Topics in Performance Evaluation

Dror Feitelson
Hebrew University

Lecture 9 – Networking and the Internet

Communication protocols and Internet applications should be evaluated in their natural habitat

Their natural habitat is the Internet

Evaluations in Internet context are hard to do

Two approaches:

- The ns2/ns3 network simulator
- The PlanetLab testbed

Why It Is HARD

Paxson & Floyd, Difficulties in simulating the Internet.
IEEE/ACM T. Networking **9(4)** Aug 2001

Simulating the Internet is hard because

- It is very big
- It is very heterogeneous
- It keeps changing

1990s:

- Growth from essentially nothing to 100,000,000 computers
- Growth rate estimated at 100% a year
- Traffic also grows exponentially, with reports ranging from 50% a year to doubling every 7-8 weeks

2000s:

- Nobody even tries to measure how big it is (home computers with intermittent connectivity, NATs multiplex addresses, ISPs hide data, etc.)
- Tens of thousands of autonomous systems (ISPs, companies, government, universities) each with many routers and many many computers

The effect of scale:

- Protocols may work fine on small or medium size networks, but fail on a really large one
  - use of flooding to find info
  - use of superlinear algorithms
- In a large enough network, rare events will occur regularly
  - flash crowds
  - various failures and error conditions

Heterogeneity: the Internet is by definition composed of multiple diverse networks, providing uniform connectivity but not uniform behavior

Highly variable topology with no single representative structure, and dynamically changing routing

Links range from modems to high-speed fiber optics (huge bandwidth gaps), direct connections to broadcast and wireless (way different congestion and loss properties), terrestrial to satellite (differences in latency)

Protocols are standard but subject to variations (different implementations, different bugs), e.g. hundreds of variants of TCP congestion control

Traffic patterns of different applications are distinct

- Plus shaping due to congestion control
- Plus adaptiveness at the application
- Plus various load levels and resulting congestion

Unpredictable changes:

Exponential growth over time (size, connections, traffic volume)

Big changes in statistics, e.g. instability of median ftp connection size

Complete change in dominant use: email, ftp, Mbone, web, P2P file sharing, media streaming

Evolution of protocols and technology – not only the one you are studying, but also all the rest of the Internet

Expected surprises:

- More use of wireless
- More use of native multicast
- Adoption of differentiated services (QoS)
- New "killer apps" arrive
- New business models and changes in pricing (leading to changes in usage)
- New technology such as scheduling in routers

Strategy I: look for invariants
things that empirically hold for a wide range of conditions

- Diurnal activity patterns (hours to days)
- Self similar traffic (sub-second to minutes)
- Poisson session arrivals (independent users)
- Heavy-tailed distributions
- Invariant aspects of topology (distances between continents/cities)

Strategy II: explore design space

Maybe use some factorial design – e.g. change one parameter at a time

But this may miss (nonlinear) interactions

- Protocol parameters and behavior
- Technological variations (router queue behavior)
- Different congestion levels
- Different network topologies
- Different traffic mixes

Note that parameter value ranges may span orders of magnitude

The ns2 Network Simulator

http://nsnam.isi.edu/nsnam/

Why simulate:

- Unlike measurements, can explore new architectures
- Unlike analysis, works with complex scenarios rather than (over)simplified models
  - complex topologies
  - complex traffic patterns
  - adaptive congestion control

In Internet context: simulating new architectures may help avoid "success disasters" – when a design becomes widespread before being fully developed and debugged

Pitfalls:

- Difficulty of verifying that the simulation indeed reflects the intended model
- Subject to all the problems listed above
  - Need to simulate multiple heterogeneous scenarios, as none is fully representative
  - Cannot anticipate innovations that will need to be considered in the future

Community effort:

- Uniform methodology allows for easier comparison of results
- Higher quality and more features than a single group can accomplish
- Create a pool of detailed models that can be used by others
    - Topology generators that create realistic complex topologies
    - Implementations of routing algorithms and protocols, including importing real implementations

Additional benefits:

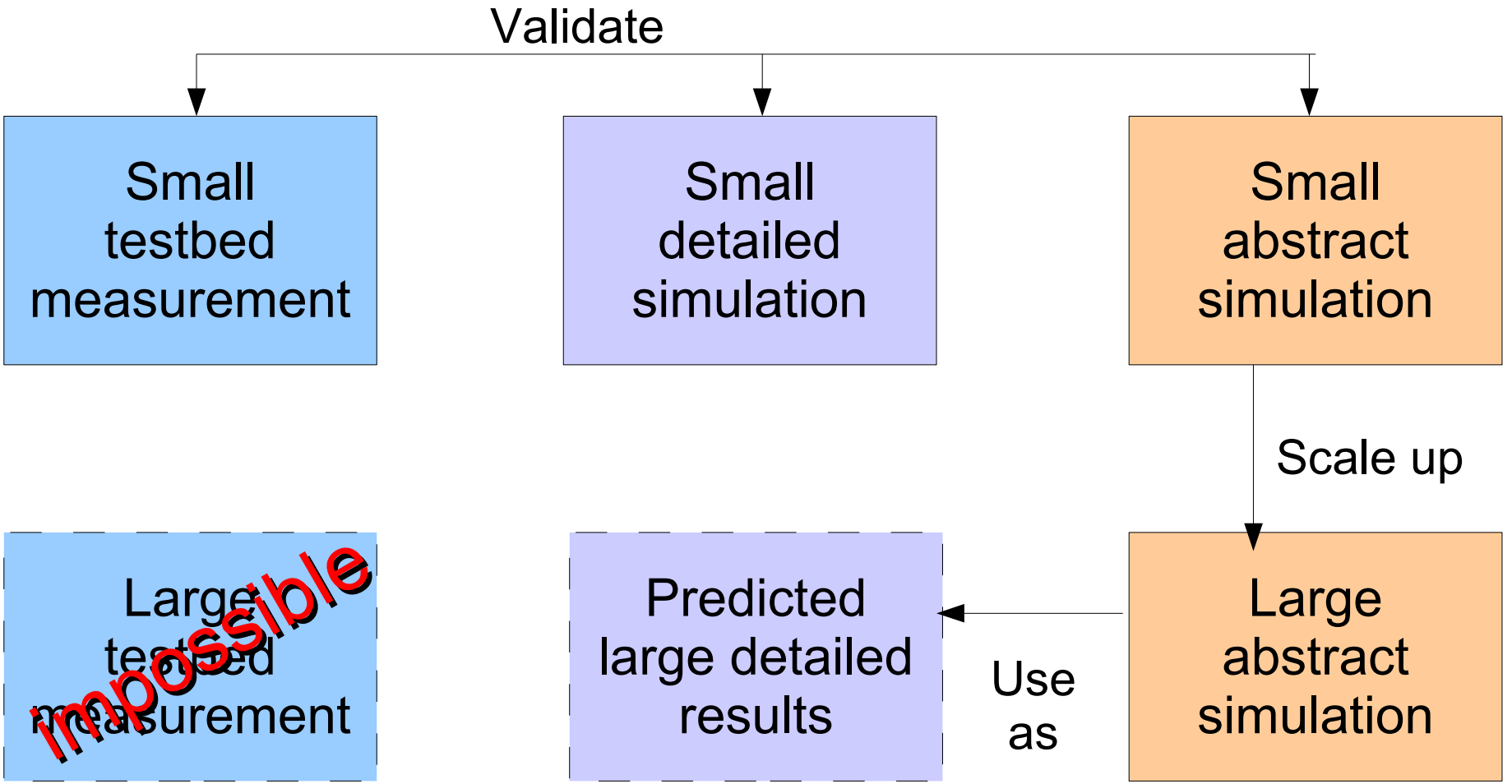
- Users can focus on their networking research and avoid repeated investment in infrastructure
  - For example, study multi-protocol interactions without having to implement all those protocols
- Comparisons against previous work can use the original implementation of that previous work
  - Compare against the right version
  - Compare against a good implementation of the competition

Scaling by abstraction: trade off simulation accuracy for reduced simulation time

- Default simulation is detailed hop-by-hop packet forwarding and dynamic routing changes
- Centralized routing just computes route changes and does not simulate route-change messages
- Session-level packet forwarding replaces hop by hop simulation by precomputed propagation delays
- Tree-based routing instead of shortest paths reduces memory requirements

Validation: compare simulated results with measurements on an experimental testbed

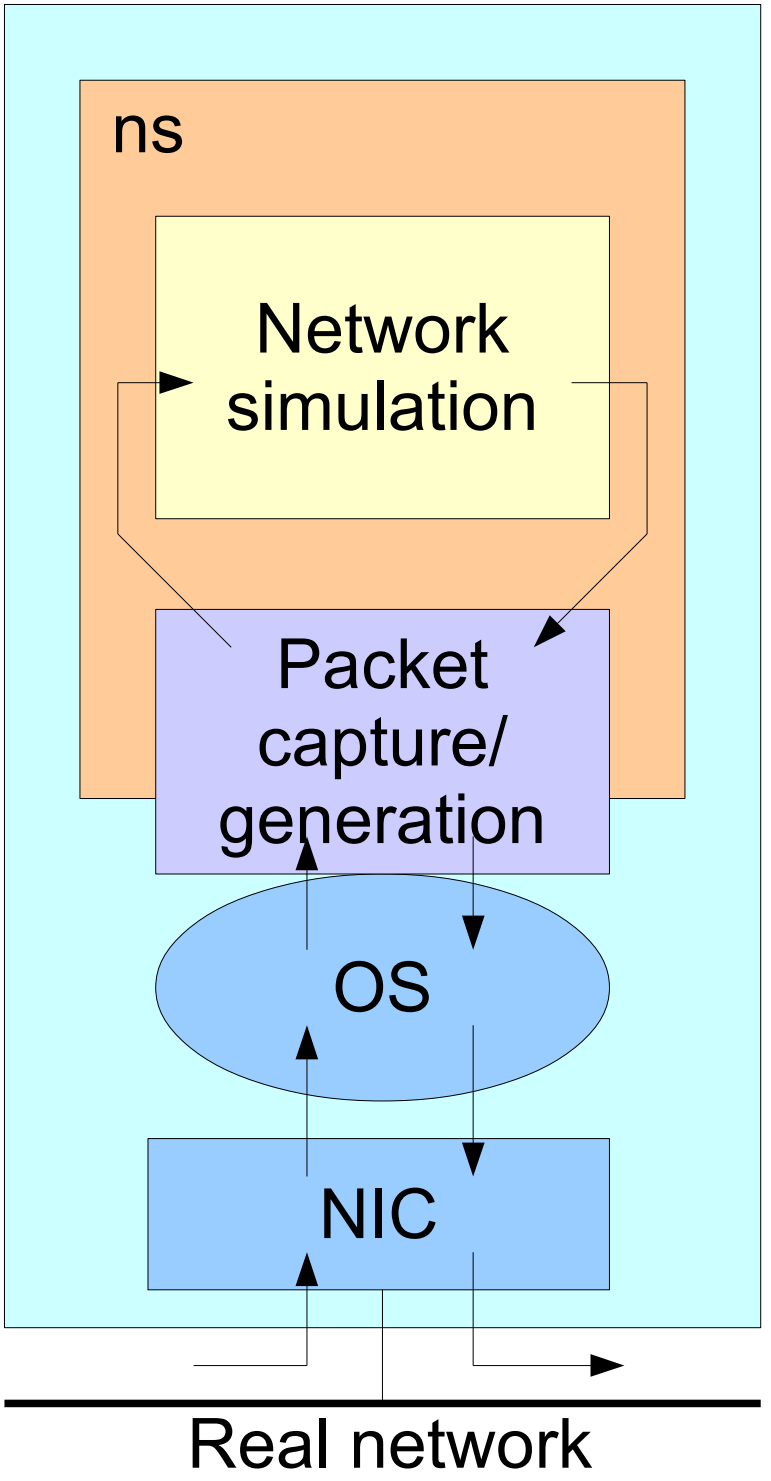
Validate small-scale abstract model and only then scale up



Emulation mode: pass real network traffic through the simulator

Can be used as part of a real-world measurement study to create controlled network behavior that would be impossible to control and reproduce

- Packet reordering
- Packet drops
- Specific delays

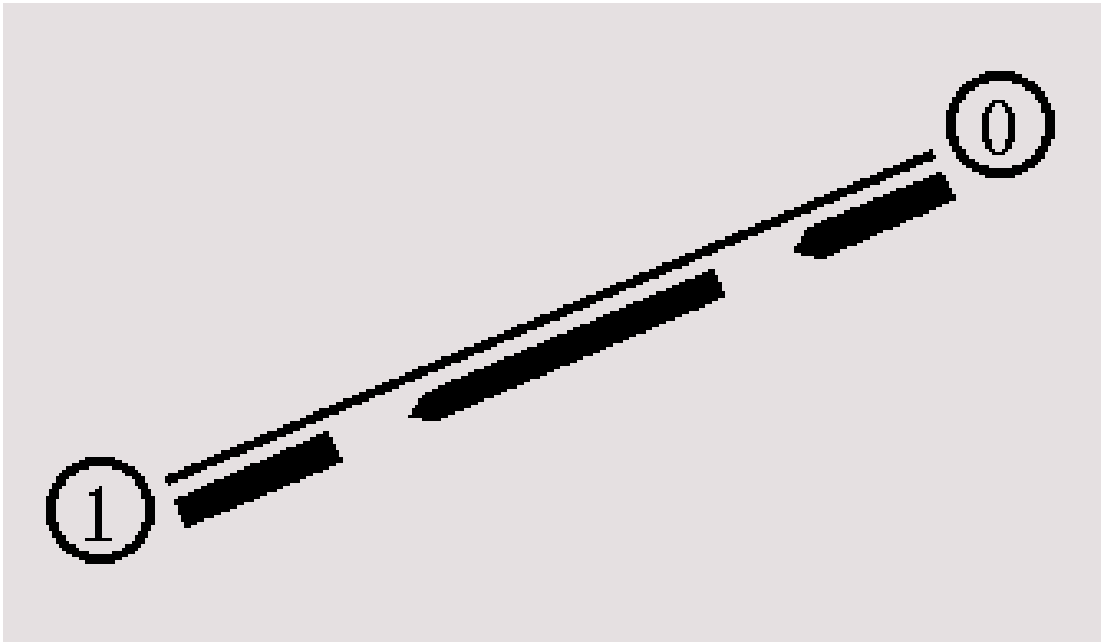


Split-programming approach:

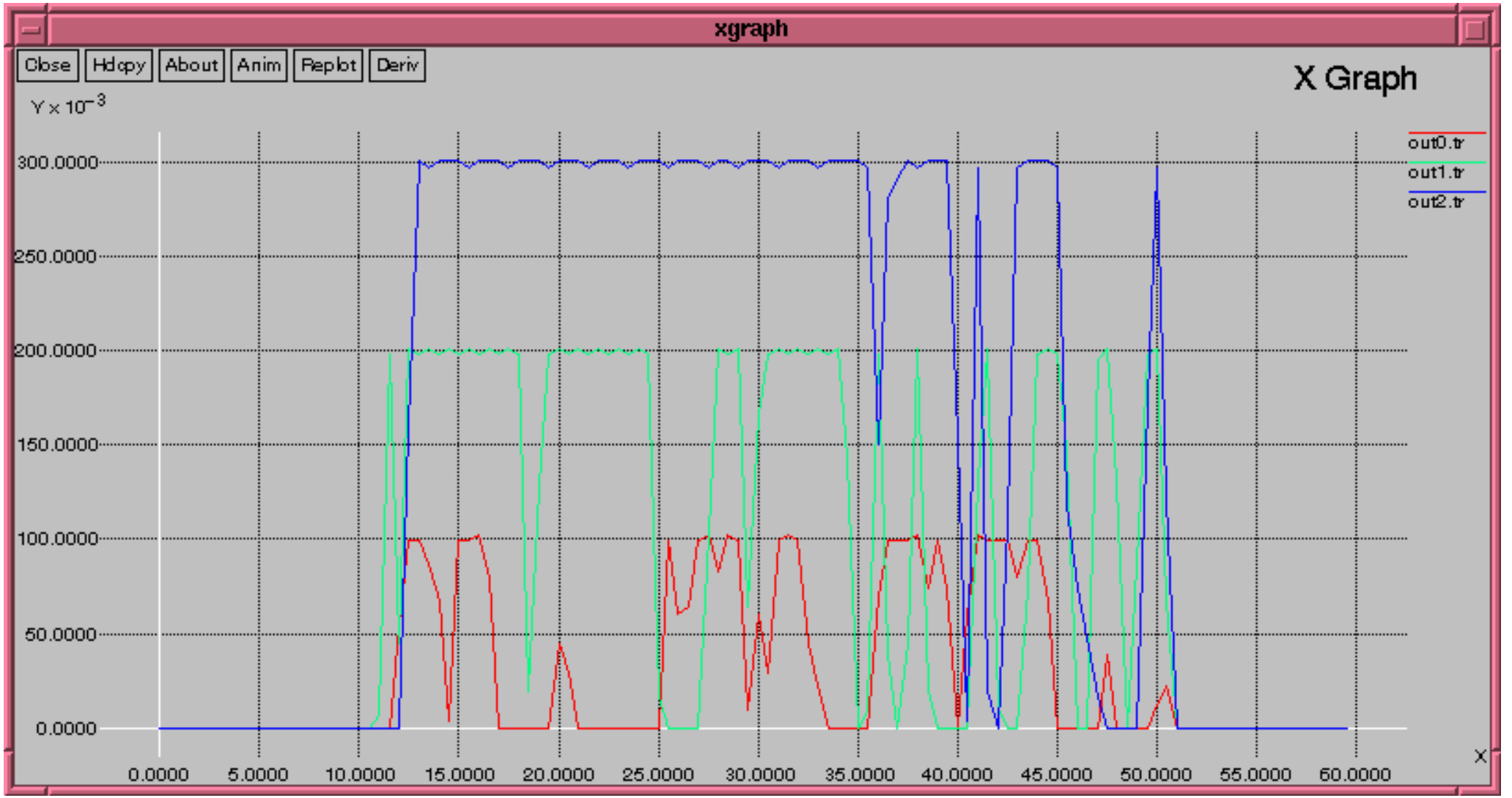
- Core simulation engine written in C++ and compiled for best performance
  - includes packet forwarding
  - includes detailed models for protocols and traffic
- Configuration for specific simulation described using a Tcl script
  - allows iterative refinement without recompilation

```
set ns [new Simulator]

#Create two nodes and a link
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

```
#Create null agent as data sink
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the two agents
$ns connect $udp0 $null0

#Set timing for activity
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```



nam: network animation during the simulation
Xgraph: plot of output performance data



PlanetLab

http://www.planet-lab.org/

Observation: new world-wide distributed services are emerging as the way to go

Hard to design due to distribution and scale

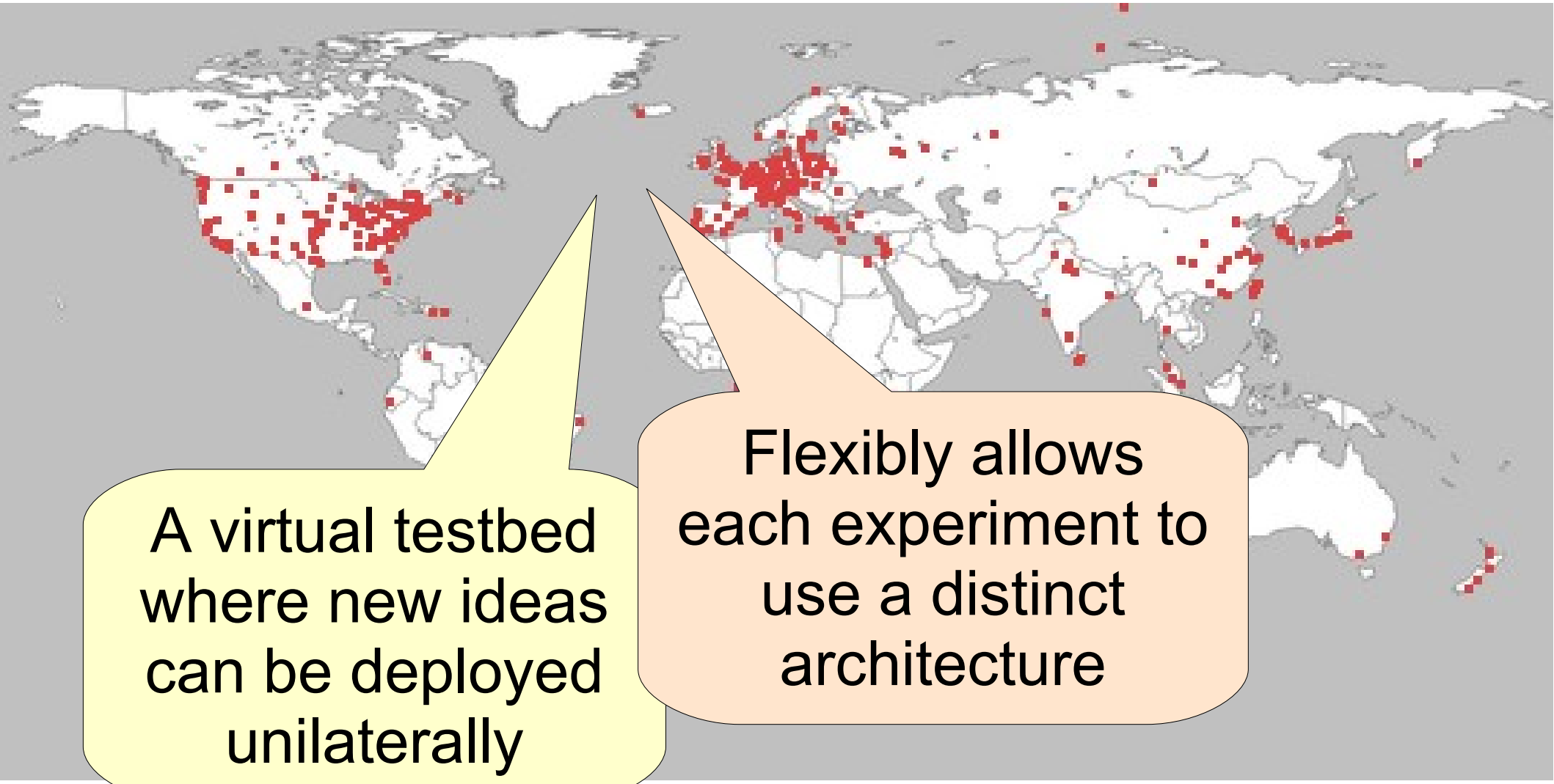
Hard to evaluate due to size/heterogeneity/changes of the Internet

Hard to deploy because counter to current technology

Same problems apply to the Internet's own architecture and protocols

The solution: use a planetary scale overlay network

Started in summer of 2002 with 100 nodes
In 2014: 1188 nodes at 587 sites

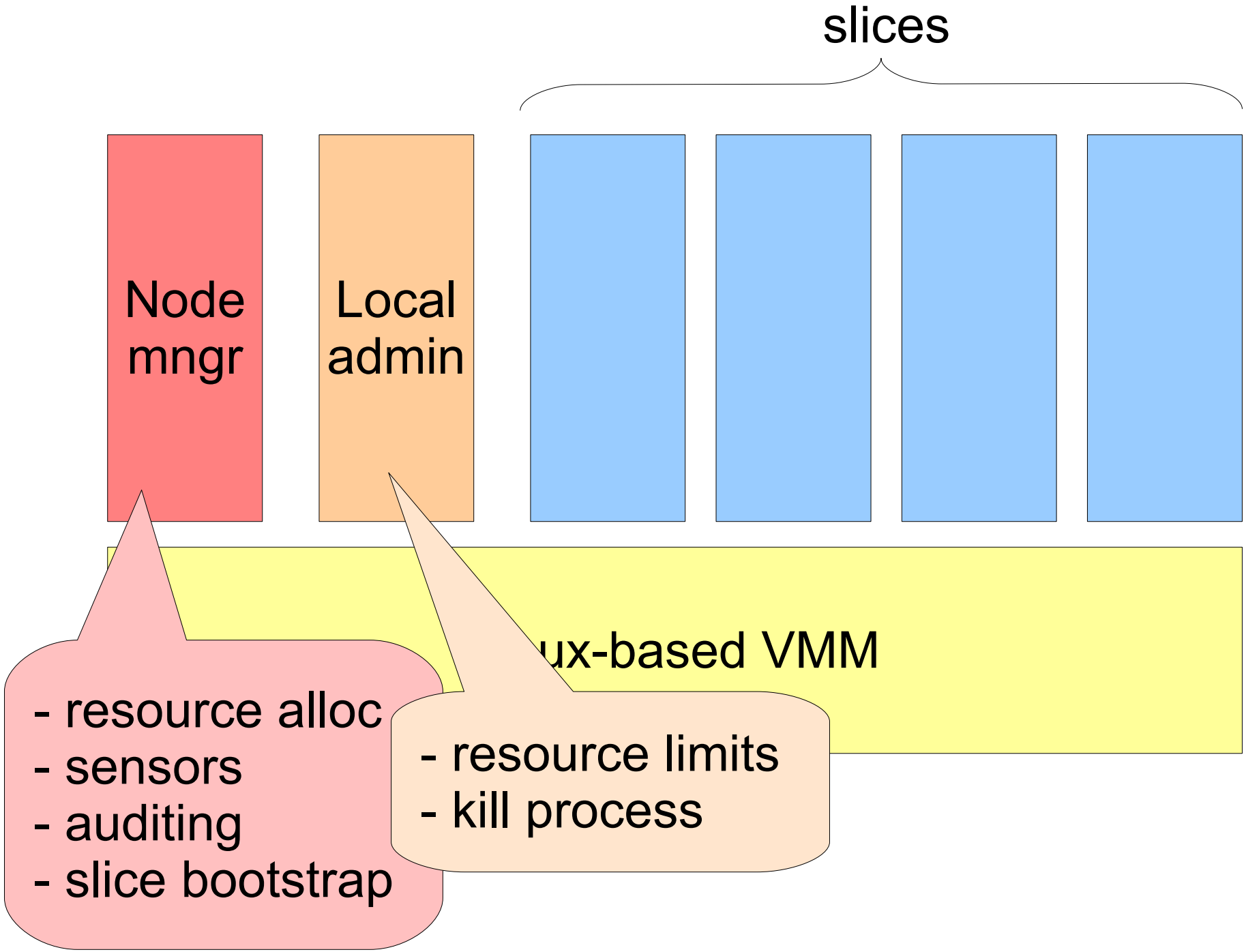


A virtual testbed where new ideas can be deployed unilaterally

Flexibly allows each experiment to use a distinct architecture

Distributed virtualization: provide users with a "virtual Internet", complete with real latencies, real competing/cross traffic, etc.

- A VMM on each node supports multiple virtual machines
- A set of virtual machines across multiple nodes is a slice
- Combine centralized control and allocation with local policies restricting resource use
- Infrastructure services also run in a slice, rather than being bundled with the underlying kernel


slices
Node mngr
Local admin
ux-based VMM
- resource alloc
- sensors
- auditing
- slice bootstrap
- resource limits
- kill process

Use unbundled management:

- Having management functions run in a slice allows for faster evolutionary development
- Alternative control functions can be implemented and compete with each other
- Requires mechanism to allow one slice to selectively manage another slice
  - allocate resources
  - kill processes

The need for isolation:

- Isolate slices from each other, so that each user can deploy an independent service and conduct independent research
- Isolate PlanetLab activities from the host, so as not to disrupt the host's activities
- Isolate PlanetLab as a whole from the Internet, so that experiments/services do not escape and cause global problems

PlanetLab also provides a deployment path for new services

- Services within PlanetLab itself are built this way
- PlanetLab-based services can be used by anyone throughout the Internet
- By really building and deploying a service under real conditions you find out what really matters
  - Engineering tradeoffs in the design
  - What users really want from the service

Evaluation as in a field experiment, as opposed to simulation or a controlled lab experiment

Lessons learned:

- Using a real system exposes "unimportant" assumptions, leads to new research opportunities
- In a real system you need to balance objectives, not optimize one thing as much as possible (as you would for publishing a paper)
- Robust reasonably good systems are better than complicated fragile excellent systems
- Real implementations foster iterative refinement beyond the first idea