University of Amsterdam

Faculty of Science
System and Network Engineering

Reliable network booting of cluster computers

Research project 2

Ing. Matthew Steggink
<matthew.steggink@os3.nl> <msteggink@msteggink.com>

Coordinator: Dr. Ir. C. de Laat, University of Amsterdam
Supervisor: Drs. M. de Vries, ClusterVision B.V.

Abstract

Network booting involves interaction between a client and DHCP server and TFTP server. These protocols use UDP which is unreliable compared to TCP. When a large number of cluster computers are booting simultaneously, some will get stuck in the process. These errors can be caused during a DHCP session or TFTP transfers. The errors are caused by packet loss, which UDP does not respond to well. The errors cause the client or server to terminate the connection, halting the boot process.

I have investigated an alternative method of booting, by using a more reliable protocol: HTTP on TCP. TCP is more reliable then UDP because it uses an acknowledgement mechanism to ensure reliable transfers. To use HTTP booting I have investigated and tested a gPXE implementation against the current situation.

I have found that the booting process can be more reliable using gPXE. Also the gPXE boot ROM can be chainloaded, therefore no modifications to the clients are required. This is very practical with a large number of clients. Using gPXE cancels the TFTP stage, making the process more reliable. However the DHCP bottleneck still exists.

I have also investigated an alternative DHCP implementation: DNSMasq. I found that DNSMasq is currently not suitable to implement gPXE because they are not fully compatible.

Acknowledgement

Amsterdam, June 30, 2008

I would like to take this opportunity to thank the people who have made this thesis possible.

First of all, I would like to thank my supervisor drs. Martijn de Vries from ClusterVision B.V., for the very challanging assignment and for giving me the opportunity to conduct this research project at their facility. Also I thank drs. Martijn de Vries for his time, the guidance and making it possible to test my findings on physical hardware. Without his aid I could not have finished this research project in time.

I would like to thank the System and Network Engineering team (dr. ir. Cees de Laat, dr. Karst Koymans, Jaap van Ginkel and drs. Eelco Schatborn) for asking critical questions at the beginning of this research project. This has given me a head start in finding a suitable strategy and a solution.

I want to express my gratitude to the developers of the etherboot/gPXE project and the SYSLinux project for making great products like PXELinux and gPXE.

I also thank my family and friends who have supported my through these 4 weeks.

Matthew Steggink

System and Network Engineering,
Faculty of Science, University of Amsterdam, the Netherlands

This document is © 2008
Matthew Steggink <matthew@msteggink.com>
<Matthew.Steggink@os3.nl>

Some rights reserved: this document is licensed under the Creative Commons Attribution 3.0 Netherlands license. You are free to use and share this document under the condition that you properly attribute the original authors. Please see the following address for the full license conditions: http://creativecommons.org/licenses/by/3.0/nl/deed.en

Version 1.0.0 and compiled with LaTeX.

Contents

1 Introduction **7**
1.1 Introduction . . . 7
1.2 Research question . . . 7

2 Theory **8**
2.1 Theory . . . 8
2.1.1 Cluster computer set-up . . . 8
2.1.2 Concept of network booting . . . 9
2.1.3 Bootstrap procedure . . . 10
2.1.4 PXE Linux . . . 11
2.1.5 PXE boot sequence . . . 11
2.2 Booting procedure . . . 12
2.2.1 DHCP . . . 12
2.2.2 ARP . . . 17
2.2.3 TFTP . . . 18
2.2.4 Post PXE . . . 20
2.2.5 Components . . . 20

3 Traffic control **22**
3.1 Virtualization and traffic control . . . 22
3.2 Traffic control in VMWare Workstation . . . 22
3.3 Traffic control in Linux . . . 23
3.3.1 Types of qdisc's . . . 25

4 Testing **30**
4.1 Traffic control tests . . . 30
4.2 Configuring traffic control . . . 31
4.2.1 Test setup . . . 31
4.2.2 Reliability of traffic control . . . 32
4.2.3 Used software . . . 33
4.2.4 Test methodology . . . 33

5 Observations **35**
5.1 Measurements 35
5.1.1 4.000 kilobit 36
5.1.2 2.000 kilobit 36
5.1.3 1.500 kilobit 37
5.1.4 1.250 kilobit 38
5.1.5 1.000 kilobit 38
5.1.6 500 kilobit 39
5.1.7 250 kilobit 39
5.1.8 100 kilobit 40
5.1.9 50 kilobit 40

6 Analysis and discussion **41**
6.1 Analysis 41
6.1.1 Identified problems 41
6.1.2 DHCP Stage 41
6.1.3 ARP Stage 42
6.1.4 TFTP Stage 42
6.1.5 Boot Stage 44
6.1.6 Problems matrix 45
6.1.7 Conclusion 46

7 Alternatives **48**
7.1 Alternatives 48
7.2 PoC: DHCP3 and gPXE 49
7.2.1 gPXE flashed into NIC boot ROM 51
7.2.2 Starting with gPXE 52
7.2.3 gPXE dataflow 53
7.2.4 gPXE tests 55
7.3 gPXE measurements 56
7.3.1 Compatibility 56
7.3.2 Performance 56
7.3.3 1000 kbit 57
7.3.4 500 kbit 57
7.3.5 250 kbit 57
7.3.6 100 kbit 58
7.3.7 Conclusion 58
7.4 Proof-of-concept: DNSMasq 60
7.4.1 Installation 60
7.4.2 Configuration 60
7.4.3 Test setup and complications 61
7.4.4 Test method 64
7.4.5 Conclusion 64

8 Conclusion and future work **65**
8.1 Conclusion 65
8.2 Future work 67

9 Appendices **71**
9.1 Appendix A: Traffic control tests 71
9.1.1 Traffic control tests 71
9.2 Appendix-B Shellscripts 74
9.2.1 tc-limit 74
9.2.2 bridge 75
9.3 Appendix-C ISC DHCP3 configuration 76
9.4 Appendix-D DNSMasq Configuration 77

Chapter 1

Introduction

1.1 Introduction

This research project is part of the Master of Science course "System and Network Engineering" of the Faculty of Science, University of Amsterdam. The objective of a research project is to conduct autonomous research to answer the research question, using literature searches, study, experiments, a proof-of-concept and/or development of software.

When network booting a large number of computers, some will get stuck in the process. The purpose of this research project is to analyze the bottlenecks of the current situation, and to find a solution to upscale the number of booting nodes without failing. An analysis must be done to investigate which problems occur, and under which circumstances they appear. Time permitting, alternatives can be investigated to see what influence they have on the current setup.

1.2 Research question

The research question consists of the subjects:

- Literature study on network booting (PXE) as implemented by PXELinux;
- Simulation of network booting (a part of) a cluster with virtualization software to discover different characteristics on how it operates with aspects like congestion, latency or limited bandwidth;
- Try to determine the root cause of the computers being stuck in the proces;
- Try alternative implementations of network booting and compare them to the original implementation;
- Create a report and advice;

Chapter 2

Theory

2.1 Theory

Cluster computers are a group of computers that work together. They usually (but not necessarily) have the same hardware and software configuration. To ensure that all the cluster computers will have the same operating system and software and configuration options, you can use a single image to boot from. This can be done using the network as a boot method, instead of using local disks. This makes things easier, as you can centralize disk management.

This chapter describes the setup, and the procedures the computers must follow to boot off the network succesfully.

2.1.1 Cluster computer set-up

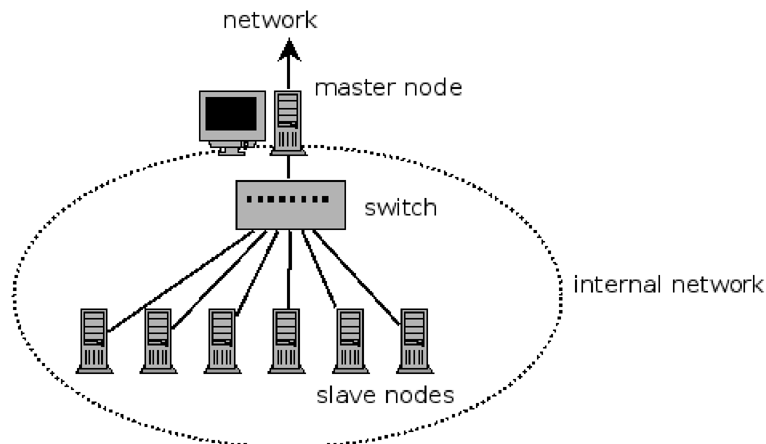


Figure 2.1: Cluster computers set-up

Clusters, deployed by ClusterVision consist of one (or two if there is a fail-over

master node) master node, a local network and multiple slave nodes (see figure 2.1 on the preceding page. The master node is connected to the local network, and the internal network which is used by the master and slaves only to communicate with eachother. The slaves are not directly connected to the outside. The internal network can be build on different technology like Gigabit Ethernet, Myrinet, Infiniband or any other technology the customer wants to use.

The master node controls the slave nodes. The management is done by the master node, it gives out IP addresses using DHCP and uploads the images using TFTP. The master node is the only node which has contact with the local network.

The slaves are stateless, they do not remember the configuration settings. The slaves obtain an IP address, a kernel and an initial ramdisk from the master node by booting off the network. This procedure will be discussed in the next section.

The slaves can be powered up, rebooted or powered down remotely using an APC powerswitch. This setup gives the administrator full control over the setup. It easy to easily add or replace nodes, and keeps image and slave management easy.

2.1.2 Concept of network booting

Network booting is a procedure, where computers use a network source instead of a local disk to boot from. The computers download an image from the network where they boot from. By using the same image, they ensure that they are installed and configured in the same manner as other computers which use the image. Using the network booting method has several advantages:

- Network booting makes it easier to roll out new computers, as they only have to use the network to boot themselves (remote setup);
- By centralizing the image you increase your control over your computers in the network: each client will have the same configuration and installation;
- You can create personal bootsettings per MAC or IP address if needed;
- It is possible to use diskless computers, which limits the impact of failing harddisks. Also this introduces more advantages: lower power consumptions and less heat generation, this is important when a lot of computers are put in a rack;

There are several ways to use network booting [1]:

- Use the BIOS to boot from Preboot eXecution Environment or PXE (by Intel) or Remote Program Loading also known as RPL (by Novell);

- If the NIC does not have PXE, you can use Etherboot/gPXE[1] or network boot using a supported[2] network interface card (NIC) which has a network boot ROM. More information about Etherboot in section 7.1 on page 48;
- Use etherboot/gPXE with a floppy, CD-ROM or USB-key to start the initial bootstrap: The image contains just enough code to boot from the network. The BIOS must support booting off these media;

The bootprocess is similar between these methods. In the next section I will write more about this. Bootstrapping can be done using the PXE protocol (see section 2.1.5 on the following page) or by alternatives like etherboot/gPXE [2]. I will discuss the PXE protocol used by PXELinux.

2.1.3 Bootstrap procedure

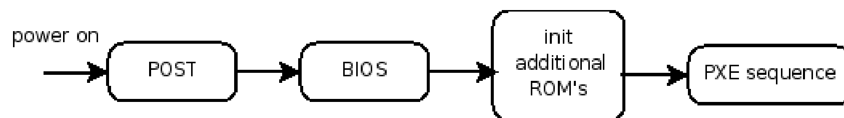


Figure 2.2: Bootstrap procedure

After the computer has been turned on and completed it's Power-On-Self-Test (POST), the second step will be executed by the BIOS (Basic Input Output System). When the BIOS is initializing ROM's in the system, the networkcard boot ROM will be found, and hooked into the BIOS boot code.

The BIOS will go through a list of bootable devices which have been defined in the boot order. When selected, the BIOS can then use the networkcard to boot using PXE (see more in section 2.1.5 on the next page about PXE) by handing over control to the networkcard boot ROM.

If the networkcard does not have PXE support, etherboot/gPXE can be flashed into the networkcard, only when the networkcard is supported by etherboot/gPXE.

When the BIOS has transferred the control to the network boot code, the network boot process starts as described in section 2.1.5 on the following page [1]. The first step in the bootstrap process has been started.

[1] Etherboot ROM's can be obtained here: http://rom-o-matic.net/

[2] The etherboot supported PCI ID's: http://rom-o-matic.net/etherboot/etherboot-5.4.3/src/bin/NIC

2.1.4 PXE Linux

ClusterVisionOS is specifically made for clusters. It contains a set of software packages and tools which can be used to manage a cluster. ClusterVisionOS is built upon other Linux distributions. It uses PXELinux [3] to boot linux off a network server, using a network boot ROM conforming to the Intel PXE (Pre-Execution Environment) specification. PXELinux uses similar steps like PXE does (see section 2.1.5) to boot computers.

2.1.5 PXE boot sequence

In this section I will document the steps PXELinux takes in booting off the network.

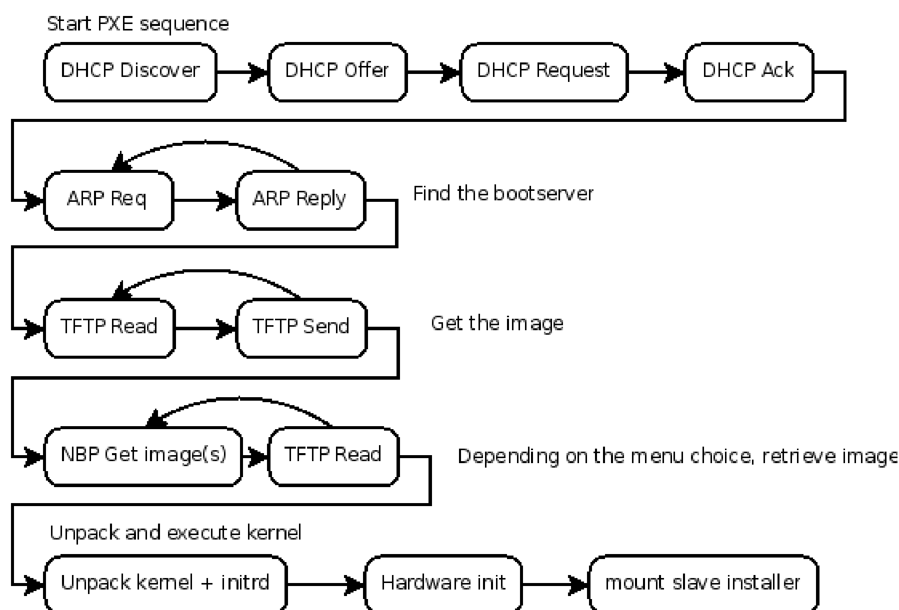


Figure 2.3: PXE Sequence

Preboot eXecution Environment (PXE) is a method to boot computers using the networkcard, instead of the harddisk or other devices to boot the system [4]. PXE leans on different mainstream protocols: Dynamic Host Configuration Protocol (DHCP), Trivial File Transfer Protocol (TFTP), User Datagram Protocol (UDP) and the IP stack. It uses the DHCP (or BOOTP) to obtain an IP and locate a bootserver, and uses TFTP to download the bootstrap program.

PXE uses extensions which it puts in the option fields of the DHCP protocol, as defined in RFC2132 [5]. PXE does not interfere with the standard operation of DHCP, as these options are standardized in this RFC. The only difference is the extended information which are put in the DHCP `options` field. A non-PXE client can just discard or ignore

the additional information without influencing the normal operation. PXELinux does not use the PXE specific fields.

Sending out a DHCP packet with these Vendor Option values, with PXE information encapsulated, is called an *extended message* [4].

The fields and options are defined in RFC2131 [6]. The PXE specific options are encapsulated in tagnumber 43 “`Vendor Options`” as described in RFC 2132. The tagnumbers are registered by IANA[3] [7] .

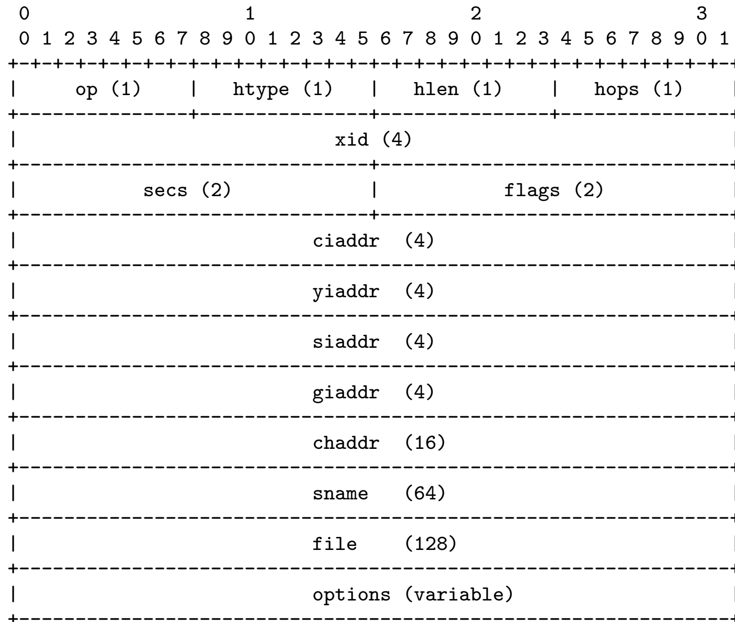
Because PXE is based on several protocols, I’ve split the boot procedure up per protocol.

2.2 Booting procedure

The procedure has been split up into the protocols DHCP, ARP and TFTP.

2.2.1 DHCP

The Dynamic Host Configuration Protocol (DHCP) passes configuration information (IP address, booting options) to hosts on the network. The DHCP protocol is a client-server model. The packetformat of DHCP are based on BOOTP packetformats. The packet format looks like:



[3] The Internet Assigned Numbers Authority

The packet fields:

Field	Octets/(bits)	Description
op	1 (8)	Message operation code / message type.
htype	1 (8)	Hardware address type. These numbers are registered by IANA [8]
hlen	1 (8)	Hardware address length (6 bytes for ethernet MAC addresses)
hops	1 (8)	Client sets to zero, optionally used by relay agents when booting via a relay agent
xid	4 (32)	Transaction ID, a random number chosen by the client, used by both to identify messages and responses
secs	2 (16)	Filled in by client, seconds elapsed since client began the DHCP process
flags	2 (16)	Flags. First bit denotes is broadcast, the remaining are reserved
ciaddr	4 (32)	Client IP address
yiaddr	4 (32)	'your' (client) IP address. Assigned to the client by the server
siaddr	4 (32)	IP address of next server to use in bootstrap; returned in DHCPOFFER and DHCPACK by server
giaddr	4 (32)	Relay agent IP address, used in booting via a relay agent
chaddr	16 (128)	Client hardware address. For MAC addresses, only 6 octets are used
sname	64 (512)	Optional server host name
file	128 (1024)	Boot file name
options	variable	Optional parameters field.

Operation: for DHCP, RFC describes 8 different packettypes:

	Client - Server		Server - Client
Value	Message Type	Value	Message Type
1	DHCPDISCOVER	2	DHCPOFFER
3	DHCPREQUEST	5	DHCPACK
4	DHCPDECLINE	6	DHCPNAK
7	DHCPRELEASE		
8	DHCPINFORM		

The `Discover` is broadcasted by the client to the network, to discover a DHCP server. Because this is restricted to the local broadcast domain only, network administrators can configure a local router to forward DHCP packets to a DHCP server on a different broadcast domain [6].

The `Offer` is a packet type sent by the server to the client to offer an IP address [6].

The `Request` is the packet sent by the client to all the DHCP servers, that it has accepted and confirmed the IP address that has been offered, and to give notice to other DHCP servers that it has declined their IP offer [6, 9].

The `Decline` is sent from the client to the server indicating network address is already in use. The client can discover that the address is already in use, through ARP. After sending the `Decline` message, the configuration process will be restarted [6].

The `ACK` is sent from the server to the client, to confirm the configuration parameters [6].

The `NACK` is sent from the server to the client, to notice the client that his network address is incorrect. This can be invoked because the client has moved to another subnet or the lease expired [6].

The `Release` message is sent from the client to the server, that the client does not need his IP address anymore, and is cancelling the lease [6].

The `Inform` message is sent by the client to server, asking only for local configuration parameters. In this case, the client already has externally configured network address [6].

Step 1: DHCPDISCOVER

```
▷ User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
▽ Bootstrap Protocol
    Message type: Boot Request (1)
    Hardware type: Ethernet
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x2a81065a
    Seconds elapsed: 4
  ▷ Bootp flags: 0x8000 (Broadcast)
    Client IP address: 0.0.0.0 (0.0.0.0)
    Your (client) IP address: 0.0.0.0 (0.0.0.0)
    Next server IP address: 0.0.0.0 (0.0.0.0)
    Relay agent IP address: 0.0.0.0 (0.0.0.0)
    Client MAC address: Vmware_81:06:5a (00:0c:29:81:06:5a)
    Server host name not given
    Boot file name not given
    Magic cookie: (OK)
  ▷ Option: (t=53,l=1) DHCP Message Type = DHCP Discover
  ▷ Option: (t=55,l=24) Parameter Request List
  ▷ Option: (t=57,l=2) Maximum DHCP Message Size = 1260
  ▷ Option: (t=97,l=17) UUID/GUID-based Client Identifier
  ▷ Option: (t=93,l=2) Client System Architecture = IA x86 PC
  ▷ Option: (t=94,l=3) Client Network Device Interface
  ▷ Option: (t=60,l=32) Vendor class identifier = "PXEClient:Arch:00000:UNDI:002001"
    End Option
    Padding
```

Figure 2.4: PXE DHCP Discover

To initiate a transmission between the client and server, the PXE client will send out a DHCPDISCOVER, with extensions in the options field [4, 7]. The DHCP discover message is specified in paragraph 4.3.1 in RFC 2131 [6]. The first field, `operations` (op), is set to `1`. This means the message type is a BOOTP request.

The **options** contain [6, 7]:

Name	Tag	Comment
DHCP message type	53	Message type
Parameter request list	55	This option is used by a DHCP client to request values for specified configuration parameters (RFC2132) [5]
Message length	57	This option specifies the maximum length DHCP message that it is willing to accept (RFC2132) [5]
Vendor Class Identifier	60	This option is used by DHCP clients to optionally identify the vendor type and configuration of a DHCP client (RFC2132) [5]
Client identifier	61	Client Identifier (RFC2132) [5]
Client Network Device Interface	94	Interface type, only supported option is 1 (UNDI) (RFC4578) [10]
Client system architecture	93	The architecture (e.g. IA x86 PC)
Client UUID	97	Unique Cient Identifier

These options are used by DHCP clients to specify their unique identifier and other variables. DHCP servers use this values to index their database of address bindings [5]. By using the option "`vendor class identifier`", it identifies the request as coming from a client which implements the PXE protocol. Normal operations do not use this option field.

The Vendor Class Identifier field looks like:

```
PXEClient:Arch:xxxxx:UNDI:yyyzzz

xxxxx = Client Sys Architecture 0 - 65535
yyy = UNDI Major version 0 - 255
zzz = UNDI Minor version 0 - 255
```

Step 2: DHCPOFFER

When the server understands this request, it will send out an DHCPOFFER. The server will notice by the options that it is enabled to recieve an extended reply. The first field, `operations` (op), is set to 2. This means the message type is a BOOTP reply.

```
▷ User Datagram Protocol, Src Port: bootps (67), Dst Port: bootpc (68)
▽ Bootstrap Protocol
    Message type: Boot Reply (2)
    Hardware type: Ethernet
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x2a81065a
    Seconds elapsed: 4
  ▷ Bootp flags: 0x8000 (Broadcast)
    Client IP address: 0.0.0.0 (0.0.0.0)
    Your (client) IP address: 10.141.0.1 (10.141.0.1)
    Next server IP address: 10.141.255.254 (10.141.255.254)
    Relay agent IP address: 0.0.0.0 (0.0.0.0)
    Client MAC address: Vmware_81:06:5a (00:0c:29:81:06:5a)
    Server host name not given
    Boot file name: default-image/boot/pxelinux.0
    Magic cookie: (OK)
  ▷ Option: (t=53,l=1) DHCP Message Type = DHCP Offer
  ▷ Option: (t=54,l=4) Server Identifier = 10.141.255.254
  ▷ Option: (t=51,l=4) IP Address Lease Time = 1 day, 4 hours
  ▷ Option: (t=1,l=4) Subnet Mask = 255.255.0.0
  ▷ Option: (t=3,l=4) Router = 10.141.255.254
  ▷ Option: (t=6,l=4) Domain Name Server = 10.141.255.254
  ▷ Option: (t=12,l=7) Host Name = "node001"
  ▷ Option: (t=15,l=30) Domain Name = "cvos.cluster clustervision.com"
    End Option
```

Figure 2.5: PXE DHCP Discover

The options contain [6, 5, 7]:

Name	Tag	Comment
DHCP Message type	53	Message type
Server identifier	54	Server identifier is used by both to identify a DHCP server as a destination address
Address lease time	51	Time in seconds. `000189C0` (hex) 100800 (dec) is 28 hours
Subnet mask	1	Subnet mask value
Router	3	Router address
Domainname	6	DNS Server
Host name	12	Client hostname
Domain name	15	Domain name

Step 3: (Optional) Select IP address

If the client selects an IP address offered by a DHCP Service, then it must complete the standard DHCP protocol by sending a request for the address back to the Service and then waiting for an acknowledgment from the Service [4, 6].

Name	Tag	Comment
Address Request	50	Requested IP address
DHCP Server ID	54	DHCP Server Identification

With tag 54, it knows which server to request it from. If there are multiple DHCP servers, they know who it is for, when the packet is broadcasted.

Step 4: (Optional) DHCP Ack

The DHCP server will now acknowledge the configuration parameters [6]. The packet is identical with step 2, but now it acknowledges the request.

The above 4 steps were responsible for discovering a DHCP server (1), receiving a reply from DHCP servers (2), selecting an IP and options (3) and recieving an acknowledgement (4).

Now the client can proceed to the download state. It can now receive a Network Bootstrap Program (NBP). To read more about NBP, see section 2.2.5 on page 20

2.2.2 ARP

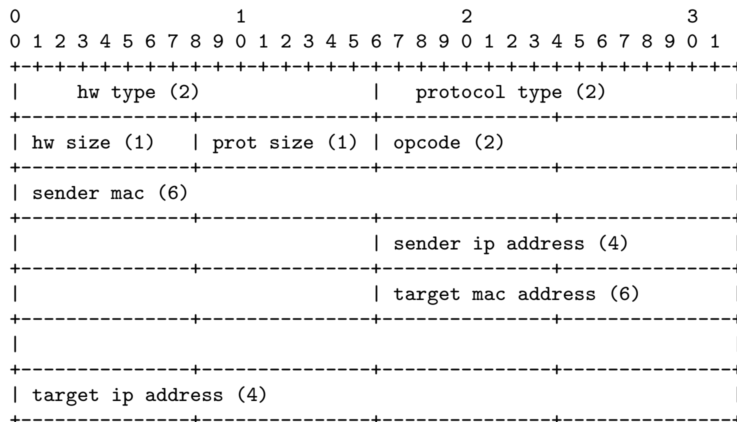
Step 5: ARP request and reply

Because it got the IP address of the TFTP server, it can now access it. However, it cannot contact it, because it only has it's IP address and not his data-layer link address (also known as MAC address) (layer 2). To obtain the MAC address it has to do an ARP request.

It will broadcast an ARP packet, filling in his MAC and IP address in the sender field, and the server IP address in the target ip address field. Because he does not know the MAC address of the server he will put all zero's in the target mac address, and will also put all one's in the ethernet destination (ffffff) to denote it is a broadcast packet.

The target can reply because his sender's MAC address is in the ethernet source field and the sender MAC address field.

The operation field has 2 options: request (0x01) or reply (0x02).




```
▷ Frame 6 (60 bytes on wire, 60 bytes captured)
▷ Ethernet II, Src: Vmware_4f:cd:df (00:0c:29:4f:cd:df), Dst: Vmware_81:06:5a (00:0c:29:81:06:5a)
▽ Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: Vmware_4f:cd:df (00:0c:29:4f:cd:df)
    Sender IP address: 10.141.255.254 (10.141.255.254)
    Target MAC address: Vmware_81:06:5a (00:0c:29:81:06:5a)
    Target IP address: 10.141.0.1 (10.141.0.1)
```

Figure 2.6: ARP Reply

The server will pick up the broadcast and reply to the sender with his MAC address. Now the client can access the server, and will proceed to use TFTP.

2.2.3 TFTP

Step 6: Read request bootfile

The PXE bootrom then uses the TFTP protocol to download the bootfile. The PXE bootrom has a limitation of only being able to load small (up to 32kb) files. It isn't able to load a full Linux kernel. So instead, PXELinux set it up to load a small bootloader called 'pxelinux.0'. That bootloader is then able to load larger images, such as the Linux kernel and the initrd image [11].

When booting, it will try to request a read operation on the pxelinux.0 bootimage. First it wants to know the transmission size of the file (tsize). When the server has returned that value, it will calculate the most optimal transmission size (blksize). By default the TFTP protocol intends data transmission by 512 bytes blocks. Regarding the fact that present local networks MTU is usually equal to 1500 bytes or more, this

```
▷ Frame 7 (88 bytes on wire, 88 bytes captured)
▷ Ethernet II, Src: Vmware_81:06:5a (00:0c:29:81:06:5a), Dst: Vmware_4f:cd:df (00:0c:29:4f:cd:df)
▽ Internet Protocol, Src: 10.141.0.1 (10.141.0.1), Dst: 10.141.255.254 (10.141.255.254)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 74
    Identification: 0x0002 (2)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 20
    Protocol: UDP (0x11)
  ▷ Header checksum: 0x9188 [correct]
    Source: 10.141.0.1 (10.141.0.1)
    Destination: 10.141.255.254 (10.141.255.254)
▷ User Datagram Protocol, Src Port: 2070 (2070), Dst Port: tftp (69)
▽ Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: default-image/boot/pxelinux.0
    Type: octet
  ▽ Option: tsize = 0
      Option name: tsize
      Option value: 0
```

Figure 2.7: Read request bootfile

block size is not effective. TFTP blksize option let change the block size, improving data transmission effectiveness.

The PXELinux specifications describes a procedure to load the bootfile. It will append values to the pxelinux.cfg in this order:

1. UUID (e.g. `/mybootdir/pxelinux.cfg/b8945908-d6a6-41a9-611d-74a6ab80b83d`)

2. MAC address (e.g. `/mybootdir/pxelinux.cfg/01-88-99-aa-bb-cc-dd`)

3. IP address in hex notation (e.g. `/0A8D0001` (hex notation for 10.141.0.1)

4. IP address in hex notation, minus one nibble per step (e.g. `/0A8D000`, `/0A8D00`, `/0A8D0`, etc.)

5. default (`/mybootdir/pxelinux.cfg/default/`

The bootfile can contain a bootmenu, from which several bootfiles can be chosen. Also, the bootfile that has been downloaded can also initiate a download of other bootfiles. Regardless of what action is chosen, for every transaction it must send a read request, calculate optimal blksize and recieve and acknowledge the datablocks. Because TFTP is a lock-step protocol, asking a acknowledgement for each packet, it is not very efficient.

After downloading and acknowledging each block, the client can now load the Network Bootstrap Program into the memory. The file downloaded and the placement of the downloaded code in memory is dependent on the clients CPU architecture [4].

Step 7: Code execution

Finally, if the authenticity test succeeded or was not required, then the PXE client initiates execution of the downloaded NBP code. This code can be another NBP which points to other boot files, or it can be the kernel itself.

2.2.4 Post PXE

After the NBP has downloaded the images via TFTP, the network bootstrap program unpacks and executes the kernel. The kernel image is compressed with the zlib compression algorithm. It decompresses the kernel an the initial ramdisk images and loads them into the system memory. Then the initial ramdisk is mounted as a temporary root. It's lifetime is short, only serving as a bridge to the real root file system [12]. After both have been set up, the kernel is then executed and various hardware initializations are performed.

2.2.5 Components

Network Bootstrap Program

Network Bootstrap Program's are images, which are executed by the CPU of the client. A NBP can do much more advanced operations, because it has access to the APIs of the PXE firmware extension (protocol and UNDI extensions) [4]:

- Download other NBPs, applications, or OS images
- Communicate on the network (UNDI APIs (see section 2.2.5 on the next page))
- Shut down and stop the base code loader runtime and/or the UNDI driver.

A NBP is essentially a second stage loader. The PXE bootrom can only load small (max. 32 kbyte) images. Therefore it needs to load a small bootrom first. This bootrom, like PXELinux, can then load other larger images, like the kernel and the initial ramdisk.

The downloaded bootrom contains a menu with several options. After a selection has been made, it will do the corresponding action defined in the menu. In this context the Network Bootstrap Program's will execute a normal slave boot. It's task is to download and load two components: the kernel (`vmlinuz`) and initial ramdisk (`initrd`).

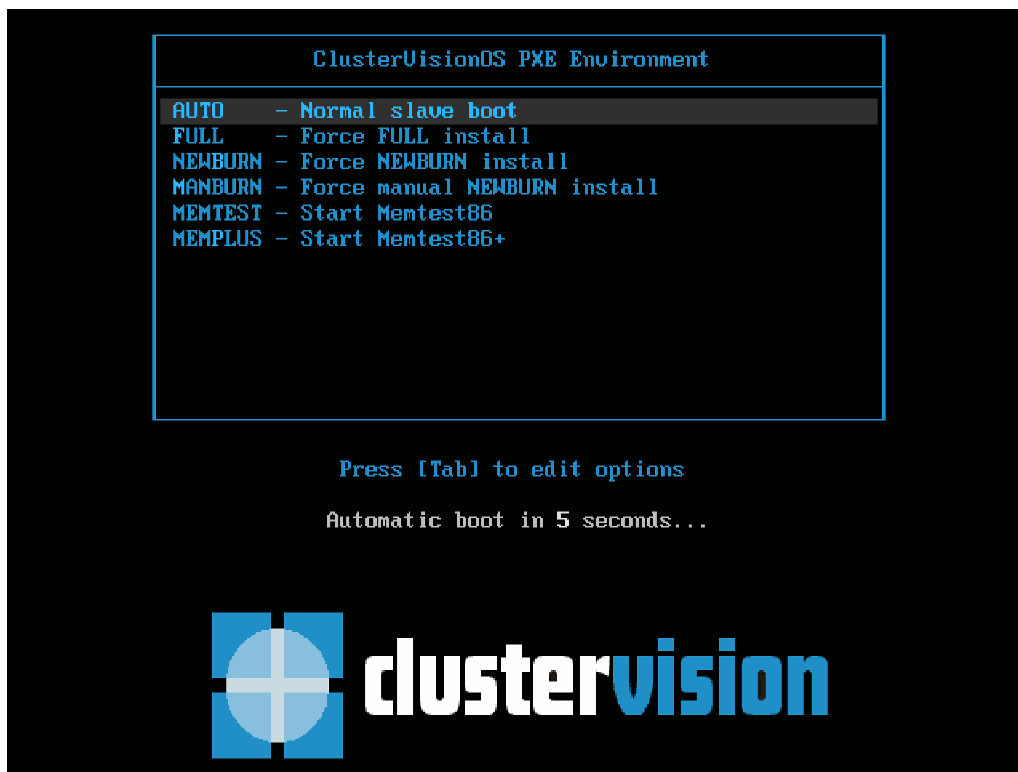


Figure 2.8: A NBP with a boot screen

Universal Network Device Interface

An Universal Network Device Interface (UNDI) enables basic control of and I/O through the clients network interface device. According to the PXE standard, PXE-enabled network cards should support the UNDI interface. This allows the use of universal protocol drivers such a way that the same universal driver can be used on any network interface that implements this application programming interface (API) [4].

Chapter 3

Traffic control

3.1 Virtualization and traffic control

Because it is not possible to use a large number of physical machines for these tests, another alternative had to be chosen. A virtual network has multiple advantages:

- All the traffic characteristics (bandwidth, latency, packetloss) can be tuned;
- A new network setup is created in a minimum timespan using scripts and linked clones;
- Every node in the network can be monitored centrally;
- Using the bridge method, you can sniff traffic from the slave and master nodes off the bridge. Using the VMWare team method, you cannot sniff traffic because the traffic is switched;
- Creating snapshots in the virtualization software can be done quickly to revert back to recent changes.

In this setup, there are two ways to limit the traffic. You can limit a whole LAN using VMWare Workstation Team function, or you can limit the traffic by placing a bridge between the slave nodes and the master node and tune the bridge. I will describe both methods in the next 2 sections.

3.2 Traffic control in VMWare Workstation

VMWare Workstation has an option to limit traffic on a given LAN segment: Teams. These teams are grouped together in a team, and the network settings can be influenced. You can adjust several parameters which should impact the LAN team: bandwidth and packetloss.

In figure 3.2 on the following page, there are 4 computers in the LAN segment. The "virtual switch" can be considered the LAN Segment that you can influence. The

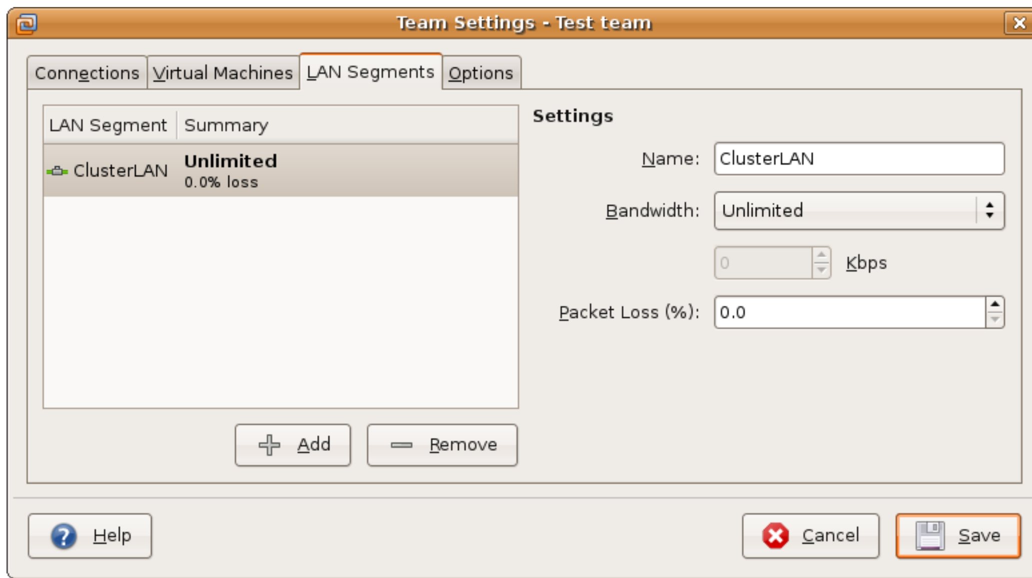


Figure 3.1: VMWare Team - LAN segment

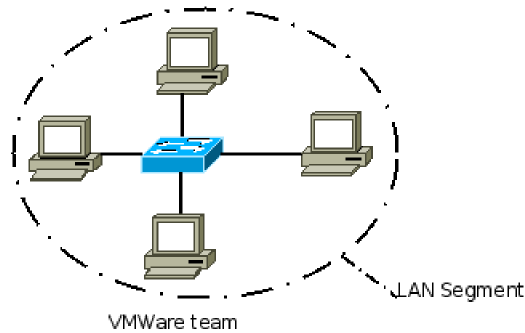


Figure 3.2: VMWare team: 4 computers in one LAN segment

network traffic cannot be sniffed in this LAN, to do that, you need a bridge that connects the LAN to the master node. A better explaination and figure can be found in section 4.1 on page 30.

3.3 Traffic control in Linux

The traffic control management is fairly unknown in Linux, but it comes standard with the iproute2 suite. The utility is named `tc`. With `tc` you can simulate network characteristics, together with netem. Netem is an extension to tc [13, 14]. To use `tc`, you do not need to load an additional module. Netem however needs a module. In a standard

7.10 server Ubuntu installation the kernel module is not loaded by default. This can be done by using modprobe to load it into the memory:

```
modprobe sch_netem
```

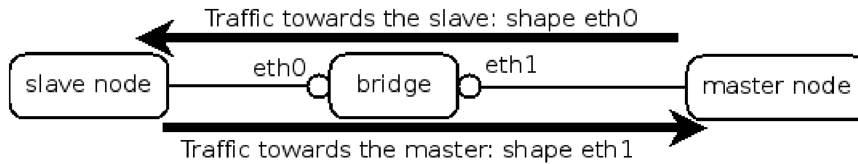


Figure 3.3: Interface traffic flow shaping

The traffic control can only shape data flows that are transmitted by the computer. You cannot influence the data flows being sent to you, only the data sent by yourself. If you want to shape traffic from the slave node to the master node, you have to shape the interface that is sending data to the master node or to the slave node(see figure 3.3) [15].

The data waiting to be transmitted is put in a queue. The queue can be influenced in how long data may stay in the queue, how long the queue is, how fast the data gets transmitted from the queue to an interface, when it should be dropped or discarded, how it should be treated: should it get reordered or have a higher priority (QoS) over other data flows. The algorithm influencing the queue is called a Queueing Discipline. The Queueing Discipline can shape both outgoing traffic (called egress) or incoming traffic (ingress) on the interface [15] (See section 3.3.1 on the next page for more information).

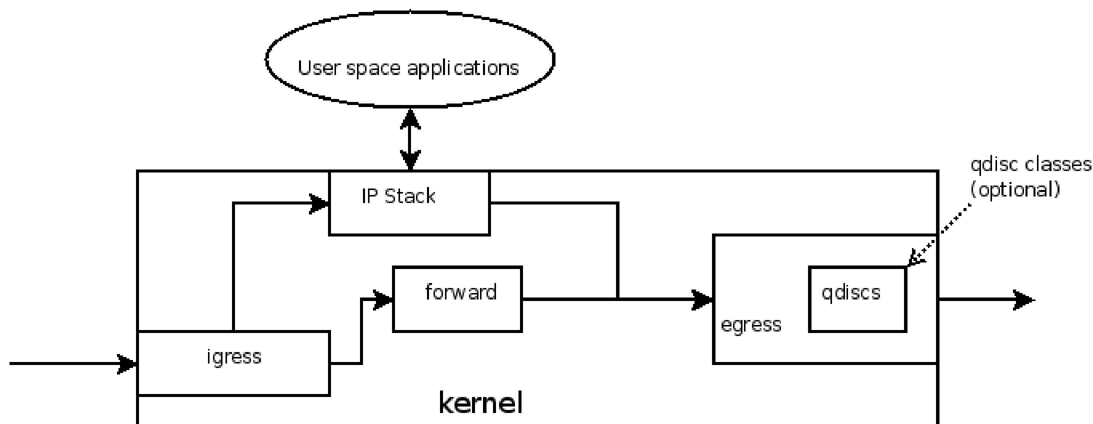


Figure 3.4: Ingress - Egress policy

The traffic controlling happens at a very early stage (see figure 3.4 on the preceding page). It begins with data entering the kernel. The kernel passes the packets to the first qdisc: the igress qdisc. The igress qdisc can apply filters to the data. If the data is allowed to pass the igress (e.g. no filter set or positive match), it will be handed to the IP stack where a userspace application can use the data, or the packet can be forwarded to the egress filter. The egress qdisc can filter the data and send it out, or it can classify the data into egress qdisc classes where they will be processed.

Applying igress filters has a distinct advantage: by early filtering of packets, it does not have to pass through all the functions, saving CPU time.

Figure 3.4 on the previous page only shows one igress and egress filter. But each network adapter has an igress and egress hooks.

3.3.1 Types of qdisc's

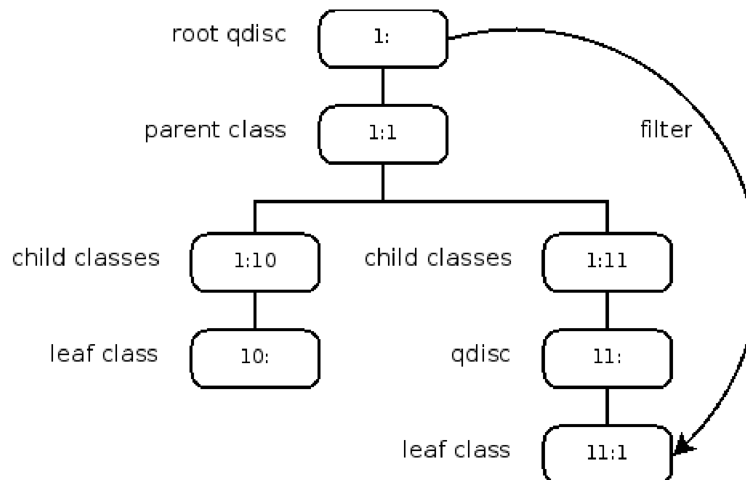


Figure 3.5: Classful Queueing discipline tree

The qdisc is the major building block on which the of Linux traffic control is built, and is also called a *Queueing Discipline* [13]. In short, it is a scheduler for the output interface [16]. It is by default based on FIFO (First In, First Out) which makes no distinction in data flow type or priority. This is the default behavoir for any unconfigured linux machine. When needed, the qdisc's can be configured to rearrange, delay or drop the packets being injected into the queue, depending on the scheduling rules set.

There are two tyes of qdiscs, classful en classless.

The classful qdiscs can contain classes, and provide a handle to which you can attach filters. A filter directs traffic directly to another object (parent,child or leaf) without having to pass all the objects in it's way. The classes form a tree (see figure 3.5). There is no restriction on using a classful qdisc without child classes.

The classless qdiscs cannot contain classes, nor is it possible to attach a filter to a classless qdisc. Because a classless qdisc contains no children of any kind, there is no need to classifying. This means the traffic cannot be specifically directed.

There are different types of qdiscs [15, 17]:

- First In First Out (FIFO), *Classless*
- Token Bucket Filter (TBF), *Classless* [18, 19]
- Hierarchical Token Bucket (HTB), *Classful* [20]
- Class based queueing (CBQ), *Classful* [21]

There are more queueing schedulers like Clark-Shenker-Zhang (CSZ), Priority Traffic Equalizer (TEQL), Stochastical Fairness Queueing (SFQ), Asynchronous Transfer Mode (ATM), Random Early Drop/Detection (RED), Generalized RED (GRED) and Diff-Serv Marker (DS_MARK). Some of these only prioritize traffic and do not shape them or are too complicated to set up. I will not discuss these, because they are beyond the scope of this project.

First In First Out (FIFO)

First In First Out (FIFO) is the default scheduler in Linux. The FIFO scheduler can be enabled or set with the `tc` command:

```
tc qdisc add dev $device root pfifo limit $queuesize
```

It is not possible to set a class or filter to use with FIFO, because it is a clasless scheduler. The FIFO qdisc name is pfifo (Packet FIFO). The queuesize is the maximum number of packets that can be in the queue. The benefits for using FIFO are minimal:

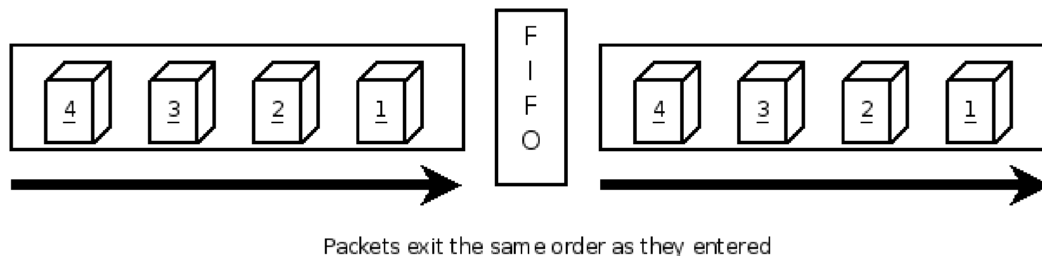


Figure 3.6: First In First Out Queueing

it uses the least CPU time of all schedulers. As long as the queue size remains short, the delay will not be increased significantly. When using a large queuesize, the packets will take longer to exit the queue, adding delay.

The FIFO scheduler cannot reorganise packets. They enter and leave in the same order. It cannot prioritize packet types, because it treats all flows the same way. During periods of congestion, FIFO queuing benefits UDP flows over TCP flows. When experiencing packet loss due to congestion, TCP based applications reduce their transmission rate, but UDP based applications remain oblivious to packet loss and continue transmitting packets at their usual rate because there is no throttling mechanism. Because TCP based applications slow their transmission rate to adapt to changing network conditions, FIFO queuing can result in increased delay, jitter, and a reduction in the amount of output bandwidth consumed by TCP applications traversing the queue.

Token Bucket Filter

The Token Bucket Filter (TBF) is a simple classless queueing scheduler, which works with tokens. To control the data rate, there is a buffer (the bucket) constantly being filled with tokens (the token-rate), until the bucket is full. Each data unit (byte) is associated with a token. Each time a new token enters the bucket, it sends out a token.

Linux uses a byte-count token bucket, this means a token refers to bytes, instead of packets. One token is the equivalent of one byte. It is not feasable to have a packet associate with a token, because most packets are not of the same size. In such a scenario, the output will have peaks and lows in the data rate constantly because large and small packets are being released. That is the reason the tokens are associated with bytes and not packets.

For example, if you want a data rate of 2 megabit/s (250.000 bytes/s) output and you associate one byte with one token. This means you need a token rate of 250.000 per second entering the bucket. If a packet is 590 bytes, you need to take 590 tokens from the bucket, and then you can release the packet. As soon as you have enough tokens, you can release the data. An important factor is that you make your bucket large enough.

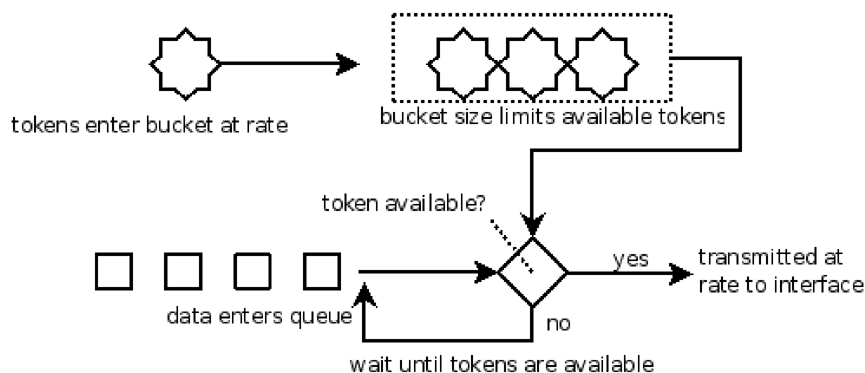


Figure 3.7: Token Bucket Filter

There are 3 actions possible [17, 16, 18]:

- $(TBF_{data} = TBF_{token})$: The data arrives in bucket at a rate that is equal to the rate of incoming tokens. In this case each incoming packet has its matching token and passes the queue without delay.
- $(TBF_{data} < TBF_{token})$: The data arrives in TBF at a rate that is smaller than the token rate. Only a part of the tokens are deleted at output of each data packet that is sent out the queue, so the tokens accumulate, up to the bucket size. The unused tokens can then be used to send data at a speed that is exceeding the standard token rate, in case short data bursts occur.
- $(TBF_{data} > TBF_{token})$: The data arrives in TBF at a rate bigger than the token rate. This means that the bucket will soon be short of tokens, which causes the TBF to throttle itself for a while. If packets keep coming in, packets will start to get dropped when the timer expires or if there is no space in the queue.

TBF can be used in tc with the command [19]:

```
tc qdisc add dev eth0 root tbf rate $rate burst $burstsize time $time
latency $latency peakrate $peakrate minburst $minburst
```

This configures the qdisc to use the tbf scheduler. The maximum sustained rate is the set `$rate`. This is comparable to the rate the tokens enter the bucket. The buffersize is `$burstsize`, also known as the bucket size. This is the maximum number of tokens available for immediate transmission, and is the maximum burst when the bucket is emptied at once. The peakrate (*ceil*) is `$peakrate` for short bursts, it is the absolute ceiling: it may not pass this maximum. The data is specified in *kbps* (kilobyte/s, *mbps* (megabytes/s), *kbit* (kilobits/s), *mbit* (megabites/s). The accumulation of tokens allows a short burst of overlimit data to be still passed without loss, but any lasting overload will cause packets to be constantly delayed, and then dropped [15].

The $time limits the number of bytes that can be queued waiting for tokens to become available. The time is specified in *s* (seconds), *ms* (miliseconds), *us* (microseconds). The latency is the *maximum* latency a packet will stay in the queue, this does not mean each packet will suffer this latency, it is a timer on the packets. Some will leave earlier, but no later then the latency described here. If the timer expires, the packers are to be dropped.

The `minburst` specifies the size of the peakrate bucket. For perfect accuracy, should be set to the MTU of the interface.

On the short term, the TBF allows a burst as large as the set minburst but no more. On the long term, it cannot be more then the set rate.

The maximum possible reliable peakrate that can be shaped by the TBF, can be calculated by multiplying the Hz rate of the kernel by the mtu in the TBF [15]. The Hz rate is the frequency with which the system's timer hardware is programmed to interrupt the kernel. It is the rate at which the TBF can process a burst in the bucket each

second.
$Rate_{max} = hz_{rate} * TBF_{burstsize}$

Because the burst size is 1540 bytes (12320 bits) and the hz rate in ClusterVisionOS on a x86-64 architecture is 1000 [22], we can shape traffic reliable up to 12.320.000 bits/s (12.320 kbit/s).

Hierarchical Token Bucket

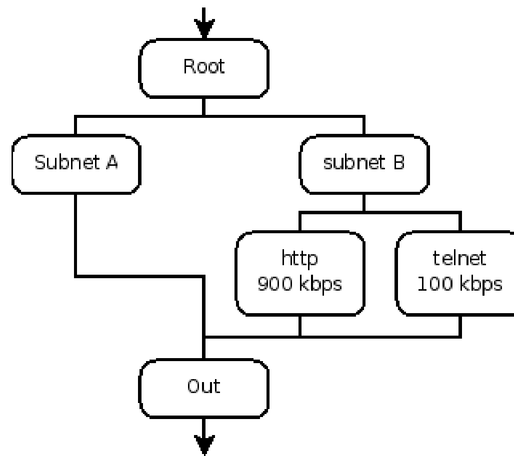


Figure 3.8: Hierarchical Token Bucket Filter

Hierarchical Token Bucket (HTB) [17, 15, 18] is essentially the same as the Token Bucket, described before. It can create multiple simulated links, to send and control data on. You can make a distinction based on protocol or address. For example, for each protocol you can create a simulated link. For each link you can specify delay, bandwidth and so forth. Say you want to send http traffic over a link with 900 kbps and telnet traffic over a link with 100 kbps. This way, you ensure a minimum data rate for each link. Another feature is that you can 'borrow' bandwidth from other links, until the links is at it's ceiling (maximum). Also links can be prioritized. It is too broad for this project, we only want to simulate a link, and not make a distinction for each protocol.

Class based queueing

The Class Based Queing (CBQ) is the most broad and complex classful scheduler. It has the same features as HTB like classes, bandwidth allocation and borrowing. But with CBQ you can assign a weight to each class. Weight helps in the Weighted Round Robin process. Each class gets a chance to send in turn. If there are classes with significantly more bandwidth than other classes, it makes sense to allow them to send more data in one round than the others.

Chapter 4

Testing

4.1 Traffic control tests

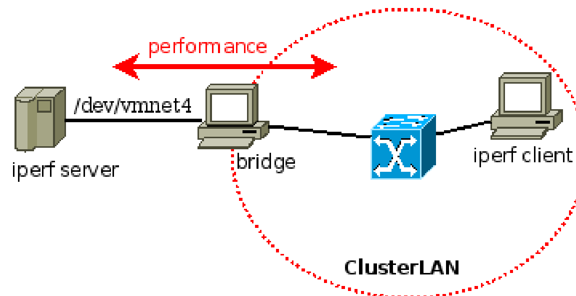


Figure 4.1: The setup scheme

The objective of these tests are to measure the maximum performance in VMWare Workstation. This is to ensure that the VMWare networking will not be a bottleneck. Also to have the experiments to be valid, the traffic control must be reliable. To determine this, we need to look at the following aspects:

1. does the team network limiter in VMWare Workstation work properly (does it shape) and how accurate is it;

2. does the limiting option in Linux using `tc` work properly (does it shape) and how accurate is it;

The software to test the bandwidth is `iperf` [23]. Iperf is capable of creating both TCP and UDP datastreams, and measuring the throughput. The server is running on ClusterVisionOS 3.1 (kernel 2.6.18), using iperf 2.0.2. The client has Ubuntu 7.10 (kernel 2.6.22) with iperf 2.0.2 During the tests, the CPU, memory usage and loadaverage will be monitored by munin and tools like vmstat and top. Ofcourse the network test will impact all running virtual machines. These tests will only test the maximum performance over

the link, to see if the traffic shaping ability works and to see if the bridge can process such data speeds.

The tests are done twice: for each run, I tested FROM the outside to the inside (ClusterLAN) and vice versa. This means switching the iperf server and client. I've used `tc` or VMWare Team to control the traffic. The full results can be found in Appendix A: Traffic control tests (section 9.1 on page 71).

4.2 Configuring traffic control

The theory of traffic control is discussed in 3.1 on page 22. To execute the tests, I've chosen to limit the outgoing interfaces eth0 and eth1 to $x$kilobits per second.

The qdisc I have chosen is a Token Bucket Filter (see section 3.3.1 on page 27). The qdiscs have been setup with a script, see the appendices.
The commandline for ratelimiting:

```
tc qdisc add dev ethx tbf rate xmbit burst 1540 latency 50ms minburst 1540
```

Where: `ethx` is the device, `xmbit` the ceiling bandwidth. To see if your settings are correct use:

```
tc -s qdisc show dev ethx
```

The `qdisc show` gives information on the interface. The `-s` switch gives statistic information on the interface. The `-s` is optional. The `-s` switch gives more information about the sent bytes, sent packets, how many dropped by the bucket and the rate limit set. To realtime follow the interface, you can use the `watch` command:

```
watch tc -s qdisc show dev ethx
```

This will monitor the output of the command every two seconds.

4.2.1 Test setup

The hardware I am using is a dual core AMD Athlon64 4200+ at 2200 Mhz with 4096 MB RAM. The harddisk is a 80 GB disk. The operating system I am using is Ubuntu 7.10 Server with kernel 2.6.22-14-server SMP x86_64.

On this server I've installed VMWare Workstation 6.0.4-93057 (x86_64). The virtual machine testing nodes all are linked cloned from the first slave node. The settings are: 256 MB memory, 1 processor, 1 harddisk (16 GB), Ethernet: ClusterLAN. The master node has been configured with 1024 MB memory, 1 processor, 1 harddisk (30 GB), 2 ethernet network interfaces: 1 bridged to the network, 1 to the bridge over a vmnet link.

The bridge has 384 MB memory, 1 processor, 1 harddisk (12 GB) and 3 ethernet

network interfaces: 1 bridged to the network, 1 to the ClusterLAN and 1 to the master node over a vmnet link. The second and third interfaces are bridged into bridge `br0` using the `bridge-utils`.

See figure 4.3 on the following page for a scheme.

4.2.2 Reliability of traffic control

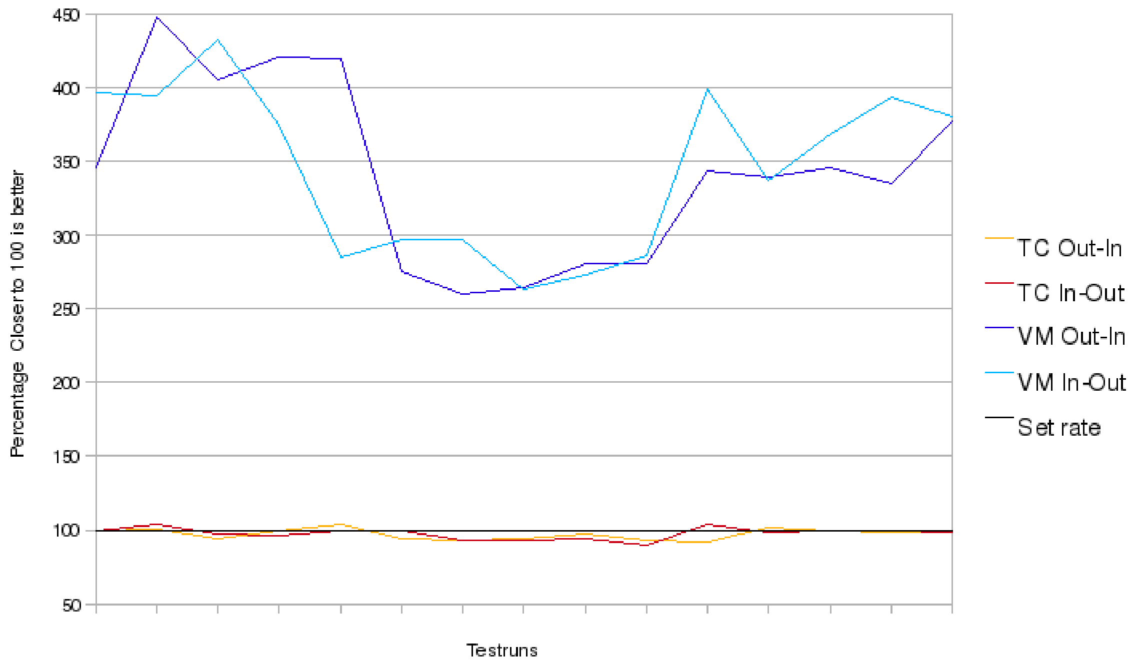


Figure 4.2: VMWare versus tc traffic control

The objective was to determine what the maximum rate of networking could be achieved and how reliable the traffic control methods are.

The set rate is 100%. **The results should be close to 100%**, the closer the better. Values that are above or below 100% means they are less reliable.

The X-axis are the test runs and the Y axis represents the percentage. E.g. a set rate of 10 mbps with a measured throughput of 15mbps will give a percentage of 150%, which is 50% off.

The set datarates are not respected by VMWare, and is therefore **not reliable** to use for the experiments. It does a little throttling, but VMWare Workstation does it very inaccurate, as it is still 3 to 4.5 times over the limit.

The tc traffic handling is correct, and is sometimes a bit above the limit. This might be due to the burst rate in the token bucket filter and the VMWare clock skew might also contribute to this fact.

4.2.3 Used software

ClusterVisionOS

ClusterVisionOS[1] is based on Scientific Linux 5.0 (Boron)[2], which is based on RedHat[3]. The difference is a set of tools and applications which make the management of slave nodes easier.

ISC DHCP3

The current DHCP server used by ClusterVision in their ClusterVisionOS (CVOS), and is the default DHCP server daemon in most BSD and Linux distributions. It is developed and maintained by the Internet Systems Consortium (ISC), a non-profit organisation. The current version installed by ClusterVisionOS is 3.0.5-redhat.

aTFTPD

The TFTP server used in ClusterVisionOS is aTFTPD. This server is multi-threaded and supports all options described in RFC2347 (option extension), RFC2348 (blksize), RFC2349 (tsize and timeout) and RFC2090 (multicast option). It also supports mtftp as defined in the PXE specification. The server handles new connections directly by starting new threads and kills itself after 5 minutes of inactivity. The aTFTP project has not been updated since March 20th 2004, but it is still in use around the world. The latest version is 0.7.

4.2.4 Test methodology

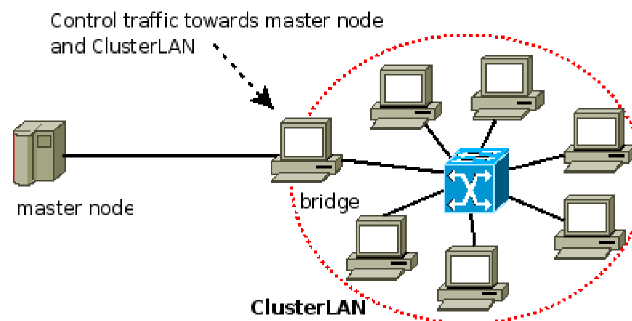


Figure 4.3: Test setup

After concluding that VMWare could not reliable control the data traffic (see figure 4.2 on the previous page), I decided to use tc in combination with a bridge. It is

[1] http://www.clustervision.com/products_os.php
[2] https://www.scientificlinux.org/
[3] http://www.redhat.com/

not possible to apply `tc` on the slave nodes. Using a bridge also gives the possibility of sniffing traffic. The bridge is transparant to the network, and can be viewed as the speed knob in the network. Tests showed it could pass traffic up to 175 Mbits per second (see section 9.1 on page 71).

Each test will be done limiting the bandwidth on the bridge. Each test will be run multiple times. The traffic statistics will be recorded: bytes, packets, dropped packets and over limit packets. The errorcodes have been classified into 11 unique codes. Some errors will reboot the node, this is still recorded as a fail because it could not complete the PXE boot sequence. Only errorcode SI (Slave Installer) will be viewed as succesful.The tests will start at 4000 kbit per second and will scale downwards until the failrate will reach 100% constantly, and it can be confirmed that no node can reach SI with this bandwidth. I've chosen the rates 4000, 2000, 1000 and 500. Other scales will be added when the tests need adjustment for increased accuracy (to show the point of failure).

The bucket needs to be adjusted when the packets are being held too long in the queue, where it is unable to transfer them out of the bucket before the token bucket filter timer expires. This is true when the bucketsize is fairly large and the latency is very low. The rate is not enough to push enough tokens (and thus data) out of the bucket. Bursts can only solve this for a limited timespan, because they cannot occur regularly, they are replenished with tokens at a fixed rate so the token bucket must contain enough tokens to allow another burst.

It is not possible to prevent drops in the token bucket when receiving too much data as you can not influence traffic being sent to the bucket, this means these drops will inevitably occur. It is only possible to prevent drops due to expiring packets (latency).

Steps

1. First the bridge will be started (if the bridge is not up already) with the shellscript I made:

   ```
   bridge up
   ```

2. Then set the tc limits (rate, bucket characteristics), and enable with the command

   ```
   tc-limit up
   ```

3. Observe the drop rate per phase with the watch command:

   ```
   watch tc -s qdisc ls dev eth0
   ```

4. Observe the results of the test (write down the tc statistics), and adjust the tc characteristics (next rate, bucket adjustment) for the next test.

5. Iteratively perform the tests until the tests start failing, then adjust the rates to pin-point a failure point.

Chapter 5

Observations

5.1 Measurements

The error messages have been uniquely identified (see section 6.1.1 on page 41), and are represented by a 2 character code. The tables all have the same headers:

Header	
no	test run number
k	connection speed limited by `tc` in kilobits (1000) per second
pl	packet loss set by `tc`
N_{1-6}	result of the corresponding node number
fail	fail percentage
comments	specific comments on the test

Errorcodes	
Error	Description
DO	No IP offer
DB	DHCP No bootfile
AT	ARP timeout
AC	ARP timeout, but continued booting somehow
TO	TFTP open timeout
TR	TFTP read timeout, cannot read from connection
TI	TFTP illegal operation
TT	TFTP server does not support tsize option
TB	TFTP Boot failed
BV	Loading vmlinuz, boot failed however
XX	Loading took too long
SI	Succesful boot to the slave installer

5.1.1 4.000 kilobit

Settings: rate 4.000 kbit, burst 16.384 (bucket size), latency 50.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	4.000	0%	SI	SI	SI	SI	SI	SI	0%	
2	4.000	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted before test
3	4.000	0%	SI	SI	SI	SI	SI	SI	0%	Server was not rebooted
4	4.000	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted, leases deleted, bridge rebooted
5	4.000	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted, bridge rebooted, VMhost rebooted

During the tests, no DHCP, ARP or TFTP problems were observed.
During the DHCP, ARP and TFTP boot, no packages were dropped, only overlimit. This is caused by too much data being sent at one time. However because no packages were dropped during the initial steps (before loading `vmlinuz` and `initrd`), they could be all processed in time (under 50 ms). The packages began to be dropped during loading vmlinuz and the initial ramdisk. The logs in `/var/log/messages` tells that the server experienced this too. It gave the error: `Timeout retrying...`, while serving the initial ramdisk (initrd). In the end, all nodes booted succesfully.

The third test produced less traffic and less drops, this may be due to caching in the server (it was not rebooted for the third test).

The table below gives interface statistics

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	131.695.922	351.905	2.311	28.539	45.434.374	345.936	0	0
2	149.453.516	364.127	2.013	43.608	46.036.752	359.358	0	0
3	127.989.392	349.310	1.359	24.465	45.405.020	345.744	0	0
4	129.231.006	350.181	1.777	18.336	45.411.722	345.787	0	0
5	147.749.769	362.903	922	41165	46.061.680	360.322	0	0

5.1.2 2.000 kilobit

Settings: rate 2.000 kbit, burst 16.384 (bucket size), latency 50.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	2.000	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted, bridge reset
2	2.000	0%	SI	SI	SI	SI	SI	SI	0%	Server not rebooted

The first tests during DHCP phase, ARP and initial TFTP, no dropped packets observed, only around 4.000 over limit packets. At the end of the test, only a small

amount (compared to the 4000k test) of packets were dropped, but there is a huge number of over limit packets. This may be due to the fairly large bucket size. They were sent in time before the timer expired, preventing a drop.

During the second test, again no drops during the intial boot, but there were again a large amount of overlimits (10.000+). The booting did take longer (loading vmlinuz and initrd), but that is no problem if they boot succesfully. The table below gives interface statistics

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	125.616.626	347.653	1016	59760	45.383.310	345.579	0	0
2	125.537.550	347.588	1002	51567	45.385.036	345.567	0	0

Now the same test, but now with a smaller bucket. Settings: rate 2.000 kbit, burst 8.192 (bucket size), latency 25.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	2.000	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted
2	2.000	0%	SI	SI	SI	SI	SI	SI	0%	Server not rebooted
3	2.000	0%	SI	SI	SI	SI	SI	SI	0%	Server not rebooted

With the smaller bucket size and lower latency, the majority of the drops only appear near the end, at the slave installer phase.

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	125.168.572	347.416	959	78684	45.391.772	345.751	0	0
2	124.823.340	351.017	754	56681	45.382.146	345.568	0	0
3	131.469.958	351.786	685	56153	45.602.285	350.520	0	0

5.1.3 1.500 kilobit

Settings: rate 1.500 kbit, burst 8.192 (bucket size), latency 25.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	1.500	0%	SI	SI	SI	SI	SI	SI	0%	Server rebooted
2	1.500	0%	SI	SI	SI	SI	SI	SI	0%	
3	1.500	0%	SI	SI	SI	SI	SI	SI	0%	

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	124.612.770	347.088	887	78.043	45.385.016	345.594	0	0
2	127.591.776	349.902	2.368	83.288	45.406.048	345.754	0	0
3	124.373.632	346.865	793	81.257	45.381.826	345.567	0	0

5.1.4 1.250 kilobit

Settings: rate 1.250 kbit, burst 8.192 (bucket size), latency 25.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	1.250	0%	SI	SI	SI	SI	SI	SI	0%	
2	1.250	0%	SI	SI	SI	SI	DO	SI	16%	
3	1.250	0%	SI	SI	SI	SI	SI	SI	0%	

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	123.667.494	346.433	359	113.567	45.368.342	345.489	0	0
2	103.437.705	288.935	471	48.366	37.814.931	287.931	0	0
3	124.435.676	346.995	735	144.915	45.380.962	345.570	0	0

Only one error occured during this test.

5.1.5 1.000 kilobit

Settings: rate 1.000 kbit, burst 8.192 (bucket size), latency 25.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	1.000	0%	BV	SI	SI	TT	SI	SI	33%	Server rebooted
2	1.000	0%	SI	SI	SI	DO	DO	DO	50%	Server rebooted
3	1.000	0%	SI	SI	SI	SI	SI	DO	16%	Server rebooted
4	1.000	0%	SI	SI	SI	SI	SI	DO	16%	Server rebooted

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	120.258.904	300.974	4.229	209.888	38.261.969	296.382	0	0
2	62.615.341	173.843	640	36.306	22.700.491	172.815	0	0
3	104.056.711	289.407	862	199.232	37.826.489	288.000	0	0
4	104.298.796	289.589	996	141.943	37.830.487	288.021	0	0

The first test showed that node 4 did fail on the TFTP server. It gave an error, but it rebooted automatically after that. Node 4 rebooted and failed again on another TFTP error. The server is reporting a timeout retried. It then got the following error from node 4:

```
illegal request <4>
```

Note: the error code has nothing to do with the node number. Node 1 failed boot trying to load vmlinuz. The server reports lots of timeouts to node 1. The error was not fatal, as it automatically reboots the node instead of halting. The first test sent lesser data compared to the other tests because of the 2 fails. It also had a huge amount of drops and over limits.

The second tests failed 3 in a very early stage. This explains the data sent is only half of the normal amount sent. The DHCPDISCOVER sent by the clients were re-

cieved by the server (logged in `/var/log/messages`), and is responded to by the server (DHCPOFFER). The clients however never recieved the response.

5.1.6 500 kilobit

Settings: rate 500 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	1.000	0%	SI	DO	SI	SI	DO	DO	50%	Server rebooted
2	1.000	0%	SI	TI	TI	SI	DB	SI	50%	
3	1.000	0%	SI	SI	SI	TI	DB	TI	50%	
4	1.000	0%	SI	TI	TI	TI	SI	SI	50%	Host rebooted

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	61.901.509	173.296	307	171.290	22.700.817	172.790	0	0
2	62.081.823	173.440	310	183.678	22.714.402	172.986	0	0
3	61.641.311	173.148	131	178.331	22.690.689	172.766	0	0
4	61.631.207	173.640	127	165.911	22.686.884	172.759	0	0

On all 500kbit tests, only 50% were succesful boots. When one clients gets to the TFTP stage, it will take up a lot of available bandwidth because UDP does not have a throttle mechanism. This very much limits the other nodes and their DHCP requests get lost, and they do not get an IP and fail to load.

5.1.7 250 kilobit

Settings: rate 250 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	250	0%	SI	DB	SI	TB	DO	DO	66%	Server rebooted
1	250	0%	TB	DB	SI	SI	DO	DO	66%	Server rebooted, test ran overnight
1	250	0%	SI	SI	BV	DB	DB	DO	66%	

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	42.277.503	116.467	567	156.365	15.210.611	116.536	0	0
2	60.928.532	130.513	12910	175.922	17.052.013	122.496	0	0
3	43.702.141	117.378	717	144.261	15.258.367	117.513	0	0

In the first run, only 2 clients made it to the last phase. It seems randomly which client can obtain an IP and continue to boot, because the clients do not fail in a particular order. Loading times are extremely long.

5.1.8 100 kilobit

Settings: rate 100 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	100	0%	TB	XX	TR	TB	TI	DO	100%	Test aborted after 45 mins
2	100	0%	TB	TB	TI	XX	TI	TI	100%	Test aborted after 10 mins
3	100	0%	TB	TR	XX	TI	DO	DO	100%	Test aborted after 10 mins

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	9.353.234	6708	422	13.323	338.099	6973	0	0
2	3.409.506	2556	46	4.879	131.177	2492	0	0
3	1.568.379	1221	84	2.352	71.285	1229	0	0

5 out of 6 failed the first run. The last node which is still continuing the boot process, is getting a lot of TFTP timeouts from the server. The server reports 45 timeouts on the `vmlinuz` file. I've aborted the test, after loading vmlinuz and the initrd took over 45 minutes. The second test only one loaded vmlinuz, and had 26 timeouts.

5.1.9 50 kilobit

Settings: rate 50 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	100	0%	TI	TB	TB	TB	XX	TB	100%	Test aborted after NBP
2	100	0%	TB	TI	TI	TI	DO	XX	100%	Test aborted after NBP
3	100	0%	TI	TI	XX	TB	TB	XX	100%	Test aborted after NBP

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	653.636	752	268	1.646	66.948	904	0	0
2	390.787	451	141	935	42.909	515	0	0
3	498.243	679	532	1734	72.487	1060	0	0

In these tests, none booted in reasonable time. One node alone will not download the images in time. In test 3, 2 nodes made it through the PXE sequence, but they kept taking bandwidth from eachother, making the download even longer. The third test had lots of timeouts and drops, due to the 2 nodes downloading.

Chapter 6

Analysis and discussion

6.1 Analysis

6.1.1 Identified problems

In the current setup there are 4 phases where it can fail:

1. DHCP Stage
2. ARP Stage
3. TFTP Stage
4. Kernel loading Stage

I've tried to analyze the problem, where it occurs and what the cause of each problem is. I've looked at the server logs (/var/log/messages) to see what I serves to the client, and sniff the bridge interfaces to see if the messages does actually pass the bridge.

I've categorized the identified error's in the DHCP, ARP, TFTP and boot phase.

6.1.2 DHCP Stage

DHCP Stage: PXE-E51 not getting an IP address

Error

```
PXE-E51: No DHCP or proxyDHCP offers were recieved.

PXE-MOF: Exiting Intel PXE ROM
```

Cause: The node is not getting an IP address. The chance of getting is message is high, especially when the network is heavily congested, and UDP packets are lost due to packet loss. DHCP relies on UDP, which is unreliable and is very sentive to packet loss, there is no acknowledgement of packets to ensure their arrival.

The server does notice DHCPDISCOVER messages (logged to `/var/log/messages`), and replies with a DHCPOFFER. This reply is never recieved by the client, although it does pass the bridge.

Result: Client will halt.

DHCP Stage: PXE-E53 no boot filename recieved

Error:

```
PXE-E53: No boot filename received

PXE-M0F: Exiting Intel PXE ROM
```

Cause: The node claims it does not receive a boot filename. This error is rare and has been described by several sources on the beowulf forum as a bug [1]. Analyzing the packets does show that the server has sent a boot filename as a response to a DHCPDISCOVER, and a second time as a response to a DHCPREQUEST. Both times the DHCPOFFER and DHCPACK contains the bootfilename. Both times, the client has not received the filename, although the DHCP message went through the bridge where it has been captured by Wireshark.

Result: Client will halt.

6.1.3 ARP Stage

ARP Stage: E11 ARP timeout

Error:

```
CLIENT IP: x.x.x.x MASK x.x.x.x DHCP IP: x.x.x.x
GATEWAY IP: x.x.x.x
PXE-E11: ARP timeout
TFTP.
```

Cause: When the client has recieved an IP address from the DHCP server, it must access the TFTP server, but it does not have it's layer 2 address yet. To get this, he does an ARP request, but no replies have been made. This error occurs when the network is heavily congested. ARP relies on UDP, which is unreliable and is very sentive to packet loss, there is no acknowledgement of packets to ensure their arrival.

Result: There are 2 options, the node halts or the node continues booting. It is unpredictable what the node will do.

6.1.4 TFTP Stage

TFTP Stage: PXE-E32 TFTP open timeout

Error:

[1] Donald Becker: http://www.beowulf.org/archive/2006-December/017023.html

```
CLIENT IP: x.x.x.x MASK x.x.x.x DHCP IP: x.x.x.x
GATEWAY IP: x.x.x.x
PXE-E32: TFTP open timeout
TFTP.....
```

Cause: This error seems to occur when the network is heavily congested and there is a lot of packetloss. TFTP relies on UDP, which is unreliable and is very sentive to packet loss, there is no acknowledgement of packets to ensure their arrival. When the server does not recieve the acknowledgements in time, it will time the node out, and the connection is terminated.

Result: The client will halt.

TFTP Stage: PXE-E35, PXE-E39 TFTP cannot read from connection

Error:

```
CLIENT IP: x.x.x.x MASK x.x.x.x DHCP IP: x.x.x.x
GATEWAY IP: x.x.x.x
PXE-E35: TFTP open timeout
PXE-E39: TFTP cannot read from connection
PXE-M0F: Exiting Intel PXE ROM
```

Cause: This error is basically the same as the previous error, only this time it was able to carry out a read operation before it timed out.

Result: The client will halt and not retry to establish the connection.

TFTP Stage: PXE-T04, PXE-E36, PXE-M0F Illegal TFTP operation

Error:

```
CLIENT IP: x.x.x.x MASK x.x.x.x DHCP IP: x.x.x.x
GATEWAY IP: x.x.x.x
TFTP......
PXE-T04: Illegal TFTP operation
PXE-E36: Error recieved from TFTP server
PXE-M0F: Exiting Intel PXE ROM
```

Cause: This error appears when the network is heavily congested (lagged). The server tries to send the packets multiple times and then times out. Because UDP does not use packet ordering, the packets may arrive in an illogical order. The client does not understand it, and sends the server a message, after which the connection is terminated. The node then halts. The server logged:

```
mylcuster atftpd[pid]: Invalid request <4> from x.x.x.x
```

Result: Client will halt.

TFTP Stage: TFTP server does not support tsize option

Error:

```
TFTP server does not support the tsize option

Boot failed: press a key to retry, or wait for reset..
```

Cause: The error is hard to reproduce, as the server is correctly configured. The node will reboot after a set timer (approx 30 seconds). The server reports timeouts and an illegal operation.

Result: The client will halt, and reboot after the timer expires.

TFTP stage: Trying to load: pxelinux.cfg/default

Error:

```
Trying to load: pxelinux.cfg/default

Boot failed: press a key to retry, or wait for reset..........
```

Cause: It has failed locating the pxelinux.cfg, and the server is reporting timeouts. This is due to heavy congestion and it times out lots of packets. The node will reboot after the timer expires.

Result: The client will halt, and reboot after the timer expires.

6.1.5 Boot Stage

Boot stage: Loading vmlinuz: Boot failed

Error:

```
Loading vmlinuz....
Boot failed: press a key to retry, or wait for reset..........
```

Cause: It has failed loading vmlinuz, and the server is reporting timeouts. This is due to heavy congestion and it times out lots of packets. The node will reboot after the timer expires.

No errors have been encountered loading initrd, it only seems to appear when loading vmlinuz.

Result: The client will halt, and reboot after the timer expires.

6.1.6 Problems matrix

Description	Cause	Result
No IP offer	Congested network will cause packetloss, resulting in a DHCP timeout	Node halted
DHCP No bootfile	Congested network will cause packetloss, resulting in a DHCP timeout	Node halted
ARP timeout	Packetloss (ARP packet lost) or master node does not respond to the ARP message in time	Node halted or node can reboot after timer expires
ARP timeout (same error as above), but continued booting somehow	(see above)	Node boots
TFTP open timeout	Packets are lost, causing the server to time the connection out	Node halted
TFTP read timeout, cannot read from connection	Packets are lost, causing the server to time the connection out	Node halted
TFTP illegal operation	Heavily congestion (lag) causes packets to lag and timeout. The packets from the clients sent are invalid	Node halted
TFTP server does not support tsize option	Packetloss	Node can reboot after timer expires
Loading vmlinuz, boot failed however	Lots of packetloss	Node can reboot after timer expires
Succesful boot to the slave installer	No errors	Succesful

6.1.7 Conclusion

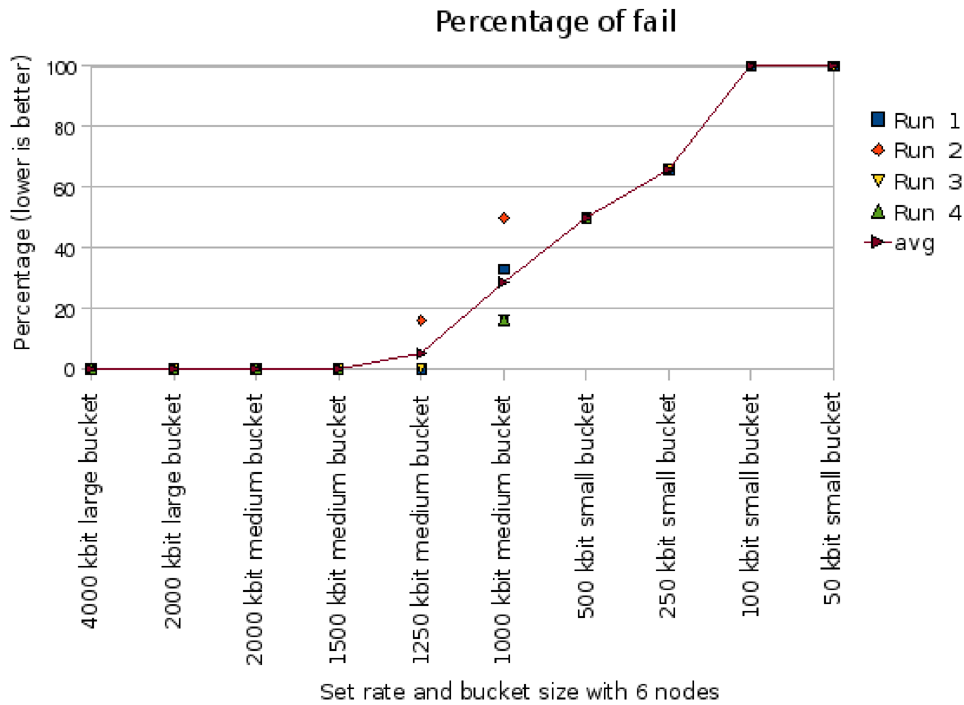


Figure 6.1: Fail percentage

The objective of these tests were to characterize the network boot process under degraded performance, and find a breaking point.

No problems have been encountered until a rate of 1250 kbit (see figure 6.1. Ofcourse the lower you go, the longer the loading times, this go as high as 45 minutes per node.

The first problems you encounter are DHCP errors (no DHCP OFFER or no boot file). This might be due to a congested network which is caused by nodes who are a step further in the bootprocess and already loading data through TFTP. The TFTP traffic creates a bigger load on the network, and these nodes can push traffic of the still starting nodes off the network, causing them to fail.

One trend which you can see are the drops by the token bucket filter. No packet was dropped from the master to the slaves. The packets which were dropped were the TFTP acknowledgements. If the server has not seen an acknowledgement in a certain timespan, it will terminate the connection. This results in the TFTP Read error and TFTP Timeout.

The further you go, the more TFTP errors you will get. The DHCP errors will disappear. Because of the limiting network speed, all nodes will have an opportunity to recieve a DHCP OFFER, before other nodes can clog the network with TFTP traffic. This results in nodes arriving at the TFTP stage at the same time, preventing pushing eachother off the network. However, until one gets an oppertunity to download TFTP data, it will start congesting the network causing packet loss. Because DHCP and TFTP rely on UDP, the traffic is unreliable; there is no mechnism to ensure arrival of packets.

It is also very difficult to reproduce errors. The ARP error's were very rare and occur sporadic, and only seen when testing the setup configuration in extreme conditions (very low bandwidth and high packetloss) using VMWare Team's. This error could not be reproduced in the setup using tc and the token bucket filter.

When tracing the packets the requested information (like boot filename or IP address) is sent by the server, and did pass the bridge but never arrived at the client. This is probably caused by packetloss and the absence of reordering in UDP. Because UDP does not cope well with packetloss, and the server will only try a few times to resend, the PXE process will fail, halting the client. The server stores it's messages about DHCP transactions and TFTP transactions in `/var/log/messages`. This can help solving problems to see if the server did recieve certain requests.

Chapter 7

Alternatives

7.1 Alternatives

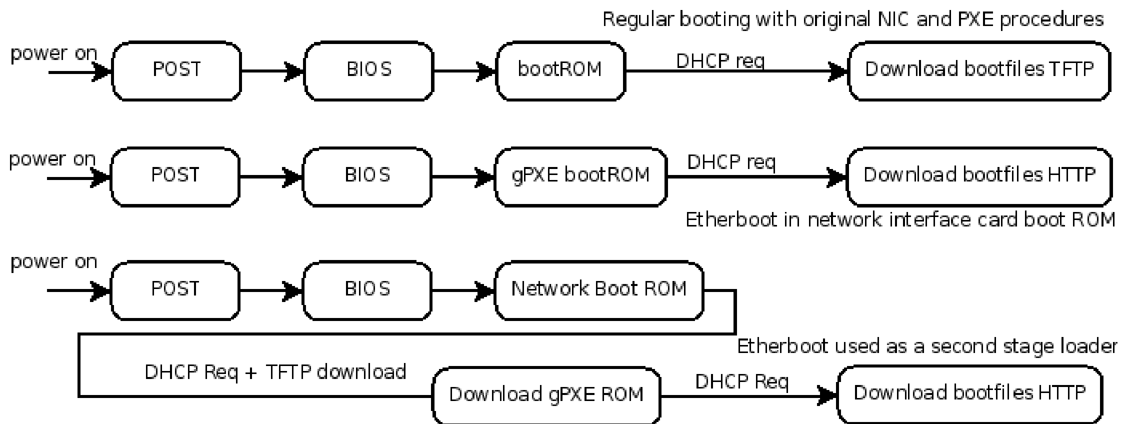


Figure 7.1: Regular PXE and gPXE in 2 modes

Alternatives have to be found for the current solution. The most important factor is reliability, performance has a lesser priority.

An alternative to the usual booting using TFTP, is booting through HTTP. HTTP has the advantage that it relies on TCP, which uses acknowledgements to ensure the arrival of packets. This makes it more reliable then a UDP based solution.

gPXE is an open source (GPL licence) network bootloader similar to PXE. It is the successor of etherboot, but both are still maintained. It is expected that gPXE will replace etherboot eventually. gPXE provides a direct replacement for proprietary PXE ROMs, with many extra features such as support for DNS, HTTP, iSCSI and ATA

over Ethernet. It is fully compatible with PXE: gPXE enabled ROM's will boot from PXE servers.

There are two options using gPXE:

1. gPXE flashed into the NIC or BIOS boot ROM;
2. gPXE loaded as second stage (PXE chainloading)[1] ;

In the first setup, gPXE bootcode will be flashed in the network or BIOS boot ROM. The advantage of this setup is that the etherboot will use TCP/HTTP from the start, instead of UDP/TFTP which is unreliable. This solution is fine for a low number of nodes (e.g. less then 10 nodes). For a large number of nodes like in a cluster computer environment, it is not feasable.

Also because not every cluster computer has the same network card or onboard network card, it is not feasable to rely on one environment. In the event the computers have exotic hardware, it may be the case that the ROM's are not even supported. Writing your own drivers to enable support for gPXE can be a hassle.

With the second option, the gPXE boot ROM will be downloaded through the use of normal PXE. Instead of serving the normal NBP pxelinux.0, the DHCP server will serve a gPXE boot ROM.

To achieve this, you can modify the DHCP server to check for gPXE extensions. If a client does not have such an extension, you serve him a gPXE boot ROM instead of the pxelinux.0 NBP. When the client has executed the gPXE bootrom, the normal boot sequence will restart, but now the client has gpXE support. It can now use HTTP to download images.

The second option is constructed in a more generic way, what means that it does not rely on the client's network boot ROM. As long as it is able to boot from the network (i.e. PXE support), it can use the chainloading feature. Also, the current slaves do not have to be flashed, only the master node needs an update of it's files and configuration. This situation is preferable over the first one.

7.2 PoC: DHCP3 and gPXE

In this proof-of-concept (PoC) I will enable gPXE support on current setup. gPXE will be configured as a second stage loader. The slave nodes need no updates or any modification, they will chainload the gPXE boot ROM's through the DHCP server.

For this setup you need a syslinux with gPXE support. SYSLinux is a total package, which also contains PXELinux. As of writing, Syslinux version 3.70 has gPXE support.

[1] http://www.etherboot.org/wiki/pxechaining

I will install 3.70 pre-19 in this setup. It can be downloaded from the kernel.org fileserver at http://www.kernel.org/pub/linux/utils/boot/syslinux/Testing/.

The current setup of ClusterVision OS uses SYSLinux 3.63. SYSLinux has support for gPXE starting at version 3.70. The SYSLinux archive holds several important files:

```
syslinux-3.70-pre19/core/pxelinux.0     The updated pxelinux.0 NBP
syslinux-3.70-pre19/gpxe/src            Sources of gPXE
syslinux-3.70-pre19/gpxe/src/bin        The undionly.kpxe boot ROM
```

You do not need to (re)compile PXELinux. The file pxelinux.0 can be directly copied to the boot directory. I've renamed the .0 extension to 3.7 to distinguish the new file.

```
cp core/pxelinux.0 /cvos/images/default-image/boot/pxelinux.3.7
```

The `undi-only.kpxe` file is not made by default. This file can be compiled using the `make` command in the directory `syslinux-3.70-pre19/gpxe/src`. The output is put in the `bin/` directory, along with a lot of other files. Copy the undionly file to the boot directory:

```
cp bin/undionly.kpxe /cvos/images/default-image/boot/
```

The ROM files are now ready for use. If you do want to recompile SYSLinux/gPXE you need nasm [2].

The DHCP3 config needs to be adjusted to recognize gPXE clients. The DHCP3 configuration file can be found in `/etc/dhcpd.conf`.

Add the following lines before any subnet or group declaration, otherwise they cannot use this information.

```
option space gpxe;
option gpxe-encap-opts code 175 = encapsulate gpxe;
option gpxe.bus-id code 177 = string;
```

The first line declares gpxe as a new space. Each gPXE DHCP packet has a private number (i.e. not officially recognized by IANA) encapsulated in the option field with the value 175 (`0xAF`). This makes it an gPXE enabled client. gPXE also has another private number in the packet with value 177 (`0xB1`). These 2 lines lets DHCP3 recognize these private options of gPXE.

Also add in the pxelinux space:

```
option pxelinux.pathprefix code 210 = text;
```

And then modify the current group:

[2] http://nasm.sourceforge.net/

```
#next-server (comment this line)
option pxelinux.pathprefix "http://";
```

The first line must be commented, otherwise it will revert to using a TFTP server. To force the client to address HTTP instead of TFTP, you append a prefix. A gPXE enabled client will recognize this, other clients will fail, trying to address `tftp://http://server`.

In the subnet declarations, you must create an if/else statement to push the ROM's to the clients.

```
if not exists gpxe.bus-id {
    filename "default-image/boot/undionly.kpxe";
}else{
    filename "http://10.141.255.254/default-image/boot/pxelinux.3.7";
}
```

Clients who do not use gPXE enabled packets will be recognized, and they will recieve a gPXE boot ROM. Clients who can be positively identified of using gPXE will recieve the NBP pxelinux3.7, and will use this. Now restart the DHCP server to apply the changes:

```
/etc/init.d/dhcpd restart
```

To offer your boot image to the clients through your webserver, you have to copy the bootimages to `/var/www/html/default-image` or create a symlink from default-image to the `/cvos/images/default-image`. The procedure the gPXE clients will follow:

1. `dhcp net0` : Request IP address for interface net0
2. `kernel http://10.141.255.254/default-image/boot/pxelinux3.7` : Follow the filename in the DHCP OFFER and ACK and load the file
3. `boot` : Boot with the options above.

The commands can also be typed in manually, by entering the gPXE console with `CTRL + B`.

7.2.1 gPXE flashed into NIC boot ROM

In this proof-of-concept, I will use gPXE in combination with VMware. It is also possible to use it with physical machine, by flashing the boot ROM into the networkcard. To use Etherboot in VMWare [3] [2], you need to know which driver to emulate and which ROM you should use. First, you can use `lspci` and `cat /proc/bus/pci/devices` to get the PCI device type and description. When you look at the NIC description in VMWare, the returned value is `8086100F`. The first value '8086' notes that the manufacturer is Intel and the device type is 100F (e1000). At the etherboot/gPXE ROM site ROM-o-Matic [4] [24], the device can be searched with the found string in the list:

[3] http://www.etherboot.org/wiki/vmwarebios
[4] http://rom-o-matic.net

```
e1000-0x100f 8086,100f e1000-0x100f
```

When you found the string, it means it is a supported NIC. To generate a ROM, go to the gPXE section, and select the NIC. For output choose a *Binary ROM Image (.rom)*. Then press “get ROM” to save the ROM. When you are using physical network cards, you need a ROM burner. In this proof-of-concept, I will be using virtual hardware in combination with VMWare. Edit your virtual machine configuration file using a text editor. The file extension is `.vmx`. Add the following lines:

```
ethernet0.present = "TRUE"
ethernet0.virtualDev = "e1000"
e1000bios.filename = "/root/vmware/gpxe-0.9.3-pci_8086_100f.rom"
```

Please note, the first 2 lines may already exist when you have added a network card. The third line notes that that it should not use the default VMWare boot ROM image, but your custom made one.

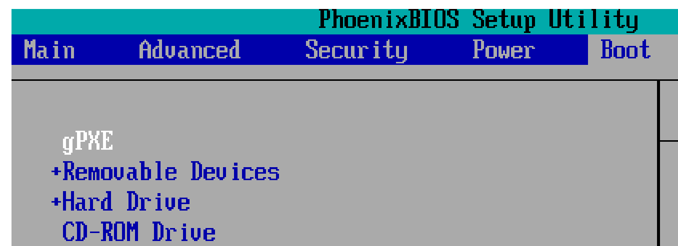


Figure 7.2: gPXE boot ROM in VMWare BIOS

Then start your virtual machine and press `F2`, then go to the boot menu and check if your gPXE boot ROM image has been loaded (figure 7.2). Place the network card first in the sequence and then save the settings.

7.2.2 Starting with gPXE

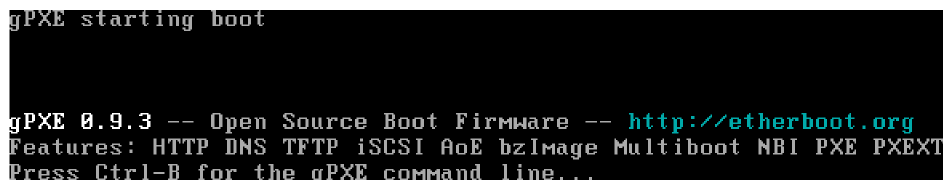


Figure 7.3: Etherboot boot ROM

When using a flashed ROM, the gPXE boot ROM will be the first to load. If you are using it as a second stage loader, the regular boot ROM will download the gPXE boot ROM and execute it. From this point, both methods are continue in the same way.

After initializing the ROM, it will give a message if you want to enter the gPXE command line. If there is no interaction, it will continue unattended booting. When the timer expires, it will then search for a DHCP server to start the PXE sequence, like the normal implementations do. The server setup is similar to the procedure described above, only you do not need to push the gPXE boot ROM to the client. If it configured correctly, the DHCP server will detect the gPXE extensions on the client and not push it anyway.

For an unknown reason, the flashed boot ROM is not completely bugfree. The server says it does not recognize the gPXE extenions sent by the client. It is to be expected that the Etherboot/gPXE project will update and correct these ROM's.

When using it as a second stage bootloader, no problems during boot were encountered with virtual hardware with version 0.9.3. There are problems with virtual hardware with 0.9.3+ (testing), where it would not find the DHCP server after loading gPXE. Physical hardware had no problems loading 0.9.3.

7.2.3 gPXE dataflow

To check if the image is indeed transferred using TCP/HTTP instead of UDP/TFTP, I captured the full traffic stream off the bridge using Wireshark.

gPXE Step 1: DHCP Request

1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x2a8da3ed
2	0.028475	10.141.255.254	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x2a8da3ed
3	0.564013	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x2a8da3ed
4	0.577986	10.141.255.254	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x2a8da3ed
5	0.581978	Vmware_8d:a3:ed	Broadcast	ARP	Who has 10.141.255.254? Tell 10.141.236.255
6	0.583357	Vmware_54:97:a5	Vmware_8d:a3:ed	ARP	10.141.255.254 is at 00:0c:29:54:97:a5
7	0.585974	10.141.236.255	10.141.255.254	TFTP	Read Request, File: default-image/boot/undionly.kpxe,
8	0.587973	10.141.255.254	10.141.236.255	TFTP	Option Acknowledgement, tsize=45147
9	0.590371	10.141.236.255	10.141.255.254	TFTP	Error Code, Code: Not defined, Message: TFTP Aborted
10	0.594352	10.141.236.255	10.141.255.254	TFTP	Read Request, File: default-image/boot/undionly.kpxe,
11	0.600000	10.141.255.254	10.141.236.255	TFTP	Option Acknowledgement, blksize=1456

Figure 7.4: Regular PXE DHCP request

The DHCP and TFTP packets are similar to the regular booting procedure, described in section 2.2 on page 12. Instead of a NBP, the client is now recieving another boot ROM: `undionly.kpxe` (see figure 7.5 on the next page). The client will execute the ROM.

gPXE Step 2: gPXE DHCP Request

Now the gPXE is loaded and will do another DHCP Discover/Request. The difference between the regular DHCP packet and the gPXE packet is the gPXE extension:

81	2.347007	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x298da3ed
82	2.351104	10.141.255.254	10.141.238.252	DHCP	DHCP Offer - Transaction ID 0x298da3ed
83	2.495017	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x298da3ed
84	2.503074	10.141.255.254	10.141.238.252	DHCP	DHCP ACK - Transaction ID 0x298da3ed
85	2.511067	Vmware_8d:a3:ed	Broadcast	ARP	Who has 10.141.255.254? Tell 10.141.238.252
86	2.511810	Vmware_54:97:a5	Vmware_8d:a3:ed	ARP	10.141.255.254 is at 00:0c:29:54:97:a5
87	2.539083	10.141.238.252	10.141.255.254	TCP	1024 > www [SYN] Seq=0 Len=0 TSV=8834415 TSER=
88	2.540052	10.141.255.254	10.141.238.252	TCP	www > 1024 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len
89	2.543040	10.141.238.252	10.141.255.254	TCP	1024 > www [ACK] Seq=1 Ack=1 Win=4096 Len=0 TS
90	2.543745	10.141.238.252	10.141.255.254	HTTP	GET /default-image/boot/pxelinux.3.7 HTTP/1.0
91	2.547120	10.141.255.254	10.141.238.252	TCP	www > 1024 [ACK] Seq=1 Ack=97 Win=5792 Len=0 T

Figure 7.5: gPXE DHCP request

```
Option: (t=175,l=37) Private
Value: b105018086100f130101170101150101110101120101180101220101190101210101100102
```

This value is the same for all clients using the gPXE boot ROM 0.9.3+. You can extract the network vendor (0x8086 Intel) and the card type (0x100f E1000 Gigabit card) from the string.

After recieving the bootfile name through a DHCP OFFER or ACK package, it will search for the image. Note that the HTTP protocol is prepended. If you do not prepend the HTTP protocol, it will assume TFTP.

```
Boot file name: http://10.141.255.254/default-image/boot/pxelinux.3.7
```

gPXE Step 3: Download menu

The rest of the datastream is downloading the image based on TCP/HTTP.

```
GET /default-image/boot/pxelinux.3.7 HTTP/1.0
GET /default-image/boot/pxelinux.cfg/default HTTP/1.0
GET /default-image/boot/menu.c32 HTTP/1.0
GET /default-image/boot/pxelinux.cfg/default HTTP/1.0
GET /default-image/boot/ClusterVision.png HTTP/1.0
```

The code above tells that the client was downloading the menu (menu.c32). After a 4 second timeout it will boot the first option in the menu. This is by default a normal slave boot (download vmlinuz and initrd).

gPXE Step 4: Download images

```
GET /default-image/boot/vmlinuz HTTP/1.0
GET /default-image/boot/initrd HTTP/1.0
```

After this step, it has the same steps as when using regular PXE.

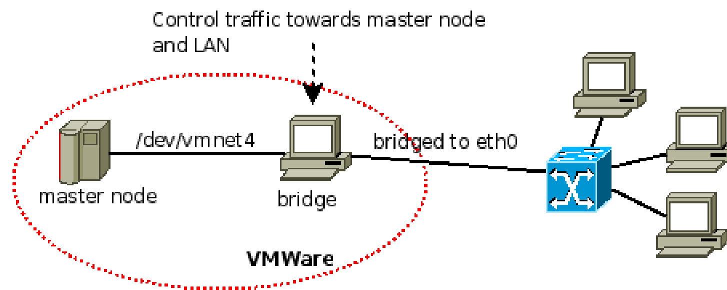


Figure 7.6: gPXE with 3 physical nodes

7.2.4 gPXE tests

Because the network adapters in VMWare gave DHCP/TFTP errors with gPXE 0.9.3+, I wanted to see if it is a VMWare specific problem (emulation of network adapers) or a gPXE problem. I added another network interface card to the physical machine to bridge the master node over that interface. After this, run the vmware networking script (`vmware-config.pl`) to enable the second interface to be bridged [25]:

```
The following bridged networks have been defined:

.vmnet0 is bridged to eth0
.vmnet2 is bridged to eth1
```

After the wizard completes you now have 2 bridged interfaces. Use the device `/dev/vmnet1` for the second interface of the master node. The `eth1` interface has a physical connection to the slave node. The settings for the master node is now:

```
Ethernet: bridged (to officenet)
Ethernet 2: custom, /dev/vmnet2 (to the cluster LAN)
```

I recieved 3 physical nodes, with a variety of Intel, Broadcom and Nvidia network cards, this gave kernel module errors after initializing the kernel and ramdisk. This could be fixed by adding the kernel module to the initial ramdisk and rebuilding the initial ramdisk.

The testruns will be similar to the virtual nodes, now the traffic will be limited in steps from 1500 kbit to 500 kbit.

7.3 gPXE measurements

Due to hardware shortage I conducted these tests with only 3 nodes. Because the BIOS and hardware configurations were different, some nodes were already booting, while some nodes were still in the POST or BIOS phase. To let them all reach the network boot phase as close as possible to eachother, I used a stopwatch to time the average boot time twice, until it reached the network boot phase. Then I programmed the slowest to start up immediately, and then delay the rest in the APC so they will all reach the network boot phase as close as possible to eachother.

7.3.1 Compatibility

Type NIC	Compatible	Remarks
VMWare Intel 1000	vmware rom	Problems with 0.9.3+ (beta)
Intel 10000	Yes	Intel Boot Agent 1.2.19
Broadcom 1000	Yes	
NVidia 1000	Yes	

7.3.2 Performance

I've taken half of the bandwidth for these tests.

Header	
no	test run number
k	connection speed limited by `tc` in kilobits (1000) per second
pl	packet loss set by `tc`
N_{1-6}	result of the corresponding node number
fail	fail percentage
comments	specific comments on the test

Errorcodes	
Error	Description
DO	No IP offer
DB	DHCP No bootfile
AT	ARP timeout
AC	ARP timeout, but continued booting somehow
TO	TFTP open timeout
TR	TFTP read timeout, cannot read from connection
TI	TFTP illegal operation
TT	TFTP server does not support tsize option
TB	TFTP Boot failed
BV	Loading vmlinuz, boot failed however
XX	Loading took too long
SI	Succesful boot to the slave installer

7.3.3 1000 kbit

Settings: rate 1000 kbit, burst 8.192 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	1000	0%	SI	SI	SI	NP	NP	NP	0%	
2	1000	0%	SI	SI	SI	NP	NP	NP	0%	
3	1000	0%	SI	SI	SI	NP	NP	NP	0%	

Note: NP: Not present. No drops observed during retrieval of the gPXE boot ROM, and no drops observed when the HTTP phase is active.

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	27.726.473	45.986	0	48.183	5.575.452	46.588	0	0
2	27.039.017	43.449	0	41.276	5.242.135	44.093	0	0
3	27.468.237	45.012	0	43.623	5.450.567	45.719	0	0

7.3.4 500 kbit

Settings: rate 500 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	500	0%	SI	SI	SI	NP	NP	NP	0%	
2	500	0%	SI	SI	SI	NP	NP	NP	0%	Test ran longer then first
3	500	0%	SI	SI	SI	NP	NP	NP	0%	Test ran longer then first

Note: NP: Not present. No drops observed during retrieval of the gPXE boot ROM. The drops occur when the HTTP transfer is active. These drops are not fatal to the bootprocess.

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	22.156.881	23.720	599	38.377	2.474.219	25241	0	0
2	26.770.363	40.194	629	38.377	4.835.160	41860	0	0
3	34.013.368	67.029	646	96.828	8.530.162	69097	0	0

7.3.5 250 kbit

Settings: rate 250 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	250	0%	SI	SI	XX	NP	NP	NP	33%	One client took a very long time
1	250	0%	SI	SI	SI	NP	NP	NP	0%	Client was still active, test statistics reset to find out if the client was stuck.
1	250	0%	XX	SI	SI	NP	NP	NP	33%	

Note: NP: Not present. No drops observed during retrieval of the gPXE boot ROM. The drops occur when the HTTP transfer is active. These drops are not fatal to the bootprocess.

	to master				to slaves			
	bytes	packets	drop	o/limit	bytes	packets	drop	o/limit
1	9.501.911	1.911	47	13.660	654.916	8.884	0	0
2								
3	19.788.348	34.442	237	53.3252	4.376.232	37.071	0	0

7.3.6 100 kbit

Settings: rate 100 kbit, burst 4.096 (bucket size), latency 12.0 ms (bucket timer), mtu 1.540 bytes

no	k	pl	N_1	N_2	N_3	N_4	N_5	N_6	Fail	Remarks
1	100	0%	XX	XX	XX	NP	NP	NP	100%	

Note: NP: Not present. This test is unusable. While the DHCP and TFTP transactions go without errors, the loading of the menu and downloading vmlinuz and initrd takes an extremely long time.

7.3.7 Conclusion

The objective of these tests were to characterize the network boot process under degraded conditions. It was expected that the gPXE would be more reliable then the current setup, but introduces a second DHCP transaction to the network.

The tests show that gPXE is more reliable in degraded conditions (low bandwidth and packet loss). This is due to the robustness TCP/HTTP brings. The setup still has possible failing points.

The intial DHCP and TFTP stage are the first possible failing points, similar to the regular PXE process. When the network is congested, DHCP and TFTP errors can occur, and the connection may fail, halting the client. The size of the gPXE boot ROM image is 16 kbyte, which is so small it is transferred so fast that it minimizes the risk of failing at that stage.

Once it is past the first phase, it will then load the gPXE image. The gPXE ROM will request an IP address for the second time, introducing the second failing point. If it does not receive an IP address it cannot contact the HTTP server and it will halt.

The first failing point can be alleviated by flashing the network cards with a gPXE boot ROM. This is an awkward solution especially when you have hundreds of computers.
Another method is to pass the IP address to the gPXE boot ROM, so it does not need to do the second DHCP request. At the moment of writing gPXE does not support

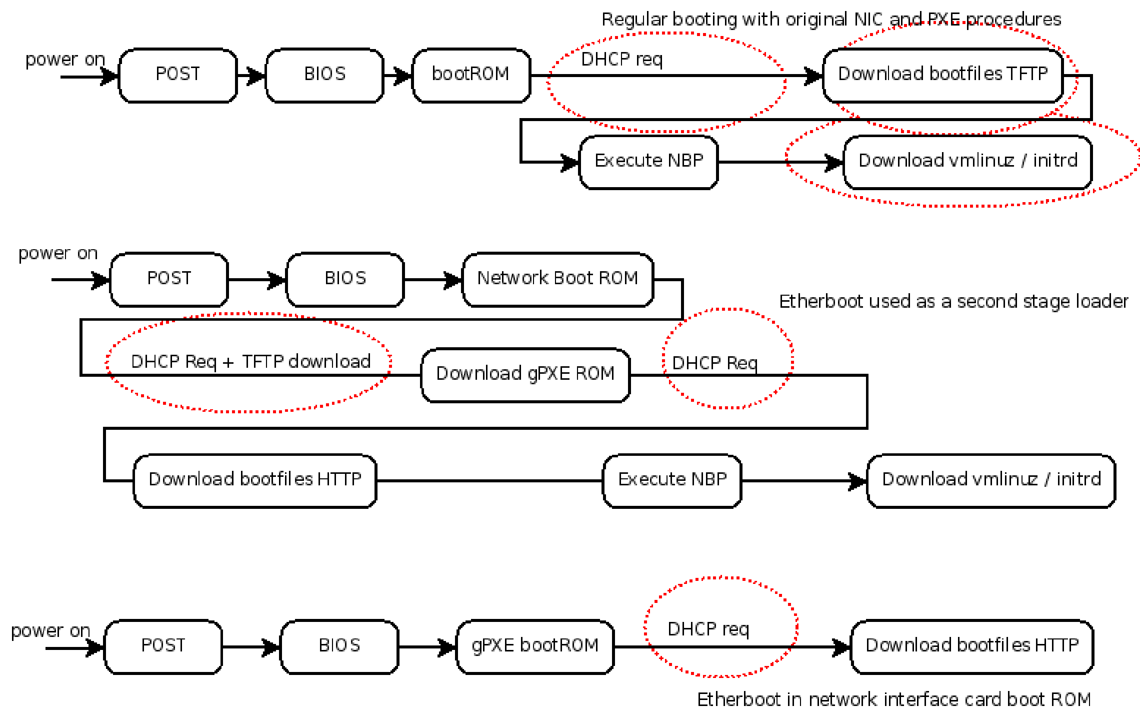


Figure 7.7: Possible failing points in the current and gPXE setups

static IP's based on the first IP the client, this feature is planned to be implemented by the gPXE project in the future.

The second failing point is still a problem, as you need at least one DHCP request for a client to pass the bootfile and IP address to the client.

While the loading of vmlinuz and initrd is reliable up to very low speeds and packetloss, although it will take a very long time.

The TCP throttling mechanism alleviates the network by sending less data. The UDP protocol does not have such a mechanism and will try to send data at the same rate, clogging the network and causing packetloss. This results in nodes pushing eachother off the network.

7.4 Proof-of-concept: DNSMasq

DNSMasq is a lightweight DNS, TFTP and DHCP server. It has TFTP support since version 2.36, it has been optimised for netboot according to the changelog. Also, it has gPXE support since version 2.41 which means it can detect gPXE options in DHCP requests and serve the bootfile.

The reason I am considering it an alternative is that it has been deployed for cluster computers according to users on the Beowulf forum [5]. Also the combination of TFTP and DHCP in one tool makes it a very useful and easy tool, as you only need to maintain and update one tool.

7.4.1 Installation

I used DNSMasq version 2.42[6] (release date May 30th 2008). Documentation for DNS-Masq is included in the file, but it is also available online at:http://www.thekelleys.org.uk/dnsmasq/docs/. To compile DNSmasq, make sure you have the development libraries and compilers present. On ClusterVisionOS 3.1, they are present. Installation is very easy, unzip and untar the file, then invoke the make command in the directory:

```
make install
```

This will install DNSMasq in the `/usr/local/sbin` directory and the manpages of dns-masq. An example configuration is included. Edit the file (see appendix D at section 9.4 on page 77).
DNSMasq assumes that the configuration file is located at: `/etc/dnsmasq.conf`

I've made a full clone of the original master node in VMware Workstation to test DNSMasq on. The testsetup is the same as the previous setups. Because I now know that the failpoint lies around 1250 kbit, I will start testing at 1500, and proceed to go down to see if I get different results.

7.4.2 Configuration

To configure DNSMasq, you have to edit the `/etc/dnsmasq.conf` file. To see the full configuration file, see Appendix D (Section 9.4 on page 77).

I will only discuss the most important configuration settings. First, define the interface or address DNSMasq will listen on. This can be done by defining the interface or listen addres:

```
interface=eth0
listen-address=10.141.255.254
```

Defining the DHCP range:

[5] http://www.beowulf.org/archive/2006-December/017023.html
[6] http://www.thekelleys.org.uk/dnsmasq/dnsmasq-2.42.tar.gz

```
dhcp-range=10.141.0.0,10.141.255.254,255.255.0.0,12h
```

Then defind the vendor class. These values are taken from the Vendor Class Identifier field (Option 60). I took the values from the DNSMasq mailing list. The dhcp-match defines the gpxe class, similar to DHCP3. It links DHCP messages with option 175 with gPXE [7].

```
dhcp-vendorclass=pxe,PXEClient
dhcp-vendorclass=eth,Etherboot

dhcp-match=gpxe,175
dhcp-vendorclass=gpxe,gPXE
```

Now the vendors have been defined, it should make a distinction between the different type's of requests, and DNSMasq can serve the bootfiles.

When a standard PXE client has been detected, it will send the undionly.kpxe boot ROM, when it is an etherboot or gPXE client it will serve the HTTP URL (in this example text, the URL has been shortened).

```
dhcp-boot=net:pxe,default-image/boot/undionly.kpxe,,10.141.255.254
dhcp-boot=net:eth,http://10.141.255.254/.../pxelinux.3.7,10.141.255.254
dhcp-boot=net:gpxe,http://10.141.255.254/.../pxelinux.3.7,10.141.255.254
```

7.4.3 Test setup and complications

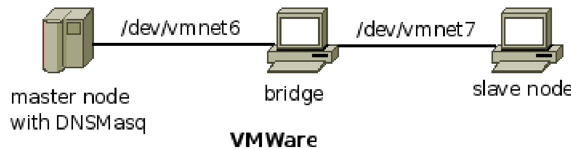


Figure 7.8: DNSMasq setup

I have created a setup where a single client, with a standard, unmodified client would do a PXEBoot from a DNSMasq server. The DNSMasq configuration was gPXE enabled (detect option 175) in the same manner as DHCP3. The client would use gPXE as a second stage loader, it would first download and then load the `undionly.kpxe` boot ROM.

During testing, the second stage did not work. It would download and execute the undionly gPXE boot loader (first stage), but the second time it could not locate the DHCP server. I used a packet sniffer (Wireshark) and compared the DHCP OFFERS from the servers. The DNSMasq DHCP server does serve different OFFERS then described in section 2.2 on page 12, which DHCP3 serves.

[7] http://lists.freegeek.org/pipermail/lessdisks/2005-June/000709.html

Differences in the first DHCP Offer

The first DHCP Offer sent to the PXE Client:

	DHCP3	DNSMASQ
Message type	0x02	0x02
Hardware type	0x01	0x01
Hardware address length	0x06	0x06
Hops	0x00	0x00
Transaction ID	Present	Present
Seconds elapsed	0x0004	0x004
Bootp flags	0x8000	0x8000
Client IP address	Zero	Zero
Your address	Present	Present
Next server	Present	Present
Relay	Zero	Zero
Client MAC	Present	Present
Server host	Not present	Not present
Boot filename	Present	Not present
Magic cookie	Present	Present

And the DHCP extensions:

	DHCP3	DNSMASQ
53	0x35 (DHCP Message type)	0x35 (DHCP Message typ
54	0x36 (Server identifier)	0x36 (Server identifier)
51	0x33 (IP address lease time)	0x33 (IP address lease time)
58	Not used	0x3A (Renewal Time Value)
59	Not used	0x3B (Rebinding Time Value)
67	Not used	0x43 (Boot file name)
28	Not used	0x1C (Broadcast address)
1	0x01 (Subnet mask)	0x01 (Subnet mask)
3	0x03 (Router)	0x03 (Router)
6	0x06 (Domain Name Server)	0x06 (Domain Name Server)
12	0x0C (Host name)	Not used
15	0x0F (Domain name)	0x0F (Domain name)

The difference in the second DHCP Offer

The second OFFERS sent to the gPXE client:

	DHCP3	DNSMASQ
Message type	0x02	0x02
Hardware type	0x01	0x01
Hardware address length	0x06	0x06
Hops	0x00	0x00
Transaction ID	Present	Present
Seconds elapsed	0x0000	0x004
Bootp flags	0x0000	0x0000
Client IP address	Zero	Zero
Your address	Present	Present
Next server	Zero	Present
Relay	Zero	Zero
Client MAC	Present	Present
Server host	Not present	Not present
Boot filename	Present	Not present
Magic cookie	Present	Present

And the DHCP extensions:

	DHCP3	DNSMASQ
53	0x35 (DHCP Message type)	0x35 (DHCP Message typ
54	0x36 (Server identifier)	0x36 (Server identifier)
51	0x33 (IP address lease time)	0x33 (IP address lease time)
58	Not used	0x3A (Renewal Time Value)
59	Not used	0x3B (Rebinding Time Value)
66	Not used	0x42 (TFTP Server Name)
67	Not used	0x43 (Boot file name)
28	Not used	0x1C (Broadcast address)
1	0x01 (Subnet mask)	0x01 (Subnet mask)
3	0x03 (Router)	0x03 (Router)
6	0x06 (Domain Name Server)	0x06 (Domain Name Server)
12	0x0C (Host name)	Not used
15	0x0F (Domain name)	0x0F (Domain name)

When comparing the servers packets, it shows DNSMasq uses different fields, but not the `boot file` field, like DHCP3 uses. It appends a DHCP option (67) to the message, which the gPXE client does not seem to understand. The regular PXE bootrom recognizes the DHCP Option 67 field, and is able to boot.

When looking at the clients DHCP DISCOVER packet, it shows that it lists a lot of options which it requested, including the option 67. It is strange that it asks for option

67, but does nothing with it. If gPXE does not request 67, DNSMasq would respond to it, by sending the bootfile name in the regular way, instead of option 67. This behaviour seems broken according to the DNSMasq mailing list[8].

7.4.4 Test method

The plan was to use a DHCP client loader, or write a similar script that produces x requests per second to the DHCP server with random MAC addresses, to see which server can handle the requests the fastest and with the least cpu load.

This test was not completed due to time complications (not enough time), and the fact that DNSMasq does not supply the correct DHCP OFFERS to the gPXE client.

7.4.5 Conclusion

DNSMasq has some advantages over DHCP3. It has support for DHCP, DNS and TFTP in one package, using one configuration file. This makes it easy to maintain and to configure. According to sources on the Beowulf mailing list, it should be more suitable for cluster computers[9].

The configuration is very different from DHCP3 and may seem complicated at first. It also does not allow to make if statements to distingush types and options.

The DNSMasq DHCP performance could not be tested because DNSMasq sends the wrong DHCP Offers because gPXE requested them and gPXE does not do anything with the Offer.

DNSMasq is therefore, at the time of writing, not usable for deploying gPXE at this moment.

[8] http://lists.thekelleys.org.uk/pipermail/dnsmasq-discuss/2007q4/001642.html
[9] http://www.beowulf.org/archive/2006-December/017027.html

Chapter 8

Conclusion and future work

8.1 Conclusion

When the network gets congested, packet loss will occur inevitably. The currently used protocols DHCP and TFTP use UDP, which does not cope well with packet loss. UDP does not guarantee reliability or ordering in the way that TCP does. Datagrams may arrive out of order, appear duplicated, or go missing without notice. This causes the client to fail. The boot failures discovered have been identified and described in section 6.1.1 on page 41.

In the current setup, I have identified four phases where the process can fail. All these phases use UDP for data transport. This makes booting unreliable. To solve this, a more reliable protocol must be chosen for data transport.

A newer version of PXELinux, version 3.70, supports booting over HTTP, which relies on TCP and is therefore more reliable. To implement gPXE and the updated PXELinux 3.70 there are 2 options. These are described in section 7.2 on page 49.

1. Using PXELinux 3.70 and gPXE flashed into the boot ROM, the only bottleneck is the DHCP stage. The server has then 4 chances to respond to the DHCP Discover before the client halts with an error. The failures that occur during the TFTP phase does not exist.

   In theory this setup is the best solution to minimize the risk of failing nodes. However, this solution is very awkward, because all the network boot ROM's or BIOS ROM's have to be flashed. This makes it an impractical solution to large number of clusters. Tests have also shown that the current version of the flashable gPXE boot ROM is not fully compatible with PXELinux.

2. Use gPXE as a second stage loader. This means the default, unmodified boot ROM will attempt a PXE session like usual, but instead of downloading a network boot program, it will download a gPXE boot ROM. This boot ROM will be executed,

and the client will become gPXE enabled. The PXE session will be restarted by making a DHCP DISCOVER. The connection will continue using HTTP for the remainder of the PXE session. Packetloss is not an issue for HTTP because TCP uses re-transmission with acknowledgement to ensure reliable delivery. Tests have shown the connection will not halt when packet loss occurs during the HTTP boot.

The second option is the most likely to implement and has been tested succesfully. It does not require modifications to the clients, and only few modifications to the master. This makes it a very practical solution especially with a large number of nodes.

However this setup introduces 2 DHCP sessions, that are fragile. To limit the risk, it should be brought back to one DHCP session. To do this, a static IP, based on the information in the first session, must be given to the gPXE boot ROM . This is, at the time of writing this report, not possible. This leaves the solution with bottlenecks, but these DHCP bottlenecks have a smaller risk then the TFTP problem. The DHCP stage takes a few seconds, while the TFTP stage might take up to several minutes whereby the likelyhood of failure is much higher.

To alleviate the DHCP problem, I have looked at another lightweight implementation: DNSMasq. This implementation is not ready for gPXE, even though the documentation states that it is. Tests show that gPXE and DNSMasq are not as compatible to eachother as gPXE and DHCP3. DNSMasq is at the time of writing not usable as an alternative in the setup.

The conclusion is that the largest bottleneck (TFTP transactions) can be taken away by booting off a more reliable protocol at the expense of a second DHCP session. When a solution has been found for the second DHCP session, it will make the setup more reliable.

8.2 Future work

To alleviate the remaining bottlenecks, the gPXE setup needs improvement on 2 aspects:

1. The second DHCP Request when using the gPXE boot ROM;
2. Replace the DHCP server for a lighter one.

To minimize the risk of the second DHCP request when using the gPXE boot ROM, a parameter must be given to the boot ROM. At present, this is not possible. It is expected that the gPXE project will implement this in the future.

The DHCP implementation might be replaced by a lighter version. DNSMasq might be an interesting alternative to look at when it fully supports gPXE. I have have described the installation and configuration of DNSMasq in section 7.4 on page 60. To determine the difference in the performance between DHCP3 and DNSMasq, the requests must be benchmarked. Due to time limitations these tests could not be carried out.

Bibliography

[1] Ewen McNeill and Naos Limited. *Linux Based Diskless Workstations.* http://www.naos.co.nz/papers/diskless/index1.html, 2000.

[2] Etherboot Project. *Etherboot/gPXE Project.* http://www.etherboot.org, 2008.

[3] H. Peter Anvin. *PXELinux - Syslinux for network boot.* http://syslinux.zytor.com/pxe.php, 2000.

[4] Intel Corporation. *Preboot Execution Environment (PXE) Specification Version 2.1.* http://www.pix.net/software/pxeboot/archive/pxespec.pdf, 1999.

[5] S. Alexander and R. Droms. *DHCP Options and BOOTP Vendor Extensions.* http://tools.ietf.org/html/rfc2132, 1997.

[6] R. Droms. *Dynamic Host Configuration Protocol.* http://tools.ietf.org/html/rfc2131, 1997.

[7] IANA. *Dynamic Host Configuration Protocol (DHCP) and Bootstrap Protocol (BOOTP) Parameters.* http://www.iana.org/assignments/bootp-dhcp-parameters, 2008.

[8] IANA. *Address Resolution Protocol (ARP) Parameters.* http://www.iana.org/assignments/arp-parameters, 2006.

[9] Wikipedia. *Dynamic Host Configuration Protocol.* http://en.wikipedia.org/wiki/DHCP, 2008.

[10] M. Johnston and S. Venaas. *DHCP Options for the Intel Preboot eXecution Environment (PXE).* http://www.ietf.org/rfc/rfc4578.txt, 2006.

[11] Xavier Brochard. *Using PXE to boot an LTSP workstation.* http://wiki.ltsp.org/twiki/bin/view/Ltsp/PXE, 2007.

[12] M. Tim Jones. *Linux initial RAM disk (initrd) overview.* http://www.ibm.com/developerworks/linux/library/l-initrd.html, 2006.

[13] Ariane Keller. *Manual tc filtering and netem.*
http://tcn.hypert.net/tcmanual.pdf, 2006.

[14] Linux Foundation. *Net:Netem.*
http://www.linuxfoundation.org/en/Net:Netem, 2008.

[15] Bert Hubert et. al. *Linux Advanced Routing & Traffic Control HOWTO.*
http://lartc.org/howto/, 2004.

[16] Martin A. Brown. *Traffic control.*
http://tldp.org/HOWTO/Traffic-Control-HOWTO/components.html, 2006.

[17] Leonardo Balliache. *Differentiated Service on Linux.*
http://www.opalsoft.net/qos/DS.htm, 2003.

[18] Wikipedia. *Token bucket.*
http://en.wikipedia.org/wiki/Token_bucket, 2008.

[19] Alexey N. Kuznetsov. *Traffic control Token Bucket Filter.*
http://linux.die.net/man/8/tc-tbf.

[20] Martin Devera and Don Cohen. *HTB Linux queuing discipline.*
http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm, 2003.

[21] Wikipedia. *Class based queueing.*
http://en.wikipedia.org/wiki/Class_Based_Queueing, 2008.

[22] corbet. *How fast should HZ be?*
http://lwn.net/Articles/145973/, 2005.

[23] Mark Gates, Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, Kevin Gibbs, and John Estabrook. *Iperf project.*
http://sourceforge.net/projects/iperf, 2008.

[24] Etherboot project. *ROM-o-matic.net dynamically generates gPXE and Etherboot network booting images.*
http://rom-o-matic.net/, 2008.

[25] VMware Inc. *VMWare Workstation User's manual*, 6.0 edition, 09 2007.

[26] Wikipedia. *Network booting.*
http://en.wikipedia.org/wiki/Network_booting, 2008.

[27] Cor Bosman. *Installing and maintaining clusters of UNIX servers using PXE and rsync.*
http://www.nluug.nl/events/sane2002/papers/SANE.ps, 2002.

[28] Microsoft Knowledge Base. *Description of PXE Interaction Among PXE Client, DHCP, and RIS Server.*
http://support.microsoft.com/kb/244036, 2007.

[29] Markus Gutschke Ken Yap. *Etherboot Project.*
http://etherboot.berlios.de/doc/html/userman/t1.html, 2006.

[30] Intel Corporation. *How do I troubleshoot PXE boot with a network protocol analyzer?*
http://www.intel.com/support/network/sb/cs-028533.htm, 2008.

[31] David C. Plummer. *An Ethernet Address Resolution Protocol.*
http://tools.ietf.org/html/rfc826, 1982.

[32] A. Emberson. *TFTP Multicast Option.*
http://www.ietf.org/rfc/rfc2090.txt, 1997.

[33] Tandem Systems. *TFTP Protocol.*
http://www.tftp-server.com/tftp-server-help/tftp-protocol.html, 2007.

[34] Remi Lefebvre. *Advanced TFTP.*
http://freshmeat.net/projects/atftp/, 2004.

[35] Nicolas Bouliane. *Analysis of the simple token bucket filter algorithm implementation inside the netfilter's limit module.*
http://people.netfilter.org/acidfu/papers/limit-tbf-analysis.pdf, 2007.

[36] Simon Kelley. *Dnsmasq.*
http://thekelleys.org.uk/dnsmasq/doc.html, 2008.

[37] Kurt Wagner. *Short evaluation of the Linux Token-Bucket-Filter Queueing Discipline.*
http://www.docum.org/docum.org/docs/other/tbf02_kw.ps, 2001.

[38] IXIA. *Performance testing library: test plans.*
http://www.ixiacom.com/library/test_plans/, 2008.

[39] Bruce Bahlmann. *The ABCs of Understanding DHCP Performance.*
http://www.birds-eye.net/article_archive/abc_of_understanding_dhcp_performance.htm, 2001.

Chapter 9

Appendices

9.1 Appendix A: Traffic control tests

9.1.1 Traffic control tests

Unlimited

This test is set with no limits in either VMWare or tc. The following settings apply:
Client: TCP Port random pick, TCP window size 16 Kbyte (default)
Server: TCP port 5001, tcp window size 85.3 KByte (default)

	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	2000 MB	165 Mbits/s	159 Mbits/s
Run 2	2000 MB	162 Mbits/s	164 Mbits/s
Run 3	2000 MB	160 Mbits/s	163 Mbits/s
Run 4	2000 MB	150 Mbits/s	172 Mbits/s
Run 5	2000 MB	160 Mbits/s	175 Mbits/s
Min		150 Mbits/s	159 Mbits/s
Avg		159.4 Mbits/s	166.6 Mbits/s
Max		162 Mbits/s	175 Mbits/s

10 Mbps

This test is set with a 10Mbps (10000kbit/s) limit in VMWare and tc. The following settings apply:
Client: TCP Port random pick, TCP window size 16 Kbyte (default)
Server: TCP port 5001, tcp window size 85.3 KByte (default)

	Using `tc`		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	100 MB	9.42 Mbits/s	9.98 Mbits/s
Run 2	100 MB	9.26 Mbits/s	9.33 Mbits/s
Run 3	100 MB	9.38 Mbits/s	9.36 Mbits/s
Run 4	100 MB	9.79 Mbits/s	9.39 Mbits/s
Run 5	100 MB	9.27 Mbits/s	9.01 Mbits/s
Min		9.26 Mbits/s	9.01 Mbits/s
Avg		9.42 Mbits/s	9.41 Mbits/s
Max		9.79 Mbits/s	9.98 Mbits/s
	Using VMWare Workstation Team		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	100 MB	27.5 Mbits/s	29.7 Mbits/s
Run 2	100 MB	26.0 Mbits/s	29.7 Mbits/s
Run 3	100 MB	26.4 Mbits/s	26.3 Mbits/s
Run 4	100 MB	28.1 Mbits/s	27.3 Mbits/s
Run 5	100 MB	28.1 Mbits/s	28.6 Mbits/s
Min		26.0 Mbits/s	26.3 Mbits/s
Avg		27.22 Mbits/s	28.32 Mbits/s
Max		28.1 Mbits/s	29.7 Mbits/s

2 Mbps

The following test is set to 2.000kbit, using VMWare and tc. The following settings apply:
Client: TCP Port random pick, TCP window size 16 Kbyte (default)
Server: TCP port 5001, tcp window size 85.3 KByte (default)

	Using `tc`		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	2 MB	1.84 Mbits/s	2.08 Mbits/s
Run 2	2 MB	2.03 Mbits/s	1.98 Mbits/s
Run 3	2 MB	2.00 Mbits/s	1.99 Mbits/s
Run 4	2 MB	1.97 Mbits/s	2.00 Mbits/s
Run 5	2 MB	1.98 Mbits/s	1.96 Mbits/s
Min		1.84 Mbits/s	1.96 Mbits/s
Avg		1.96 Mbits/s	2.00 Mbits/s
Max		2.03 Mbits/s	2.08 Mbits/s

	Using VMWare Workstation Team		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	10 MB	6.87 Mbits/s	7.97 Mbits/s
Run 2	10 MB	6.78 Mbits/s	6.73 Mbits/s
Run 3	10 MB	6.92 Mbits/s	7.36 Mbits/s
Run 4	10 MB	6.70 Mbits/s	7.87 Mbits/s
Run 5	10 MB	7.54 Mbits/s	7.60 Mbits/s
Min		6.70 Mbits/s	6.73 Mbits/s
Avg		6.96 Mbits/s	7.51 Mbits/s
Max		7.54 Mbits/s	7.97 Mbits/s

128 kbps

The following test is set to 128kbit, using VMWare and tc. The following settings apply:
Client: TCP Port random pick, TCP window size 16 Kbyte (default)
Server: TCP port 5001, tcp window size 85.3 KByte (default)

	Using `tc`		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	500kB	127 kbits/s	127 kbits/s
Run 2	500kB	129 kbits/s	133 kbits/s
Run 3	500kB	121 kbits/s	125 kbits/s
Run 4	500kB	128 kbits/s	123 kbits/s
Run 5	500kB	133 kbits/s	128 kbits/s
Min		121 kbits/s	123 kbits/s
Avg		127.6 kbits/s	127.2 kbits/s
Max		133 kbits/s	133 kbits/s

	Using VMWare Workstation Team		
	Transferred	Throughput Outside-Inside	Throughput Inside-Outside
Run 1	500kB	443 kbits/s	508 kbits/s
Run 2	500kB	573 kbits/s	505 kbits/s
Run 3	500kB	519 kbits/s	553 kbits/s
Run 4	500kB	538 kbits/s	480 kbits/s
Run 5	500kB	537 kbits/s	365 kbits/s
Min		121 kbits/s	123 kbits/s
Avg		127.6 kbits/s	127.2 kbits/s
Max		133 kbits/s	133 kbits/s

9.2 Appendix-B Shellscripts

9.2.1 tc-limit

```
#!/bin/bash
if [ $1 = "up" ];
then
        echo "Setting qdiscs"
        modprobe sch_netem
        echo "Applying token bucket to interface eth0"
        tc qdisc add dev eth0 root tbf rate 1mbit burst 1540 latency 50ms minburst 1540
        echo "Applying token bucket to interface eth1"
        tc qdisc add dev eth1 root tbf rate 1mbit burst 1540 latency 50ms minburst 1540
        tc -s qdisc ls dev eth0
        tc -s qdisc ls dev eth1
fi

if [ $1 = "reset" ];
then
        tc qdisc del dev eth0 root
        tc qdisc del dev eth1 root
        echo "QDiscs resetted"
fi

if [ $1 = "status" ];
then
        echo "Bridge configuration"
        brctl show
        echo "Traffic control"
        tc qdisc show
fi
```

9.2.2 bridge

```
#!/bin/bash
if [ $1 = "up" ];
then
        brctl addbr br1
        brctl addif br1 eth0
        brctl addif br1 eth1
        ip link set br1 up
        brctl setfd br1 0
        brctl show
fi
if [ $1 = "down" ];
then
        ip link set br1 down
        brctl delbr br1
        brctl show
        echo "Bridge deleted"
fi
```

9.3 Appendix-C ISC DHCP3 configuration

```
not authoritative;
ddns-update-style            none;
default-lease-time           100800;
#default-lease-time          -1;
use-host-decl-names          on;

option space gpxe;
option gpxe-encap-opts code 175 = encapsulate gpxe;
option gpxe.bus-id code 177 = string;


option space pxelinux;
option pxelinux.reboottime code 211 = unsigned integer 32;
option pxelinux.configfile code 209 = text;
option pxelinux.pathprefix code 210 = text;
group {
  option domain-name            "cvos.cluster clustervision.com";
  option domain-name-servers    10.141.255.254;
  option subnet-mask            255.255.0.0;
  option broadcast-address      10.141.255.255;
  option routers                10.141.255.254;

  option pxelinux.reboottime 60;
  server-identifier             10.141.255.254;
  #next-server                   10.141.255.254;
  option pxelinux.pathprefix "http://";
  #option pxelinux.configfile  \\
         "http://10.141.255.254/default-image/boot/pxelinux.cfg/default";

  host node001 {
    #filename "default-image/boot/pxelinux.0";
    #filename "http://10.141.255.254/default-image/boot/pxelinux.0";
    hardware ethernet 00:0C:29:81:06:5A; fixed-address 10.141.0.1;
  }

  subnet 10.141.0.0 netmask 255.255.0.0 {
    range 10.141.232.0 10.141.239.255;
    #filename "default-image/boot/pxelinux.0";
    filename "http://10.141.255.254/default-image/boot/gpxelinux.0";
    if not exists gpxe.bus-id {
            filename "default-image/boot/undionly.kpxe";
    }else{
            filename "http://10.141.255.254/default-image/boot/pxelinux.3.7";
    }
    max-lease-time 100800;
  }
}
```

9.4 Appendix-D DNSMasq Configuration

```
# Configuration file for dnsmasq.

#interface=eth0

#listen-address=10.141.255.254

domain = cvos.cluster.clustervision.com

dhcp-range=10.141.0.0,10.141.255.254,255.255.0.0,12h



dhcp-vendorclass=pxe,PXEClient
dhcp-vendorclass=eth,Etherboot

dhcp-match=gpxe,175
dhcp-vendorclass=gpxe,gPXE


dhcp-option=1,255.255.0.0     # subnet-mask
dhcp-option=3,10.141.255.254      # routers
#dhcp-option=pxe,67,/cvos/images/default-image/boot/undionly.kpxe   #bootfile-name
#dhcp-option=eth,67,/cvos/images/default-image/boot/pxelinux.3.7   #bootfile-name
#dhcp-option=gpxe,67,/cvos/images/default-image/boot/pxelinux.3.7   #bootfile-name

#dhcp-boot=default-image/boot/undionly.kpxe          # gpxe,undionly.kpxe
#dhcp-boot=net:gpxe,http://10.141.255.254/default-image/boot/pxelinux.3.7,,


dhcp-boot=net:pxe,default-image/boot/undionly.kpxe,,10.141.255.254
dhcp-boot=net:eth,http://10.141.255.254/default-image/boot/pxelinux.3.7,10.141.255.254
dhcp-boot=net:gpxe,http://10.141.255.254/default-image/boot/pxelinux.3.7,10.141.255.254



# Enable dnsmasq's built-in TFTP server
enable-tftp


# Set the root directory for files availble via FTP.
tftp-root=/cvos/images/
```