A P2P-Based Architecture for Secure Software Delivery Using Volunteer Assistance

Purvi Shah, Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204-3010
{purvi, paris}@cs.uh.edu

Jeffrey Morgan, John Schettino,
Chandrasekar Venkatraman
Hewlett-Packard Laboratories
Palo Alto, CA, 94304-1126
{jeff.morgan, john.schettino, chandra}@hp.com

Abstract

We present a content delivery infrastructure distributing and maintaining software packages in a large organization. Our work based on a trace-based analysis of an existing software delivery system that we conducted to find general principles and properties that could be used to devise a better solution. Our design combines a conventional server with volunteer nodes that expand its scalability. We rely on Peer-to-Peer technology to speed up content synchronization among the volunteer nodes while maintaining a conventional client/server interface for the service customers. Finally our system includes a novel load balancing mechanism that considers both the synchronization workload and the customer-generated workload of the volunteer nodes. Our simulation results indicate that the feedback information currently available at the server/tracker of the P2P system offers enough information to ensure a fair load distribution among the peers.

1. Introduction

Peer-to-Peer (P2P) systems are a natural form of autonomic systems. Member peers offer and receive content from each other in a collaborative environment without distinguished roles as either pure servers or pure customers. They can be programmed to self-organize and self-improve the distributed system. In this paper we apply P2P technology to resolve the scalability problems observed in current techniques used to provide management and maintenance services in enterprise networks.

We present a content delivery network (CDN) that can be used by managed services organizations, on the basis of donated servers, to distribute and maintain software packages. We also introduce a mechanism to optimize load balancing on these donated resources. Our CDN is designed to be used in the real world to improve the downloading rates of the customers.

The main contributions of this work include:

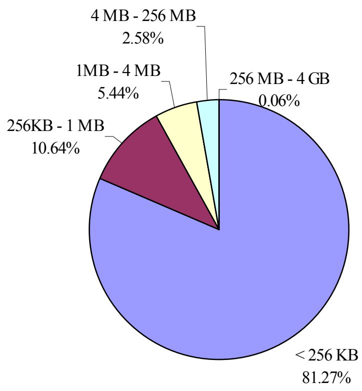

Figure 1. File size distribution in the central software server.

- A trace-based analysis of the current software delivery system to find general principles and properties that could be used as a strategy to devise a better system.
- A scalable CDN architecture for delivering software using volunteer nodes.
- An efficient mechanism for load balancing in the proposed design. This mechanism is instrumental for improving the performance and the fairness of the service.

The remainder of the paper is organized as follows. In the next section, we present our trace analysis. Section 3 introduces our tool to synchronize the content from the server on the edge nodes using P2P delivery. Section 4 starts with a description of the basic design of the CDN and explains its operation using volunteer nodes. Related work is discussed in section 5. Finally concluding remarks are given in section 6.

2. Trace analysis

We began by analyzing ten days worth of logs associated with a software delivery system supporting various Linux installations and distributing their updates in a corporate environment.

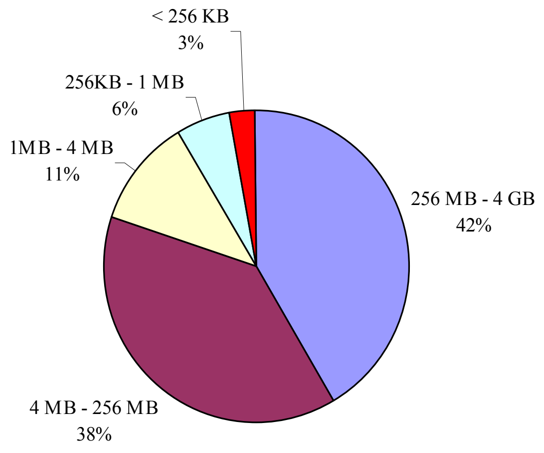

Figure 2. Fractions of space occupied by files of different sizes.

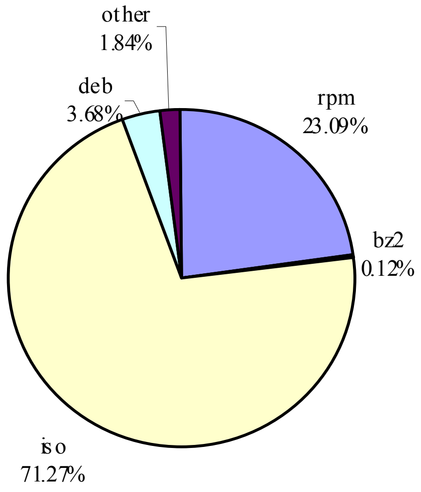

Figure 3. Types of files distributed by the central server.

2.1. Central software repository

This repository served system software and updates for approximately ten different Linux distributions and consisted of roughly 2.2 million files. As we can see on Fig. 1, 81 percent of these files are smaller than 256 KB and 91 percent of them are smaller than 1 MB. In our study, we considered only files that are *software packages,* that is, applications, libraries, software updates and, more generally, any piece of software that is published as an independently named file. The total size of the repository is approximately 2.86 TB. As Fig. 2 shows, files larger than 4 MB occupy 80 percent of the disk space even though these files account for less than 9 percentage of the total number of files.

2.2. Access patterns

As Fig. 3 indicates, image (*.iso* files) downloads comprise 71 percent of the total server workload, compared to the other packages. At the same time, the percentage of downloads for update packages (such as *.rpm* files) is large. As seen in Fig. 4, almost two thirds of the files uploaded by the server are smaller than 256K and 83 percent of them are smaller than 1 MB.

We observed several interesting access patterns such as flash crowds at the time of new package releases. Fig. 5 shows the percentages of the customer requests received each day by the server after the release of an update package over a span of

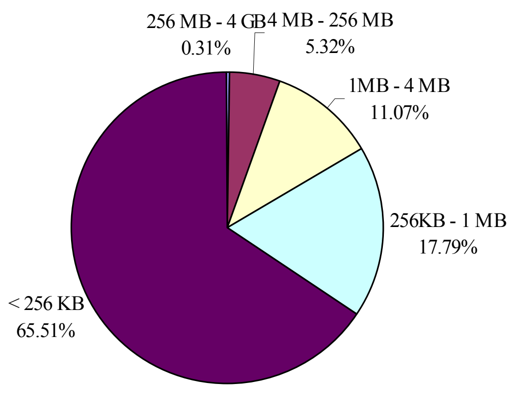

Figure 4. Sizes of files uploaded by the server.

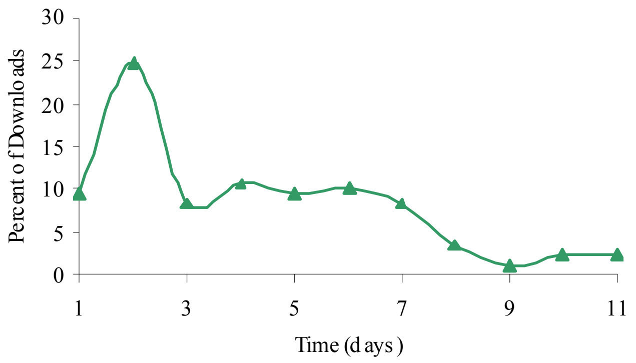

Figure 5. How download request rates for a specific package varies over time since its release day.

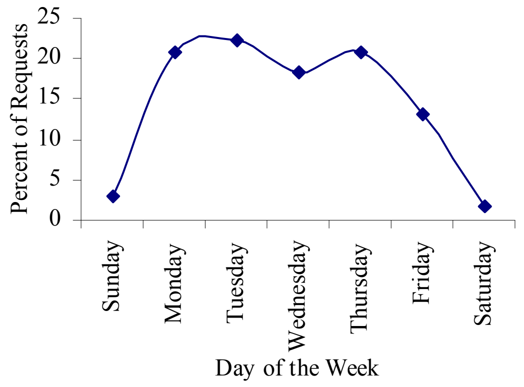

Figure 6. How download request rates vary over the days of the week.

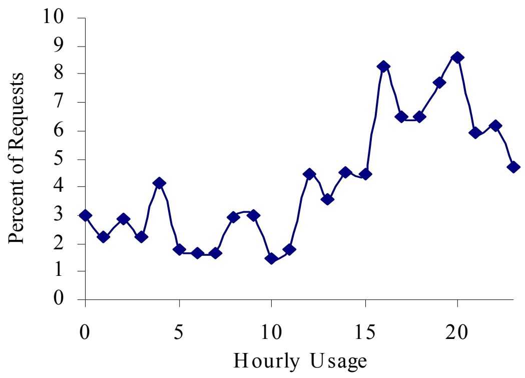

Figure 7. How download request rates vary over the time of day.

ten days. From it we can observe that more than one third of all customer requests for a given package are received before the end of the day following the package release. The number drops by the third day and is further reduced one week after the release. This traffic surge suggests that increasing the server capacity to control flash crowds is not a practical solution.

Fig. 6 shows that the percentage of customer requests during weekdays is much higher than during weekends. Similarly, Fig. 7 shows the percentage of customer requests during evenings is much higher than during the work hours.

2.3. Number of identical files

We found out that 17 percent of files larger than 1 MB were identical and differed from other files only in name. In addition, 17 percent of these identical files were source-code packages by more than one Linux distribution. They comprised up to 85 GB of the disk space and accounted for 30 percent of the identical files in terms of size.

2.4. Similarity

We also looked for similarity among files as they may exist among different versions and variants of the same source-code package. The majority of packages in the repository are compressed files. They are compressed using tools such as *gzip* [12] and *bzip2* [5]. This lack of a standard has some very unfavorable effect on exploiting file similarity using the tools such as *rsync* [20]. Unfortunately the *gzip-rsyncable* patch [13] has not been adopted widely by the software vendors. Our findings in this sub-section provide an incentive to the vendors to adopt **rsync**-friendly compression algorithms.

Fig. 8 suggests that considerable similarity exists among the uncompressed versions of the same software. Exploiting this similarity can considerably reduce the server workload in terms of the data being transferred, that is, the network workload.

We also observed that software has *variants,* that is, different packages for clients with different architectures and operating systems. Since the variants of a single package tend to be updated around the same time, we observed a strong

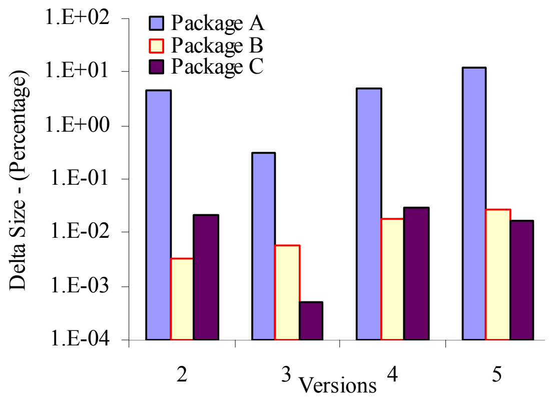

Figure 8. Differences between versions of the same software package.

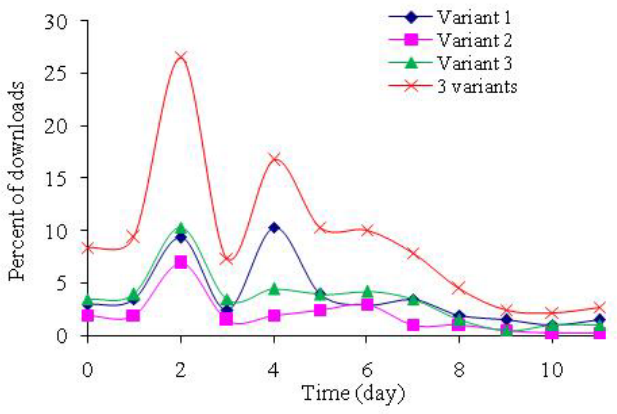

Figure 9. Download rates for different variants of a newly released update package.

correlation among downloads of different variants of the same package (see Fig. 9). Utilizing the similarity among these variants would greatly benefit any P2P solution as peers could find several more potential neighbors and use these neighbors improve their download rates.

2.5. Synchronization workload

We also observed that the various departments within the enterprise manage around forty edge nodes that maintain complete or partial mirrors of the software repository for serving updates to a small set of machines. By studying data traces collected over a period of 35 days by one of these edge nodes we observed that the node spent on average 1.81 hours daily synchronizing its repository with the server. As seen in Fig. 10, the number of hours spent that way varied between a minimum of 0.54 hours and a maximum of 2.53 hours. Fig. 11 shows that the download bandwidth observed by the edge node was approximately 2.25 Mbps even though its download capability was 1 Gbps and the capacity of the narrowest link from the server to the edge node was approximately 155 Mbps.

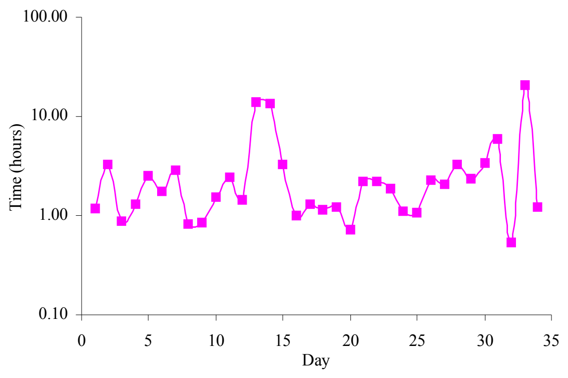

Figure 10. Number of hours spent daily by an edge node synchronizing its software repository.

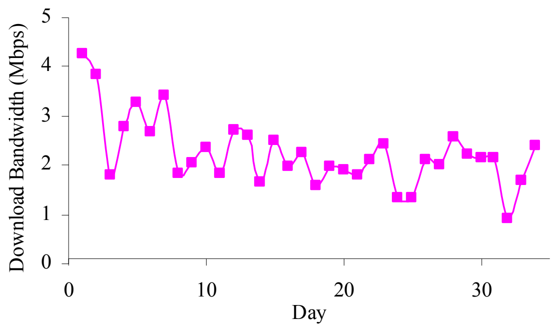

Figure 11. Download bandwidth observed by an edge node synchronizing its software repository.

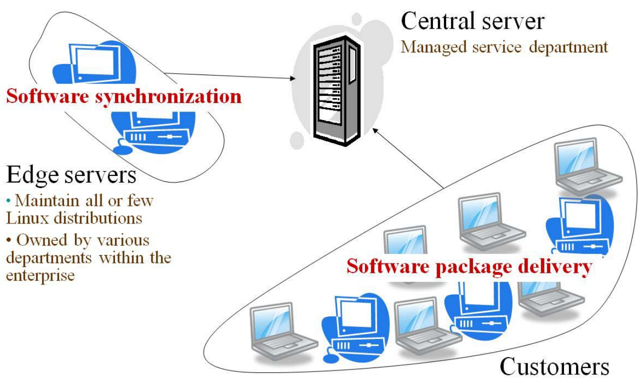
Central server
Managed service department
Software synchronization
Edge servers
• Maintain all or few Linux distributions
• Owned by various departments within the enterprise
Software package delivery
Customers

Figure 12. How the existing system works.

2.6. Summary

In summary, we identified interesting properties of the current software delivery system such as customer request patterns, package attributes and more. Fig. 12 gives an idea of the current system design and its workload. Based on our observations we believe this software delivery system should be optimized for efficient delivery of both small and large sized files. In addition, it should be capable of managing flash crowds and able to exploit identicalness and similarity among files.

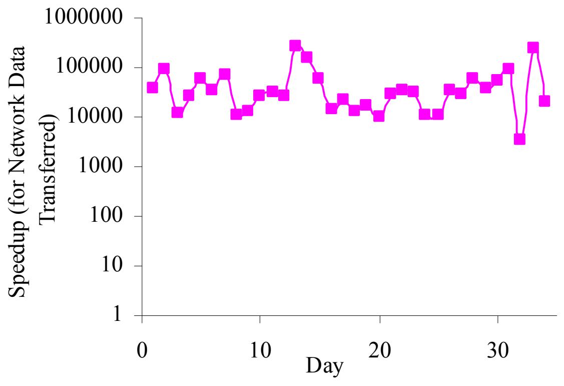

Figure 13. Speedup attained by using *rsync* to synchronize the edge server repository.

3. Software synchronization by the edge nodes

With insights from our trace analysis, we first developed *PRsync*, a tool that uses P2P technology based on *BitTorrent* [6, 4] for synchronizing the repositories on the edge nodes in an efficient fashion. Our tool integrates P2P technology into the current synchronization tool, **rsync**. It is also able to exploit both the presence of identical files and similarities existing among different versions of the same package. This integration was feasible because all edge nodes use the same **rsync** tool to synchronize their repository with the central server.

In its current form, **rsync** synchronizes remote repositories during an interactive network session involving two sites, namely the server and a specific client. Fig. 13 shows the *speedup* (that is, the ratio of the total repository size to actual the data actually transferred) attained when synchronizing the edge server repository using **rsync**.

Using **BitTorrent** among the edge nodes would allow us to further improve the efficiency and scalability of the synchronization system. With our new tool, the server can now reduce the processing on the server repository for multiple edge nodes. This processing includes computing MD4 checksums, and as a result can be compute-intensive for large individual packages. Our tool uses the *torrent metadata files*, which contains the information for each chunk being distributed in the *BitTorrent system* [6]. These files store the pre-computed checksums and help to reduce the server's processing workload. While **rsync** required the server to transmit the data to each edge node in a totally separate fashion, using BitTorrent allows data transfers among the edge nodes themselves.

A major aspect of **PRsync** is that it separates content delivery from synchronization: the first task is shared by the edge nodes and the server while synchronization remains the sole responsibility of the server. This separation removes redundant processing at the server and permits the use of a P2P protocol to deliver the content.

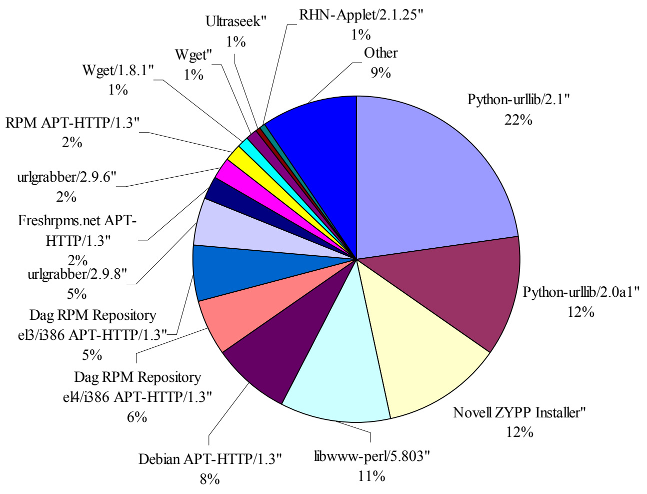

Figure 14. Top 15 applications/tools used by customers.

4. Software delivery to the customers

Next we explored strategies to improve the software delivery. A major problem in integrating pure P2P solutions in this scenario is that customers use different tools. Fig. 14 shows the top 15 of 120 different tools that were used to download packages during our trace data collection period. As we can see, some of these tools are different variants of the same software. A client-transparent solution can achieve a widespread use and is much easier than configuring most P2P systems. Since we are providing a delivery service, we should also avoid consuming customer bandwidth if there is an alternative way to obtain the required bandwidth.

4.1. Software delivery system design

We contacted the administrators of the edge nodes and asked them whether they would be willing to volunteer their nodes for distributing software to machines not under their direct administration. Their acceptance allowed us to propose a better system design.

Fig. 15 illustrates the basic components of our new software delivery system. Customers first send their file requests to the server. Using the data provided by the BitTorrent tracker, the server then redirects the customer request to one of the volunteer nodes[1] hosting the file. The customer downloads the entire file from that volunteer node.

Like existing CDNs our solution does not require custom tools. In addition, all the volunteer nodes maintain their own administrative organization. They still download packages as they do in the current system. In addition, they respond to requests from their peers (via PRSync and BitTorrent) as well

[1] Note that we use the terms *edge nodes*, *volunteer nodes* and *peers* interchangeably in what follows, since edge nodes are volunteered to form a self-organizing P2P system to provide the content delivery service to the customers.

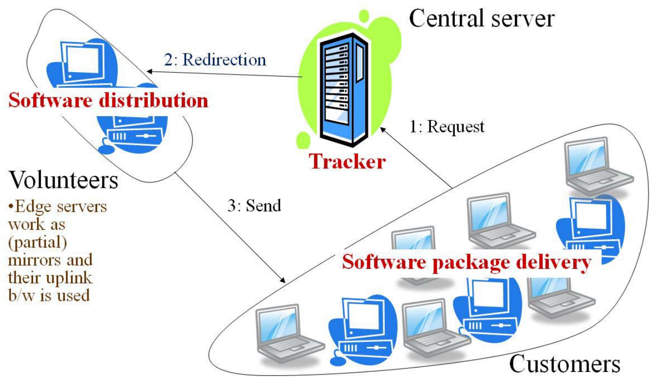
Central server
2: Redirection
Software distribution
1: Request
Tracker
Volunteers
•Edge servers work as (partial) mirrors and their uplink b/w is used
3: Send
Software package delivery
Customers

Figure 15. How our proposed system utilizes volunteer nodes.

as requests from customers (via the *Apache* [3] web server and HTTP). The resultant system is not a pure P2P system, but a client-server system based on P2P technology. Instead of requiring a cluster of dedicated nodes, the server consists of an inexpensive group of volunteer nodes updating themselves through a P2P protocol.

There are several challenges involved in designing such a distributed architecture. While CDNs typically use either dedicated connections or a private high-speed networks for synchronization of their edge nodes, our system uses the same network for both synchronization and delivery tasks. As a result, using a load balancer to distribute the customer requests without considering the synchronization loads of each volunteer cannot guarantee that the loads of our volunteer nodes will be well balanced. Thus we need a load balancing mechanism that can account for both workloads.

Security is another important issue, as we cannot fully trust the volunteer nodes. To address this issue, our system requires that its customers first communicate with the server before contacting any of the volunteer nodes. As a result, the customers will be able to verify the package received from the volunteer nodes through the use of MD4 checksums provided by the server. This mechanism can be implemented as a customer side transparent lightweight proxy that intercepts and verifies the package received from the volunteer node.

In addition, this design enables the server to attempt to optimize the individual workloads of the volunteer nodes. Our first step is to collect information on the volunteer nodes and to find out which volunteer nodes have which files. To know this, we plan to rely on the tracker of the BitTorrent system. This way the load balancer will forward the incoming customer request only to volunteer nodes that have the package.

Recall that a BitTorrent *tracker* is a peer that assists in the communication between peers using the protocol by keeping complete membership information. As peers enter the system, they first connect to the tracker. Peers that have already begun downloading also communicate with the tracker at fixed *update intervals* to learn about new peers and provide statistics.

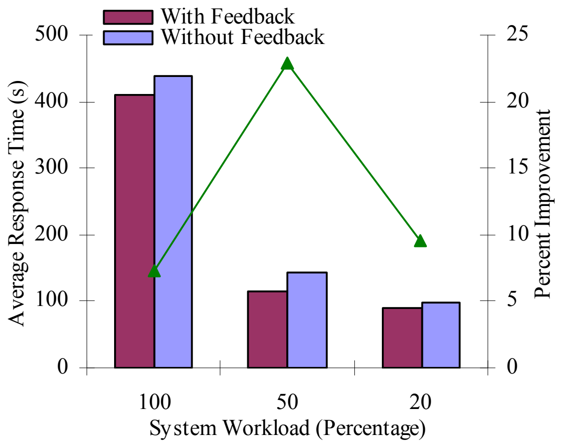

Figure 16. Average response time performance.

To balance the workload among the volunteer nodes we use a *feedback-controlled load balancing mechanism*. We propose to require peers to add information on their current workload to the messages they already send to the tracker. This way we can identify the volunteer nodes that are currently overloaded and redirect fewer customers to such volunteer nodes. Observe that our approach does not result in any increase in the network traffic as the feedback is merely piggybacked to the messages already exchanged between the tracker and the peers.

4.2. Evaluation

In this subsection, we present a simulation study of our proposed feedback-controlled load balancing mechanism. Our simulations are implemented using the JAVA based discrete-event General P2P Simulator (GPS) [23]. GPS models concurrent uploads of the peer under the various algorithms of the BitTorrent protocol and calculates the download rates from its neighbors. This way we can model the synchronization workload. We modified GPS to also model the client-server interactions of our software delivery system.

We modeled the network transmission and queuing delays but assumed that the network propagation delays could be neglected since they are relevant only for small sized control packets. To keep our model simple, we ignored the complexity of the dynamics of TCP connections. We assumed the idealized performance of TCP and assumed that connections traversing a link shared its bandwidth equally. Like previous simulation studies [3, 22], we assumed that bandwidth bottlenecks only occurred at the edge and did not model shared bottleneck links in the interior of the network. This is the same as requiring all the nodes to be in a single Internet region.

We take an algorithm-independent approach in describing a feedback-controlled load balancing mechanism. Nevertheless, the effectiveness of any load balancing mechanism depends on the algorithm used by the load balancer. For our simulations, we used the *request counting* algorithm provided by the **Apache** load balancer [3]. This algorithm counts the number of customer requests and distributes requests across workers until they have each served an equal number of requests.

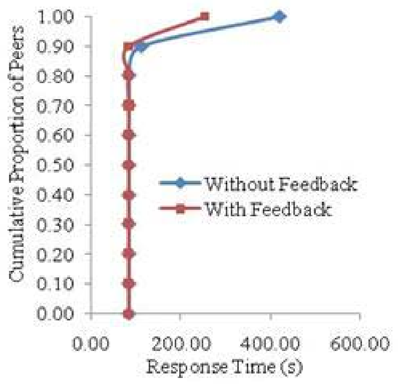

Figure 17(a). Response time performance for a 20% workload.

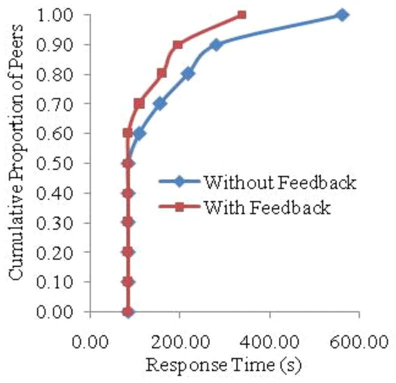

Figure 17(b). Response time performance for a 50% workload.

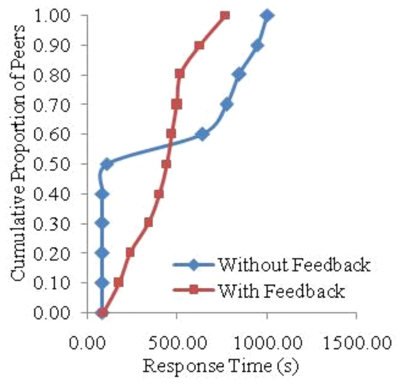

Figure 17(c). Response time performance for a 100% workload.

In this study, we consider both the workload resulting from the synchronization activity of the volunteer nodes and the customer generated workload. This combination of workload introduces some unpredictability in customer response time as it causes a customer request not always represent a fixed duration of work. As a result, uniformly distributing the customer requests will not result in a uniform distribution of the workload among the volunteer peers. We added therefore *feedbacks* that measure number of active connections maintained by each volunteer peer and will be retrieved at each tracker update interval connections. Since these feedbacks take into account the peer synchronization activity, they provide a better tool for dispatching customer requests.

By varying the rate of customer request arrivals, we observed the system performance at different workloads. We ran our experiments for eight volunteer nodes and assumed that all the packages have the same size. To simulate the underlying synchronization workload, we had one half of the volunteer nodes periodically download files. This created a state where one half of the volunteer nodes had a much higher workload than the others because they had to handle at the same time download requests from their customers and chunk exchange request from the other peers.

We evaluate the scalability of the load balancer with and without our mechanism. We want to determine whether or not this mechanism can rebalance the workload on the volunteer nodes and improve performance. We measure the *response time*, that is, the intervals between the submission of a customer request and its completion of the response as seen by the customers. Fig. 16 and 17 show the response times for the customers.

As Fig. 16 shows, our load-balancing scheme improves average customer response times at all system workload levels. Fig. 17 indicates that there is some unevenness in the response times. Without our feedback mechanism the load balancer sometimes overloads some volunteer nodes due to poor load balancing, which results in higher response times and higher unevenness in response times. As Fig. 17c shows, our load balancing mechanism was particularly effective in reducing response time unevenness at a 100 percent workload. Without it, one half of the requests went to the four lightly shared peers while the other requests went to the four heavily loaded peers. As a result, one half of the requests completed within a few seconds while the other requests had to wait between 8 and 15 minutes. Our load balancing mechanism minutes and no request took more than 12 minutes. In addition, our simulations also showed an improvement in the download times of the underlying synchronization activity.

Next we measured the volunteer node *workload*, that is, the volunteer node network usage. Fig. 18 presents the respective workload of the server and the volunteer nodes. The difference in network usage between volunteer nodes when the load balancing mechanism is not used clearly shows that one half of the volunteer nodes are being underutilized while the central server and the other volunteer nodes have a higher workload. The results presented here show a definite improvement in the performance with the addition of our mechanism. The network usage of the volunteer nodes is much closer when the tracker limits the number of customer requests sent to peers that are busy updating their contents.

4.3. Implementation issues

The current web server used at the server and volunteer nodes is the **Apache** web server. The **Apache** distribution contains a number of modules and third party modules can be integrated into it. We are now in the process of modifying the **Apache** load balancer module to provide our proposed feedback-controlled load balancing mechanism at the server. In addition modifications will be made in **PRsync** so that the volunteer nodes can provide their synchronization workload to the tracker, which in turn can be used by the server.

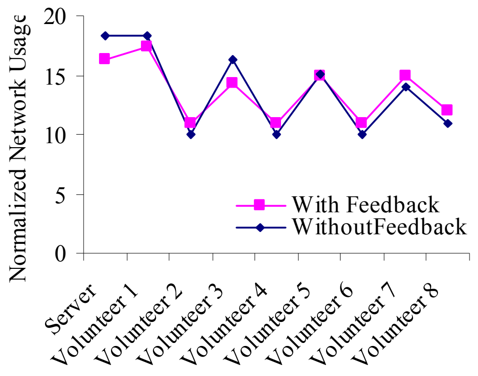

Figure 18(a). Normalized network usage for a 50% workload.

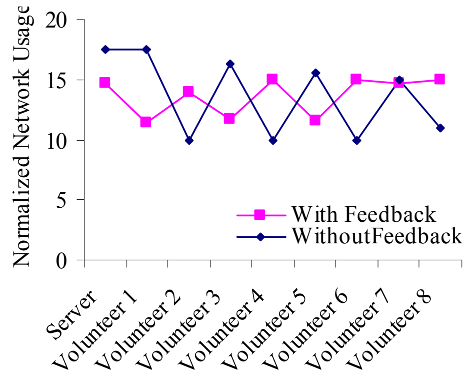

Figure 18(b). Normalized network usage for a 100% workload.

5. Related work

Gkantsidi*s et al.* [10] analyzed traces from a Windows update system. We study a software delivery system supporting system software and updates for more than ten different Linux distributions. We evaluate the current system from a networking point of view and look at ways to improve the current system design.

This work builds on a large amount of effort in several areas: content delivery networks and P2P systems, synchronization software and load balancing. In this section, we briefly consider the work accomplished by the research community on these topics.

5.1. Content delivery networks

Today CDNs are applied to improve performance and scalability of content delivery. While existing CDNs such as *Akamai* [2] and *Digital Island* [7] can provide very large service capacity, the cost of maintaining such CDNs is very high.

Previous work on volunteer computing has shown that distributed computing projects such as *SETI@Home* [1] using donated machines can work as well or even better than the largest supercomputers. While CDNs such as *CORAL* [9] offer an attractive solution, our approach involving volunteer nodes is more economical and has considerably lower administration overheads. At the same time CDNs including donated resources have to address additional security and load balancing issues.

Possibly the work more closely related to our is Pierre and Steen's work [19]. What distinguishes our CDN from their work is that we let volunteer nodes form a self-organizing and self-improving P2P system. We use widely accepted and efficient **BitTorrent** to monitor our volunteer nodes and redirect customer requests based upon their workload.

The deployable version of our CDN will contain tools used by the current software delivery system such as **rsync** and **Apache** and tools that have been built before such as **BitTorrent**. We think that constructing a solution on top of the current system will save replication of some complicated technical effort. Thus we argue this approach has a potential of much further distributed reach.

5.2. Synchronization software

Several synchronization systems have been developed, mainly in the domain of distributed file systems [21] for instance *AFS* and *NFS* where stringent data consistency is of main importance. For efficient broadcasts of the differences between files, *rsync+* [8] can be used.

These systems assume a much tighter pairing of the individual nodes than in our environment, which incorporates edge nodes under different administrative organizations. Moreover, in our environment the repositories are kept autonomously and edge node administrators decide the synchronization.

We believe that the ease of integration and performance benefits make **PRsync** an attractive tool for the edge nodes wishing to synchronize their repository. We decided to use BitTorrent system for P2P distribution because like rsync it distributes files at the chunk level. This feature also allows parallel downloading and is not supported by other P2P protocols such as *Gnutella* [11], *KaZaA* [14] and *Napster* [16]. In addition, BitTorrent systems are self-improving since they use an incentive mechanism to adapt to the network conditions and evolve into a better overlay network. Finally, unlike completely decentralized protocols such as Gnutella, the BitTorrent tracker allows us to improve system performance with some degree of centralization.

5.3. Load balancing

Most CDNs employ their own private high-speed networks. Their load balancing mechanism does not need to be concerned with the internal traffic due to synchronizing the edge nodes. In contrast our system is built over public networks so that traffic among the volunteer nodes has a significant influence on the performance and thus needs to be taken into account.

In previous research, a lot of effort has been spent in the field of load balancing. Possibly the work more closely related to ours is that of Kerr [15]. He proposed using dynamic feedback system to optimize load-balancing decisions. This approach has not been designed for P2P environments. In addition, we exploit the information readily available at the BitTorrent tracker instead of deploying a monitoring system. As a result the overhead of our approach should be minimal.

6. Conclusions

We have presented a content delivery infrastructure that can be used by managed services organizations. We started with a trace-based analysis of an existing software delivery system to find general principles and properties that could be used as a strategy to devise a better solution. Our proposal consists of supplementing a conventional server with volunteer nodes that expand its scalability. Our design is unique in that it combines the concept of volunteering with the P2P technology. We rely on P2P technology to speed up content synchronization among the volunteer nodes while maintaining a conventional client/server interface for the customers of the service. Finally, our system includes a novel load balancing mechanism that considers both the synchronization workload and the customer-generated workload of the volunteer nodes. We show that with a small of modification to the server side software our load balancing mechanism provides better performance.

In the future, we plan to study content placement policies that can handle *volatile* volunteer nodes. This would allow individuals to donate idle machine time to improve the software delivery process. Our trace analysis indicates that by replicating only the newly released packages we can make use of the volatile volunteer nodes to diminish the flash crowd effects.

As the volunteer nodes could be globally distributed, it is desirable to select the volunteer nodes that are near to the customer. Next, we plan to take into account the round trip time as an additional metric when performing load balancing [17]. By means of existing tools such as *ping* or *traceroute* the volunteer node can provide the proximity information between itself and the customers it has served to the BitTorrent tracker periodically. Even after applying this step, the server will still have a choice of few volunteer nodes to pick for a customer via our feedback controlled load balancing mechanism.

So far we have considered exploiting similarity between different versions of a package. Similarity Enhanced Transfer (SET) [18] exploits chunk level similarity in downloading related files. Note their data analysis is fairly different from our work given that much of the similarity comes from files with the same content but with slightly different metadata information in the header. However, we can apply their proposed approach to additionally exploit similarity between different variants of a package. Though to utilize this similarity SET proposes to integrate the chunk level mapping information with the CDN infrastructure.

References

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, *SETI@home: An experiment in public-resource computing*. Communications of the ACM (CACM), 45(11), Nov. 2002.

[2] Akamai white papers. On-line at: *http://www.akamai.com/html/perspectives/whitepapers.html.*

[3] Apache HTTP server documentation, Online at: *http://httpd.apache.org/docs/.*

[3] A. Bharambe, C. Herley and V. Padmanabhan, Analyzing and improving a BitTorrent network's performance mechanisms, *Proc. IEEE Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, 2006.

[4] Get BitTorrent, Online at: *http://www.bittorrent.com/download.*

[5] bzip2 documentation, Online at: *http://www.bzip.org/docs.html.*

[6] B. Cohen, Incentives build robustness in BitTorrent, *Proc. Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, Berkeley CA, June 2003.

[7] Digital Island Inc., Online at*: http://www.digitalisland.com.*

[8] B. Dempsey and D. Weiss. On the performance and scalability of a data mirroring approach for I2-DSI. *Proc. Internet2 Network Storage Symposium* (*NetStore*), Seattle, WA, Oct. 1999.

[9] M. Freedman, E. Freudenthal and D. Mazières, Democratizing content publication with Coral, *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004.

[10] C. Gkantsidis, T. Karagiannis, P. Rodriguez and M. Vojnovic, Planet scale software updates, *Proc. ACM Special Interest Group on Data Communication Conference (SIGCOMM)*, Pisa, Italy, Sept. 2006.

[11] Online at: *http://www.gnutella.com.*

[12] Gzip-rsyncable patch, On-line at: *http://rsync.samba.org/ftp/unpacked/rsync/patches/gzip-rsyncable.diff.*

[13] Gzip user's manual, Online at: *http://www.gnu.org/software/gzip/manual/gzip.html.*

[14] The KaZaA guide, Online at: *http://www.kazaa.com/us/help/index.htm.*

[15] J. Kerr, Dynamic feedback in LVS, *Proc. Australia's National Linux Conference (Linux.Conf.Au)*, Perth, Australia, Jan. 2003.

[16] Online at: *http://www.napster.com.*

[17] K. Obraczka and F. Silvia, Network Latency metrics for server proximity, *Proc IEEE Global Telecommunications Conference (GLOBECOM)*, San Francisco, CA, Nov. 2000

[18] H. Pucha, D. Andersen, and M. Kaminsky, Exploiting similarity for multi-source downloads using file handprints, *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Cambridge, Massachusetts, Apr. 2007.

[19] G. Pierre and M. Steen, *Globule: a collaborative content delivery network,* IEEE Communications Magazine, 44(8), Aug. 2006.

[20] Rsync: Remote file synchronization system, On-line at: *http://samba.anu.edu.au/rsync/documentation.html.*

[21] M. Satyanarayanan. *Distributed File Systems, Chapter 14*, Distributed Systems. Addison-Wesley, 1993.

[22] P. Shah and J.-F. Pâris, Peer-to-Peer multimedia streaming using BitTorrent, *Proc. IEEE International Performance Computing and Communications Conference (IPCCC)*, New Orleans, LA, Apr. 2007.

[23] W. Yang and N. Abu-Ghazaleh, GPS: a general Peer-to-Peer simulator and its use for modeling BT, *Proc. International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Atlanta, GA, Sep. 2005.