Experiences booting one million virtual machines (and a few tools we developed)

Ron Minnich Don Rudish

08961 - Scalable Computing R & D

Eurosys 2010

The idea

- We're working to boot ten million machines
- We'd like to run a real botnet at scale and scale seems to be "huge"
- Of course, the numbers are open to argument, but ...
- "A computer botnet is known to have breached almost 75,000 computers in 2,500 organizations around the world," – last week
- Found almost by accident

Over 10M compromised in US

- No. 1: Zeus: 3.6 million
- No. 2: Koobface: 2.9 million
- No. 3: TidServ: 1.5 million
- No. 4: Trojan.Fakeavalert: 1.4 million
- No. 5: TR/Dldr.Agent.JKH: 1.2 million
- No. 6: Monkif: 520,000
- No. 7: Hamweq: 480,000
- No. 8: Swizzor: 370,000
- No. 9: Gammima: 230,000
- No. 10: Conficker: 210,000 <<= we thought this was bad!
- Source: http://www.networkworld.com/news/2009/072209-botnets.html

Previous work

- 2008 – boot 50,000 VMs on Talon (128 nodes)
- 2009 – boot one million VMs on Thunderbird (4460 nodes)
- Create "sandbot" – prototype bot network bot
- Showed pretty pictures at SC2009

How are botnets built?

- Typically "overnet" (nice writeup at wikipedia)
  - So-called because it is an overlay network
  - i.e. it has structure "overlaid" on th internet
- Many use edonkey2 protocol
- The legal overnet taken down 2006
- Like that did any good, because:
- The illegal version out there, alive, and kicking
  - Just try to tell the RIAA that!
- If p2p is outlawed only outlaws will have p2p

Edonkey implemented kademlia protocol

- That's another long talk . . . and wikipedia does a better job than I can do
- Kademlia implements a Distributed Hash Table (DHT)
- Hash is 128 bits
- Nodes have a hash (i.e. 128-bit ID)
- Nodes contain information stored by hash as (key,value) pairs
- Hash uses XOR for "distance" metric

Kademlia network operations

- PING(hash)
  - what you expect
- STORE(hash, value)
- FIND_NODE(hash)
  - recipient of request returns set of nodes with least "distance"
  - For nodes, you want "close to", because you already know yourself
- FIND_VALUE(hash)
  - return value of exact match of hash
  - For values, you want an exact match of course

DHT information values

- For talking to a node: (IP, port)
  - can be used to contact other nodes
- Otherwise, whatever you want
  - Movies
  - Songs
  - RIAA takedown notices
- And here's an interesting thought:
  - Executables
  - Command files
  - commands

Put it together

- You have a way to uniquely name a node with low probability of collision
- You have a distributed way to:
  - Find a node
  - Join the set of nodes
  - store information
  - query information
- So you've got a fault-tolerant, distributed, programming support environment

RIAA shut down the legal uses

- But it's all there for the bad guys
  - And they use it
- Again, that's another very long talk but, as usual, wikipedia has great foundation article
- The statistics are overwhelming
- And kind of hard to verify: how do you really know, if every attempt to probe it is foiled?
- But it's real enough to scare researchers
  - Some are physically afraid!

So what to do?

- One possibility is to apply High Performance Computing (HPC) resources to attempts to understand behavior
  - 180,000 core/30,000 node "Jaguar" at Oak Ridge
  - 20,000 core/5,000 node "Thunderbird" at Sandia
  - And all those little 10,000 core systems out there
- They all run Linux

What we're doing

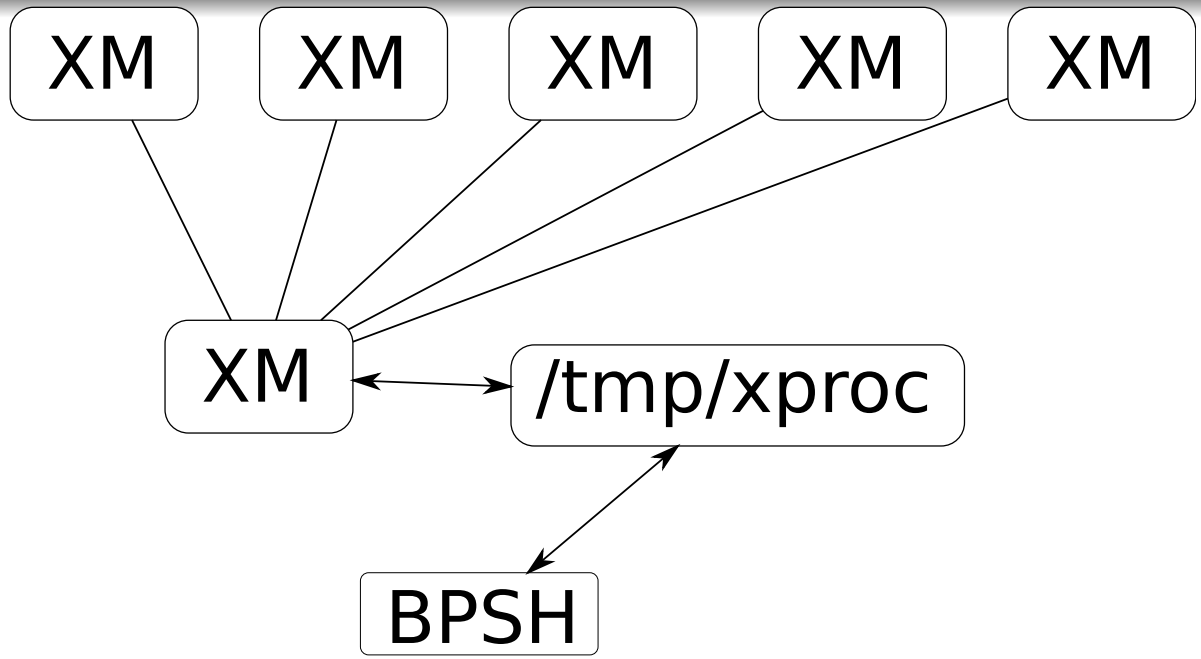
- Use OneSIS cluster software (onesis.org)
  - Used to bring up 4600-node cluster (T-bird)
  - Relied on NFS root in earlier version
- Extend OneSIS with what we learned from Los Alamos Clustermatic (9grid.net/clustermatic)
  - Extremely light-weight, RAMdisk-based nodes
  - Can boot a node w/20M footprint
  - Compare to huge footprint of most cluster software

Result: extremely light nodes

- With lots of room for . . . lots of Virtual Machines
- On T-bird nodes, 250 are easy, x4600 nodes
- Modern nodes, 1000 are easy, x10K on Cray XT4
- So we've gone to 1M on T-bird
- And we hope to go to 10M on Cray XT4
- Was it easy? No. Success once, failure once
  - How I hate IPMI ...
- But it can be done.

Plus new stuff

- Xproc (bitbucket.org rminnich/clustermatic)
- Pushmon
- Vmatic



bpsh -a date
bpsh bundles up:
date command
list of nodes
environment
Date and all
.so's needed
as cpio

Xproc

- Allows you to get remote processes started fast
- Great deal of flexilibity in terms of what files go along
- Since it uses a cpio to move the files anyway ...
- bpsh -f <file or dir) [-f <file or dir>]*
- Allows users to tag things to carry along
- Demo time

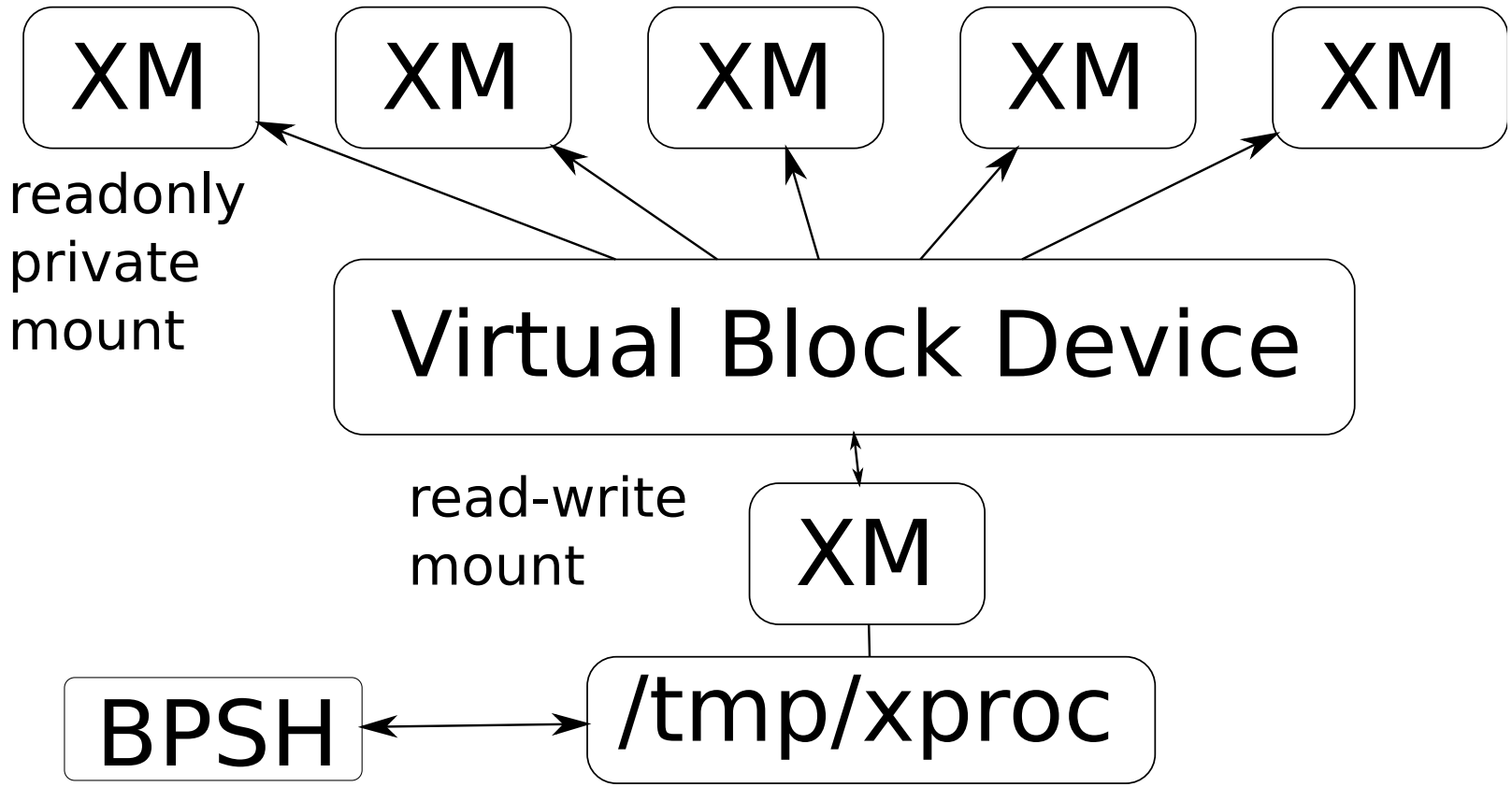
Privacy

- Before xproc starts a child, it forks with CLONE_NEWNS and then creates a private mount on /xproc
- cpio stream is unpacked into that private mount
- Result: that mount is there for proc and all children
- evaporates when proc and all children exit
- Not visible "outside"
- Demo

Efficient mounts for host/guest communications

- Problem: model is that we send cpio over each link
- On the same hosts, with 1000 guests, that's stupid
- So we exploit the private mount and set up a shared host/guest block device
- Files are placed in there by host, read by guest
- The trick: it's read-only and evaporates when process is done
- Avoids conflicting block writes and other issues form multiple guest IOs
- If guest wants to write data it has to send it out over per-guest block device or socket

Block device for efficient IO



Xproc tricks

- Install all libraries to all disks: `bpsh -a cp -a lib/* /lib`
- strace a binary `bpsh 0 -f /bin/cp strace bin/cp`
- Simple change: bpsh 0 -f 'useslibs date' cp lib/* /lib
- Run a command locally: `bpsh -l 0 /bin/date`
- nested execution: `bpsh -f /bin/cp -a bpsh -a bin/cp`

The funnest Xproc trick: local disk

- Create a local script like this:

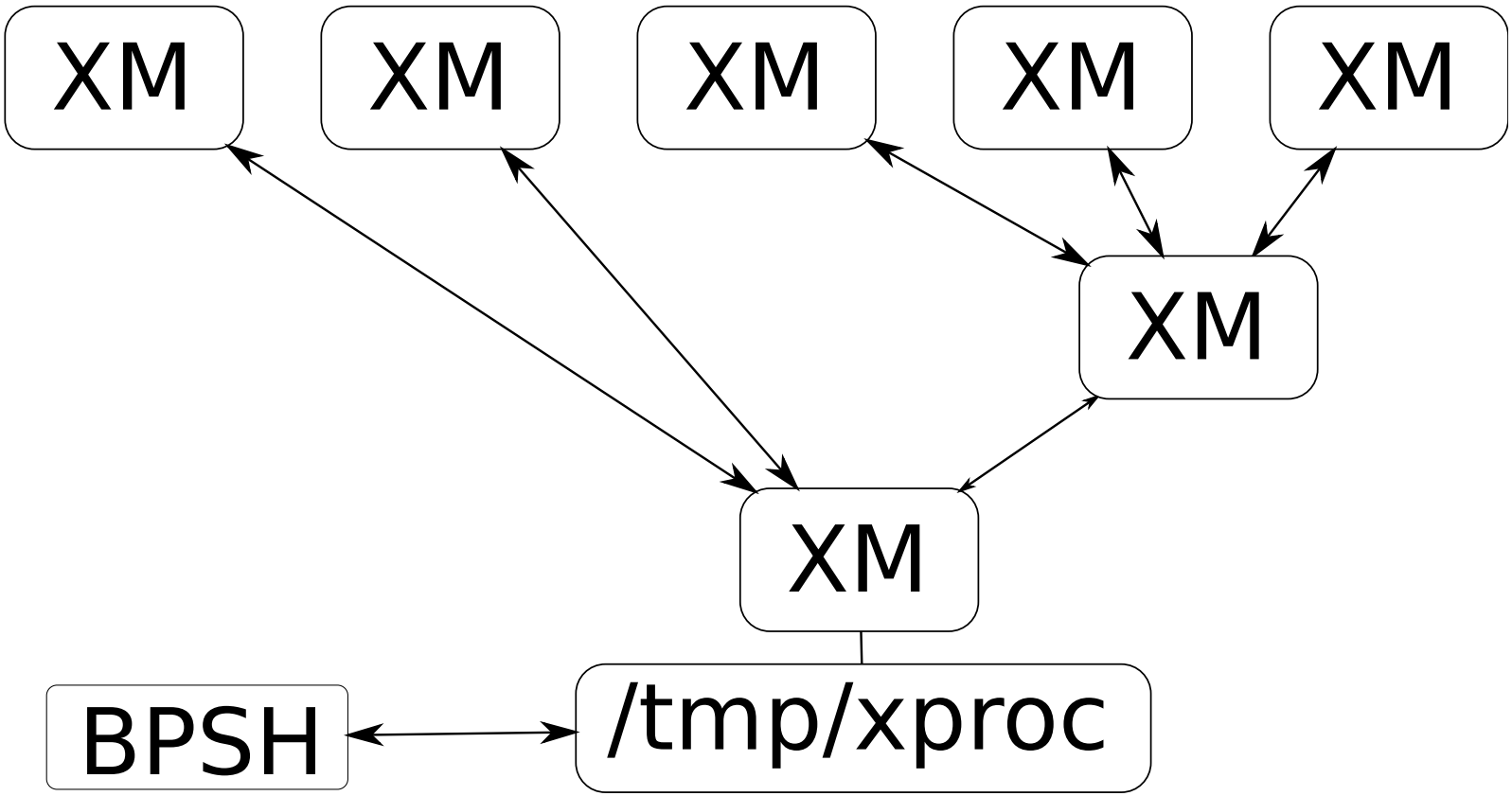
```
#!/bin/sh
mount /dev/vda /xp
LD_LIBRARY_PATH=/xp /xp/date
```

- `bsph -f script -l 0 /bin/sh script`
- And, the shared block device will be used to run the command
- Note this mount evaporates!
- Because the mounts all run in a private name space
- So we get two evaporating mounts for the price of one :-)
- And you *really* want that mount to evaporate since the guests cache disk blocks ...

Tree Spawn

- Around about 500 nodes tree spawn is attractive
- Xproc masters can be slaves too
- And hence we can have persistent tree structure
- Which nodes play which role? Currently determined statically (ran out of time)
- Will be moving to dynamic determination shortly (nothing else is feasible)

Tree Spawn



Hierarchical node numbering

- On bproc we had "flat node space"
- On Xproc we will have a 2-level hierarchy (for now; good to 1M nodes with ease)
- Node IDs below a certain range refer to physical nodes
- Nodes above that range refer to guests (and are the same on all nodes)
- Example: nodes 1-1000 are physical and unique; nodes 1001-2000 are per-node
- To run a command on all nodes:
- bpsh 1-1000 bpsh -l 1001-2000 /bin/date
- Each of nodes 1-1000 runs a command on each of nodes 1001-2000

Pushmon

- Set up hierarchy of processes
- Processes at the leaves push S-expressions "higher up"

```
(("MARK: 1266084142.710273")0x4336 "o0x4336 s1 #0")
(("MARK: 1266084143.272552")0x924e "o0x924e s1 #0")
(("MARK: 1266084145.387336")0x9879 "o0x9879 s1 #0")
```

- S-expressions aggregated as they move "up the tree"
- Result: very low overhead data collection
- Earlier version: each node of 50,000 sends a small UDP packet to an aggregator that writes each packet to NFS mount – destroyed network and server
- Pushmon is very low overhead on networks and file servers

Building a nation scale network on a cluster

- For each cluster node ("host"), start 250-1000 VM's ("guests")
- Need very low-overhead control system
  - Want the time to go to the VM's, not the host
- Need very small memory footprint
  - Primary limit on number of guests is memory
- Need to be able to efficiently start programs on 100s of thousands of guests at a time

At 10M scale

- A DHCP file is at least 350 Mbytes
- DNS (dnsmasq) hash algorithm collapses
- If not routing (as HPC does it), ARP table is at least 100 Mbytes
- Parsing /etc/hosts dominates startup time
- If all nodes talk to all nodes, kernel tables consume a lot of memory
- The tools we use today are designed for a small world
- This is a large world

Lessons learned at this scale

- Tree structure: the node address space, communications, and control must function in a hierarchical manner.
- No node should ever need to communicate with all other nodes – it is unlikely that they will all be up at the same time anyway.
- Ad-hoc: the creation of the hierarchy is dynamic, not controlled by a configuration file. Configuration information, such as node naming, must be described by an algorithm of polynomial, not a static list.
- Dynamic: the hierarchy is continuously changing in response to failures and node restarts.

Continued on next slide.

Lessons learned at this scale
Cont...

- No specialization: nodes must be able to perform any role. A node that is functioning as a compute node must be able to take on the role of manager of other nodes on demand.
- Aggregation model: Individual nodes are not visible; applications operate in terms of groups of nodes.
- Who cares: The fate of an individual node is unimportant – even the root node
- Decoupled (or asynchronous) operation: the application can inititate an operation (e.g. start a program on a set of nodes) but should not depend on an ACK

One lesson we did not expect to learn

- Lguest won out over KVM in performance, memory footprint, convenience – even on platforms with hardware virtualization
- By some measures

	Lguest	KVM
Memory footprint	VM size	VM size + 100 MB (qemu)
startup time 500 VMs	180 seconds	1200+ seconds
OS perturbation	"low"	"high"

- KVM is still more consistently reliable; we see lguest guests dying for now reason
- Do we care? See above: we don't
- At this scale it may not be possible to find out what goes wrong

To sum up

- Lots of machines
- No central configuration possible
- Status unknown/unknowable
- Firehose of data
- Hierarchy is fine but it has to be resilient
- But we know the bad guys do it
  - Botnets of 100s of thousands are old news
  - One vendor had a 2M node botnet ca. 2001
- Might use botnet tools to run this network

What is VMatic?

- Tool for rapidly provisioning virtual machine environments
- Evolved from:
  - Booting VMs on laptops
  - Uses elements from OneSIS
- Now is used to deploy VMs on HPC platforms
  - Small Testbeds (Used in this demo)
  - Hyperion
  - Thunderbird

Kernel

Kernel Attributes:

- Uses minimal kernel configurations to keep the OS from interfering with performance
- Patching elements that are not necessary for VMs and hinder the ability to scale
  - Example
    - Virtio wants to allocate ~40 MB of pages for GSO (Generic Segmentation Offload) TOO AGGRESSIVE for a 20 MB virtual machine

Filesystem

- Ram based
  - Chose Initramfs
    - In the kernel
    - Efficient
    - Simple to use

Image Template

- 1st level of files in the filesystem
- Statically linked programs
- Try to use BusyBox as much possible
- When not possible: we use light weight alternative programs
  - Similar to those seen in embedded systems
- Idea
  - Have most common UNIX utilities already installed
    - Mostly for automated scripting

Filesystem Overlays

- Supplicant to the default template
  - Simple way for users to add files and directories to an image
- Overwrites defaults by taking precedence in filesystem creation
- Binary files
  - VMatic will automatically find and copy all shared library dependencies

Computational Configuration

- Started off by
  - Assigning: IP & HW address to VMs and routers
  - Creating: Routing tables
- Technique
  - Helps create a fast way to realize a chosen network topology
- Minimal burden on cluster administrators
  - Just change the boot target

DEMO

Deploying VMatic on a Cluster

- Cluster name: Drastic Research Testbed
- Consist of 12 nodes
  - Node Names: dn[4-11, 16-19]
    - dn = Drastic Node
  - 4 Core AMD Opteron
  - 32 GB of RAM

Build Image Requirements

- Copy of the clusters existing host table