

# Variation on AFS as root filesystem

wgehrke@dia.uniroma3.it

<http://www.dia.uniroma3.it/~wgehrke/>

Wolfgang Gehrke

DIA  
Univ. Roma Tre

Dipartimento di Informatica e Automazione  
Università degli Studi Roma Tre



# Outline

1 Problem

2 Idea

3 Implementation

4 Outlook

# Problem statement

# Problem statement

- setup
- laboratories ca. 100 clients = 50 + 40 + 10
  - Linux (Debian, Ubuntu) installation on local HDs
  - activation of auto updates
  - some clients out of sync
  - exposure to local tampering

# Problem statement

- setup
- laboratories ca. 100 clients = 50 + 40 + 10
  - Linux (Debian, Ubuntu) installation on local HDs
  - activation of auto updates
  - some clients out of sync
  - exposure to local tampering

wish for centralized image plus clean-up on boot

# Problem statement

- setup
- laboratories ca. 100 clients = 50 + 40 + 10
  - Linux (Debian, Ubuntu) installation on local HDs
  - activation of auto updates
  - some clients out of sync
  - exposure to local tampering

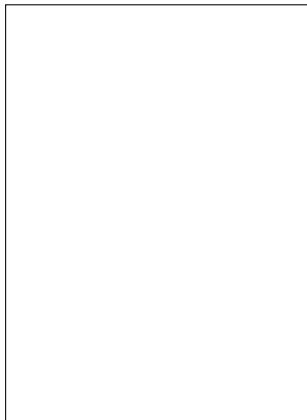
wish for centralized image plus clean-up on boot

- context
- PXE boot already in use
  - NFS root file system not desirable
  - AFS available for HOMEs
  - yet no redundancy for RW volumes
  - PCs with more than 1GB RAM and 40 GB HD

# Sketch

# Sketch

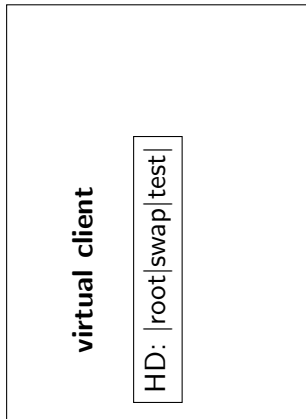
## host with VirtualBox





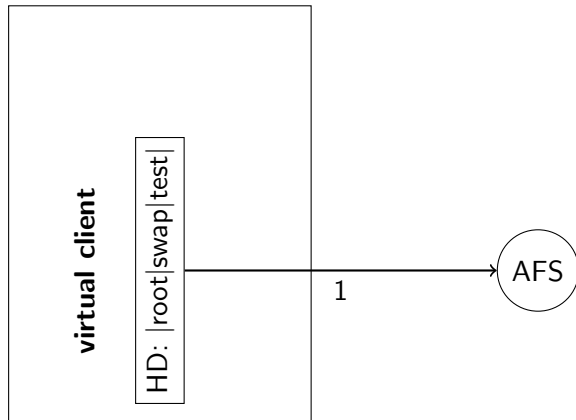
# Sketch

## host with VirtualBox



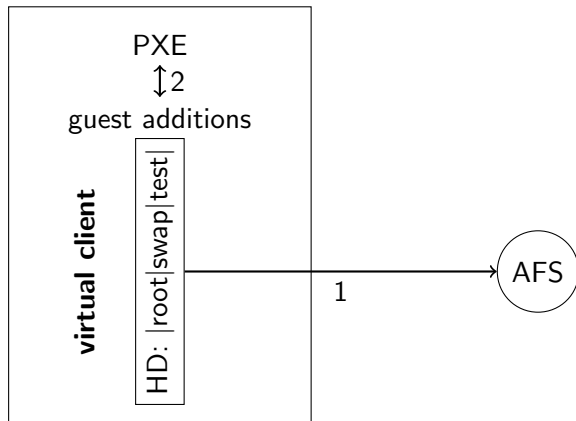
# Sketch

## host with VirtualBox

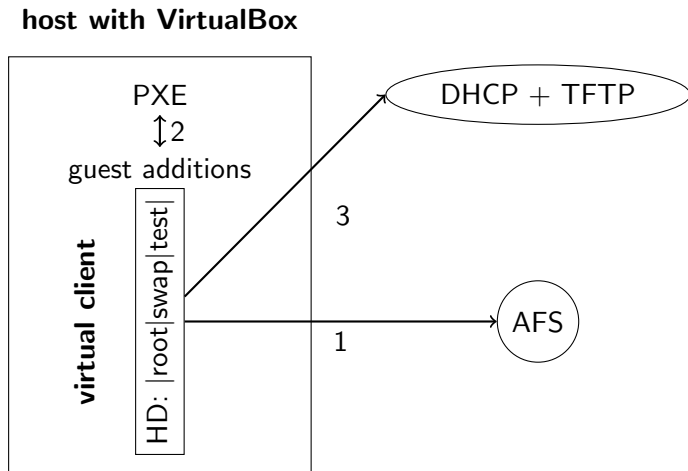


## Sketch

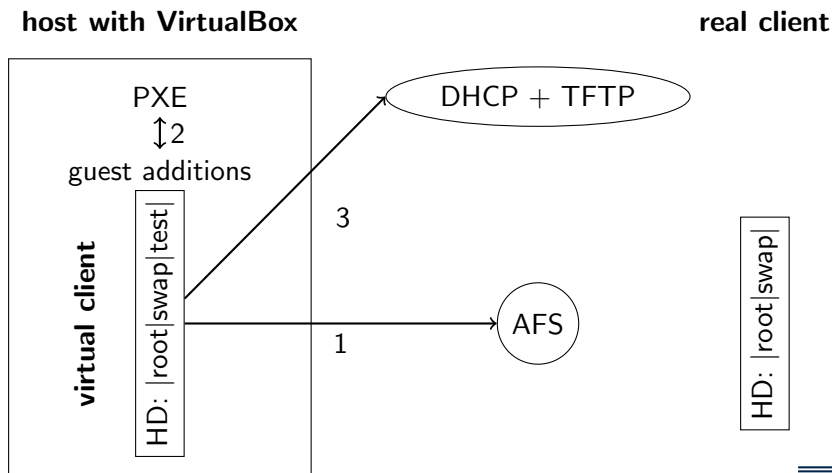
## host with VirtualBox



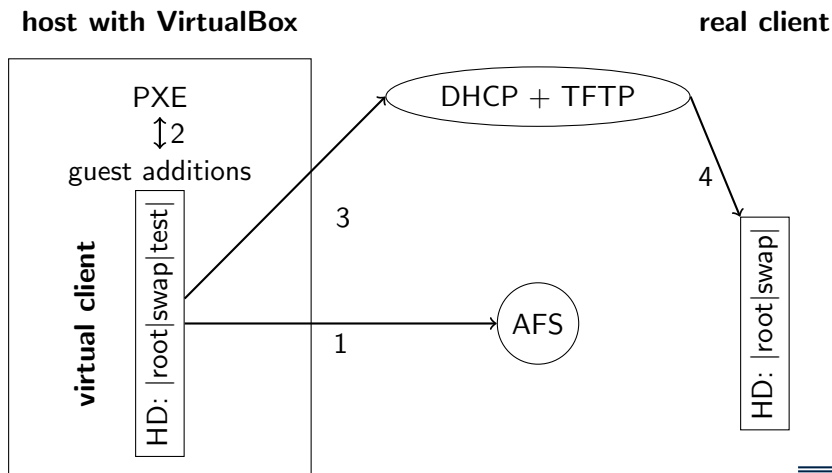
## Sketch



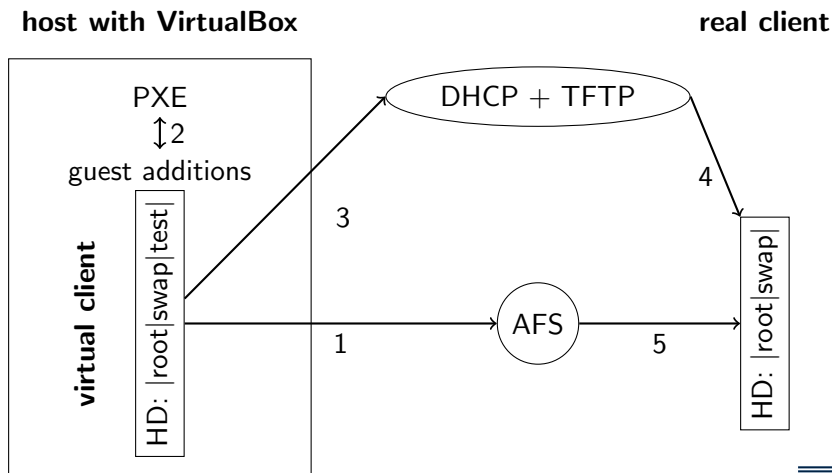
## Sketch



## Sketch



## Sketch



# Virtualization



# Virtualization

## why

- creation of one virtual master image
- test host OS independent (Linux, Mac OS X, Windows)
- image just a simple file
- snapshots possible
- several persons can update the image

# Virtualization

- why
- creation of one virtual master image
  - test host OS independent (Linux, Mac OS X, Windows)
  - image just a simple file
  - snapshots possible
  - several persons can update the image

software Virtualbox with guest additions

# Virtualization

- why
- creation of one virtual master image
  - test host OS independent (Linux, Mac OS X, Windows)
  - image just a simple file
  - snapshots possible
  - several persons can update the image

software Virtualbox with guest additions

- benefits
- allows for PXE boot to try resulting kernel and ramdisk
  - partition client's HD accordingly
  - e.g. 1 root + 1 swap + 1 test partition
  - DHCP client as in real installation
  - different virtual network configurations possible

# PXE boot

- 1 boot computer with (PXE supporting) network card
- 2 contact *proxy* DHCP server for boot server and network settings
- 3 contact TFTP server for network bootstrap program
- 4 load NBP into RAM
- 5 optionally verify check-sum then execute the program
- 6 Debian Linux uses pxelinux
- 7 pxelinux: pxelinux.0 and pxelinux.cfg/default
- 8 Linux kernel over TFTP ca. 1.5MB for Debian
- 9 initial ramdisk over TFTP ca. 6.0MB for Debian (max. 32MB ?)
- 10 potentially *multicast* TFTP

# AFS part

- 1 no RW replication hence same problem as NFS
- 2 but RO replication for redundancy
- 3 allowing just RO from clients adds further security
- 4 move as much as possible of the system into RO space
- 5 \*.tar or \*.tgz for variable parts of the file system
- 6 AFS in that case is needed early in the boot process
- 7 client caching reduces network traffic
- 8 vos release for live upgrades
- 9 kernel + ramdisk upgrades on TFTP server potentially serving from RW space in AFS

# Modifications in initial ramdisk

## Modifications in initial ramdisk

- prepare `/afs` mount point and kernel module for AFS
- add all `/bin` and `/sbin` contents
- append all necessary libraries for dependencies
- copy `/usr/bin/fs`
- etc.tar for settings like keytabs or ssh keys

## Modifications in initial ramdisk

- prepare /afs mount point and kernel module for AFS
- add all /bin and /sbin contents
- append all necessary libraries for dependencies
- copy /usr/bin/fs
- etc.tar for settings like keytabs or ssh keys
  
- take care of parsing extra boot parameters
- activate network and start AFS client with memcache
- prepare a swap and root partition with symlinks into AFS
- untar variable space, also /etc from ramdisk
- move /afs mount to destination before pivot\_root



## Modifications in initial ramdisk

- prepare /afs mount point and kernel module for AFS
- add all /bin and /sbin contents
- append all necessary libraries for dependencies
- copy /usr/bin/fs
- etc.tar for settings like keytabs or ssh keys
  
- take care of parsing extra boot parameters
- activate network and start AFS client with memcache
- prepare a swap and root partition with symlinks into AFS
- untar variable space, also /etc from ramdisk
- move /afs mount to destination before pivot\_root
  
- requires at least 1GB RAM and ca. 15MB ramdisk

# Boot parameters

`initrd=initrd.new` name of initial ramdisk

`ramdisk_size=128000` enlarge the size reservation

`ip=dhcp` for the network settings

`root=/afs` similar to NFS for setup script

`afssrc=/afs/dia.uniroma3.it/projects/debian` source for file system  
(bin/ boot/ home.tar lib/ opt/ sbin/ srv/ usr/ var.tar)

`afdst=/dev/sda2` root partition

`swap=/dev/sda3` swap partition

`cell=dia.uniroma3.it` home cell and activation of *setuid*

`cache=100000` AFS cache size for memcache

# Upgrade procedure

# Upgrade procedure

scripting with Makefile under root account in virtual client

# Upgrade procedure

scripting with Makefile under root account in virtual client image in AFS space

- 1 upgrade master client
- 2 activate AFS on master client
- 3 work as root with system:administrators credentials
- 4 rsync of RO parts and tar of RW parts (no hard links)
- 5 small corrections are necessary  
(/lib/udev/devices/ /lib/init/rw/)
- 6 release the master image volume

# Upgrade procedure

scripting with Makefile under root account in virtual client image in AFS space

- 1 upgrade master client
- 2 activate AFS on master client
- 3 work as root with system:administrators credentials
- 4 rsync of RO parts and tar of RW parts (no hard links)
- 5 small corrections are necessary  
(/lib/udev/devices/ /lib/init/rw/)
- 6 release the master image volume

## kernel and initial ramdisk

- 1 dist-upgrade master client
- 2 copy new kernel to TFTP server
- 3 create new ramdisk with modifications
- 4 copy new ramdisk to TFTP server

## Benefits of this setting

- master client as image file within virtualization
- resulting setup can be tested in the virtual environment
- `rc.local` can provide customization based on hostname
- all essential parts are provided from redundant RO space in AFS
- security relevant parts are coded into initial ramdisk
- Linux kernel remains unchanged
- upgrades can be pushed with running clients
- logging is centralized with `rsyslogd` over TCP
- central logserver with MySQL backend
- users can activate further services with `sudo` as necessary using local space below `/var` (deploy `/etc/sudoers`)

## How does it work so far?

- tests done with Debian Linux 5.0 i386
- Ubuntu 10.04 different because of new start procedure for services
- 10 clients work reliably booting in ca. 3 minutes (10/100 network)
- laboratory with 50 clients needs reconfiguration of network hardware
- image file for master client on a *Mac mini* running *Mac OS X server*
- image can be mounted over Kerberized NFS from two *Mac OS X* clients
- *VirtualBox* used with *Mac OS X* as host operating system
- for Debian different *runlevels* can be prepared
- specialized `rc.local` can activate a kiosk mode



## What next?

- extend this approach to Linux servers
- requires special treatment of persistent RW part of the file system
- possible solution could be *GlusterFS* over dedicated network

## What next?

- extend this approach to Linux servers
- requires special treatment of persistent RW part of the file system
- possible solution could be *GlusterFS* over dedicated network
  
- look into alternatives for client
- push changing parts of file system into RO, too
- *unionfs* like in KNOPPIX could be applied

## What next?

- extend this approach to Linux servers
- requires special treatment of persistent RW part of the file system
- possible solution could be *GlusterFS* over dedicated network
  
- look into alternatives for client
- push changing parts of file system into RO, too
- *unionfs* like in KNOPPIX could be applied
  
- reflect to support other client OS
- Linux can make use of just a single image but not Windows (SID ?)
- could also a virtualization layer fit into the initial ramdisk?  
this goes in the direction of *desktop virtualization*