x86 Network Booting: Integrating gPXE and PXELINUX

H. Peter Anvin
rPath, Inc.
<hpa@zytor.com>

Marty Connor
Etherboot Project
<mdc@etherboot.org>

Abstract

On the x86 PC platform, network booting is most commonly done using software that follows the Preboot Execution Environment (PXE) specification. PXELINUX from the SYSLINUX Project and gPXE from the Etherboot Project are popular Open Source implementations of key PXE components.

In this presentation, we will describe how these two projects were able to jointly develop an integrated PXE-compatible product that provides additional network booting capabilities that go well beyond the PXE specification. We will also discuss some of the organizational challenges encountered during this collaboration between two Open Source projects with different priorities, design goals, and development strategies.

1 Motivation

Open Source software development by definition allows and encourages code sharing and collaboration between projects. There are, however, costs associated with these endeavors, and these costs must be weighed against potential benefits associated with using code developed by another project.

In the case of improving integration between gPXE and PXELINUX, developers from SYSLINUX and the Etherboot Project were motivated to collaborate because they believed there might be significant benefits to leveraging work already done by the other project. Although the process of creating better interoperability between products required significant communication and effort, it was a useful and rewarding exercise for both development teams.

To understand how these two Open Source projects reached this point of collaboration we will examine the history of network booting on the x86 PC platform, as well as the development journey each project took prior to this collaborative effort.

2 The PC Platform: Ancient History

Network booting has been implemented in various forms for many years. To appreciate how it evolved it is instructive to examine the early days of PC computing when standards were few, and achieving consensus between vendors on any technical innovation was even more difficult than it is today.

The x86 PC platform has a direct lineage to the original IBM PC 5150 released in 1981. This machine, an open platform, and its successors, the 1983 IBM XT and the 1984 IBM AT, were widely copied by a number of manufacturers. The PC industry largely followed IBM's technical lead until the disastrous 1987 attempt at reclaiming their initial monopoly position with the closed platform PS/2 line erased their technical and marketplace leadership positions in the industry.

As a result, for the first half of the 1990s there was no clear path for new standards to become accepted on the PC platform, and PCs were becoming little more than massively sped-up versions of the IBM AT. Thus, even as new media such as networking and CD-ROMs became available to the platform, there was little support for booting from anything other than the initially supported floppy and hard disk, although PCs could optionally use an expansion card carrying proprietary booting firmware.

TCP/IP networking, as something other than a niche product, came late to the x86 PC platform. For many years, memory limitations when running MS-DOS and its derivatives meant that simpler, proprietary network stacks were used; the primary ones being NetBIOS from IBM and Microsoft, and IPX from Novell.

The response of early network card manufacturers, to the extent they supported booting from networks at all, was simply to provide a socket into which a ROM (usually an EPROM) could be inserted by the end user. This ROM had to be preprogrammed with firmware specific

both to the network card and the network protocol used; as a result, these ROMs were frequently expensive, and few PCs were ever so equipped. To further complicate the situation, the size of ROMs varied widely depending on the card. Some cards supported as little as 8K of ROM space, greatly limiting the amount and complexity of boot software that could be supplied.

In the early 1990s, as the use of TCP/IP became more prevalent, some manufacturers provided TCP/IP-based booting solutions using the then-standard BOOTP and TFTP protocols, often based on downloading a floppy image to high memory (above the 1 MB point addressable by DOS.) These solutions generally did not provide any form of post-download access to the firmware; the downloaded floppy image was expected to contain a software driver for a specific network card and to access the hardware directly.

By the mid 1990s, the PC industry, including IBM, was seriously suffering from having outgrown IBM AT standards. In January 1995, Phoenix and IBM published the "El Torito" standard [3] for booting PCs from CD-ROM media. Support for this standard was initially poor, and for the rest of the decade most operating systems that were distributed on CD-ROM media generally shipped with install floppies for booting PCs that lacked functional CD-ROM booting support.

3 NBI and PXE: Network Booting Strategies

As the cost of network interface cards (NICs) dropped dramatically in the early 1990s, the cost of a network boot ROM could be a substantial fraction of the cost of the NIC itself. The use of OS-dependent, user-installed ROMs as the only method for network booting had become a significant limitation. In the Open Source world, this issue was further complicated by the need to supply a physical piece of hardware, since a software-only distribution would require end users to have access to an expensive EPROM burner to program the software into an EPROM.

In 1993, Jamie Honan authored a document titled "Net Boot Image Proposal" [4] which defined image formats and methods for downloading executable images to a client computer from a server. A key feature of Jamie's proposal was the specification of *Network Boot Image* (NBI) file format, "a vendor independent format for boot images." This may have been the first attempt in the Open Source world at specifying an OS-independent method for network booting on the PC platform.

To keep NBI loader code uncomplicated, a utility called `mknbi` was used to convert OS specific files such as kernels and initrds into NBI format for loading into memory. This simplified loader code because it only needed to load a single, uncomplicated image format. It did, however, require an extra step to convert OS images to NBI format prior to booting.

NBI was never used in the closed-source OS world. Instead, vendors continued to offer incompatible solutions, usually based on their respective proprietary network stacks.

In 1997, Intel *et al.* published the Wired for Management Specification (WfM) [5]. It included as an appendix a specification for the *Preboot Execution Environment* (PXE), a TCP/IP-based, vendor-neutral network booting protocol for PCs, which included an application programming interface (API) for post-download access to the firmware driver. The specification, as well as its current successor [6], both have numerous technical shortcomings, but it finally made it possible for NIC and motherboard vendors to ship generic network booting firmware. Higher-end NICs began to have onboard flash memory preprogrammed with PXE from the factory instead of providing a socket for a user-installable ROM. Motherboards began to have PXE firmware integrated into their BIOSes.

The PXE specification defines a set of software components (Figure 1) including a PXE Base Code (BC) stack, a Universal Network Driver Interface (UNDI) driver—both in ROM—and a Network Boot Program (NBP), which loads from a server. Of these components, only the UNDI is specific to a certain NIC. Just as with the Open System Interconnect (OSI) networking model, this model does not completely reflect the actual division into components, but is nevertheless useful as a basis for discussion.

The PXE approach is significantly different from the one taken by NBI. NBI simply loads a `mknbi`-prepared bootable OS image into memory with no further access to the ROM-resident network routines.[1] In contrast, PXE BC first loads an NBP, which then loads a target

[1] Some NBI-compliant ROMs would provide APIs to request additional services, but those APIs were never standardized.

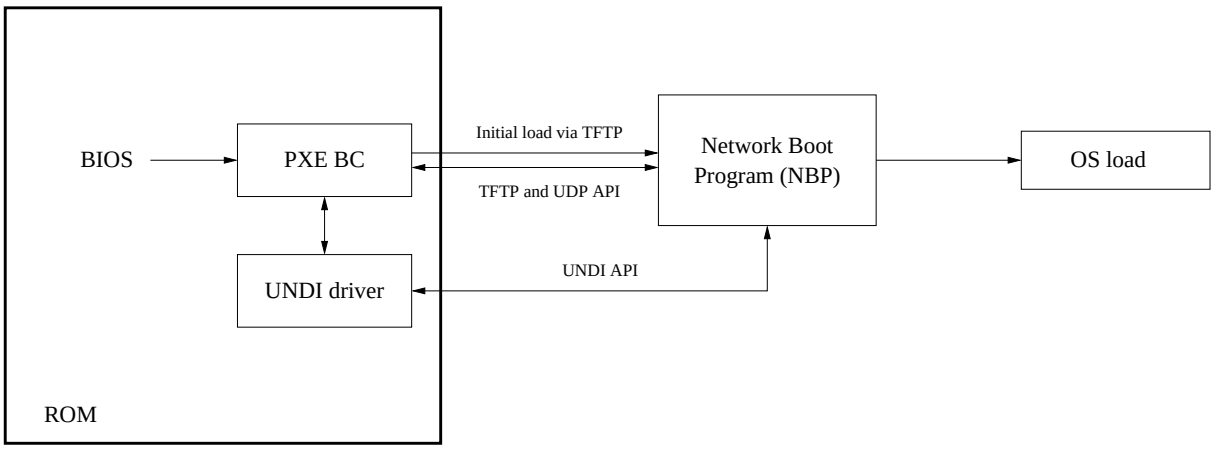

Figure 1: The PXE concept model

OS using still-resident BC and/or UNDI stacks from the ROM using an API defined in the PXE specification.

These differences in approach allow PXE ROMs to contain simple, compact loader code while enabling more specific and capable second-stage loader code to load native OS image formats without pre-processing. Further, NBP code resident on a network server can be upgraded centrally to fix bugs or add capabilities. Added costs to this approach versus NBI include extra time and server load to transfer an NBP before the target OS image is loaded, and the need to maintain multiple components on the boot server. These costs have minimal impact when using modern servers and LAN hardware.

4 The SYSLINUX Project

The SYSLINUX [1] project was started in 1994 by H. Peter Anvin as a way to allow creation of Linux boot floppies without requiring Linux-specific tools. Until that point Linux boot floppies, universally required to install Linux on the PC platform, had been distributed as raw images to be written to a bare floppy disk. On an MS-DOS machine this meant using a tool called RAWRITE. The resulting floppy was an opaque object and was considered unformatted by most non-Linux operating systems.

The SYSLINUX installer (named by analogy to the MS-DOS SYS command) ran directly under MS-DOS and all data was stored in conventional files on an MS-DOS FAT filesystem. Since it was designed for running on floppies it had to be small (the 18K overhead from the FAT filesystem itself was a point of criticism in the early days): as of version 1.30 (November 3, 1996) the SYSLINUX binary was 4.5K in size. Even so, it contained a reasonably flexible configuration system and support for displaying online help; the latter was particularly important for install disks.

In 1999 Chris DiBona, then of VA Linux Systems, provided an early PXE-equipped system as a development platform for a PXE loader. Since the Intel PXE specification at the time specified that only 32K was available to the NBP it was decided that basing the PXE loader code on SYSLINUX—an existing, compact, configurable loader—would make sense. Thus, SYSLINUX 1.46, released September 17, 1999, included PXELINUX, a PXE network bootstrap program with a SYSLINUX-based user interface. Subsequently SYSLINUX acquired support for other media, specifically ISO 9660 CD-ROMs in El Torito "native mode" and hard disks with standard Linux ext2/ext3 filesystems.

As support for CD-ROM booting—and later, USB booting—on PCs became more universal, pressure to keep code size minimal waned since storage capacities were less restrictive. At the same time network administrators in particular requested a more configurable user interface. To avoid burdening the SYSLINUX core (written in assembly language and challenging to maintain) with additional user interface features, an API was developed to allow user interfaces to be implemented as independent, loadable modules. The first such interface was a very sophisticated system written by Murali Krishna Ganapathy, based on a text-mode windowing interface. Though very advanced and capable of almost

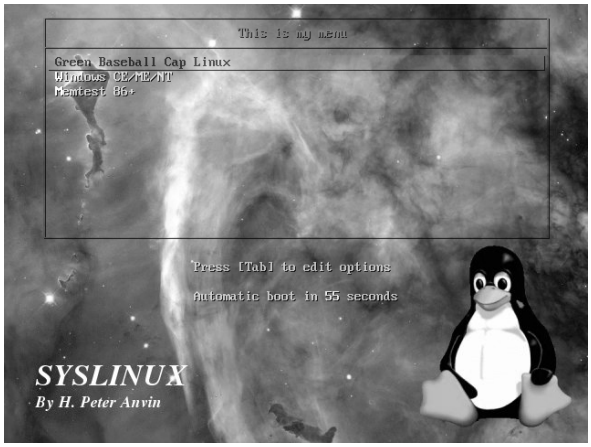

Figure 2: The SYSLINUX simple menu system

infinite customization, it turned out to be too difficult for most users to configure. To address this issue, the "simple menu system" (Figure 2) was implemented and is now used by most SYSLINUX users.

The API also allows support for new binary formats to be written as well as "decision modules" (boot selection based on non-user input, such as hardware detection). An 88,000-line library, derived from `klibc`, is available to developers to make module development easier and as similar to standard applications-level C programming as possible.

Historically SYSLINUX has focused on the PC BIOS platform, but as the bulk of the code has been migrated from the core into modules and from assembly language into C, the feature set of the core has become bounded. The intent is for the core to become a "microkernel" with all essential functionality in (possibly integrated) modules; this would permit the core to be rewritten to support other platforms such as EFI.

PXELINUX has not, however, implemented any protocols other than TFTP. The PXE APIs only permit access at one of three levels: TFTP file download, UDP, or raw link layer frames. No method to access the firmware stack at the IP layer is provided. This means that to support TCP-based protocols, such as HTTP, a full replacement IP stack is required.

It is worth noting that although a large number of people have contributed to SYSLINUX over the years, it has largely remained a one-person project. As of this writing there is a serious effort underway to grow the SYSLINUX project developer base. To facilitate this process the SYSLINUX project will participate in Google Summer of Code for the first time in 2008.

5 The Etherboot Project

In 1995 Markus Gutschke ported a network bootloader, Netboot, from FreeBSD. Netboot followed Jamie Honan's 1993 "Net Boot Image Proposal." Since the first OS targeted for loading was Linux, `mknbi` was used to combine kernel and initrd images into a single NBI file before loading.

Since Netboot did not support his network card and Netboot drivers at the time had to be written in assembly language, Markus implemented a new driver interface allowing drivers to be written in C. He called his code Etherboot.

Markus released Etherboot 1.0 in 1995 and it proved to be popular enough that a small community called the "Etherboot Project" [2] formed to support and improve it with additional functionality and drivers. In late 1996 one of the group's more active contributors, Ken Yap, took over leadership of the project.

In 1997, when Intel published the PXE specification, Ken began work on NILO, a first attempt at an Open Source PXE implementation. In 1998 Rob Savoye, with funding from NLnet Foundation, took over NILO development. For various reasons the project was unsuccessful, and development of NILO officially ceased in 2000 [7].

Etherboot development continued, however, and in 1999 Marty Connor became involved with the project, having discovered it through conversation with Jim McQuillan of LTSP (Linux Terminal Server Project). Marty ported several of Donald Becker's Linux NIC drivers to Etherboot to provide support in Etherboot for popular cards of the day.

In 2000, Marty created `rom-o-matic.net` [8], a web-based Etherboot image generator that created customized Etherboot images on demand. This made it much easier for people to create and test Etherboot because no specific build environment or command line expertise was required. Usage and testing of Etherboot increased dramatically.

Another boost to Etherboot use came in 2001 when the Etherboot Project first exhibited in the .ORG Pavilion

at the IDG LinuxWorld Expo and invited LTSP to share their booth. Live demos of Etherboot network booting and LTSP thin clients sparked the interest of many potential users.

In 2002 Michael Brown first encountered Etherboot while trying to find a solution for booting wireless thin clients. He developed and submitted an Etherboot driver to support Prism II-based wireless cards, and became a regular contributor to the project.

About this time Marty became concerned that PXE was fast becoming a de facto standard for network booting since it was being included in a significant number of motherboards and mid-to-high-end NICs. Although there was strong opposition within the project to supporting PXE for technical reasons, he felt that unless Etherboot supported the PXE specification Etherboot would quickly become irrelevant for most users.

Added incentive to support PXE came in 2004 when Marty and H. Peter Anvin spoke about creating a complete, compliant Open Source PXE implementation to support PXELINUX. Later in 2004 Michael added partial PXE support to Etherboot, which was then capable of supporting PXELINUX though it lacked full PXE functionality.

In 2005 Marty and Michael created gPXE, a major rewrite of Etherboot with PXE compatibility as a key design goal. Soon after, Marty became the third Etherboot Project Leader and Michael became the project's Lead Developer. Primary development energy was then redirected from Etherboot to gPXE.

In 2006 Michael, with help from Google Summer of Code student Nikhil C. Rao, added more robust and compliant TCP support to gPXE. This enabled Michael to add TCP-based protocols such as iSCSI, which in turn allowed gPXE to network-boot exotic operating systems such as Windows Server 2003.

In 2007 the Etherboot Project exhibited in the LinuxWorld Expo .ORG Pavilion for the 12th time, this time demonstrating gPXE booting of various OSes via HTTP, iSCSI, AoE, and other protocols. Michael and Peter created, coded, and demonstrated a first API for gPXE to PXELINUX integration.

As of 2008 `rom-o-matic.net` had generated over two million Etherboot and gPXE images, with a typical size of 40K for a ROM image containing support for DHCP, DNS, TFTP, HTTP, iSCSI, AoE, and multiple image formats including PXE, bzImage, Multiboot, and gPXE scripts.

A large number of people have generously contributed to the success of the Etherboot Project over the years. Many of their names can be found on the project's acknowledgments web page [9]. There are also many users who contribute on the project's mailing lists and IRC channel. Their help with documentation, testing, and support greatly contributes to the quality and popularity of Etherboot and gPXE.

6 The Strengths of Each Project

Given the primary focuses of the projects it is not surprising that each brings different strengths to the collaboration. gPXE supports a wide range of protocols, can be integrated in ROM rather than relying on shipped firmware, and supports other architectures; however, its user interface is limited. PXELINUX has advanced user interfaces and, because it is a part of the SYSLINUX suite, has cross-media support, but its protocol support is limited to TFTP.

Within the PXE concept model PXELINUX strictly acts as the NBP, whereas gPXE can act either as NBP, BC, or BC and UNDI combined depending on how it is configured and compiled. gPXE configured to function as BC and UNDI is most common when it is used in ROM or loaded from disk.

gPXE is also able to be loaded from a server as an NBP and then take over the functions of either the BC only, or the BC and UNDI combined, and then load another NBP such as PXELINUX to perform a target OS load. This configuration, referred to as "chainloading," can be used either to substitute functionality from a partially working PXE stack or to get the enhanced capabilities of gPXE, either way without having to actually modify the ROM on the device.

7 Choosing a Strategy for Collaboration

Collaborative projects carry significant risks and rewards over single-team development. Because of this, potential costs and benefits should to be considered carefully before embarking on such a journey.

Rather than seeking to collaborate with the Etherboot Project, the SYSLINUX project could have implemented its own TCP/IP stack, HTTP client, and iSCSI and AoE initiators for PXELINUX. Alternatively, it could have reused the code from gPXE or used another Open Source TCP/IP stack, such as lwIP [10].

Collaboration, though requiring some additional development effort, had several potential advantages:

- Using gPXE's protocol support would mean that SYSLINUX maintainers would not have to integrate and support additional code to support new protocols.
- Code improvements to either project could be of benefit to users of both projects.
- Users of both gPXE and PXELINUX could share a single user interface for accessing features.

In light of these potential advantages, the developers of both projects decided to explore ways of working together.

Popular strategies for collaboration between Open Source projects differ primarily based on whether it is the intention of one project to take over maintenance of code produced by another project or whether the projects intend to maintain separate code bases which interoperate based on well-defined interfaces.

Some common strategies for collaboration between projects are:

- *Componentization*, where one project's code and development team simply becomes part of another project. The second project then ceases to exist as an independent project.
- *Aggregation*, where one project includes the other's code as a component, possibly in modified form, but the second project's code continues to be developed as a separately maintained project. In this model, the first project can be considered a consumer of the second project. This is particularly common with application programs that depend on libraries that are not widely available.
- *Cooperation*, where the two projects mutually agree on a set of APIs and independently implement their respective parts. The projects are maintained separately, and aggregation into a combined product is performed by the distributor or end user.
- *Stacking*, in which one project independently defines an interface available to all potential users of the code, which is completely sufficient (without modification) for the needs of the second project. In this case, the combination is strictly a case of the second project being a consumer of the first, and final aggregation is typically performed by the distributor or end user; this strategy is typified by widely used libraries.

Each of these strategies has advantages and pitfalls, based on the nature of the projects, the development teams, and any corporate entities that may be involved. The tradeoffs between these strategies can be quite different in the Open Source world over what they might be in analogous corporate environments.

8 Integration, so Far

Initial steps toward integrating gPXE and PXELINUX were taken in 2004 when Etherboot first became capable of acting as a PXE ROM (BC and UNDI combined). This allowed Etherboot to replace defective vendor PXE implementations, either by replacing the ROM or by chainloading, but did not provide any additional capabilities. Nevertheless, this approach has been widely used, especially with SiS900 series NICs, a briefly popular NIC with a notoriously buggy vendor PXE stack.

PXELINUX users had been requesting additional protocol support for quite some time, especially the ability to download via HTTP. Not only is HTTP, a TCP-based protocol, faster and more reliable than TFTP (based on UDP), but HTTP servers have better support for dynamic content, which is frequently desired for generating configuration files.

At LinuxWorld Expo San Francisco in 2006, SYSLINUX and Etherboot Project developers met to discuss the situation. At that meeting, the following constraints were established:

- The primary focus of gPXE is as ROM firmware. The continued utility of gPXE in ROM must be maintained.

- Extended protocol support must work in PXELINUX when it is loaded from a vendor PXE stack. Supporting extended protocols only with gPXE in ROM is not acceptable.
- Although gPXE already had support for extended protocols by accepting a URL via the PXE API's TFTP (`PXENV_TFTP_OPEN`) routine, the PXE TFTP interface is inadequate for PXELINUX; a new API is necessary for PXELINUX to access functionality beyond what standard PXE APIs permit.

In electronic discussions afterwards, the following high-level plan was agreed to by both development teams:

- An extended PXE API for PXELINUX will be developed.
- A technique will be developed to aggregate gPXE and PXELINUX into a single binary to simplify deployment in existing PXE environments.

Unfortunately around this time both projects became distracted by other priorities: the Etherboot Project focused on providing new and improved network drivers, support for SAN protocols (iSCSI and AoE), and completing an initial gPXE release; the SYSLINUX project on user interface and module API improvements. A test version without the aggregate binary (and certainly not stable enough to be deployed in a real environment) was demonstrated at IDG LinuxWorld Expo 2007, but after that the collaboration languished for months.

Toward the end of 2007 improved protocol support was becoming a high priority for the SYSLINUX project, while Etherboot developers were pushing toward a mid-February initial release (gPXE 0.9.3). Over the holidays developers from both projects conducted a sizable joint development and debugging effort, implementing binary encapsulation support and tracking down a number of issues that occurred when gPXE was chainloaded from a network server, as opposed to running from ROM. Having more developers testing the code helped find bugs that only manifested on certain hardware and particular PXE implementations. Fortunately (for some meaning thereof), the combined team had access to to a large and eclectic collection of troublesome hardware.

After the initial beta release of gPXE 0.9.3 on February 14, 2008, the original plan was reviewed by both development teams. As there had been significant changes in both code bases, the plan was revised as follows:

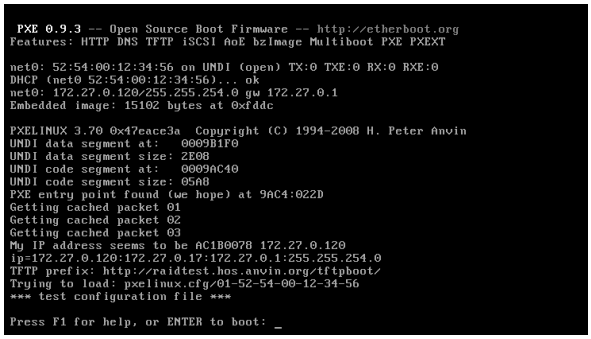

Figure 3: `gpxelinux.0` loading its config file via HTTP

- An initial implementation will be based on the already implemented extended PXE API, plus any additions necessary.
- This API will be considered private and not guaranteed to be stable between revisions. Thus, the only supported use will be between gPXE and an an embedded PXELINUX; if gPXE is used in ROM it should still chain-load the combined image.
- As SYSLINUX code has increasingly moved toward having most of its code in modules, with a well-defined application binary interface (ABI), the projects will eventually migrate to a model where gPXE implements the SYSLINUX module ABI directly; at that time the private API will be deprecated.

The third item on this list was accepted as a Google Summer of Code project under Etherboot Project mentorship for 2008.

More powerful functionality is possible when gPXE is also used in ROM (or provided on CD-ROM, USB stick, or floppy). gPXE can either be used as the native PXE stack on the system using its own network device drivers, or it can use the UNDI driver from the vendor PXE stack. With suitable configuration gPXE can download an initial NBP image from a specific URL set at compile time or saved in nonvolatile storage. This capability can be used to invoke a service facility with a very small investment in ROM; the only local network resources required are working DHCP and DNS. If the downloaded image is `gpxelinux.0`, a full range of PXELINUX modular functionality becomes available.

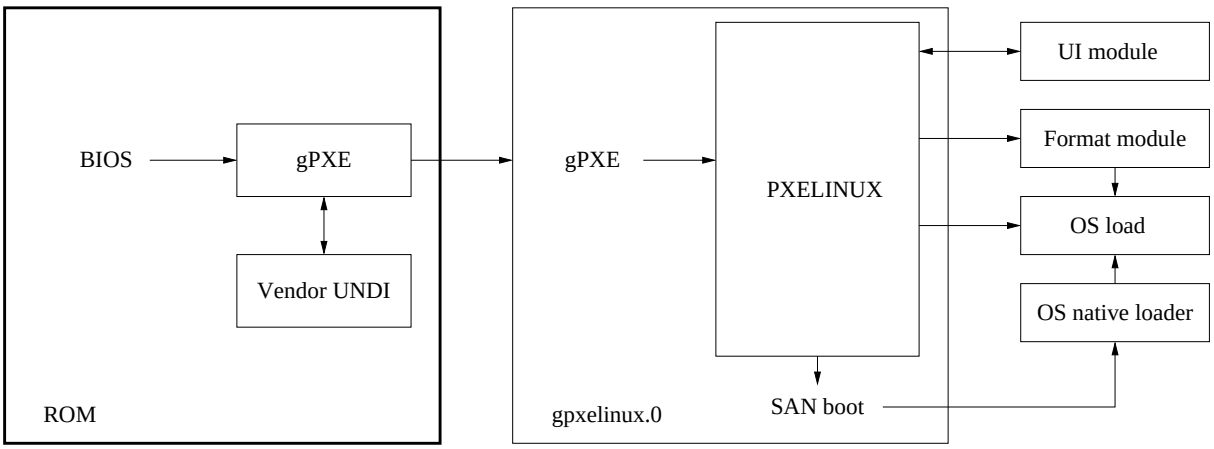

Figure 4: gPXE loading `gpxelinux.0`, using a vendor UNDI driver

Ultimately SYSLINUX and the Etherboot Project decided not to combine code or development teams but rather to modify both of their code bases to support a jointly developed API (a *cooperation*). However, to make it easier for end users, the SYSLINUX distribution now contains a snapshot of gPXE sources so that the combined image can be built from a single download; in this sense, it is also an *aggregation*. However, the intent or the projects to emphasize the cooperative aspects of supporting a common API and to have the SYSLINUX combined source code tree have minimal differences from the primary gPXE distribution.

It is the desire of both projects that this will permit each project to retain its particular focus and identity, while giving end users access to functionally contained in both code bases.

As of April 15, 2008, the combined product is available in beta form as part of SYSLINUX 3.70-pre9. This distribution includes a slightly modified snapshot of the gPXE git repository (containing a few changes necessary for the usage model, but in need of cleanup before being fed upstream into the gPXE tree). When built, this produces `gpxelinux.0`, a combined binary, in addition to the conventional `pxelinux.0`. If loaded from a standard vendor PXE stack, `gpxelinux.0` can be redirected to non-TFTP protocols via the PXELINUX Path Prefix DHCP option [11] or via explicit URL syntax in the configuration file. A DHCP option, in particular, allows even the PXELINUX configuration file to be acquired through non-TFTP protocols such as HTTP, making it much easier to generate configuration files dynamically.

As of the SYSLINUX 3.70-pre9 release, `gpxelinux.0` is not yet a drop-in replacement for `pxelinux.0` in all situations because some issues relating to chainloading another NBP remain. It is expected that these issues will be relatively easy to resolve.

9 Next Steps

At the time of this writing the primary collaborative development focus of the projects is to resolve a few remaining interoperability issues and to clean up SYSLINUX-local modifications to gPXE so that they may be integrated into the official gPXE source base.

The combined `gpxelinux.0` image using the current approach is expected to be released with SYSLINUX 3.70. Over the summer of 2008 considerable progress on implementing the SYSLINUX module API in gPXE will hopefully be made. This effort will also serve as a trailblazing project for the "microkernelized" rewrite of the SYSLINUX core across all media.

10 Lessons Learned Along the Way

When integrating Open Source projects, especially ones developed outside the influence of a corporate or sponsorship structure, one must consider at least the following pitfalls and concerns:

- *Motivation:* For collaboration between two projects to succeed it is important that there be incentives for both of their user communities and de-

velopment teams. Without a shared sense of purpose, enthusiasm for the project may quickly wane on the part of one or both projects.

In the case of the SYSLINUX-Etherboot collaboration, both projects recognized the opportunity to leverage each others work and both were motivated to explore how they might productively work together. Understanding the motivations of other project participants was an important part of keeping the collaboration moving forward.

- *Focus:* The primary reason for combining two projects is that each brings different strengths to the table. It is likely that each project has development goals aimed toward improving its respective strengths. A goal related to facilitating code combination might therefore be relatively low on the priority of either one or *both* the parent projects! A good working relationship is likely to improve joint focus, but other driving forces may still prevail, such as funding issues.

  Focus differences were a significant issue early in the SYSLINUX-Etherboot collaboration. Rather than completely executing the original project plan, each project ended up working more on other priorities, especially SAN support for gPXE and improved user interfaces for SYSLINUX. Not until late 2007 discussions did both projects agree on the priority of the joint development effort and commit to the shared goal of producing a test release in the March 2008 timeframe.

- *Culture:* Every Open Source project has a unique culture that generally depends on the preferences of the original or principal developers or administrators. Just as in a corporate collaboration or merger, culture clashes can manifest themselves as subtle but significant roadblocks to progress. In Open Source projects, such issues may include frequency and style of developer communication, review and commit policies, and coding style. In large projects, these processes are often more formalized than in smaller projects. Nevertheless, it is important to recognize, respect, and address differences between collaborative partners as they can significantly affect the success of the joint effort.

  Whereas the SYSLINUX project has a single central maintainer responsible for all technical direction, Etherboot Project decision making is somewhat more distributed. This difference complicated some early discussions until it became clear that for actionable agreement to be achieved, all relevant Etherboot Project team members needed to be included in technical discussions.

- *Credit where credit is due:* These days many, if not most Open Source software developers are deriving their income either directly or indirectly from Open Source work. Others, such as students, may be concerned about future marketability. Still others may consider recognition a major driver for their Open Source involvement. Accordingly, *recognition is very valuable currency in the Open Source world.* A perception, true or not, that one project is trying to usurp credit for another project's work is likely to create ill will and poor relations.

  Discussions of credit can be sensitive, as some people may feel their concerns aren't appropriate or valid. In the particular case of the SYSLINUX-Etherboot Project collaboration, both sides had concerns, but some went unvoiced for a long time. Although both sides had good intentions, these unresolved concerns slowed the collaboration considerably, until they were discussed and addressed as legitimate issues.

By recognizing that with the best intentions such issues can and do occur—even in a collaboration involving relatively small projects—one can significantly improve the chances for a successful and timely joint project.

Learning to work together benefited the code bases and development teams of both projects. Code changes needed to support jointly developed interfaces required optimization and auditing of critical code sections. In addition, communication, confidence, and trust between developers significantly improved during the process of working together to achieve a shared goal.

References

[1] SYSLINUX project web site, `http://syslinux.zytor.com/`

[2] Etherboot Project web site, `http://www.etherboot.org/`

[3] C. Stevens and S. Merkin, *"El Torito" Bootable CD-ROM Format Specification*, Version 1.0,

January 25, 1995, http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf or http://tinyurl.com/99c5f

[4] J. Honan and G. Kuhlmann, "Draft Net Boot Image Proposal," Version 0.3, June 1997, http://www.nilo.org/docs/netboot.html

[5] Intel Corporation *et al.*, *Network PC System Design Guidelines*, Version 1.0b, August 5, 1997, http://www.intel.com/design/archives/wfm/

[6] Intel Corporation, *Preboot Execution Environment (PXE) Specification*, Version 2.1, September 20, 1999, http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf

[7] K. Yap, "NILO; organization and status," May 2000, http://www.nlnet.nl/project/nilo/how.html

[8] Etherboot project, rom-o-matic.net, http://www.rom-o-matic.net/

[9] Etherboot project, Acknowledgements, http://etherboot.org/wiki/acknowledgements

[10] A. Dunkels *et al.*, *The lwIP TCP/IP Stack*, http://lwip.scribblewiki.com/

[11] D. Hankins, *Dynamic Host Configuration Protocol Options Used by PXELINUX* (RFC 5071), December 2007, http://www.ietf.org/rfc/rfc5071.txt

Proceedings of the Linux Symposium

Volume One

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*