Network Virtualization and Data Center Networks
Network Testbeds
Qin Yin
Fall Semester 2013
Material uses slides from Timothy Roscoe, Larry Peterson, Jay Lapreau, and GENI.net

Outline
• PlanetLab – Real Network Environment
• Emulab – Network Emulation Testbed
• GENI

PlanetLab

• 1165 machines spanning 558 sites
• A research testbed and a deployment platform for broad-coverage network services

PlanetLab is ...
• Large collection of machines spread around the world for distributed systems research
  – I can deploy and run my code in Seoul, San Francisco, Rio de Janeiro, Moscow, Mumbai, ...
• Focus/catalyst for networking and systems community
  – Most major Universities now host sites

What is PlanetLab good for?
• Planetary-Scale network applications:
  – Low latency to widely spread users
  – Span boundaries: jurisdictional and administrative
  – Simultaneous viewpoints: on the network or sensors
  – Hardware deployment is undesirable
• Long-running services, not just experiments
• Overlay networks

PlanetLab is not ...
• A distributed supercomputer
• A simulation platform
• An Internet emulator
• An arena for repeatable experiments
• Completely representative of the current Internet
PlanetLab is ...

- An opportunity to qualitatively validate distributed system research in a real deployment environment
- An opportunity to gain valuable experience about what works and what doesn’t in the wide area at scale

Why is it successful?

- Community “P2P”-like model
  - “network effects”
  - Lots of benefit from small entry fee
- Sliceability
  - Enables multiple approaches
  - Sharing of the platform
- Virtual machine interface
  - Emphasis on multiplexing the machine
  - Isolation left to the VMM

Motivation

- New class of services & applications emerging
  - Spread over a sizable fraction of the net
  - CDNs, P2P as the first examples
- Architectural components are beginning to emerge
  - Distributable hash tables provide scalable translation
  - Distributed storage, caching, instrumentation, mapping, ...
- The next Internet will start as an overlay on the current one
  - as did the last one...
  - it will be defined by its services, not its transport
    - translation, storage, caching, event notification, management

Lots of work done in big distributed systems

- Researchers had NO vehicle to try out their next n great ideas in this space
  - Lots of simulations
  - Lots of emulation on large clusters
  - Lots of folks calling their 17 friends before the next deadline
- But not the surprises and frustrations of experience at scale to drive innovation

What do people use it for?
Some early examples:

- Overlay Networks
  - RON, ROM++, ESM, XBone, ABone, etc.
- Network measurement
  - Scriptroute, *Probe, I3, etc.
- Application-level multicast
  - ESM, Scribe, TACT, etc.
- Wide-area distributed storage
  - Oceanstore, SFS, CFS, Palimpsest, IBP
- Resource allocation
  - Sharp, Slices, XenoCorp, Automated contracts
- Distributed query processing
  - PIER, IrisLog, Sophia, etc.
- Content Dist. Networks
  - CoDeeN, ESM, UltraPeer emulation, etc.
- Management and Monitoring
  - Ganglia, InfoSpect, Scout Monitor, BGP Sensors, etc.
- Distributed Hash Tables
  - Chord, Tapestry, Pastry, Bamboo, etc.
- Virtualization and Isolation
  - Denali, VServers, SilK, Mgmt VMs, etc.
- Router Design implications
  - NetBind, Scout, NewArch, Icarus, etc.
- Testbed Federation
  - NetBed, RON, XenoServers

What is it good for?

- PlanetLab addresses the related problems of:
  - Deploying widely-distributed services
  - Evaluating competing approaches in a realistic setting
  - Evolving the network architecture to better support such services
- Only game in town for most networking research
  - Other than building into Azureus ...
PlanetLab is ...

- A network testbed
  - 450 active research projects
    - Experiment at scale
    - Experiment under real-world conditions
    - Potential for real workloads and users
- A deployment platform
  - 15 continuously running servers

Slice

- Host
  - Slice
    - Slice pl_green
    - Slice pl_blue
    - Slice pl_red

Guidelines

- Each service needs an overlay covering many points
  - logically isolated
- Many concurrent services and applications
  - must be able to slice nodes => VM per service
  - service has a slice across large subset
- Must be able to run each service/app over long period to build meaningful workload
  - traffic capture/generator must be part of facility
- Consensus on “a node” more important than “which node”
PlanetLab relationships

- PlanetLab <-> member institutions
  - Shared control of nodes
- PlanetLab <-> research users
  - Distributed virtualization, slices
- PlanetLab <-> research builders
  - Shared interfaces, unbundled management
- PlanetLab <-> rest of the Internet
  - Isolation, security, packet auditing
Global View

Evolution vs Intelligent Design

- Favor design principles over a fixed architecture
- Let experience dictate what problems to solve
- Tactically...
  - Leverage existing software and interfaces
  - Keep VMM and control plane orthogonal
  - Exploit virtualization
    - Vertical: management services run in slices
    - Horizontal: stacks of VMs
  - Give no one root (least privilege + level playing field)
  - Support federation (divergent code paths going forward)
    - Minimize universal interface

Distributed virtualization

- Slice abstraction in shared infrastructure
  - set of virtual machines (slivers)
  - Initialized with boot state
- Isolate VMs/services from each other
  - Allocate and schedule node resources
  - Partition or contextualize the available namespaces
  - Provide a stable programming base
- Isolate the Internet from PlanetLab
  - Resource usage accounting and limits
  - Auditing of slice actions

Unbundled management

- Allow parallel infrastructure services to run in slices and evolve over time
- OS defines only local (per-node) behavior
  - Global (network-wide) behavior implemented by infrastructure services
- Multiple competing services running in parallel
  - Shared, unprivileged interfaces between OS and infrastructure services

4 main areas

- Node virtualization
- Resource allocation and isolation
- Network virtualization
- Distributed monitoring

Virtualization Design Alternatives

- Unix processes
  - Isolation is problematic
- Java Virtual Machine
  - Too high-level
- Hypervisors (e.g., VMWare)
  - Don’t scale well – lot of memory consumed by each image
  - Don’t need multi-OS functionality
- Paravirtualization (e.g., Xen, Denali)
  - More efficient
  - A promising solution (but still has memory constraints)
- Virtualize at system call interface (e.g., Jail, Vservers)
  - Reasonable compromise
  - Support large number of slices with reasonable isolation
Node Virtualization

- Linux vservers - virtualizes at the system call level
  - Each vserver is a directory in a chroot jail
    - Shares binaries, but has its own packages and services
  - Each vserver runs in its own security context
    - Own UID/GID name space
    - Limited superuser capabilities (e.g., no CAP_NET_RAW)
    - Confined to using some IP numbers only
    - Confined to some areas of the file system
  - Scales to 1000 vservers per node (29MB each)
- Vserver isolation
  - Each vserver has its own security context
  - Using a combination of security context and UID/GID

Node Virtualization (cont.)

- Communication among vservers
  - No local sockets or IPC
  - But via IP
    - Simplifies resource management and isolation
    - Interaction is independent of their locations
- Reduced Resource Usage
  - Physical memory
    - Single copies of the kernel and daemons
    - Shared read-only and copy-on-write memory segments across unrelated servers
  - Unification (disk space)
    - Share files across contexts
    - using file system immutable invert bit for a primitive form of COW

Vserver Implementation

- Initialize vserver
  - Create a mirror of reference root file system
  - Create two identical login account
- Switching from default shell (modified)
  - Switch to the slice’s vserver security context
  - Chroot to vserver’s root file system
  - Relinquish subset of true super user privileges
  - Redirect into other account in that vserver

Isolation and Resource Allocation Alternatives

- KeyKOS - strict resource accounting
- To reduce crosstalk among application requiring soft real-time guarantees
  - Processor Capacity Reserves
    - Accounting across control transfers
  - Nemesis
    - Restructuring OS to eliminate server processes in the data path
  - Scout - scheduling along data paths (SILK)
    - CPU scheduling, network accounting and safe raw socket

Interface

- Rspec (resource specification)
  - Resource requirements about
    - Reservation for physical resources (CPU, bw, disk)
    - Limits on logical resource usage (file descriptors)
    - Assignments on shared name spaces (port numbers)
    - Slice privileges
    - Start and end time
- Rcap (resource capability)
  - The right to use a set of node resources
- Node manager:
  - Rcap <- acquire(rspec)
  - Bind(slice_name, rcap) – slice creation time (vserver creation)

Slice Creation
Implementation of Resource Allocation

- **Non renewable resource**
  - Disk space, memory pages, file descriptors
  - Appropriate system calls wrapped to check with per slice limits
- **For renewable resources (CPU, line bw):**
  - Fairness: each of N vservers gets 1/N of the resources during contention
  - Guarantees: each slice reserves certain amount of resources (e.g., 1Mbps bandwidth, 10Mcps CPU)
    - Left-over resources distributed fairly
- **Cap per-vserver total outgoing bandwidth**
  - Hierarchical token bucket queuing discipline (htb)
- **CPU scheduling**
  - SILK kernel module leveraging Scout
  - Proportional share scheduling using resource containers

Network Virtualization

- **Using filters on network send and receive to access raw network devices**
  - like Exokernel and Nemesis.
- **Sharing and partitioning a single network address space**
  - Planetlab: using a safe version of raw sockets.
- **Alternative approach (similar to xen)**
  - Assign different IP address to each VM, each using the entire port space and manage its own routing table.
  - Problem: unavailability of enough IPV4 addresses in the order of 1000 per node.
Protected Raw Sockets

• Services may need low-level network access
  – Cannot allow them access to other services’ packets
• The Scout module
  – manages all TCP and UDP ports and ICMP IDs
  – ensures that there are no collisions between safe raw sockets and TCP/UDP/ICMP sockets
• Provide “protected” raw sockets
  – TCP/UDP bound to local port
  – Incoming packets delivered only to service with corresponding port registered
  – Outgoing packets scanned to prevent spoofing
• ICMP also supported
  – 16-bit identifier placed in ICMP header

Monitoring

• Serve several purposes
  – discover/select resources for a slice
  – monitor node/network health
  – measure/monitor Internet activity
• Http Sensor server collects data from sensor interface on each node
  – local state (/proc) + local view of the network (ping)
  – http://localhost:33080/nodes/ip/name
• Clients can query form the sensor database

Aggregated Bandwidth Usage

![Graph showing aggregated bandwidth usage with labels for 9TB/day, NSDI deadline, SOSP deadline.](image)

Aggregate CPU Load

![Graph showing aggregate CPU load with labels for NSDI deadline.](image)

How to Use PlanetLab

• Don’t expect:
  – Repeatable experiments, other than very long-running studies
  – Large numbers of nodes (approx 500)
  – Lots of CPU (machines are loaded!)
  – High availability (machines reboot without warning)
• Do expect:
  – The unexpected!
  – Real experience running a service
  – Real users (if you want them)
  – Lots of interesting challenges
  – To find out if your idea really works

Best Practice

• Build a real system
• Debug it in the lab on your own network
• Try it out on PlanetLab to ensure it works
• Experiment on EmuLab for repeatability
• Use simulation for scalability
• Cross-validate your results!
• Deploy on PlanetLab to get real experience
Lessons from PlanetLab

- Nothing works as expected at scale!
  - Many unintended and unexpected consequences of algorithmic choices
  - Simulation results do not carry over well
    - Simulate, deploy, measure, edit cycle
- Evaluating competing approaches “in the wild” refines techniques
- The ability to try things out “for real” seems to stimulate ideas

Long-term aims

- PlanetLab incubates the next Internet
  - Now: GENI (more info later)
- New networks deployed as overlays over existing Internet
- Service-oriented network becomes the norm
- Computation as a localizable network resource

PlanetLab Conclusion

- Think of PlanetLab as a communal shared artifact for researchers
- Provides many diverse, overlapping projects around the world with a stable place to stand to change things
- Forum for exchange and composition of services and applications
- Selection environment based on real deployment and use
- Bottom-up approach to changing the world

Emulab Philosophy

- Live-network experimentation
  - Achieves realism
  - Surrenders repeatability
  - e.g., MIT “RON” testbed, PlanetLab
- Simulation
  - Presents controlled, repeatable environment
  - Loses accuracy due to abstraction
- Pure emulation
  - Introduces controlled packet loss and delay
  - Requires tedious manual configuration

Emulab Philosophy (cont.)

- Emulab approach
  - Brings simulation’s efficiency and automation to emulation
  - Artifact free environment
  - Arbitrary workload: any OS, any “router” code, any program, for any user
  - So default resource allocation policy is conservative:
    - allocate full real node & link: no multiplexing; assume max. possible traffic
Key Ideas

- **“Emulab Classic”**
  - Brings simulation’s efficiency and automation to emulation
  - 2 orders of magnitude improvement in configuration time over a manual approach

- Virtual machine for network experimentation
  - Lifecycle & process analogy
  - Integrates simulation, emulation, and live-network experimentation

Two Emulation Goals

- **Accurate:**
  - Provide artifact-free environment
- **Universal:**
  - Run arbitrary workload: any OS, any code on “routers”, any program, for any user

- Therefore, the default resource allocation policy is conservative:
  - Allocate full real node and link: no multiplexing
  - Assume maximum possible traffic

Emulab Virtual Machine

- Achieved through OS techniques:
  - Virtualization/abstraction
  - Single namespace
  - Conservative resource allocation, scheduling, preemption
  - Hard/soft state management

- Benefits:
  - Facilitates interaction, comparison, and validation
  - Leverages existing tools (e.g., traffic generation)
  - Brings capabilities of one technique to another (e.g., nse emulation of wireless links)

Experiment

- Acts as central operational entity
- Represents …
  - Network configuration, including nodes and links
  - Node state, including OS images
  - Database entries, including event lists
- Lasts minutes to days, to weeks, to … forever!

Emulab

- Allow experimenter complete control, i.e., bare hardware with lots of tools for common cases
  - OS’s, disk loading, state mgmt tools, IP, traffic generation, batch, ...
- Virtualization
  - of all experimenter-visible resources
  - topology, links, software, node names, network interface names, network addresses
  - Allows swapin/swapout
- Remotely accessible
- Persistent state maintenance (in database)
- Separate control network
- Configuration language: ns

Experiment Life Cycle

- Specification
- Parsing
- Global resource allocation
- Node self-configuration
- Experiment control
- Preemption and swapping
Experiment Life Cycle

Experiment Control

$ns duplex-link $A $B 1.5Mbps 20ms

Experiment Life Cycle

Swap Out

$ns duplex-link $A $B 1.5Mbps 20ms

Experiment Life Cycle

Swap In

$ns duplex-link $A $B 1.5Mbps 20ms

Experiment Life Cycle

ns Specification

- $ns: de-facto standard in network simulation, built on Tcl
- Important features:
  - Graceful transition for $ns users
  - Power of general-purpose programming language
- Other means of specification:
  - Java GUI
  - Standard topology generators
Assign: Mapping Local Cluster Resources

- Maps virtual resources to local nodes and VLANs
- General combinatorial optimization approach to NP-complete problem
- Based on simulated annealing
- Minimizes inter-switch links & # switches & other constraints ...
- All experiments mapped in less than 3 secs [100 nodes]
- WANassign for Mapping Global Resources (uses generic algorithm)

VLANs and Delay Nodes

- Isolation done with Virtual LANs (VLANs) on our switches
- Traffic shaping done with transparent bridges
  - Invisible to nodes
  - Regular nodes running FreeBSD
  - Dummynet used for traffic shaping
  - Listens for events related to its links

Frisbee Disk Loading

- Loads full disk images
- Performance techniques:
  - Overlaps block decompression and device I/O
  - Uses a domain-specific algorithm to skip unused blocks
  - Delivers images via a custom reliable multicast protocol

Experiment Creation

- Virtual node information
- Physical node mapping
- LAN/link information
- Delay node information
- Log of experiment creation

Experiment Creation Scaling
GENI Design

• Key Idea
  – *Slices* embedded in a *substrate* of networking resources

• Two central pieces
  – Physical network substrate
    • expandable collection of building block components
    • nodes / links / subnets
  – Software management framework
    • knits building blocks together into a coherent facility
    • embeds slices in the physical substrate

National Fiber Facility

+ Programmable Routers

+ Clusters at Edge Sites

+ Wireless Subnets
Summary of Substrate

- **Node Components**
  - edge devices
  - customizable routers
  - optical switches
- **Bandwidth**
  - national fiber facility
  - tail circuits (including tunnels)
- **Wireless Subnets**
  - urban 802.11
  - wide-area 3G/WiMax
  - cognitive radio
  - sensor net
  - emulation

References