale to be useful. Also, the rate of status changes can guide the decisions regarding peer group size upper bounds, since larger groups will require more frequent monitoring to maintain tolerable staleness.

Our measurements confirm our earlier hypothesis about the importance of taking a monitoring and avoidance approach. They show that our system exhibits fairly dynamic liveness behavior. Avoiding bad peers is essential and most failure time is in long failures so avoidance is an effective strategy. Figure 9 depicts the stability of the CoDeeN system with 40 proxies from four of our CoDeeN redirectors' local views. We consider the system to be stable if the status of all 40 nodes is unchanged between two monitoring intervals. We exclude the cases where the observer is partitioned and sees no other proxies alive. The x-axis is the stable period length in seconds, and the y-axis is the cumulative percentage of total time. As we can see, these 4 proxies have very similar views. For about 8% of the time, the liveness status of all proxies changes



(a) Divided into 2 Groups (b) Divided into 4 Groups

Figure 10: System stability for smaller groups

every 30 seconds (our measurement interval). In Table 1, we show the 50^{th} and the 90^{th} percentiles of the stable periods. For 50% of time, the liveness status of the system changes at least once every 6-7 minutes. For 90% of time, the longest stable period is about 20-30 minutes. It shows that in general, the system is quite dynamic – more than what one would expect from few joins/exits.

The tradeoff between peer group size and stability is an open area for research, and our data suggests, quite naturally, that stability increases as group size shrinks. The converse, that large groups become less stable, implies that large-scale peer-to-peer systems will need to sacrifice latency (via multiple hops) for stability. To measure the stability of smaller groups, we divide the 40 proxies into 2 groups of 20 and then 4 groups of 10 and measure group-wide stability. The results are shown in Figure 10 and also in Table 1. As we can see, with smaller groups, the stability improves with longer stable periods for both the 50^{th} and 90^{th} percentiles.

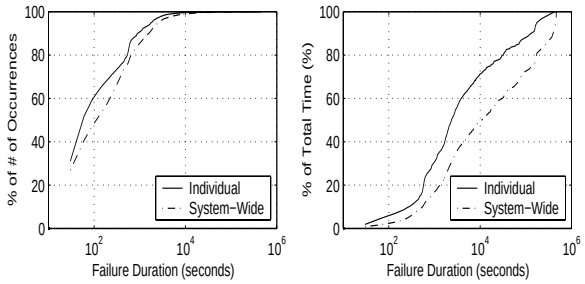
The effectiveness of monitoring-based avoidance depends on the node failure duration. To investigate this issue, we calculate node avoidance duration as seen by each node and as seen by the sum of all nodes. The distribution of these values is shown in Figure 11, where "Individ-

	40-node		2 × 20-node		4 × 10-node	
	50%	90%	50%	90%	50%	90%
pr-1	445	2224	1345	6069	3267	22752
ny-1	512	3451	1837	10020	4804	25099
uw-1	431	2085	1279	5324	3071	19579
st-1	381	2052	1256	5436	3008	14334

Table 1: System Stable Time Period (Seconds)

Site	Fetch	Miss ACKs	Node Down	Late ACKs	DNS
pr-1	6.2	18.3	29.6	13.6	32.1
ny-1	4.7	16.1	31.7	14.0	33.9
uw-1	10.4	16.8	30.0	12.8	29.7
st-1	5.0	14.7	27.2	15.4	34.3

Table 2: Average Percentage of Reasons to Avoid A Node



(a) CDF by # of Occurrences

(b) CDF by Total Time

Figure 11: Node Failure Duration Distribution. Failures spanning across a system-wide downtime are excluded from this measurement, so that it only includes *individual* node failures. Also, due to the interval of node monitoring, it may take up to 40 seconds for a node to be probed by another nodes, thus failures that last a shorter time might be neglected.

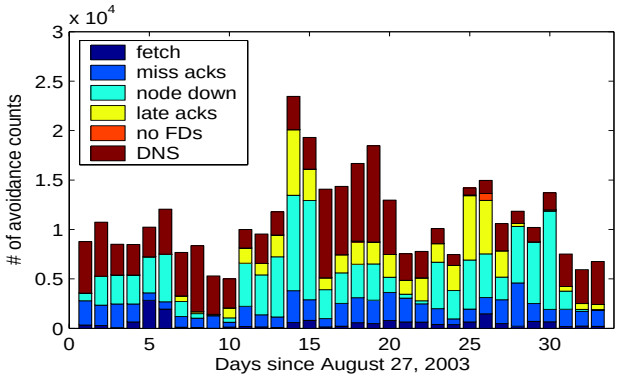


Figure 12: Daily counts of avoidance on ny-1 proxy

ual" represents the distribution as seen by each node, and "System-Wide" counts a node as failed if all nodes see it as failed. By examining the durations of individual failure intervals, shown in Figure 11a, we see that most failures are short, and last less than 100 seconds. Only about 10% of all failures last for 1000 seconds or more. Figure 11b shows the failures in terms of their contribution to the total amount of time spent in failures. Here, we see that these small failures are relatively insignificant – failures less than 100 seconds represent 2% of the total time, and even those less than 1000 seconds are only 30% of the total. These measurements suggest that node monitoring can successfully avoid the most problematic nodes.

5.4 Reasons to Avoid a Node

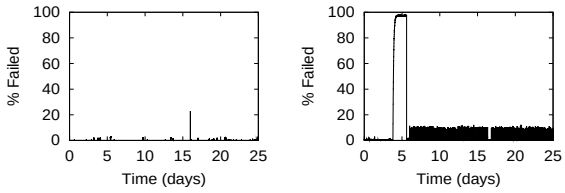
Similar to other research on peer-to-peer systems, we initially assumed that churn, the act of nodes joining and leaving the system, would be the underlying cause of staleness-related failures. However, as can be seen from the stability results, failure occurs at a much greater rate than churn. To investigate the root causes, we gather the logs from 4 of redirectors and investigate what causes nodes to switch from viable to avoided. Therefore, our counts also take time into account, and a long node failure receives more weight. We present each reason category with a non-negligible percentage in Table 2. We find that the underlying cause is roughly common across nodes – mainly dominated by DNS-related avoidance and many nodes down for long periods, followed by missed ACKs. Even simple overload, in the form of late ACKs, is a significant driver of avoidance. Finally, the HTTP fetch helper process can detect TCP-level or application-level connectivity problems.

In terms of design, these measurements show that a UDP-only heartbeat mechanism will significantly underperform our more sophisticated detection. Not only are the multiple schemes useful, but they are complementary. Variation occurs not only across nodes, but also within a node over a span of multiple days. The data for the ny-1 node, calculated on a daily basis, is shown in Figure 12.

5.5 DNS behaviors

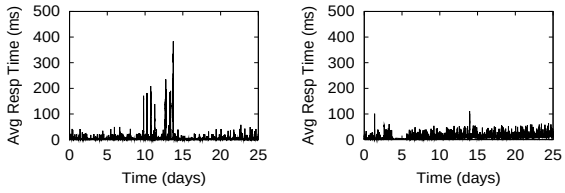
As described earlier, during our HTTP fetch tests, we measure the time of local DNS lookups. When local name servers are having problems, DNS lookups can take many seconds to finish, despite usually taking only a few milliseconds. We further investigate how DNS lookups behave on each proxy by looking at DNS failure rates and average response time for successful queries. If a DNS lookup takes longer than 5 seconds, we regard it as a DNS failure, since this value is the resolver's default timeout.

Figures 13 and 14 show the DNS failure rates and DNS average lookup time for successful queries on 2 of our sampling proxies, ny-1 on east coast and st-1 on west coast. DNS lookup time is usually short (generally well below 50ms), but there are spikes of 50-100ms. Recall that these lookups are only for the controlled set of the intra-CoDeeN "fetch" lookups. Since these mappings are stable, well-advertised, and cacheable, responses should be fast for well-behaved name servers. Anything more than tens of milliseconds implies the local nameservers



(a) ny-1 proxy (b) st-1 proxy

Figure 13: DNS lookup failure at different proxies



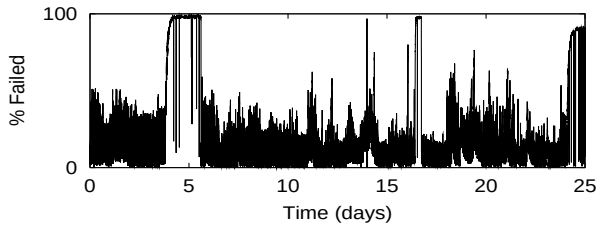
(a) ny-1 proxy (b) st-1 proxy

Figure 14: DNS lookup time at different proxies

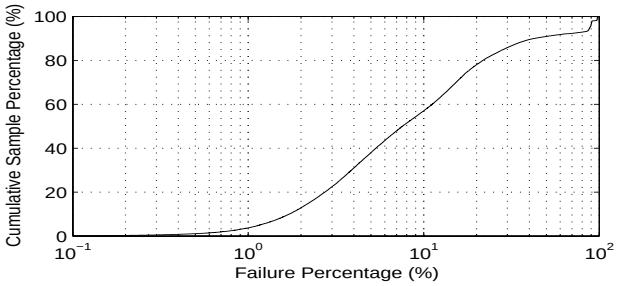
are having problems. These statistics also help to reveal some major problems. For example, the st-1 proxy has a period of 100% DNS failure rate, which is due to the name server disappearing. The problem was resolved when the node's resolv.conf file was manually modified to point to working name servers.

Though DNS failure rates on individual proxies are relatively low, the combined impact of DNS failures on web content retrieval is alarming. Downloading a common web page often involves fetching the attached objects such as images, and the corresponding requests can be forwarded to different proxies. Supposing an HTTP session involves 20 proxies, Figure 15 shows the probability of incurring at least one DNS failure in the session. From the cumulative distribution we can see that for more than 40% of time we have DNS failure probability of at least 10%, which would lead to a pretty unpleasant surfing experience if we did not avoid nodes with DNS problems. Note that these problems often appear to stem from factors beyond our control – Figure 16 shows a DNS nameserver exhibiting periodic failure spikes. Such spikes are common across many nameservers, and we believe that they reflect a cron-initiated process running on the nameserver.

To avoid such problems, we have taken two approaches to reduce the impact of DNS lookups in CoDeeN. The first is a change in redirector policy that is intended to send all requests from a single page to the same reverse proxy node. If a request contains a valid "referer" header, it is used in the hashing process instead of the URL. If no such header exists, the last component of the URL is omitted when hashing. Both of these techniques will tend to send all requests from a single page to the same reverse proxy.



(a) Aggregate DNS failure rate over 25 days



(b) Cumulative distribution DNS failure rate

Figure 15: DNS failure rate of 20 nodes, i.e. the probability of at least one node having DNS difficulty. The abnormal peak around day 5 in (a) is caused by the same peak in Figure 13(b). Thus when computing the cumulative distribution in (b) we only considered the last 15 days.

Not only will this result in fewer DNS lookups, but it will also exploit persistent connections in HTTP. The second modification to reduce the problems stemming from DNS is a middleware DNS brokering service we have developed, called CoDNS. This layer can mask local DNS resolver failure by using remote resolvers, and is described elsewhere [17].

5.6 Requests Rejected for Security Reasons

In Section 3, we explored the idea of rejecting requests that could cause security problems or abuse system resources. Figure 17 shows a snapshot of the statistics about various reasons for rejecting requests. Three major reasons include clients exceeding the maximum rate, requests using methods other than GET and requests with no host field, indicating non-standard browsers. Most of the time, these three comprise more than 80% of the rejected traffic. The query count represents the number of bandwidth capped CGI queries which include all sorts of malicious behaviors previously mentioned. Disallowed CONNECTs and POSTs indicate attempts to send spam through our system. CONNECTs alone constitute, on the average, over 5% and sometimes 30% of all rejected requests. From this graph, we can get an idea of how many scavenging attempts are being made through the open proxies like CoDeeN.

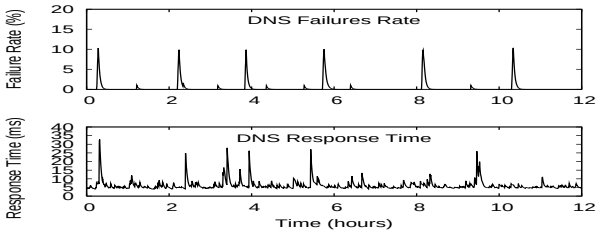


Figure 16: DNS failures, response times for the Stanford proxy

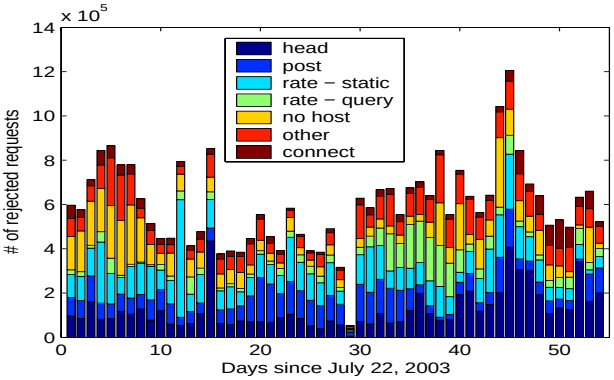


Figure 17: Daily counts of rejected requests on CoDeeN

We assume most of this traffic is being generated automatically by running some custom programs. We are now studying how to identify these malicious programs versus normal human users and innocuous programs like web crawlers, in order to provide an application-level QoS depending on client classification.

6 Related Work

Similar to CoDeeN, peer-to-peer systems [20, 22, 24] also run in a distributed, unreliable environment. Nodes join or depart the system from time to time, and node failures can also happen often. Besides maintaining a membership directory, these systems typically follow a retry and failover scheme to deal with failing nodes while routing to the destinations. Although practically, these trials can be expected by peer-to-peer system users, the extra delays in retrying different next hops can cause latency problems. For latency-sensitive applications implemented on a peer-to-peer substrate, multiple hops or trials in each operation become even more problematic [7]. The Globe distribution network also leverages hierarchy and location caching to manage mobile objects [3]. To address multiple-hop latency, recent research has started pushing more membership information into each node in a peer-to-peer system to achieve one-hop lookups [12, 21]. In this regard, similar arguments can be made that each node could monitor the status of other nodes.

Some researchers have used Byzantine fault tolerant approaches to provide higher reliability and robustness than fail-stop assumptions provide [1, 5]. While such schemes, including state machine replication in general, may seem appealing for handling failing nodes in CoDeeN, the fact that origin servers are not under our control limits their utility. Since we cannot tell that an access to an origin server is idempotent, we cannot issue multiple simultaneous requests for one object due to the possibility of side-effects. Such an approach could be used among CoDeeN's reverse proxies if the object is known to be cached.

In the cluster environment, systems with a front end [11] can deploy service-specific load monitoring routines in the front end to monitor the status of server farms and decide to avoid failing nodes. These generally operate in a tightly-coupled environment with centralized control. There are also general cluster monitoring facilities that can watch the status of different nodes, such as the Ganglia tools [9], which have already been used on PlanetLab. We can potentially take advantage of Ganglia to collect system level information. However, we are also interested in application-level metrics such as HTTP/TCP connectivity, and some of resources such as DNS behaviors that are not monitored by Ganglia.

Cooperative proxy cache schemes have been previously studied in the literature [6, 19, 25, 27], and CoDeeN shares many similar goals. However, to the best of our knowledge, the only two deployed systems have used the Harvest-like approach with proxy cache hierarchies. The main differences between CoDeeN and these systems are in the scale, the nature of who can access, and the type of service provided. Neither system uses open proxies. The NLANR Global Caching Hierarchy [15] operates ten proxy caches that only accept requests from other proxies and one end-user proxy cache that allows password-based access after registration. The JANET Web Cache Service [14] consists of 17 proxies in England, all of which are accessible only to other proxies. Joining the system requires providing your own proxy, registering, and using an access control list to specify which sites should not be forwarded to other caches. Entries on this list include electronic journals.

A new Akamai-like system, CoralCDN [8], is in the process of being deployed. Bad nodes are avoided by DNS-based redirection, sometimes using an explicit UDP RPC for status checking.

7 Conclusion

In this paper, we present our experience with a continuously running prototype CDN on PlanetLab. We describe our reliability mechanisms that assess node health and prevent failing nodes from disrupting the operation of the overall system. We also discuss our security mechanisms that protect nodes from being exploited and from

being implicated in malicious activities. The intentional dual use of CoDeeN both as a CDN and as an open proxy network and the resource competition on PlanetLab nodes make it a very valuable testbed. We believe that future services, especially peer-to-peer systems, will require similar mechanisms as more services are deployed on non-dedicated distributed systems, and as their interaction with existing protocols and systems increases.

Our distributed monitoring facilities prove to be effective at detecting and thus avoiding failing or problematic nodes. The net benefit is robustness against component disruptions and improved response latency. Although some of the aspects of these facilities seem application-specific, they are not confined to CDN services. Other latency-sensitive services running in a non-dedicated distributed environment can potentially benefit from them, since they also need to do extra reliability checks. Our experiences also reveal that reliability-induced problems occur almost two orders of magnitude more frequently than node joins/leaves, which makes active monitoring necessary and important for other systems such as peer-to-peer.

Our security measures consist of classification, rate limiting, and privilege separation. They provide a model for other Web-accessible services. For example, some of the security mechanisms we are developing are suitable for ISPs to deploy on their own networks to detect misbehaving customers before problems arise. Other systems that allow open access to Web resources may face similar situations, and may be able to adopt similar mechanisms.

Our experiences with CoDeeN and the data we have obtained on availability can serve as a starting point for designers of future systems. We demonstrate that effective monitoring is critical for system proper operation, and security measures are important for preventing the system from being abused.

Acknowledgments

This research is supported in part by DARPA contract F30602–00–2–0561. We thank our shepherd, Atul Adya, for his guidance and helpful input. We also thank our anonymous reviewers for their valuable comments on improving this paper.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[2] Akamai. Content Delivery Network. http://www.akamai.com.

[3] A. Baggio, G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Efficient tracking of mobile objects in Globe. *The Computer Journal*, 44(5):340–353, 2001.

[4] BOPM. Blitzed Open Proxy Monitor. http://www.blitzed.org/bopm/.

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999.

[6] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.

[7] R. Cox, A. Muthitacharoen, and R. Morris. Serving dns using chord. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[8] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In *Proceedings of the 1st Symposium on Networked System Design and Implementation (NSDI '04)*, 2004.

[9] Ganglia. http://ganglia.sourceforge.net.

[10] GNU wget. http://www.gnu.org/software/wget/wget.html.

[11] G. Goldszmidt and G. Hunt. Netdispatcher: A tcp connection router. IBM Research White Paper.

[12] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003.

[13] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC)*, 2002.

[14] JANET Web Cache Service. http://wwwcache.ja.net.

[15] National Laboratory for Applied Network Research (NLANR). Ircache project. http://www.ircache.net/.

[16] Network Systems Group, Princeton University. CoDeeN—A CDN on PlanetLab. http://codeen.cs.princeton.edu.

[17] K. Park, Z. Wang, V. Pai, and L. Peterson. CoDNS: Masking DNS delays via cooperative lookups. Technical Report TR-690-04, Princeton University Computer Science Department, Feb. 2004.

[18] N. Provos and P. Honeyman. Detecting steganographic content on the internet. In *ISOC NDSS'02*, Feb. 2002.

[19] M. Rabinovich, J. Chase, and S. Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. *Computer Networks and ISDN Systems*, 30(22–23):2253–2259, 1998.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM'01*, Aug. 2001.

[21] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system. In *Tenth ACM SIGOPS European Workshop*, 2002.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[23] Speedera. http://www.speedera.com.

[24] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, San Diego, California, Aug. 2001.

[25] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay. Design considerations for distributed caching on the internet. In *International Conference on Distributed Computing Systems*, pages 273–284, 1999.

[26] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirecion on CDN Robustness. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[27] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles*, pages 16–31, 1999.