PROJECT SUMMARY

Multicast Protocol Interoperability Testing

Network-layer multicast, an enabler of large-scale collaborative applications, will be a key component of the future global information infrastructure. The emerging wide-area architecture for multicast has several interrelated components. Rapid development of mature, **interoperable** implementations of these components is necessary to accelerate the deployment of multicast.

This proposal addresses tools and methodologies for network protocol interoperability testing, with specific emphasis on multicast protocols. We propose to develop a software testing platform that will allow developers to emulate large test networks using few relatively inexpensive devices, to script–and thereby automate–interoperability testing between independent implementations, and to embed protocol implementations in network simulators.

Multicast Interoperability Testing

USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina Del Rey, CA 90292

1 Introduction

The Next-Generation Internet (NGI) [20] and Knowledge and Distributed Intelligence (KDI) [9] initiatives target a sophisticated global information infrastructure for scientific collaboration. A key component of this infrastructure, and one which will form the basis of several large-scale collaborative and distributed applications, is network-layer multicast.

The design of a scalable inter-domain[1] multicast architecture is a topic of much current research and standards activity. The emerging architecture for large scale multicast has several components: sparse-mode and dense-mode protocols for intra-domain multicast ([5, 7]), a protocol for inter-domain multicast that builds shared and bidirectional distribution trees [13], and an architecture for dynamic address allocation [13]. This seemingly complex, intertwined, infrastructure is necessary to simultaneously support administrative heterogeneity and to scale to large numbers of globally routed multicast groups. This latter requirement is especially crucial for KDI; we believe scalable multicast is imperative for widespread research collaboration.

The design of the multicast architecture results in a system with several software components (Figure 1). Interoperability between these software components is an important practical barrier to multicast deployment. Interoperability issues arise at both the protocol and the implementation levels. For example, not only must the protocols for address allocation must interoperate with the multicast routing protocols, but independent implementations of these protocols must correctly interoperate with each other.

To some extent, extensive simulations combined with test generation can uncover some of the protocol level interoperability problems. However, to carefully test implementation level interoperability, we believe that there is a need for a multicast protocol *software testing platform*. While developing a reference implementation for a multicast routing protocol[2], we also developed a software testing tool called *pkt*. This proposal seeks to extend *pkt* into a more flexible software testing

[1] A domain, in Internet routing parlance, is a contiguous section of the Internet routing fabric that is administratively homogeneous. Examples include campus networks and ISP transit facilities.

[2] PIM-Sparse Mode [7]

platform called *mint* [3]. A unique feature of *mint* will be its integration with *ns*, the packet-level network simulator.

Mint will simplify medium-scale laboratory testing of multicast protocol implementations. As a result, we expect faster time-to-market for mature Internet multicast deployment. In turn, this will lead to the development of large-scale multicast coordination applications that will greatly improve the quality of global scientific collaboration.

Section 2 describes the motivation for multicast interoperability testing in greater detail, and outlines our proposed work. Section 3 discusses the design of *pkt*. In Section 4, we describe our proposed work in detail. Section 5 discusses related work, Section 6 lists the plan for a two-year effort, and Section 7 discusses the expected impact of this work.

2 Background and Motivation

The widespread availability of network-layer multicast will be a key enabler of future *knowledge networking* [9]. Even today, we see multicast used for different classes of distributed applications, ranging from real-time audio and video conferencing [8], to information retrieval, distributed databases [10], and infrastructure service evolution [1]. In the future, we can expect that multicast, because it is a "natural" way of expressing of multipoint communication, will form the basis for control and real-time analysis of remote scientific instruments, for distributed modeling and simulation environments and numerous other collaborative research applications.

This wide diversity in applications is attributable to the generality of the IP multicast service model [6].[4] The service model allows group members to be identified by a single location-independent label. Receivers may dynamically join or leave groups, and sources can send data to a group without joining the group. This model is realized by a collection of mechanisms that enable the efficient and scalable construction of multicast distribution trees. These mechanisms are extremely difficult to design because they must satisfy many, often conflicting, requirements: scalability to many large and sparsely distributed groups, efficient and low-latency delivery of multicast data, administrative heterogeneity, and efficient utilization of IP multicast address space.

Nevertheless, we see emerging an inter-domain multicast architecture that reasonably satisfies these requirements. This architecture contains three logical components:

- A collection of *intra-domain* routing protocols. These are intended to be used for multicast data delivery within a single administrative domain. Examples of such protocols include PIM [7], CBT [2] and MOSPF [16].

[3]the **m**ulticast **in**teroperability **t**est platform

[4]We believe that this service model is essential for developing large-scale collaborative applications in NGI and KDI. However, the emerging next-generation Internet may support other service models that can efficiently realize multicast applications such as video distribution. This only increases the motivation for better support for interoperability testing.

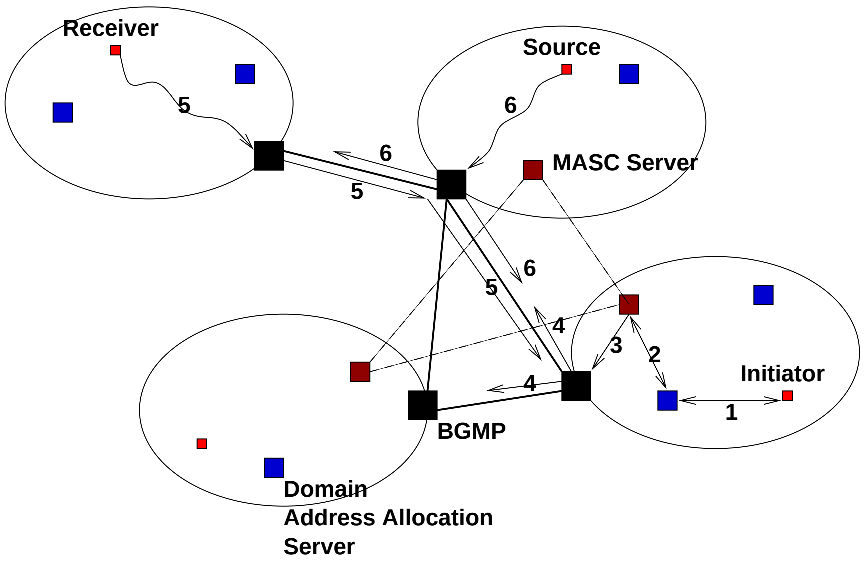

Figure 1: **Chronology of a typical multicast session:** This figure presents a simplified description of the interactions between different components of the Internet multicast architecture. Numbers adjacent to the arrows correspond to list items in the text.

- A standard *inter-domain* multicast routing protocol, BGMP [13], that builds shared, bidirectional, domain-level distribution trees.
- A distributed address allocation mechanism, called MASC [13], which dynamically allocates aggregatable multicast group addresses to individual domains, while attempting to achieve efficient address utilization.

The multicast routing protocols interact with, and derive paths from the underlying unicast routing.

It is beyond the scope of this proposal to describe and justify each component of this architecture in any detail. However, Figure 1 chronicles some of the component interactions that occur during the lifetime of a typical global multicast session:

1. An initiator for the multicast group requests a dynamic address assignment from a domain *address allocation server*. A domain may have more than one address allocation server.
2. Prior to the initiator's request, the address allocation server may have pre-allocated for its domain a *range* of multicast group addresses. It does this by contacting the *MASC server* for that domain. MASC servers in different domains run a distributed algorithm that partitions multicast address ranges.

3. In turn, the MASC server would have contacted a BGMP speaking router in order to advertise, using BGP-4+ [3], this address range. In BGMP terminology, the initiator's domain is the *root domain* for the address range. Using BGP-4+, BGMP advertises reachability to the root domain.

4. Suppose now that a receiver in a different domain chooses to join this group. The BGMP speaker at the boundary of that domain forwards[5], based on the BGP-4+ reachability propagated in the previous step, a *join* towards the root domain. This has the effect of setting up a domain-level tree branch.

5. Finally, when a source sends data to the group, its border BGMP speaker forwards the data *bidirectionally*–both towards the root domain and towards the joined receiver.

Even this simplified picture brings out several protocol interactions: between applications and address allocation servers, between allocation servers, between allocation servers and MASC servers, between MASC servers, between a MASC server and a BGMP speaking router, between BGMP and unicast routing, between BGMP speaking routers, and between BGMP and the intra-domain routing protocol.

More generally, these examples suggest two levels of interactions:

Protocols Interactions at this level are between different protocols that are designed to interoperate with each other. The interaction between BGMP and the intra-domain routing protocol (in steps 5 and 6) is one example. The interaction between BGMP and MASC (in step 4) is another.

Implementations Interactions at this level are between software components implemented by different vendors. The number of possible interactions increases approximately combinatorially with the number of independent protocol implementations. Implementation interactions may also reveal protocol interaction design flaws.

These interactions are not unlike feature interactions in telecommunication networks [4].

Together with operator-specifiable configuration knobs in implementations, these interactions, if incorrectly designed or implemented, can have adverse operational impact. For example, a recent study of routing [14] in the Multicast Backbone (MBone) revealed several subtle implementation errors which resulted in network instability. In one instance, an implementation that sent batched routing updates overwhelmed other implementations with large batches. In another, a configuration option caused unicast routes to be selectively injected into the MBone, resulting in a routing loop.

How do we alleviate the impact of undesirable interactions? Several techniques are in use, to varying degrees, today:

[5]For simplicity, we do not describe how the intra-domain routing protocol ensures that BGMP hears of receiver interest.

Modeling and Simulation Particularly for protocol or feature interactions, protocol modeling [4], or test generation and simulation [11] can reveal design flaws that result in undesirable interactions.

Small-scale Testing Protocol implementors use small laboratory test networks to detect or verify implementation interoperability. Regression and stress tests using small networks can minimize undesirable interactions after deployment.

Large-scale Deployment More widespread, given today's time-to-market pressures, is the approach of actually deploying implementations in real networks and correcting undesirable interactions after occur.

None of these approaches is superior to the other. Simulation may not reveal all protocol interaction errors. Small-scale testing may not uncover all undesirable implementation interactions. For these reasons, post-deployment correction of implementation errors will continue to be prevalent. However, effective simulation and testing methodologies can reduce the expense associated with post-deployment correction of design and implementation errors.

In this proposal, we focus on small-scale testing of low-level networking protocols, and more specifically on multicast protocols. This focus is motivated by the lack of widely-available, inexpensive and flexible, software platforms for such testing. Today, some vendors usually verify their implementations by using ad-hoc test networks. These ad-hoc tests may not verify interoperability with other implementations, and may be tedious to construct. The very small number of proprietary platforms that exist today are typically inflexible, expensive or require specialized hardware (Section 5). We believe that this situation is likely to continue–that vendors, because of competitive pressures, are unlikely to invest effort in methodologies for careful *interoperability* testing. This will, we believe, adversely affect the rapid introduction of new network-layer technologies in KDI.

Based upon our experience with implementing and testing multicast protocols, we propose to design a *software platform for protocol testing.* This software platform will have three components:

- A high-level *language* for expressing a protocol test. The language will contain primitives for manipulating packet contents at a high-level, for generating packets at specified instants, and for matching received packets for specific content.

- A protocol-independent test description *interpreter*, which may allow distributing parts of the test to various *testers* in the test network.

- A *library of tests* that implementors might use to compose other tests.

3 *pkt* Design

This design for a software testing platform is motivated by our experience with *pkt*, a tool that we developed for testing and debugging our public-domain implementation of Protocol Independent Multicast–Sparse Mode (PIM-SM [7]). *pkt* allows users:

- To create fairly complex protocol tests using relatively simple network configurations.
- To rapidly prototype, and easily reconstruct, automated test sequences.

In order to achieve this functionality, *pkt* unifies two technologies: high-level scripting languages such as Tcl [18], and low-level packet capture facilities such as the Berkeley Packet Filter [19]. In *pkt*, tests are described using Tcl extensions. These extensions allow users to symbolically manipulate the contents of packet headers, and to express symbolic predicates defined on the contents of received packets. This functionality enables rapid test specification and easy test reuse. Because *pkt* uses low-level packet capture facilities, users have nearly complete control over the entire contents of the generated packets. With this feature, a single test description may be used to generate data or protocol control traffic with arbitrary contents. This allows users to *emulate* larger network using a single test device.

To illustrate this last idea, we describe how *pkt* can simplify the testing of unicast routing protocol convergence. One way to test the reaction of a routing protocol, such as RIP, to link failures is to construct the test network shown in Figure 2(a). This network contains parallel paths between the data source and destination. RIP chooses the more preferred path when both are available, but cuts over to the backup path when the more preferred path fails. To test this, we might construct the test network of Figure 2(a), ensure that RIP selects the more preferred path (say, the one through router **B**), manually disable router **B**, then verify whether traffic now flows through **C**.

The same convergence test can be realized using the test network of Figure 2(b). In this case, the *pkt* program generates the appropriate routing protocol reachability and unreachability events, and tests the availability of the preferred and backup paths. Using this approach, the test is completely *automated*. Furthermore, in our experience, many simple test descriptions can be executed with a fixed topology like that of Figure 2(b). This allows automated completion of batteries of protocol tests. What adds to *pkt*'s power and flexibility is its ability to specify this test at a fairly high-level (Figure 3).

The *pkt* programming language is designed to be unaware of the actual protocols under test. This deliberate design choice keeps *pkt* simple. However, *pkt* is *extensible* to new protocols. Users can define new packet header formats, and can easily create libraries of tests on top of *pkt*. This generality of *pkt* is evident in the number of protocol header formats that *pkt* now supports: IP, UDP, IGMP, RIP, PIM-SM, and BGMP.

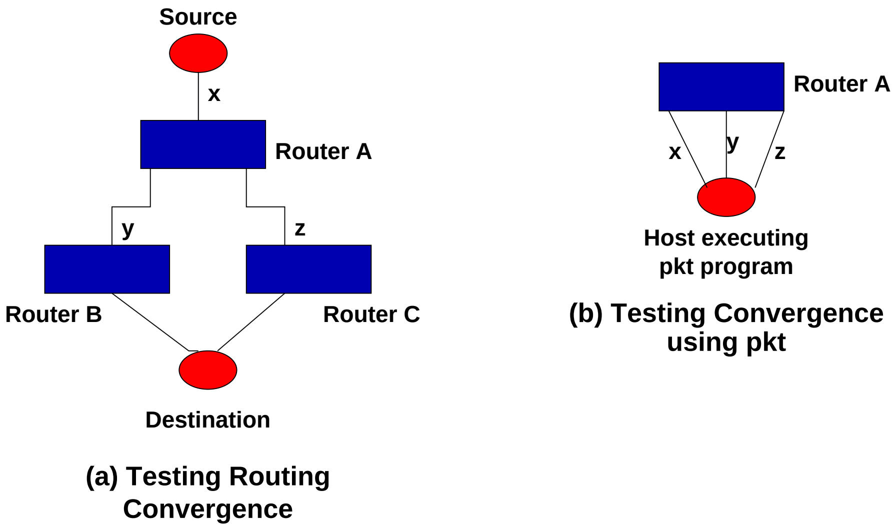

Figure 2: **Testing using *pkt*:** This figure presents a simplified description of the interactions between different components of the Internet multicast architecture. Numbers adjacent to the arrows correspond to list items in the text.

Today, a prototype implementation of *pkt* exists. Its functionality has evolved based on our use of *pkt* for debugging our PIM-SM implementation. There are several ways in which *pkt* has been, or can be, used:

Debugging development code: *pkt* provides an alternative to writing scaffolding code for modular testing of software implementations. For example, a *pkt* program can be used to generate a packet that invokes the protocol module of interest.

Regression testing: *pkt* is equally useful for scripting repeatable tests that verify the correctness of successive versions of a protocol implementation.

Interoperability testing: Because *pkt* does not require instrumenting protocol implementations, it can be used to test proprietary products for which sources are not available.

Stress Testing: Protocol implementations are notoriously susceptible to timing-related errors. *pkt* can be used to generate random event sequences in order to tickle such bugs.

Problem Reconstruction: Finally, because it is well integrated with low-level packet capture facilities, *pkt* can be used to replay entire packet traces. This feature can reproduce bugs observed in actual deployments.

```
header new ip iphdr udp udphdr rip riphdr

header set iphdr  dst r1.test.net ttl 1 src r2.test.net proto RIP
header set udphdr sport 520 dport 520
header set riphdr response dst-net metric 1

# create the first RIP advertisement
packet assemble rip1 iphdr udphdr riphdr

# create another rip packet source'd by r3
header set riphdr src r3.test.net
packet assemble rip2 iphdr udphdr riphdr

# get the devices ready for writing, and send rip
device open x y z
device send x rip1
device send y rip2

# now create a data packet
header set iphdr dst dst.test.net src src.test.net ttl 255
header set udp sport 1234 dport 5678
packet assemble p1 iphdr udp data

# send a data packet and wait to see it
device send z p1

device filter x ``dst dst.test.net and udp port 1234''
device read -timeout 2 x p2
if {[device list p2] == ""} { error ``no packet read'' }

# now we'll create the packet that advertises no more access
header set iphdr src r2.test.net dst r1.test.net ttl 1
header set riphdr response dst-net metric 255
packet assemble rip3 iphdr udphdr riphdr

device send x rip3
device filter y ``dst dst.test.net and udp port 1234''
device read -timeout 10 y p2
if {[device list p2] == ""} { error ``convergence failed'' }
pass ``this test passed''
```

Figure 3: ***pkt*** **Script for Convergence Test:** This packet script emulates the test of Figure 2(a) using a single test device.

4 Proposed Work

pkt is currently being used at several places, including Sun Microsystems, the University of Oregon, and the Gated Consortium housed at Merit Network Inc. Leveraging our experience with *pkt*, and our expertise in designing and implementing multicast protocols, we propose to design the next-generation multicast protocol testing platform, called *mint*. Two substantive features will distinguish *mint* from *pkt*:

Support for Distributed Testing *pkt* was designed assuming a single *tester*. *Mint* will support multiple observation points, enabling larger test topologies. Such topologies may be necessary, for example, to simultaneously test all the components of Figure 1.

Integration with a Network Simulator *Mint* will interoperate with the *ns* network simulator in a manner that will allow an actual implementation to be "inserted" into a simulated network. This is useful, for example, to test an implementation in the context of a larger network.

Finally, even though the overall design of *pkt* is sound, the current *pkt* implementation lacks several desirable features. *Mint* will also contain several usability enhancements over *pkt*: support for explicitly timed packet generation and reception, a general logging and reporting facility, flexible packet storage and editing features, integration with traffic generators such as `tcplib` and network animators such as `nam`, and support for a variety of packet capture facilities.

4.1 Support for Distributed Testing

The programming model for *pkt* assumes that test descriptions are executed at a single *tester*.[6] The number of physical interfaces on the tester constrain the different network configurations that can be emulated using *pkt*. While *pkt* was sufficient for testing a single protocol (PIM-SM), it may not be sufficient for testing multicast interoperability.

Figure 4 illustrates this limitation of *pkt*. Many PCs and workstations today ship with a single network interface; in most cases, this interface slot can contain up to four network ports. Figure 4 shows an example of an end-to-end multicast interoperability test, involving MASC, BGMP, and PIM-SM, that cannot be realized using a single four-port tester. There are two ways to overcome this drawback:

- One approach is to decompose a more complicated test into simpler components each of which can be accommodated by a single tester. This approach assumes that the success of each component test indicates the success of the compound behavior—that assumption is sometimes incorrect.

[6] A tester is a node that executes a software test description. In practice, testers are inexpensive pieces of equipment, like user workstations or PCs.

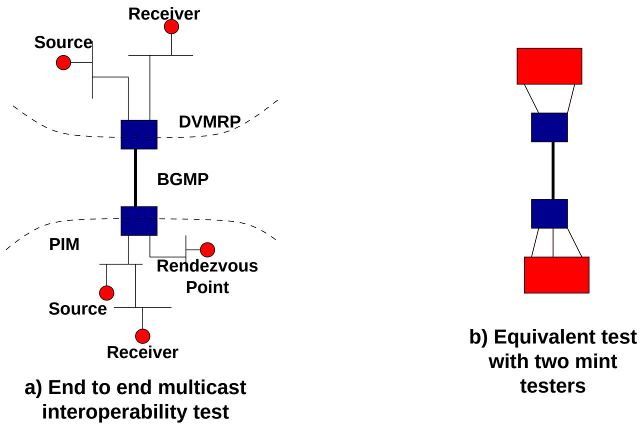

Figure 4: **End-to-end tests using *mint*:** *pkt* may be limited in its ability to conduct simple end-to-end interoperability tests. For example, to test correct inter-domain multicast packet delivery between two domains, each running different multicast protocols, we need the experimental setup shown in (a). Using *mint*, and two testers, the test network configuration becomes that shown in (b).

- Another, more correct approach, would be to explicitly develop two or more separate test descriptions that execute in different testers. While workable, this approach is a bit cumbersome.

Mint's programming model will explicitly provide support for multiple testers. Not only will this address the *pkt* limitation described above, it will also enable tests that span a geographically distributed network (*e.g.*,, across a wide-area link connecting two laboratories). To do this, *mint* will contain two important features:

- Users will be able to specify a virtual test topology and define packet generation and reception events in terms nodes in that topology.

- *Mint* will contain mechanisms that enable transparent relocation of libraries to testers, relieving the programmer of the details of realizing the distributed test.

Mint's programming model will include a *logical* test network topology description. Specifically, users will be able to assign symbolic names to network nodes and links between nodes. Then, they may qualify *mint*'s packet generation and filtering primitives (*e.g.*,the `device send` and `device filter` commands in Figure 3) with the symbolic names of nodes and interfaces. The binding between the logical test topology and the physical topology is delayed until the test is actually executed: this ensures test *portability* to different locations.

Mint will also provide support for transparent remote execution of test commands. While this may seem trivial at first blush, there are some interesting systems design challenges. In order to keep the programming model intuitive, *mint* will not require users to syntactically partition primitives executing at different testers. That is, there will be no trivial syntactic approach to separate code that executes at one tester from code that executes at another tester. How, then, does *mint* transparently execute the appropriate commands at remote testers?

Replication In this approach, *mint* initially distributes the entire test description (and all protocol libraries) to all tester nodes. Each tester node executes the script, but only processes those packet generation and predicate checking primitives that are relevant to itself. This approach is conceptually simple, but may need some complex mechanisms for downloading interpreter libraries to testers. Furthermore, the testers may need to synchronize their packet generation in order to preserve test semantics.

Master-Slave In this approach, the test description is executed at one node (the *master* tester). This tester, however, hands off packet generation and predicate testing primitives (possibly conveniently encoded in a lower-level description) to the appropriate *slave* tester. This approach promises a simpler implementation, at the expense of possibly interfering with test traffic.

We propose to investigate these issues.

4.2 Integration with a Network Simulator

Using work done by the VINT project, it has become possible to analyze network protocol performance using packet-level simulations. Today, their **ns** network simulator contains fairly complete implementations of several Internet protocols, including some multicast routing protocols. In this section, we describe software techniques that will allow *mint* to embed (and test) actual implementations inside of a simulated network.

We see two advantages to integrating a simulation environment and a protocol testing platform. First, such integration can allow testing an implementation in the context of a larger (albeit simulated) network than would be possible with *mint* alone. Second, integration can increase confidence in an implementation's compliance with protocol specifications. The assumption here, of course, is that a simulated protocol can be relatively easily verified for compliance.

Mint will support two kinds of integration with *ns*. *Off-line* integration only loosely couples *mint* and *ns*. In this approach, developers will first run a simulated protocol in a network topology of their choosing, and instrument one of the nodes to collect a trace of packets received and sent by that node. They will then replay received packets to a real implementation, and compare that implementation's generated packets against those sent by the simulated node.

This form of integration can be achieved with relative simple functionality: 1) *Ns* modules for instrumenting simulated nodes to collect packet traces, and 2) *Mint* translators that can replay

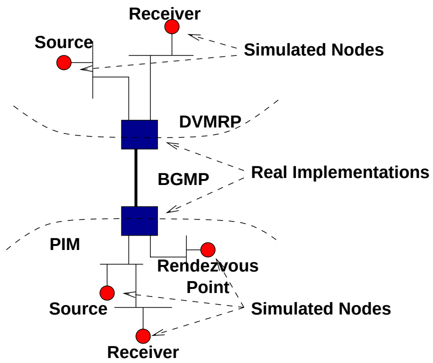

Figure 5: **On-line integration with *ns*:** *Mint* users will be able to embed their implementations in a simulated network. Such on-line integration poses significant design challenges.

collected traces and verify the implementation's generated packets. Aside from protocol compliance, it can be used for:

- Testing implementation sensitivity to event timing. By varying packet replay times, protocol implementors can hope to tickle timing-related errors.
- Stress testing implementations. Traces from nodes at the center of large simulated networks can be used as synthetic protocol traffic workloads.

Off-line integration is not without its limitations. Many protocols allow the implementor some latitude in including or excluding optional features, adjusting values of protocol constants, and adding configuration knobs. If the simulator does not capture these implementation-specific features, then, an off-line *mint* test on a real implementation may incorrectly fail. For example, in a test of unicast routing, a simulator may not be able to predict which of two equal-cost paths an implementation may choose. In that case, the simulated node may make a different choice than an actual implementation. When the trace is replayed to the implementation, the test might appear to fail.

We believe off-line integration is but a step towards the goal of more seamlessly integrating *mint* with *ns*. This *on-line* integrated test platform would allow any *mint* test description to be realized on a topology that contains a mixture of simulated and actual nodes. For example, the test topology of Figure 4(a) can be realized by simulating some nodes as shown in Figure 5.

On-line integration poses these implementation design issues:

- The preservation of temporal information between simulated and real nodes. Specifically, one or more simulated nodes may run ahead of real-time; a packet generated by an implementation may be injected into a simulated node in its past. Conceptually simple, but slightly tricky to implement, is the notion of forcing the simulator clock to adhere (as closely as possible) to real-time. Techniques used in distributed simulation (such as Time Warp [12]) may be applicable here.

- *mint* support for transparently injecting real-world packets into the simulator, and vice-versa. This software will be based on the low-level packet capture facilities already integrated with *mint*, and will perform the appropriate header translations before inserting the packet into *ns*'s event queue.
- Higher-level *mint* support for dynamically instructing *ns* nodes to generate packets or tests for network events. Using this, *mint* can parse descriptions of distributed tests, and hand off tests to the appropriate simulated node. To do this, we will need to modify *ns*'s core.[7]
- Auxiliary mechanisms to automatically generate *ns* configurations (topology descriptions, and protocol commands etc.) from a description of the distributed test. In this way, users of *mint* need not explicitly configure *ns* in order to execute a test.

5 Related Work

To our knowledge, there does not exist a software testing platform that contains the functionality we expect from *mint*. *Mint* is most closely related to two other commercial network emulators: Packet Shell [15] from Sun and ANVL [17] from Midnight Networks. Packet Shell is also Tcl-based and supports symbolic manipulation of packet contents. Both products are integrated with low-level packet capture facilities. However, they both lack support for *mint*'s distributed testing and simulator integration.

6 Plan of Work

Section 4 identified three separable *mint* tasks: usability enhancements over *pkt*, support for distributed testing, and integration with *ns*. We expect to structure our plan of work as three parallel threads of development that span a two-year time frame. Our plan of work is best described by what we expect to accomplish at the end of each half-year interval of the project:

- By the end of the first six months, we will have completed the implementation of the usability enhancements to *mint*, and completed the design and evaluation of the distributed testing support.

[7]Basic networking constructs, such as nodes and links, are implemented in *ns*'s C++ core. Many other protocols are implemented in Tcl. A stub library translates Tcl calls to C++ where necessary.

- By the end of the first year, we will have a working prototype of distributed testing. We will also have begun preliminary investigations into the design of on-line integration with *ns*.
- At the eighteen month mark, we expect to have a working prototype of off-line integration and a documented solution to the timing issues in the design of on-line integration.
- Finally, by the end of the second year, we expect to demonstrate the execution of *mint* test descriptions on a network containing some simulated nodes.

7 Expected Impact

Mint will provide a much-needed software testing platform for network-layer protocols. Since *mint* provides tools that can rapidly increase software developers' confidence in the correctness of their protocol implementations, we believe it will significantly reduce the time-to-market of vital network technologies such as multicast.

In turn, this can have far reaching extrinsic impact. Multicast will form the basis for large-scale, wide area, research collaboration applications. The testing methodologies that *mint* introduces will, we expect, be even more relevant in the design of emerging new network technologies such as networked *smart* spaces, and sensor networks.

Results From Prior NSF Support

Ramesh Govindan has been supported by NSF Cooperative Agreement NCR-93-21043 since 1994. The PIs on this NSF Routing Arbiter project are Deborah Estrin and Jon Postel. Prof. Govindan currently leads the Routing Arbiter effort at USC's Information Sciences Institute.

The overall goals of the Routing Arbiter project are to provide advanced Internet routing services, and enable an Internet-wide routing policy analysis framework. Of particular relevance to this proposal, the Routing Arbiter project has contributed significantly to the development of Internet multicast protocols such as PIM and BGMP/MASC.

References

[1] E. Amir, S. McCanne, and R. H. Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM 1998*, Vancouver, British Columbia, September 1998.

[2] A. Ballardie, P. Francis, and J. Crowcroft. Core-based trees. In *Proceedings of ACM SIGCOMM 1993*, San Francisco, CA, September 1993.

[3] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol Extensions to BGP4. Internet-Draft, `draft-ietf-idr-bgp4-multiprotocol-v2-01.txt` , August 1998.

[4] K. H. Braithwaite and J. M. Atlee. Towards Automated Detection of Feature Interactions. In *Proceedings of the Second International Workshop on Feature Interactions in Telecommunications Software Systems*, pages 36–59, May 1994.

[5] S. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of 1988 ACM SIGCOMM Conference on Communication Architectures and Protocols*, pages 55–64, August 1988.

[6] S. Deering. Host Extensions for IP Multicasting. Request for Comments 1112, Internic Directory Services, August 1989.

[7] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE Transactions on Networking*, 4(2), April 1996.

[8] S. Floyd, V. Jacobson, S. McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proceedings of the ACM SIGCOMM '95*, Aug 1995.

[9] National Science Foundation. Knowledge and Distributed Intelligence. Program Solicitation 98-55, May 1998.

[10] R. Govindan, H. Yu, and D. Estrin. Large-Scale Weakly Consistent Replication using Multicast. Technical Report 98-682, Computer Science Department, University of Southern California, 1998.

[11] A. Helmy and D. Estrin. Simulation-based STRESS Testing Case Study: A Multicast Routing Protocol. In *Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, Montreal, Canada, July 1998.

[12] David Jefferson, Brian Beckman, Fred Wieland, Leo Blume, Mike DiLorento, Phil Hontalas, Pierre Laroche, Kathy Sturdevant, Jack Tupman, Van Warren, John Wedel, Herb Younger, and cs555-sp1998 Steve Bellenot. Distributed simulation and the Time Warp operating system. In *Eleventh*, pages 77–93, November 1987.

[13] K. Kumar, P. Radoslavov, C. Alaettinoglu, D. Estrin, M. Handley, and D. Thaler. The MASC/BGMP Architecture for Wide-Area Multicast Routing. In *Proceedings of ACM SIGCOMM 1998*, Vancouver, British Columbia, September 1998.

[14] D. Massey and B. Fenner. Multicast Routing Summary Report. `http://ganef.cs.ucla.edu/~masseyd/Route/`, 1998.

[15] Sun Microsystems. The packet shell protocol testing tool. http://playground.sun.com/psh.

[16] J. Moy. Multicast Extensions for OSPF. *Communications of the ACM*, 37:61–66, 1994.

[17] Midnight Networks. Automated network validation library. http://www.midnight.com/ANVL.

[18] J. K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, April 1994.

[19] V. Jacobson S. McCanne. The bsd packet filter. a new architecture for user-level packet capture. In *Proceedings of the 1993 Winter USENIX Conference*, San Diego, CA, January 1993. USENIX.

[20] J. E. Smith and F. W. Weingarten. Research Challenges for the Next-Generation Internet. Workshop on Research Directions for the Next-Generation Internet, May 1997.

BIOGRAPHICAL SKETCHES

Ramesh Govindan
USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina Del Rey, CA 90292.
Phone: +1-310-822-1511 (x103)
Email: `govindan@isi.edu`

Education

- Ph.D in Computer Science from the University of California at Berkeley, 1992.
- M.S. in Computer Science from the University of California at Berkeley, 1989.
- B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, 1987.

Experience

- Research Assistant Professor, University of Southern California, and Scientist, USC/Information Sciences Institute, 1994-present.
- Member of Technical Staff at Bell Communications Research, Morristown, NJ, 1992-1994.

Related Publications

Prof. Govindan has several publications in operating systems and internetworking.

- *An Analysis of Internet Inter-Domain Topology and Route Stability*, Ramesh Govindan and Anoop Reddy, Proceedings of IEEE Infocom 1997, Kobe, Japan.
- *Time/Space Trade-offs in Route Server Implementation*, Ramesh Govindan, Internetworking: Research and Experience, Feb 1995.
- *Flexible Routing and Addressing in a Next-Generation Internet Protocol*, Paul Francis and Ramesh Govindan, Proceedings of ACM SIGCOMM 1994, London, England.
- *Scheduling and IPC Mechanisms for Continuous Media*, Ramesh Govindan and David P. Anderson, Proceedings of the 13th ACM Symposium on Operating Systems Principles, Asilomar CA, 1993.

Other Publications

Prof. Govindan has also designed multimedia software systems.

- *A File System for Continuous Media*, David P. Anderson, Yoshitomo Osawa, and Ramesh Govindan, ACM Transactions on Computer Systems, vol 10, no 4, November 1993.
- *Abstractions for Continuous Media in a Network Window System*, David P. Anderson, Ramesh Govindan, and George Homsy, First International Conference on Multimedia Information Systems, Singapore, January 1991.

George Eddy
USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina Del Rey, CA 90292.
Phone: +1-310-822-1511
Email: eddy@isi.edu

George Eddy has worked in the area of computer systems and networks for the last ten years. His work has involved distributed application development, real-time embedded systems, network protocol development and testing, and network management. George received a BSCS from the California State University, Long Beach and a MSCS from the University of Southern California, where his focus was computer networks and multicast routing.

Over the last two years George has developed the Protocol Independent Multicast (PIM) intra-domain multicast routing protocol, and is in the process of developing the inter-domain multicast routing protocol, the Border Gateway Multicast Protocol (BGMP). During this period George has obtained significant practical experience with testing multicast routing protocols.