NEPI: An Integration Framework for Network Experimentation

Alina Quereilhac, Mathieu Lacage, Claudio Freire, Thierry Turletti, Walid Dabbous
INRIA, Sophia Antipolis, France
{alina.quereilhac,mathieu.lacage,claudio-daniel.freire,thierry.turletti,walid.dabbous}@inria.fr

Abstract: **Many different experimentation environments address complementary aspects of network protocol evaluation, but because of their disparities and complexities it is often hard to use them to reproduce the same experiment scenario. NEPI, the *Network Experimentation Programming Interface*, was created to make evaluation of network protocols and applications easier and more reproducible using different experimentation environments, by providing a uniform object model for designing, deploying, and controlling experiments. In this paper we describe how we enhanced the design of NEPI and provided experiment validation, distributed experiment control, and failure recovery functionalities. We also validate the NEPI approach by implementing support for three complementary environments, a physical testbed, a network emulator, and a network simulator. Furthermore, we show with a concrete experiment use case, available online for reproduction, how easy it is with NEPI to integrate these environments for hybrid-experimentation.**

1. INTRODUCTION

Simulators, emulators and physical testbeds are different approaches to network experimentation that provide a varying degree of repeatability, scalability, instrumentation and realism. These dimensions are all important to guarantee accurate results but can not be completely satisfied by the use of a single approach. While simulators allow fine grained control of experimentation parameters, easy instrumentation, good scalability and perfect repeatability[1], they lack the realism that physical testbeds provide, due to the use of simplified models of the underlying physical network. Emulators are a middle term between these two options, having better repetability and scalability than physical testbeds, and better realism than simulators.

The use of different approaches to conduct experiments grants a better insight on the behavior of the evaluated protocols and applications. In this context, several complementary environments for network experimentation have proliferated. But due to their heterogeneities and complexities it is often hard to use them to reproduce the same experiment scenario, or to make them collaborate to conduct hybrid-experiments. This has introduced the need for an integration framework to simplify conducting network experiments under arbitrary experimentation environments, avoiding human errors, and also to enable experiment reproducibility [2].

[1]Perfect repeatability in simulators is given by the absolute control over experimentation parameters.

[2]Experiment reproducibility requires full knowledge of the experiment methodology and parameters.

NEPI [11], which stands for *Network Experimentation Programming Interface*, was created to address the need of easily managing network experiments across arbitrary experimentation environments, enabling design, deployment and control of complex experiments. The initial challenge, as described in [11], consisted in producing an adequate object model that would be generic enough to unify the description and control interfaces of experiments under different environments, without losing the ability to express the different levels of detail provided by them. In this paper we apply NEPI's object model to support heterogeneous experimentation environments and enhance the framework by adding new functionalities. We enforce separation between experiment design and execution stages, with off-line experiment validation. We also introduce a hierarchical distributed monitoring scheme to control experiment execution, we implement a stateless messages based communication scheme, and add failure recovery mechanisms to improve robustness.

In Section 2 of this paper we introduce the related work. In Section 3, we explain the enhancements made to the framework, and in Section 4, we demonstrate the feasibility of the NEPI approach by implementing support for three complementary environments, *PlanetLab testbed* [9], the *netns emulator* [3], and the *ns-3 simulator* [5]. Finally, in Section 5, as a proof of the framework's ability to conduct experiments under heterogeneous experimentation environments using a uniform object model, we provide a concrete experiment use case. The scripts and steps to reproduce the experiment are made available online.

2. RELATED WORK

There are several experimentation tools available today that facilitate conducting network experiments under different experimentation environments. Among the ones that most approximate NEPI's objectives, we can mention *Emulab* [14], *OMF* [13], and *CORE* [8].

Emulab is a network experimentation framework that unifies emulation facilities with physical testbeds, and also integrates a modified version of the *ns-2* [4] simulator. Similarly, *OMF, cOntrol, Management and Measurement Framework*, and *CORE, Common Open Research Emulator*, are frameworks that support a limited form of the three experimentation approaches, simulation, emulation and physical testbeds. The main drawback in all cases is that the choice of the experimentation environment remains limited to a few available options, and interconnection between different environments is not always transparent.

What sets NEPI aside from other tools, is that it was designed to unify the use of *arbitrary* heterogeneous experimentation environments under a uniform object model, while preserving their different features. And also to enable transparent interaction between them by implementing the necessary communication mechanisms.

3. THE FRAMEWORK

In the present section, we first give a brief introduction to the framework, a more detailed description can be found in [11]. Then, we explain how we addressed enhancements related to experiment validation, control, and robustness.

The experiment's work flow starts with the *design stage*, when the user specifies the network topology and applications to execute. After this, follows the *execution stage*. NEPI takes care of *deploying* the experiment by allocating necessary resources, configuring the environments, and instantiating and interconnecting components according to the user specifications. A *control* step occurs during experiment runtime, where the user can modify the experimental parameters through NEPI's user interface. Finally, NEPI *collects* experiment results, called *Traces*, from remote locations into the local machine, at the user's request. NEPI can be used programmatically by writing a Python script, or graphically with NEF, *Network Experimentation Frontend* [2], a graphical user interface for NEPI.

To model experiments using heterogeneous experimentation environments in a generic way, and still expose environment specific characteristics to the user, NEPI proposes a modeling abstraction based on *"Boxes and Connectors"*. All elements in the experiment, regardless of the environment, are represented as *boxes* associated to other boxes though special ports called *connectors*. Each box has a collection of *attributes*, which hold configuration information, and a collection of *traces*, which represent results associated to the element. For example, a network interface element can be associated to a tcpdump trace result. For each supported experimentation environment, NEPI defines specific box types, and the connector types, attributes, and traces associated to those box types. From the box and connector types of a given association between boxes, NEPI can decide what actions should be performed to instantiate and connect the corresponding elements during the deployment of the experiment. By defining environment specific box and connector types, attributes, and traces, NEPI provides specific information on top of the generic experiment modeling abstraction.

Figure 1 shows two representations of the same experiment reflecting two possible levels of detail exposed by different environments. The representation on the left reflects an environment that allows less detail in the description of a point-to-point connection between two nodes, than the one on the right. This simple example shows how NEPI allows to describe arbitrarily complex experiments in a generic way, providing enough flexibility to adapt to the varying degrees of detail provided by different experimentation environments.

We have seen how a simple experiment can be described using interconnected boxes, but this is not enough to describe an experiment composed of many environment instances, whether

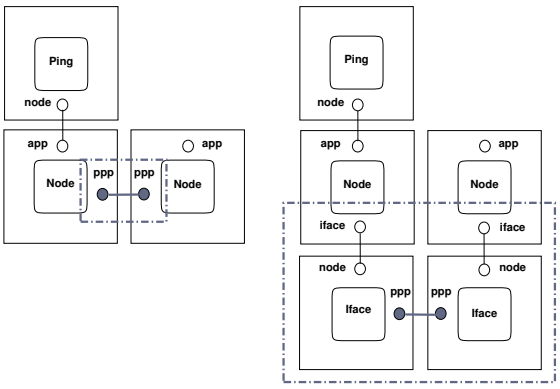

Figure 1: Different levels of detail to represent the same experiment

they correspond to the same or different experimentation environments. To represent this information NEPI provides two other modeling elements, the *TestbedDescription* and the *ExperimentDescription* objects. A *TestbedDescription* object groups the boxes associated to a certain environment instance, and the *ExperimentDescription* object groups *TestbedDescription* objects and supports the serialization of the experiment description to *xml* format. This *xml* description can be used as design documentation, and for later reproducing the experiment in NEPI.

So far we have only discussed how an experiment is described using NEPI, this corresponds to the experiment *design stage*. But NEPI also takes care of performing deployment, control, and results gathering, which belongs to the experiment *execution stage*. Since the initial framework prototype, we have enhanced the architecture and design of NEPI, specially in relation with the *execution stage*, which presented some problems. The initial NEPI prototype used a single *Controller* object to orchestrate the complete experiment execution, this object was also involved in design, which tied design and execution stages. A *Server* object was used to run environment specific code during execution. Having a single controlling entity to monitor the complete experiment and be aware of all experiment components proved to be a deficient approach as any failure in the *Controller* would risk the whole experiment to fail. Furthermore, the communication between *Controller* and *Servers* relied on the *RPyC* [6] Python library to access remote objects as if they were local. This communication mechanism associates a complex state of object references to the *RPyC* connection object. In the eventuality of a connection rupture, recovering the state of object references between local and remote side was not easily done. This issues required enhancements in the design and architecture of NEPI.

3.1. Enhancements

Off-line Validation. One important enhancement consisted in enforcing a clear separation between *design* and *execution* stages. This change was intended to allow to design experiments without the need of interacting with the experimentation environments. Making design completely independent from

execution excludes the need of having testbed resources, or internet access, available and ready to use at design time. This is important when design can take a long time and testbed resources are expensive or scarce. But this flexibility does not come at no cost. The main reason for imposing *on-line* design, with full resource availability, is to support experiment *validation*. In order to defer execution and still be able to validate environment specific configurations during design, we chose to add special *metadata* information for each supported environment version. The validation data is represented by a *Metadata* object that should be reimplemented for each supported environment, and holds sufficient information to perform *offline* evaluation of the attributes values and allowed connections between boxes. Additionally, in environments where the programmatic interface and the behavior can change from one version to another, like *ns-3* or *PlanetLab*, it is necessary to add validation information per environment version basis. To satisfy this requirement a new *Metadata* class can be implemented to support the new environment version in NEPI.

Hierarchical Distributed Monitoring and Control. We added to NEPI a hierarchical structure to monitor and control the different parts of the experiment during execution. Separate specialized *TestbedControllers* are each responsible for monitoring and controlling one environment instance, while a single *ExperimentController* is used to control all *TestbedControllers* and provide global coordination for the experiment. This hierarchical structure allows to decentralise responsibilities, improving modularity and avoiding to concentrate the whole experiment control in a single entity. This isolates failures, facilitating error detection and recovery. If the *ExperimentController* fails, as control of environment instances occurs independently, it is possible to recover by simply relaunching and re-connecting the *ExperimentController* to every *TestbedController*.

Stateless messages based communication. When an experiment requires to be executed in a distributed way, the new implementation of NEPI, employs an ad-hoc stateless messaging protocol for communication between the remote controllers. NEPI assigns to every element in the experiment a global unique numeric identifier, as part of the experiment description, which is used in the protocol messages to identify the target element for the requested operation. This simple communication scheme turned out to be resilient to communication failures because no persistent connection state is required.

Robustness in distributed execution. The new implementation of NEPI contemplates several distributed execution failure scenarios and handles them transparently to the user whenever it is possible. An example of a failure scenario that can be handled transparently is a disconnection error. When it is not possible to handle failures in this way, the failure policy specified by the user on a *TestbedController* is taken into account. The failure policy can instruct a *TestbedController* to fail, to reconstruct, or to rerun the experiment on the testbed. The possible failure policies are limited by the testbed characteristics. For instance, the *PlanetLab TestbedController* uses *Planetlab's* xml-rpc interface to remotely interact with the slice resources, so if the *TestbedController* fails it is possible to reconstruct the controller and reconnect it to the running experiment in the slice. This is not the case for other testbeds, such as *ns-3*, where the simulated experiment is executed by the *TestbedController* process, and in case of failure the only recovery possibility would consist in reconstructing the *TestbedController* and rerunning the part of the experiment from scratch.

3.2. The new architecture

Figure 2 shows how all parts of NEPI come together to conduct network experiments. We modified the architecture of the framework to enforce separation between *Design* and *Execution* stages. The *ExperimentDescription* and *TestbedDescription* objects are used to model the experiment. Each of the latter will represent one environment instance, and will validate elements, attributes an connections using environment specific *Metadata* objects. The *ExperimentDescription* object can then produce an *xml* representation of the experiment, which is used by the *ExperimentController* object to deploy it. For each environment instance participating in the experiment, the *ExperimentController* will construct a *TestbedController* object to handle the communication and control. The *TestbedController* can be constructed locally to the *ExperimentController*, or remotely, using *ssh*, with authentication information provided by the user, to tunnel the communication protocol. For each supported experimentation environments, a *TestbedController* class must be implemented to extend a common execution interface and provide the environement specific implementation. In this way, it should be possible to create a *TestbedController* class to support any arbitrary environment in NEPI by just mapping the generic execution interface to environment specific actions.

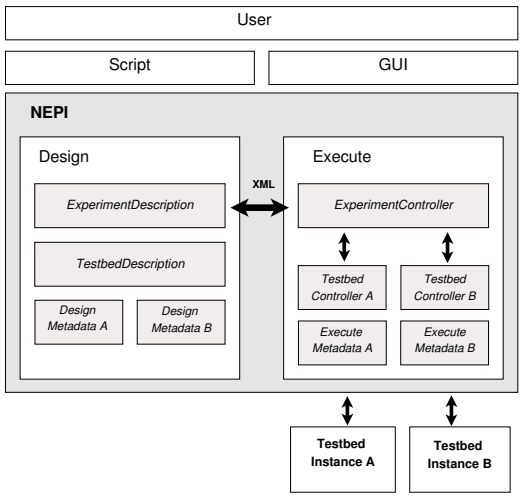

Figure 2: NEPI's architecture

4. SUPPORTED ENVIRONMENTS

In a previous publication [11], we left the open question to whether it was possible to define a uniform object model that could be used with heterogeneous network experimentation environments. In the present section we give an answer to

that question by explaining how NEPI's object model was concretely applied to capture the different degrees of granularity provided by three complementary environments, the *PlanetLab testbed* [9], the *netns emulator* [3], and the *ns-3 simulator* [5].

PlanetLab is a globally distributed network of computers connected to the Internet for research purposes. From an experiment's topology point of view, *PlanetLab* provides little flexibility as it only allows to select nodes connected to the Internet, with a predefined number of network interfaces and fixed IP addresses. NEPI models these components using *Node* boxes connected to a *Internet* box trough *NodeInterface* boxes. A *hostname* attribute in the *Node* allows to select a specific host, otherwise the number of associated *NodeInterface* and the IP addresses set on them, as *Address* attributes, is employed for resource discovery. As arbitrary applications can be executed on the nodes, *PlanetLab* constraints are minimum in this respect. An *Application* box with a *Command* attribute is used to input the execution command. TAP devices are used to create overlay topologies of bridges between environment instances, represented by a *TapInterface* box. Specific connectors in this box, such as *tcp*, *udp*, and others, will define the communication protocol.

netns emulator is a network emulator, developed by the Planete team at INRIA [10], which provides lightweight virtualisation of the Linux network stack. It uses the netns Linux Containers technology [1], to enable emulation of a complete network inside a single host. The *netns emulator* is similar to *PlanetLab* in the type of components it provides to build a network, but differs in that it permits creation of an arbitrary number of virtual nodes and interfaces, and that nodes inside the *netns emulators* are interconnected through virtual Ethernet links (veth) [7], instead of real links. To model *netns emulator*, NEPI also uses *Node*, *Application*, *NodeInterface*, and *TapInterface* abstractions represented by boxes. But *NodeInterfaces* are connected trough a *Switch* instead of an *Internet* box.

ns-3 is a discrete-event network simulator developed mainly for research and education purposes. As opposed to *PlanetLab* and the *netns emulator*, *ns-3* allows a better control of the experiment definition by providing lower level components, such as network stack protocols, loss, and delay models for certain devices. By directly mapping these *ns-3* native components to boxes, it is possible to export the same level of control on *ns-3* experiments though NEPI's object model. In order to provide interconnection with other testbed instances, we created a new type of *ns-3* called *FileDescriptorNetDevice*. This device can read Ethernet frames from a file descriptor, and write into it frames generated by the simulated experiment. When the file descriptor is associated to an external TAP device, this mechanism can be used to communicate a *ns-3* simulation with a *netns* emulation or *PlanetLab* experiment.

The main challenge in applying the object model consisted on choosing the right functional units that would map NEPI's generic modeling components, *Boxes*, *Connectors*, *Attributes*, and *Traces*, to actual environment components. Interaction between different experimentation environment is made transparent to the user by providing special boxes that take care of implementing the necessary communication mechanisms.

5. NEPI IN ACTION

To demonstrate the capabilities of NEPI we will consider a concrete experiment use case, where we want to study the impact of physical rate control algorithms on a video streaming application over a wireless network. One possibility would be to use a physical wireless testbed, but another approach that allows better control of the experiment consists in using *ns-3* to simulate the wireless and mobility models, and the *netns emulator* to emulate realistic network stack conditions, and to allow execution of arbitrary applications.

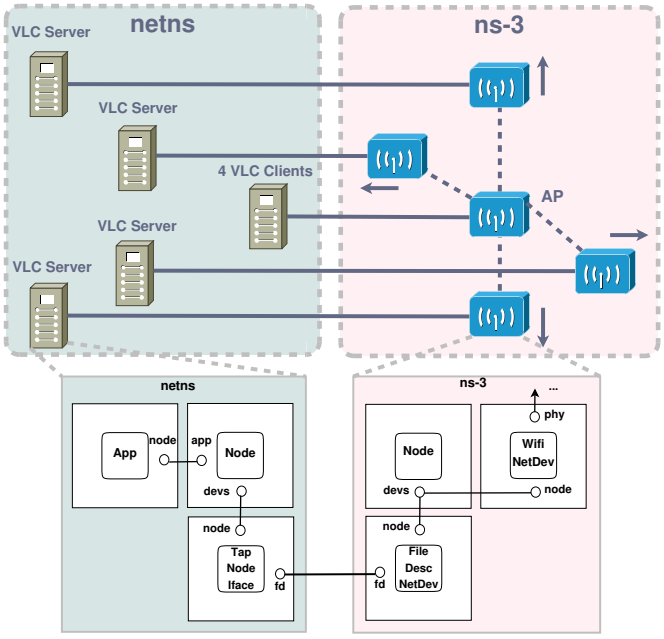

Figure 3: Conceptual diagram of experiment

Figure 3 shows a conceptual diagram of the experiment we are going to conduct using NEPI. This experiment complements the work done on efficient collision detection by Maguolo et al [12], providing validation of the *AARF-CD* rate control algorithm [12] for real video traffic. The experiment consists of a *netns emulator* instance, hosting four VLC servers in four emulated nodes. These nodes stream RTP video to a fifth emulated node hosting four VLC clients. The emulated nodes are interconnected only through a *ns-3* simulated 802.11 wireless network, so the RTP traffic is routed through the simulation.

As stated before, NEPI performs automatic validation of the experiment description during design, without requiring any testbed resources to be available. It prevents setting invalid attribute values, or creating invalid connections between boxes. As an example, NEPI will validate that IP addresses are well formed before deploying the experiment.

Additionally, NEPI takes care of implementing the necessary tunneling mechanisms for achieving data transmission between different environments, and of wrapping this mechanism in testbed specific boxes that can be interconnected. Figure 3 also shows the detail of how the interconnection between *ns-3* and *netns emulator* instances can be designed using *Boxes*

and Connectors. In this example the *TapNodeInterface* box represents a TAP device inside a *netns* emulation, and the *FileDescriptorNetDevice* box represents an *ns-3* object capable of interacting with the outside of the simulation through a file descriptor. When NEPI performs the deployment of the experiment, it will handle the passing of the file descriptor associated to the TAP device in the *netns emulator* side to the *FileDescriptorNetDevice* object in the *ns-3* side, establishing the communication line.

The wireless ns-3 network in our experiment is composed of four mobile stations and one AP, connected respectively to the emulated VLC server and VLC client nodes. The simulated mobile stations are evenly arranged 10 m apart from the AP, and they move away at a speed of 1 m/s until they reach a distance of 100 m. We consider four scenarios where we modify only the rate control algorithm to be used by ns-3 wireless nodes. The objective is to evaluate, under the presence of contention and the increment in the distance from the AP, which of three rate control algorithms, *ARF*, *AARF*, and *AARF-CD*, better preserves the streamed video quality. For the fourth scenario, we chose the *ideal* rate control algorithm, which will be used as a reference for comparison. See [12], for details about these four algorithms.

To measure the received video quality we considered the packet loss per stream, taking into account the packets sent by the streaming servers to the clients within a 3 s time window to ensure correct video playback.

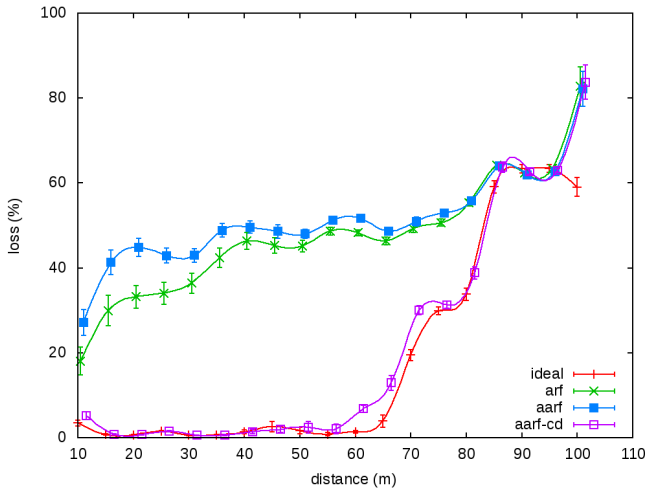

Figure 4: Streaming performance by distance, comparing four rate control algorithms, with 95% confidence interval.

Figure 4 reflects the video quality of the streams as distance from the AP increases for the four mentioned rate control algorithms. As shown in [12], *AARF-CD*, which was created to handle the presence of collisions, is the one that performs best. With this simple experiment using NEPI, we have shown the suitability of the *AARF-CD* algorithm for use in video streaming applications. We have also shown how NEPI enables easily combining different experimentation environments to satisfy research needs. The experiment presented here took only half an hour to design, and NEPI performed the deployment of each run of each scenario in only few seconds.

More information on how to reproduce this experiment and how the results were analyzed can be found at:

`http://yans.pl.sophia.inria.fr/trac/nepi/wiki/nepi/VlcExperiment`

6. CONCLUSION

NEPI intends to improve experimental network research by making it easier to use different experimentation environments to evaluate and reproduce the same experiment scenarios, granting better reliability to experimental results.

In this paper we presented the full design and implementation of NEPI, and demonstrated the feasibility of NEPIs approach by implementing support for three heterogeneous experimentation environments. We also showed, with a concrete reproducible example involving the *ns-3 simulator* and the *netns emulator*, that NEPI makes it possible to easily and transparently integrate different supported environments in a single hybrid-experiment.

For the future development of NEPI, work remains to be done in supporting more experimentation environments and building up the user community. Due to lack of space, we only presented in this paper one experiment case. A journal version is under progress which will include more experiment cases, involving also *PlanetLab*, with more details on the hierarchical distributed control and stateless communication schemes implemented in NEPI.

REFERENCES

[1] Lxc maintainers, linux containers project page. `http://lxc.sourceforge.net/`.

[2] Nef, network experimentation frontend. `http://yans.pl.sophia.inria.fr/trac/nepi/wiki#NEF`.

[3] The netns testbed. `http://yans.pl.sophia.inria.fr/trac/nepi/wiki#NETNS`.

[4] The network simulator. `http://www.isi.edu/nsnam/ns`.

[5] The ns-3 network simulator. `http://www.nsnam.org/`.

[6] Rpyc, remote python call. `http://rpyc.wikidot.com/`.

[7] Swsoft, openvz. `http://www.openvz.org/Veth`.

[8] Jeff Ahrenholz, Claudiu Danilov, Thomas R. Henderson, and Jae H. Kim. Core: A real-time network emulator. In *MILCOM 2008. IEEE*, pages 1–7, 2008.

[9] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[10] Martin Ferrari. Netns: End of studies report ubinet master. `http://hal.inria.fr/inria-00601848/fr/`.

[11] Mathieu Lacage, Martin Ferrari, Mads Hansen, Thierry Turletti, and Walid Dabbous. Nepi: using independent simulators, emulators, and testbeds for easy experimentation. *SIGOPS Oper. Syst. Rev.*, 43(4):60–65, 2010.

[12] Federico Maguolo, Mathieu Lacage, and Thierry Turletti. Efficient collision detection for auto rate fallback algorithm. In *MediaWiN 2008*, July 2008.

[13] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. Omf: a control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev.*, 43:54–59, January 2010.

[14] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. OSDI '02*, pages 255–270, Boston, MA, dec 2002. USENIX Association.