

Overlays and Virtualization

- Overlays as vehicles for change
- Overlay-based services
- Shared overlay infrastructures

Overview

- Overlays as vehicles for change
 - » IP as overlay on telephone network
 - » mbone for multicast and 6bone for IPv6
 - » what's needed to make such an overlay succeed?
- Overlay-based services
 - » content-distribution networks (e.g. Akamai, Coral)
 - » gaming networks
 - » multiparty teleconferencing (e.g. ESM, MegaMeeting)
- Shared overlay infrastructures
 - » commercial distributed hosting services (e.g. Savvis)
 - » Planetlab – for experimental distributed services
 - » GENI – Global Environment for Network Innovation

- vehicles for change
- overlay services
- shared infrastructures

Overlays as Vehicles for Change

- Internet started as telephone network overlay
 - » Arpanet used 56 Kb/s “leased lines” from phone network
 - » NSF-net connected supercomputer centers at 1.5 Mb/s
 - lower speed connections to individual universities
 - backbone later upgraded to 45 Mb/s
 - phased out as web and commercial internet providers emerged
- Mbone and 6bone were Internet overlays.
 - » sought to spur deployment of multicast and IPv6
 - » modest success as experimental vehicles, but limited impact on deployment
- What’s needed for an overlay to succeed?
 - » sustained commitment over long haul
 - in Internet, this came from organizational backing (gov’t \$\$)
 - community-driven efforts can succeed (e.g. Linux, gnu) but will fail if insufficient community commitment
 - » compelling application that appeals to many
 - key for Internet was the web

Content Delivery Networks

- CDNs deliver requested traffic on behalf of web sites
 - » reduce load on “home site”
 - » accommodate traffic variation – flash crowds
 - » reduce latency in delivery to end users
 - » improved reliability
- Why are they useful?
 - » web sites need not be equipped for peak loads
 - » distributed servers reduces impact of network congestion
 - congested peering points, sub-optimal inter-domain routes
- Examples:
 - » Akamai – original and largest commercial CDN
 - operates over 10,000 servers in over 1,000 networks
 - “Akamaized” content directed to Akamai serves
 - DNS used to re-direct traffic and load-balance
 - » Coral – research CDN on PlanetLab
 - similar techniques to Akamai
 - free, open platform (but you get what you pay for?)

- vehicles for change
- overlay services
- shared infrastructures

Basic CDN Operation

- Web site substitutes CDNized URLs for selected content.
 - » for Akamai – `www.foo.com/images/logo.gif` becomes `a836.g.akamaitech.net/7/836/123/e358f5db0045e9/www.foo.com/images/logo.gif`
 - » for Coral `www.cse.wustl.edu` becomes `www.cse.wustl.edu.nyud.net:8090`
- DNS query directed to DNS server for CDN domain.
- DNS server for CDN domain selects server and returns its IP address.
 - » server selection based on proximity to source
 - using source IP address and other information on ISP connectivity
 - » also may consider server load, likelihood of content presence, type of content
- “Old” information automatically removed from cache
 - » based on time-to-live
- Dynamic content handled using “edge-side inclusion”
 - » see www.esi.org

Akamai

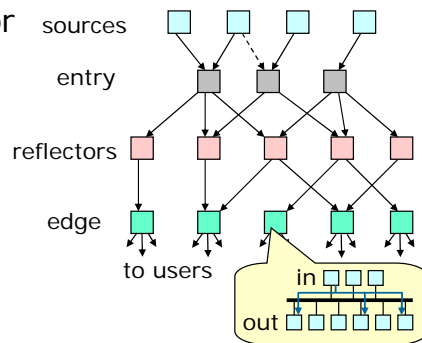
- Started in 1999, quickly became dominant CDN
 - » 20,000 servers in 1,000 networks, in 71 countries
 - » presence in many ISP nets allows traffic to bypass bottlenecks
- Data mapped to servers using consistent hashing
 - » allows DNS servers to work with inconsistent views of servers
- Load balancing
 - » within site, servers report load to local DNS server
 - » DNS server selects which server addresses to return for given content request
 - TTL of less than a minute to allow fast response to shifting loads
 - » top-level DNS server balances load across sites
- Monitoring of network structure and performance
 - » obtain information from ISP routers using BGP
 - » continuous probing using traceroute and other tools
- To improve performance for uncacheable objects
 - » split TCP connection – fast staging of data, shorter control loop

Consistent Hashing

- With 20K servers distributed across world, root DNS servers will not always have consistent picture.
- Consider n users that retrieve data from hash table with B buckets, using hashing.
 - » each user only “sees” a fraction of the buckets (say 80%)
- Construct hash function for each user so that
 - » any user can find any item
 - » items are distributed evenly across all buckets
 - » no item need be replicated many times
 - » if new bucket added to user’s view, change to hash function does not move items between buckets in original view
- Sketch of technique
 - » h_1 maps $items \rightarrow [0,1]$, h_2 maps $buckets \rightarrow [0,1]$
 - » look for item i in bucket B for which $|h_1(i) - h_2(B)|$ is smallest
 - » if users see most buckets, small amount of replication suffices
 - » use search tree to find closest bucket
 - speedup search using separate search trees for subranges

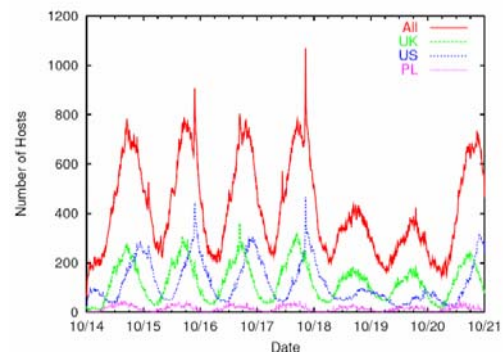
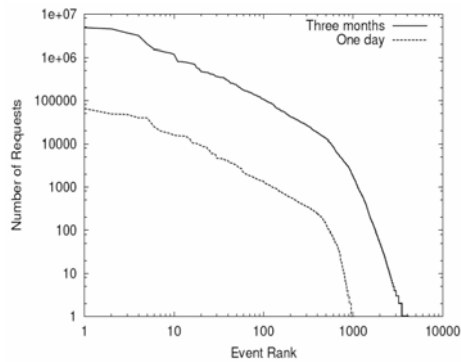
Real-Time Streaming in Akamai

- CDNs make big difference for audio and video
 - » nearby servers provide low startup delay
 - » can sustain greater consistency in delivery
- Live streaming requires multicast distribution
 - » three level organization
 - » entry points stream data from sources
 - » reflectors copy and forward
 - » edge nodes send to sinks
- Ethernet multicast used in reflectors and edge nodes.
- Announce/subscribe model used to distribute streams.



- Quality improvement
 - » re-send recent lost packets
 - » adaptive multipath trans.
 - avg < 1.2 paths
 - » pre-bursting on startup
 - use packets in retransmit buffer
 - reduces buffer delay

Real-Time Streaming Traffic Data



- Most popular sources serve many users and long tail
- Radio station traffic (at right)
 - » significant time-of-day variations
 - » occasional flash crowds

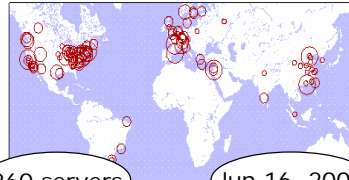
[from *An Analysis of Live Streaming Workloads on the Internet*
by Sripanidkulchai, Maggs and Zhang]

Edge Computing in Akamai

- Web apps often do significant computation at server
 - » present information collected from multiple sources and display in form based on user profile
 - » product configuration application (e.g. for ordering computer)
 - » collecting input from forms
- If loads are highly variable, can be expensive for web site to configure for worst-case
- Applications split into *edge* and *origin* components
 - » presentation processing at edge, backend databases at origin
 - can optimize communication by exchanging raw data, not html
 - » number of edge component instances varies with load
 - attempt to run on servers close to user
- Technical issues
 - » security – separate processes per application, resource limits
 - » load balancing – two level hierarchy, largely using DNS
 - » maintaining session state – session objects stored reliably by “system” with local cached copy provided to application

Content Distribution in Coral

- Coral intended to improve performance of small sites
 - » no formal infrastructure, or supporting organization
 - » uses PlanetLab or other donated resources
- Technical elements
 - » DNS redirection used to associate hosts with nearby caches
 - » hierarchical clustering mechanism
 - each node belongs to clusters (defined by delay) at several levels
 - DHT for each cluster used to index hashed pages
 - » proxy servers retrieve nearby cached copies whenever possible
 - short-lived page reference added to DHT while waiting for page
 - longer-lived reference added when page received
 - » users assigned to proxies based on measurements
 - » sloppy DHT spreads (key,value) pairs in cluster around target
 - during put, if node nearest to key in id-space already stores several references, new pair stored at next-nearest node
- Na Kika for dynamic content



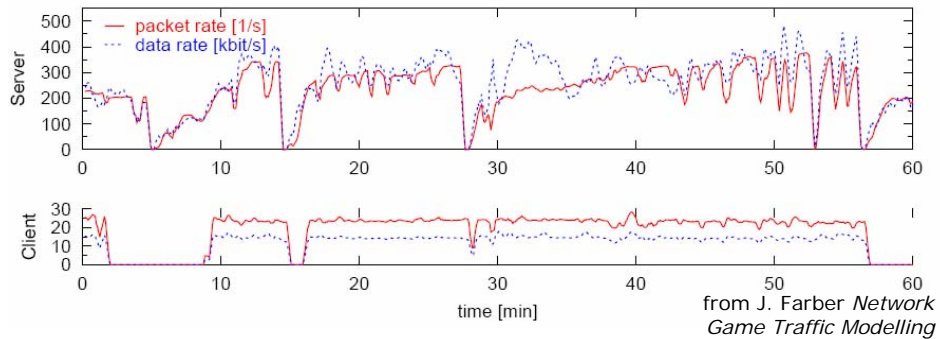
260 servers

Jun 16, 2005

Online Gaming Networks

- Centralized approach
 - » specialized collection of servers at single site
 - » may involve multiple server types, including back-end database processors to maintain game state on disk
 - » thousands of concurrent users, tens of servers
- Peer-to-peer
 - » individual peers coordinate different parts of game world
 - collect and distribute state updates
- Distributing game servers
 - » reducing latency between server and users
 - » dedicated bandwidth among server sites
- Utility computing model
 - » emergence of game service companies that provide hosting
 - » relieves game company of infrastructure maintenance
 - » ability to rapidly adjust capacity to demand
 - » more network "points of presence"

Network Traffic Issues for Games

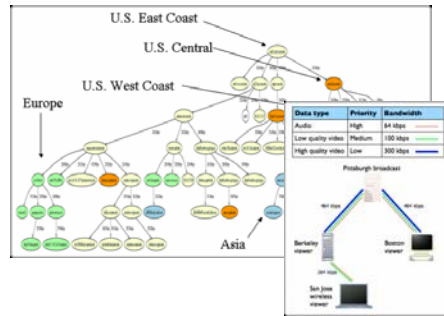


- Data from multi-player action game – LAN-party
- Modest data rates – 20 p/s, <20 Kb/s per client
 - » only state updates must be communicated
- Server load primarily function of game computation
- Latency is key issue for some games
 - » 50 ms needed for fast action, 100 ms ok for role-playing

Video Broadcasting & Conferencing

■ End System Multicast

- » peer-to-peer infrastructure for video broadcast
- » end systems self-organize into application-level multicast tree
- » layered video transmission
- » upstream bandwidth constrains performance
- » uses Planet-Lab nodes as relays



■ Megameeting

- » commercial conferencing service
- » fully browser-based
 - no software to install
 - uses Flash media server
- » centralized implementation
- » multiparty conferences (≈10 sites)
 - unicast delivery



Commercial Hosting Services

- **Web hosting**
 - » host web sites for small businesses
 - » including web site design and management
 - » eliminates need for dedicated infrastructure and skilled staff
 - » reduces under-utilization of infrastructure
 - » ability to handle unusual traffic peaks
 - » multi-site hosting better global presence
- **General IT infrastructure hosting – utility computing**
 - » hosting service maintains large data centers around the world
 - » blade servers dynamically assigned to different customers
 - » secure data storage at multiple sites to ensure against loss
 - » may also support specialized enterprise network services
 - virtual private networks, VOIP, multicast video streaming
- **What makes such services attractive?**
 - » lets organizations focus on core business concerns, not IT
 - » more efficient use of resources through sharing
 - individual organizations need not configure for peak usage

- vehicles for change
- overlay services
- shared infrastructures

PlanetLab

- PlanetLab is global overlay net testbed
- Uses commodity PCs
- Centrally administered from Princeton Univ
- Users assigned "slices"
 - » processing resources at multiple sites
 - » Linux Vservers
- Use standard tools to implement dist. apps
- Fair sharing of CPUs
- No provisioned network bandwidth



Basic PlanetLab Service Model

- Background
 - » originally designed to solve “test & measure” problem for experimental systems
 - » make it easy for researchers to obtain several distributed PCs on which to configure and run experiments
- PlanetLab users define *slices* and independently add nodes to their slices.
 - » slice definitions placed in PlanetLab Central (PLC) database
 - » nodes poll PLC, download slice defs and configure themselves
 - for each node in a slice, a virtual machine (vServer) is defined
 - no practical limit on number of vServers per node
- Best effort service
 - » no mechanism to reserve fixed percentage of CPU capacity
 - » fair sharing of network bandwidth and CPU (on per-slice basis)
 - network interface bandwidth only – no concept of virtual link
 - standard OS scheduling gives coarse-grained time slices
 - manual mechanisms for getting larger *relative* share
- Allocation services seek to pick best nodes for a slice

Running Services on PlanetLab

- Sample applications
 - » End system multicast (ESM) – p2p video distribution
 - » Coral – content delivery network
 - » Meridian – network positioning service
 - » Scalable Sensing Service (S3) – distributed system monitoring
 - » Oceanstore – persistent global data storage
- Infrastructure does not allow large-scale usage
 - » user-space applications incur high overhead
 - » <1000 nodes; many report 100% utilization for long periods
 - » largest service (Coral) reports 10^{12} bytes/day or <100 Mb/s
 - others much smaller
- Service model limits consistent performance
 - » lack of resource reservation
 - slice performance dependent on competing traffic
 - » coarse-grained time-sharing
 - 10 ms time granularity implies processing of packets may be delayed by hundreds of milliseconds

Resource Allocation in PlanetLab

- Moving to model that supports resource allocation
- Resources are dynamically *bound* to slices
 - » PLC delegates resource allocation to resource allocation services using resource capabilities
 - » services provide fine-grained allocation (in space & time)
- Slices may have CPU *reservations* or *shares*
 - » reservations implemented using tokens, where 1 token is “worth” 1 ms of CPU time
 - » slices with reservations receive new tokens every 10 ms, depending on magnitude of their reservation
 - slice’s processes can run only if it has tokens
 - » slices with shares get small reservation (32 tokens/sec), but get preferential access to “spare cycles”
 - if no slices have tokens, extra tokens created for runnable slices with shares (in proportion to their shares)
 - if no runnable slices with shares, slices with reservations get extra tokens
- Coarse timing granularity due to conventional OS/CPU

GENI

- **Global Environment for network innovation**
 - » major initiative by NSF and network research community
 - » objective: create national facility for research on novel network architectures and applications
 - » will use virtualization to enable multiple experimental networks
 - » now in planning stages – hope to start implementation fall 2008
- **Relationship to PlanetLab**
 - » similar in use of virtualization to enable multiple experimental “slices” to co-exist within a common infrastructure
 - » PlanetLab focuses on distributed applications, not networking
 - limited resource reservation mechanisms, no “virtual link” concept
 - implemented with commodity PCs only
 - » GENI will focus on new network architectures
 - major concern with network security and mobility
 - tighter control of resource usage, better isolation of slices
 - higher performance platforms
 - 10 Gb/s backbone links, Network Processor and FPGA subsystems, in addition to general purpose processors

Case for GENI

- Deficiencies in the Internet architecture
 - » insecure, vulnerable to attack – naive trust model
 - » poor fit for newer network technologies – wireless, optical
 - » poor fit for newer computing technologies – sensors, PDAs
 - » poor fit for some applications classes – QoS, group commun.
 - » insufficiently dependable for mission-critical applications or for serving public in times of crisis
 - » inadequate tools for network management – labor intensive
 - » economic incentives that inhibit investment in infrastructure
- Obstacles to correcting deficiencies
 - » industry alone will not solve the problems
 - conflicting objectives of industry stakeholders
 - fundamental changes require near-universal agreement
 - commercial objectives do not always match larger societal goals
 - » research as usual will not solve problem
 - deployment obstacles pushing researchers towards increasingly incremental research
 - most creative minds looking for more productive areas

GENI Requirements

- *Sliceable* – multiple (1000+) experimental networks
 - » using virtualization to provide isolation
- *Generality* – avoid limiting what researchers can do
 - » implies need for programmable/configurable components
- *Fidelity* – experimental results reflect “reality”
- *User access* – easy for individual users to “opt in”
 - » provide physical access, allow networks to run continuously
- *Controlled isolation* – allow slices to interact if desired
 - » isolated operation by default, interaction by mutual agreement
- *Diversity and extensibility* – use diverse technologies
 - » allow technology experimentation (including optical, wireless)
- *Wide deployment* – many sites; Internet-accessible
- *Observability* – extensive data gathering mechanisms
- *Federation & Sustainability* – long life, many participants
- *Ease of use* – make it easy, but don’t limit “power users”
- *Security* – protect GENI and “do no harm” to Internet

Tensions

- *Sliceability vs. fidelity*
 - » jitter on virtual links
 - » responsiveness of virtual machines on general-purpose OS
- *Generality vs. fidelity*
 - » performance of software/hardware implementations may differ
 - » may address using programmable hardware
- *Architectural design vs. technology development*
 - » enable evaluation of new optical switch, or radio technology
- *Performance vs. function*
 - » programmability, sliceability make high performance challenging
- *Networking vs. application research*
 - » explicitly neutral – but must support research at both levels
- *Design studies vs. measurement studies*
 - » primarily for design, but can also measure Internet
- *Deployment studies vs. controlled experiments*
 - » isolation issues, potential for misuse/abuse by slice users

Engineering Principles

- *Start with a well-crafted system architecture*
 - » modular components that can evolve independently
- *Build only what you know how to build*
 - » avoid feature creep in infrastructure
 - » do interesting stuff in experimental nets
- *Build incrementally, incorporating experience, feedback*
 - » catch problems early by exposing system to real users
- *Design open protocols and software, not stovepipes*
 - » enable ongoing evolution by users
- *Leverage existing software*
 - » don't re-invent things you don't need to
- *Leverage existing infrastructure*
 - » exploit existing Internet and access network infrastructure to reach users

GENI System Components

- High speed backbone
 - » 10 Gb/s links joining 20-30 sites around the country
 - » backbone platforms will provide variety of processing resources that can be assigned to different slices
 - » links shared through provisioned virtual links
 - » optical switching layer – role not yet clear
- Access nodes
 - » about 100 sites, connected to backbone via tail circuits
 - » scaled-down version of backbone platforms with different mix of processing resources
- Wireless networks
 - » several types – 802.11 mesh, 3G/WiMax, cognitive radio, sensor
 - » 802.11 net to have 1000 nodes in dense urban location
 - » ad hoc mesh organization with 25% connected to wired network
 - » wireless access likely at other sites as well

GENI System Management

- **Component manager**
 - » instantiate slices & allocate resources within component
 - provide interface to higher level management elements
 - implements requested allocation and ensures isolation
 - » sensor interface for status reporting
 - » audit traffic leaving GENI for internet
 - prevent renegade slices from interfering with internet usage
- **GENI management core**
 - » coordinates slice management across components
 - slice manager used to create control slices
 - resource controller – monitors component operation status
 - auditing archive – stores sensor data from component managers
 - » multiple instantiations for decentralized management
 - enable private GENI subnets, and federation with other testbeds
 - » unified view provided through common web interface
 - users configure slices, request resources through shared interface
 - masks presence of separate subnets

GENI System Management

- **Infrastructure services**
 - » key management elements, implemented as slices
 - » provisioning service – resource discovery, configuration
 - » resource broker – allocates resources to slices based on policies
 - » information plane – operational status information
 - » development tools
 - programming environments, debugging tools, tracing, logging
- **Underlay services**
 - » general support mechanisms useful to multiple experimenters
 - » security service
 - authentication & authorization support, PKI, delegation certificates
 - » topology service – information about network connectivity
 - unclear whether GENI substrate, slice or Internet
 - » file and naming service – enabling distributed storage by slices
 - » legacy internet service – IP subnet in a slice
 - serve as reference implementation for experimental slices