A P2P Network Booting Scheme Using a BitTorrent-like Protocol

Wigi Vei A. Oliveros
Networks and Distributed Systems Group
Department of Computer Science
University of the Philippines Diliman
Email: waoliveros@upd.edu.ph

Cedric Angelo M. Festin
Networks and Distributed Systems Group
Department of Computer Science
University of the Philippines Diliman
Email: cmfestin@upd.edu.ph

Roel M. Ocampo
Computer Networks Laboratory
Electrical and Electronics
Engineering Institute
University of the Philippines Diliman
Email: roel@upd.edu.ph

Abstract—**Network booting is widely used today, mostly for thin-client computing, cluster computing, and operating system installation. Popular protocols like trivial file transfer protocol (TFTP) and hypertext transfer protocol (HTTP) are used to download operating system images. Network booting clients simultaneously downloading from a single server create a bottleneck at the server's link once the link capacity has been maximized. The result is a slowdown in the download time and the overall boot process. Several P2P network booting solutions exist, but they are implemented on top of a running operating system. In this paper, we propose a distributed downloading scheme at the network bootloader level using a BitTorrent-like protocol for network booting clients. We discuss the implementation of a new download protocol for iPXE, an open-source network bootloader, and evaluate the performance of TFTP, HTTP, and the proposed BitTorrent-like protocol by measuring their download times over varying file sizes and number of nodes. On 20 clients, simultaneously downloading a 500 MiB file using the proposed protocol resulted to a decrease in average download time of 51.6% compared to TFTP and 46.6% compared to HTTP.**

I. Introduction

Network booting is widely used today in operating systems installation, cluster computing, and low-cost computing for schools and libraries. One of the benefits of network booting is centralized administration of nodes in a network. Administrators can simplify management of nodes by using only one operating system image for all booting nodes. However, having a centralized server for operating system images will result in a bottleneck when traditional download protocols like Trivial File Transfer Protocol (TFTP) and Hypertext Transfer Protocol (HTTP) are used. If the link of the server is already maximized, the boot process of the nodes is significantly slowed down.

Several network-booting systems use peer-to-peer (P2P) techniques to lessen the load of the server and to utilize other available links that interconnect nodes. To the best of our knowledge, all P2P solutions to the bottleneck problem are implemented on top of a running operating system, and no free and open-source network bootloader has any P2P downloading capabilities.

II. Related Work

Network booting allows a computer to boot from the network by downloading the files it needs to run from another computer. In many low-cost computing solutions like the Linux Terminal Server Project (LTSP) and Diskless Remote Boot in Linux (DRBL), clients use Preboot Execution Environment (PXE) to initialize the network booting process by acquiring an IP from a DHCP server and downloading a network bootloader from a TFTP server. Once the network bootloader is running, it fetches an operating system image from a TFTP server. Two of the most popular network bootloaders are PXELINUX and iPXE [1]. The former is limited to TFTP in downloading images while the latter can use HTTP (which results in faster downloads). iPXE can also be embedded onto NICs.

Simultaneous TFTP downloads of the OS from a single image server create a bottleneck at the server's link. This bottleneck results in a slowdown in the overall boot process of the nodes. Similar bottleneck scenarios in disk image transfers are addressed by using multicasting [2], [3] or peer-to-peer protocols [4]. In [2], a system named Frisbee was used in disk cloning FreeBSD and Windows XP, then multicasting was used to distribute cloned disk images simultaneously to many clients, while in [3], Ghost which has UDP multicasting features was used to distribute Windows 2000 and XP images. While multicasting alleviates the server bottleneck problem, there are cases where it is not applicable. For example, network security and stability concerns might lead network administrators to disable the use of multicasting. Network topology can also prevent one from using multicasting in distributing images [4].

We now take a look at three network booting systems that employ P2P in downloading operating system images [4], [5], [6]. The first two use virtual machines in running guest operating systems for use in classrooms. Both systems also address the bottleneck problem but they differ with our proposed solution with respect to the environment where the network booting process occurs. The third is a system that is closer to our proposed solution but imposes the limitation of booting only a Linux system.

A. *Network booting using virtual machines images for classes*

O'Donnell in [4] proposed a system of network-booting nodes that uses BitTorrent in downloading OS images for classrooms. Nodes in the system have an installed OS which runs a BitTorrent client. The client downloads virtual machine images that contain both operating system and data files. Downloading pre-constructed OS images ranging from 5–50 GB initially took 6–24 hours to complete. Shifting to BitTorrent, the download time was greatly reduced to just 1–4 hours.

Takada, et al. in [5] also proposed a network booting system that runs virtual machine images for use in university classrooms. They used MessagePack in requesting and sharing blocks of the image to boot. In their system, booting nodes connect to a centralized server to obtain a list of active nodes they can connect to. This behavior is similar to the use of a centralized tracker in BitTorrent.

The network booting process in both systems occur at the host operating system, and the images downloaded are for loading in virtual machines. While both systems address the bottleneck problem, they only do so at an environment where an operating system (host OS) is already running. The use of virtual machines, while appropriate in classroom settings due to the ease of administering images, may significantly cripple the performance of applications when the CPU is slow and RAM is small. We want to remove this limitation by proposing a solution that operates on an environment where no operating system yet is running. This can enable users to network-boot a host operating system by downloading the image using P2P.

B. *Network booting host OS for server farms*

Moobi [6] is a tool that enables thin-server farm management. Servers which do not have operating systems installed on disk are enabled using network booting. Parts of a fully functioning operating system and file system are handed out by Moobi to the thin servers by using a hybrid approach. The kernel and initial ramdisk are transferred using TFTP, the skeleton images by HTTP, and the remainder of the file system by BitTorrent. Moobi effectively addresses the bottleneck problem, but using it restricts the user to boot only a Linux system.

In our solution, the entire operating system is downloaded using the iPXE network bootloader program. All needed parts of the operating system to be booted are downloaded even before a kernel is loaded, enabling our solution to boot any operating system as long as the boot image for download is recognized by iPXE.

III. P2P Network Booting Scheme

As mentioned in earlier sections, the problem that we are trying to solve is the bottleneck occurring at the server's link when clients are simultaneously downloading boot images using protocols like TFTP or HTTP. A slowdown in the overall boot process is experienced when the server's link is maximized. In order to eliminate the bottleneck, we designed a peer-to-peer (P2P) approach in distributing the operating system image to simultaneously network-booting clients. We focused on (1) minimizing the amount of outgoing traffic from the node that contains the initial copy of the file, and (2) minimizing the overall booting time by reducing the download time of all participating nodes.

Our network booting scheme goes through the following steps to boot the entire system:

1) Nodes power on and use PXE to boot from the network.
2) Nodes download iPXE then form a P2P network overlay.
3) The OS image is downloaded simultaneously by nodes.
4) The OS images are loaded after download is complete.

A. *File sharing*

A *peer* in our system is a network-booting node participating in a simultaneous OS image download. A peer can connect to multiple peers. Initially, during a network booting session, no peer except one called the *seeder* will have the complete copy of the operating system image to be downloaded by other peers. This image will be downloaded by peers connected to the seeder piece-by-piece. As soon as a peer successfully downloads a piece, it can advertise the piece by notifying other connected peers. The peer uploads the piece to connected peers requesting for it. Notifications, requests, and responses between peers are done though the use of BitTorrent messages.

B. *BitTorrent-like Protocol*

The application-level protocol that enables our clients to do P2P communications uses messages from the BitTorrent protocol. Each peer connects to another through a handshake. A handshake is an exchange of HANDSHAKE messages that contain the hash of the file being downloaded and the ids of the peer. We adopt the format of the handshake and other messages from BitTorrent.

To download a piece from a remote peer, a REQUEST message must first be sent. If the remote peer has the piece, it will be sent to the requesting peer through a PIECE message. When a peer downloads a piece successfully, it sends a notification to all connected peers using HAVE messages.

We adopted the format of the messages from BitTorrent due to the following reasons: (a) The messages used by the BitTorrent protocol are already sufficient for requesting and exchanging pieces of a file between peers. (b) The protocol can easily be extended in the future to a full BitTorrent protocol implementation. In the future, we may want to use popular BitTorrent clients for our initial seeder, and participate in swarms of other BitTorrent clients.

C. *Modified Bootloader Program*

We implement the BitTorrent-like protocol by creating a new download protocol for the iPXE network bootloader. Our protocol runs on top of iPXE's TCP/IP stack. However, the TCP implementation in the current iPXE codebase is not a full implementation. Nodes using iPXE for network booting always run as clients. The current TCP stack only has provisions for creating client sockets and nothing for server sockets. This means that machines running iPXE can only initiate outgoing TCP connections and not accept incoming ones. Since a peer must be able to do both, we modified iPXE's TCP implementation to include support for server sockets.

D. Overlay Network

We decided to use a linear daisy-chained topology (shown in Fig. 1) for the overlay due to several reasons. First, the amount of outgoing traffic from the source of the file can be minimized to the size of a single copy with some additional protocol overhead. Second, we assume that information about the network nodes are known and this information is encoded in the iPXE binary. Also, the flow control and congestion control in the TCP implementation currently does not support simultaneous connections to three or more peers. Fig. 2 illustrates how peers in our daisy-chained topology will exchange messages using our P2P protocol. The peer directly connected to the initial seeder will have knowledge that the seeder has all the pieces of the file to be downloaded. Peer 1 requests piece n for download by sending a REQUEST message to the seeder. The seeder responds with a PIECE message containing piece n. After Peer 1 receives the piece, the peer advertises the piece to Peer 2 by sending a HAVE n message. Peer 2 can then request for piece n, receive it, then advertise it to the next peer. Using this strategy piece from the seeder get propagated to all peers down the chain. Eventually, all pieces will be received by all peers, and all will proceed to booting the downloaded file.

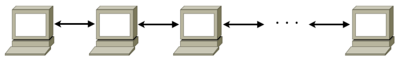

Fig. 1. Daisy chain of nodes with bidirectional links

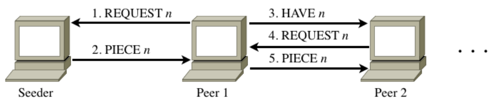

Fig. 2. Order of messages transmitted using our protocol on a daisy-chained topology

IV. Metrics

We measured the following in our experiments to evaluate the performance of our protocol relative to the existing download protocols in iPXE: *average download time* and *total bytes served*.

A. Average download time

If there are n nodes that downloaded a file, each with a download time t_i, then the average download time is equal to

$$\frac{1}{n}\sum_{i=1}^{n} t_i.$$

B. Total bytes served

The total bytes served b of a server is the number of bytes sent by the server to all other nodes. In the case of our P2P protocol, this is the amount of outgoing data from the first seeder. We only measure the amount of data at the application layer, (e.g. TFTP requests, HTTP header and data, and our P2P messages). Retransmitted data in TCP are not added to the count.

C. Experimental Setup

In the experiments, we utilized 20 nodes in a LAN setup. The nodes and the server were connected to a 24–Port 10/100 Cisco Small Business SF 300–24 Managed Switch via 100 Mbps full-duplex links.

We benchmarked the performance of TFTP, HTTP, and our P2P protocol in simultaneous downloads on the 20-node setup. We recorded the start and finish times of the nodes participating in a download run. Modifications to the iPXE code were written to enable us to synchronize the start time of all nodes participating in a download run.

For each run, we encode the start time to the iPXE code then rebuild the binary. The nodes are restarted and the updated iPXE binary is downloaded. After that, an embedded script containing commands for downloading were loaded by each node. The nodes start downloading when the encoded start time occurs. Until then, nodes will just wait.

V. Results

In this section we report the results of our experiments. We analyze the data obtained to provide an evaluation of our proposed solution. In the succeeding figures, we refer to our peer-to-peer protocol as “P2P”.

A. Effect of file size on average download time

Shown in Fig. 3 is the average download time of 20 nodes over file sizes ranging from 100–500 MiB in increments of 100. All three protocols show linear increase in average download times as the size of the file increases. As indicated by the slopes of the curves, the average download time grows the fastest when using TFTP, followed by HTTP. Among all three protocols, average download time grows the slowest when using our protocol (P2P), demonstrating effective reduction of the average download time by almost half during simultaneous downloads. The trends observed here shows that our protocol is a good alternative to TFTP and HTTP when simultaneously fetching a file using a number of nodes.

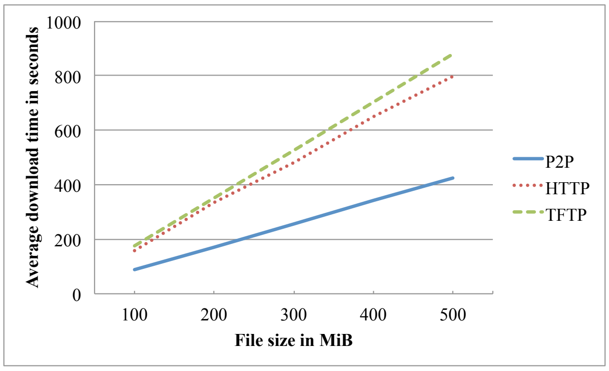

Fig. 3. Average download time on 20 nodes as file size increases

B. *Effect of number of nodes on average download time*

Shown in Fig. 4 is the average download time of a 100 MiB file over a varying number of nodes. When only a single node is downloading, HTTP outperforms both TFTP and our protocol. This behavior can be attributed not only to the low protocol overhead in HTTP, but also in its effective use of TCP as several blocks of bytes of the file can be transmitted without receiving an acknowledgement. Since TFTP uses block requests that execute in lock-step fashion (i.e. the next block is not requested until the block previously requested is received), nodes are restricted to receiving one block of 512 bytes at a time. The result is low throughput that is dependent on the round trip time (RTT) of a block. At 8 nodes, HTTP no longer shows superior performance as TFTP catches up, and P2P narrows the gap. At 10 nodes, all three protocols have the same average download time (P2P at 86 s, HTTP at 85 s, and TFTP at 87 s). And at 11 nodes and greater, our proposed protocol has a lower average download time compared to both HTTP and TFTP.

Our protocol demonstrates an almost constant average download time compared to TFTP and HTTP's linear increase at 8 nodes and beyond. While there is an increase in the download time as the number of nodes increase, the increment in time is very minimal and almost unnoticeable. From these observations, we can say that our P2P protocol, in terms of download time, scales well as the number of nodes increases.

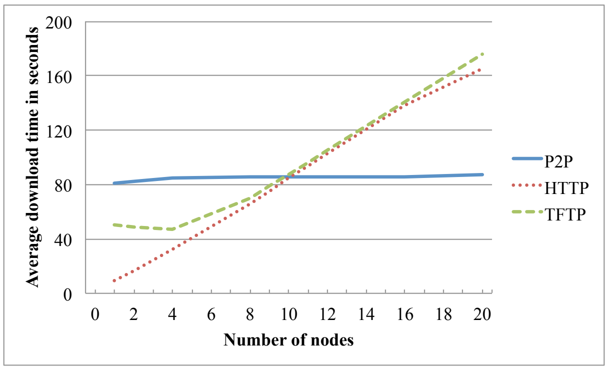

Fig. 4. Average download time of a 100 MiB file as number of nodes increase

C. *Server and Initial Seeder Traffic*

On 20 nodes downloading a 100 MiB file, the amount of outgoing server traffic generated by P2P, HTTP, and TFTP are shown in Figure 5. HTTP traffic reached 2000.01 MiB, while TFTP reached 2005.59 MiB. Both protocols generated traffic 20 times the size of the file with additional minimal overhead. We conclude that in HTTP and TFTP, the amount of traffic linearly increases as the number of nodes simultaneously downloading increases. However, our protocol only generated 102.93 MiB of outgoing traffic from the initial seeder. The traffic only consisted of one copy of the file being downloaded with additional overhead. The amount of traffic becomes independent of the number of peers simultaneously downloading. This result can be attributed to the nature of the chosen daisy chained topology.

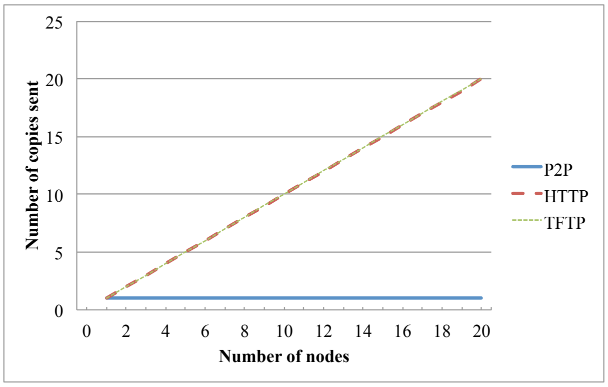

Fig. 5. Outgoing traffic as number of file copies sent by the server/seeder when downloading a 100 MiB file on 20 nodes

VI. Conclusion and Future Work

In this paper, we have described an alternative peer-to-peer download scheme for network-booting clients using a BitTorrent-like protocol and a daisy-chained network overlay. We have shown that our proposed solution to server bottlenecks occurring during simultaneous downloads can improve download times and reduce server traffic. Our scheme essentially moved the peer-to-peer approach in network booting from the operating system level to the network bootloader level.

Several improvements to our work can result in faster boot times and better scalability. One possibility is to extend the protocol into a full BitTorrent implementation to enable interconnectivity with popular clients. We are also interested in observing the performance of our protocol when used in a variety of network overlays, as using the daisy-chained topology, while resulting to fast downloads, produces articulation points in the intermediate nodes of the chain.

References

[1] iPXE, "ipxe - open source boot firmware," http://ipxe.org.

[2] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb, "Fast scalable disk imaging with frisbee," in *In Proc. of the 2003 USENIX Annual Technical Conference*, 2003.

[3] C. Sin and D. Wong, "Image baby image!: making pc cloning more efficient," in *SIGUCCS '07: Proceedings of the 35th annual ACM SIGUCCS fall conference*. New York, NY, USA: ACM, 2007, pp. 314–317.

[4] C. M. O'Donnell, "Using BitTorrent to distribute virtual machine images for classes," in *SIGUCCS '08: Proceedings of the 36th annual ACM SIGUCCS conference on User services conference*. New York, NY, USA: ACM, 2008, pp. 287–290.

[5] S. Takada, A. Sato, Y. Shinjo, H. Nakai, A. Sugiki, and K. Itano, "A p2p approach to scalable network-booting," in *Networking and Computing (ICNC), 2012 Third International Conference on*, Dec., pp. 201–207.

[6] C. McEniry, "Moobi: a thin server management system using bittorrent," in *Proceedings of the 21st conference on Large Installation System Administration Conference*, ser. LISA'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 20:1–20:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1349426.1349446