Network Visualization with Nam, the VINT Network Animator

Visualization tools such as nam, a network animator that supports packet-level animation and provides scenario-editing capabilities, make protocol design and debugging easier.

Deborah Estrin
Mark Handley
John Heidemann
Steven McCanne
Ya Xu
Haobo Yu
The VINT Project

Network protocol designers face many difficult tasks, including simultaneously monitoring state in a potentially large number of nodes, understanding and analyzing complex message exchanges, and characterizing dynamic interactions with competing traffic. Traditionally they've used packet traces to accomplish these tasks, but traces have two major drawbacks: They present an incredible amount of detail, which challenges the designer's ability to comprehend the data, and they are static, which hides an important dimension of protocol behavior. As a result, detailed analysis frequently becomes tedious and error-prone. Although network simulators such as the VINT project's ns[1] can easily generate numerous detailed traces, they provide limited help for analyzing and understanding the data.

Nam, the network animator that we developed in our work at the VINT project, provides packet-level animation, protocol graphs, traditional time-event plots of protocol actions, and scenario editing capabilities. Nam benefits from a close relationship with ns, which can collect detailed protocol information from a simulation. With some preprocessing, nam can visualize data taken directly from real network traces.

NETWORK PROTOCOL VISUALIZATION

Researchers have explored network protocol visualization in many contexts, beginning with static protocol graphs and visualization of large-scale traffic, and more recently including simulation visualizations and editors. Network visualization tools allow designers to take in large amounts of information quickly, visually identify patterns in communication, and better understand causality and interaction.

Graphs of packet exchanges are useful for understanding cause and effect in complex protocols like TCP. Work at MIT[2] and the University of Arizona[3] is typical: Graphs show time against TCP sequence numbers on a 2D graph, sometimes with annotations to show special events. Similar time-event graphs have proven useful in understanding reliable multicast behavior in SRM.[4]

Several groups have looked at visualization of large, static network data sets. Important questions include choice of layouts based on real-world geography or network topology and how best to use animation, color, and 3D. More generally, many researchers have tackled the problem of visualization of complex data.[5] Systems like these share the principle that multiple linked views are essential for visualizing complex data.

Several network simulation systems include explicit support for visualization, either customized to a particular end application or to something more general. Opnet includes visualization capabilities, and Simphony[6] explicitly includes packet-level animation. Systems such as Opnet and Parsec[7] have provided this capability for some time. CMU's ad-hockey was designed explicitly to support node movement.[8] GUI network editors are of most benefit to novice users or users running small simulations.

NAM BASICS

To animate network traffic in several ways, nam interprets a trace file containing time-indexed network events, as Figure 1 shows. Typically, an ns simulation generates this trace, but nam can also use processing data from a live network to produce a trace. Nam usually runs offline using traces stored on disk, but it can also play traces from a running program through a Unix pipe.

0018-9162/00/$10.00 © 2000 IEEE

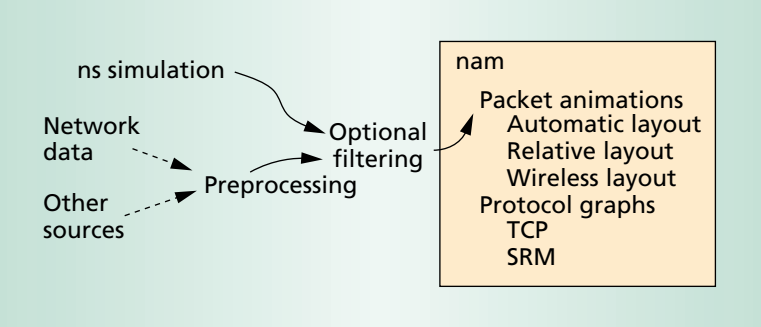

Figure 1. Data flows into nam from network data or other sources after preprocessing into the nam trace format. Optional filtering adds information to a nam trace, possibly highlighting specific flows or generating additional statistics.

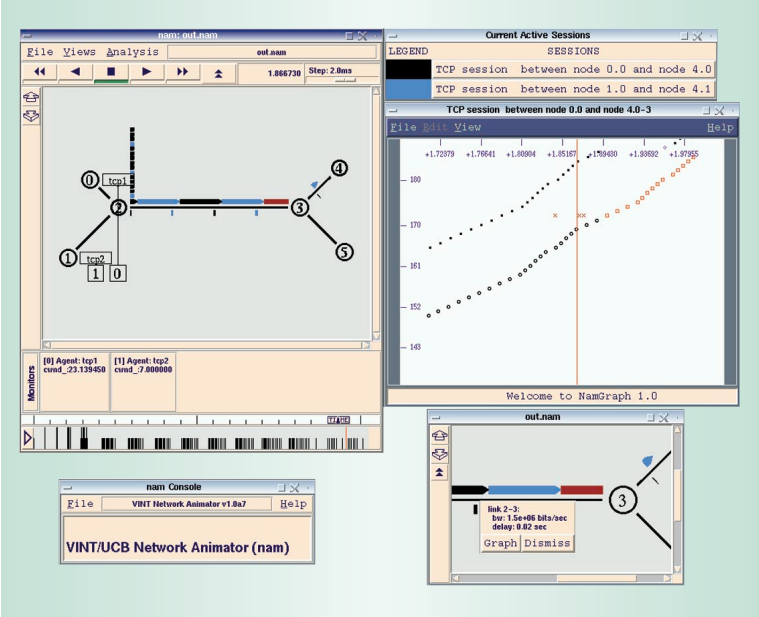

Figure 2. A typical nam session. The top-left window shows a packet animation for a network, with monitors and statistics running across the bottom. The top-right window shows TCP flows in the trace. The center-right window shows a time-sequence plot of a specific TCP flow. The bottom-right window shows a zoomed view of the packet animation and a pop-up query concerning link 2-3. The bottom-left window is the console for nam-wide operations.

The nam trace file contains all information needed for the animation—both on static network layout and on dynamic events such as packet arrivals, departures, and drops and link failures. The input file for wireless networking simulations also includes information on node location and movement.

Figure 2 shows a typical nam session. The main window, at the top left, shows packet animations, with nodes, network links, and packets flowing between nodes or queued, waiting to be sent. For example, link 2-3 shows TCP data moving along the top and return acknowledgment traffic moving along the bottom in the reverse direction. Packet color can show many things; here blue and black packets represent flows from different sources, and red packets are those carrying a congestion signal. Packets move from node to node along links and queue up when links are full. For example, a large queue near node 2 reflects the busy link between nodes 2 and 3.

Under this network are several statistical summaries. Monitor boxes show the parameters of protocols running on particular nodes. The ticks beneath them show link utilization as a function of time. The window at the bottom right of the figure zooms in on part of the network.

The center-right window shows a protocol-specific time-event graph of a particular flow on a given link. In this case, the graph plots TCP sequence numbers against time using different symbols to show data packets, acknowledgments, and acknowledgments that include explicit congestion information.

Designers can execute multiple copies of nam, even driving them in lockstep. The ability to visualize the output of more than one simulation trace file makes side-by-side comparisons possible. These are especially useful for investigating protocol sensitivity to input parameters in the same simulation scenario.[4]

Packet animation

Nam's core is packet animation. Figure 3 shows three variants of the TCP protocol nam uses to send data from Web servers on the right to clients on the left. Animation allows the user to quickly see the status of each part of the network—the top link is severely congested and dropping packets; the middle link is slightly busier than the bottom link—and quickly compare algorithms—the middle variation has one extra magenta packet, while the top version sends many back-to-back packets. Nam lets users adjust the animation speed and play it forward or backward, making it easy to find and examine interesting occurrences.

The first step in a new animation is displaying the network topology. Nam has three different topology layout mechanisms to accommodate different needs:

- *Automatic.* The default is an automatic layout algorithm based on a spring-embedded model.[9] The algorithm assigns attractive forces on all links and repulsive forces between all nodes and tries to achieve balance through iteration. Automatic layout can produce reasonable layouts of many networks without explicit user guidance, but it may not produce satisfactory results for complicated networks. If necessary, users can manually adjust the resulting layout. Figure 4 shows an automatic layout.
- *Relative.* For smaller topologies, relative layout is

possible. The user specifies the relative directions of links—left, up, down—and nam uses the directions to place nodes relative to each other. Nam sets link length proportional to bandwidth and delay. Relative layout works well for small topologies and has the advantage that packet movement rate is constant and consistent with link delay and bandwidth. Relative layout's disadvantages are that the user must specify the directions of each link, not all networks have a planar representation that satisfies delay constraints, and relative layout of a topology containing different delays can result in very short links. For example, the 10-Mbps, 1-ms delay links on the left of Figure 3 are too short for observing packet flow when shown at the same scale as the 800-Kbps, 100-ms central link.

- *Wireless.* Wireless layout associates each node with a physical location in a constrained area. Each node's 3D coordinate gives its position in the area (though visualization currently uses only two dimensions) and its velocity vector. Wireless visualizations typically lack explicit links.

Once nam lays out the topology, packet animation is straightforward. Trace events indicate when packets enter and leave links and queues. As the main window of Figure 2 shows, packets appear as rectangles with arrows at the front, and queues as arrays of squares. To identify source and destination pairs, users can select packet colors based on codes set in the simulator or set during preprocessing. When queues fill, packets literally drop, falling to the bottom of the display as small rolling squares.

One difficulty in implementing packet animation was that some events that animation requires are not present in the trace file. We wanted the trace file to be as explicit as possible, but some trace events are animation-specific and must be dynamically constructed. One example is identifying when a dropped packet leaves the screen, which the simulator would not know.

To focus on interesting parts of the simulation, users can control the animation playback rate. VCR-like buttons allow forward and backward playback, and a slider sets the playback rate. Users can skip simulation dead time or annotate interesting trace events so they can jump to them. The animation window is interactive, letting the user click on packets, links, and nodes to bring up pertinent information, including statistics.

Other visualization methods

In addition to packet animation, we experimented with other ways to visualize information. Users can specify node color and shape to indicate state such as membership in a multicast group, use protocol agents to represent the state of a protocol instance at an end node, and display agents as small labeled rectangles attached to nodes.

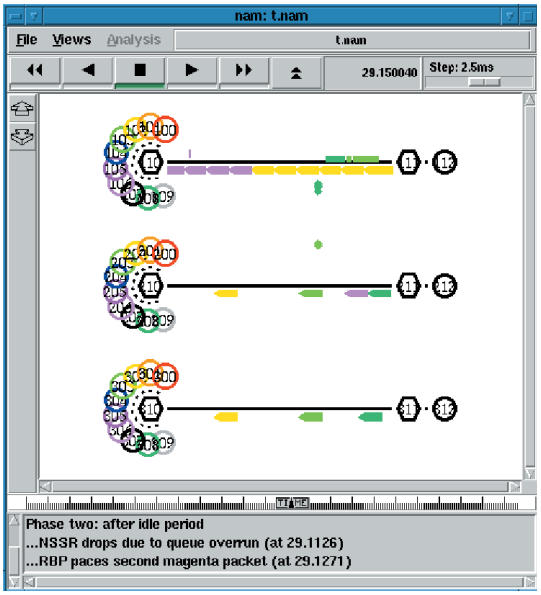

Figure 3. Using packet animation in nam. Here nam is comparing three different TCP protocols with the same workload. The different numbers of packets in flight and the packet loss in the top variant represent variations in the three protocols.

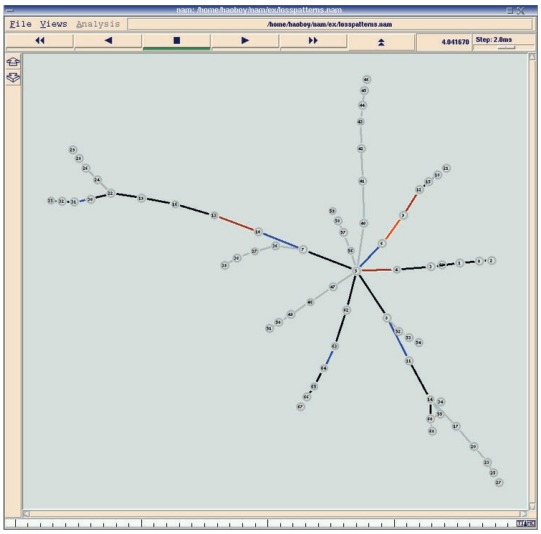

Figure 4. Link animation in a visualization of Internet multicast-backbone loss rates.

Figure 4 shows one example of non-packet-level animation—the topology of a portion of the Internet multicast backbone as of 1998. To determine if Mbone loss was primarily in the core network or the edges, we measured loss rates for various links. The

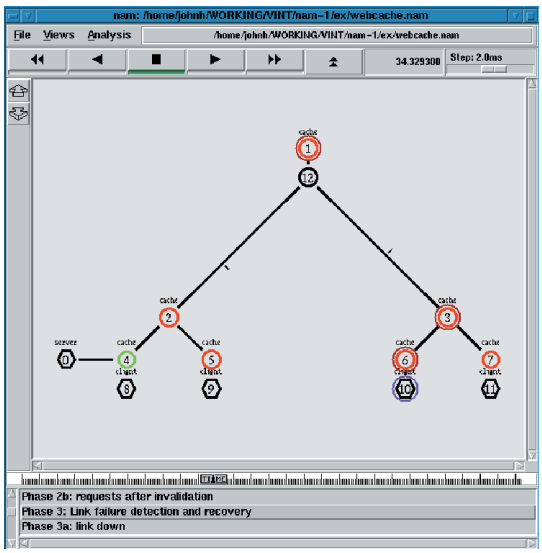

Figure 5. Visualization of applications with nam. The shapes represent node types—the hexagons are the client and server and the circles are caches. The node color shows the cache status; rings around nodes show algorithm status.

figure shows different loss rates in colors that change over time.

We have also found nam useful for application-level visualization. In Figure 5 we use nam to visualize cache-coherence algorithms in a hierarchical Web cache. Shapes show node types—the client and server are hexagons, and caches are circles. Node color shows cache status—valid or out of date. Rings around nodes show algorithm status—for example, refreshing a cache.

Simulation details

Nam's animation component displays only a subset of the simulation details present in the trace output. Other nam components handle additional information, such as packet headers or protocol state variables. The statistics component provides three ways to display this additional information:

- Clicking on any of the displayed objects—for example, packets and protocol agents—brings up a pop-up panel showing object-specific information.
- Clicking on a link brings up a selection panel, a special type of pop-up that allows the user to open a new pane (the black stripes in Figure 1) to display bandwidth utilization or packet loss.
- Pop-up monitors display object-specific information. Monitors remain associated with an object until the user explicitly removes them or until the trace indicates that the object is gone. These monitors are displayed at the bottom of nam's main window, as illustrated in Figure 1.

Protocol-specific graphs

Nam can also represent protocol-specific information with time-event graphs, which plot against events such as an advancing sequence number or message transmission. These graphs have long been used to understand TCP behavior and more recently to understand timer interaction in scalable reliable multicasts.[4]

Currently, nam supports protocol graphs for TCP and SRM, but we plan to add a pluggable API to support other, more-generic protocols. Figure 6 shows time-event graphs for SRM (top right) and TCP (bottom center and bottom right). When a graph first comes up, a nam filter scans the trace file to extract the relevant information for a specific flow or protocol.

The advantage of integrating these views with nam is that it synchronizes graphs and packet animations. Moving a time slider or clicking on an interesting event in any view will update the time in all views. To help users coordinate events, nam displays each trace event in a consistent way—through color or shape—across views.

Scenario creation and editing

We use nam in two complementary ways to assist scenario creation:

- With the scenario input facility that we recently added to nam, users can apply a traditional drawing approach to add nodes, links, and protocol agents. Nam then saves this scenario as an ns simulation script (in Tcl), which the simulator will process.
- The ns scenario generator uses nam to visualize large scenario topologies, using tools such as Georgia Tech's ITM[10] to construct these scenarios. Nam uses automatic layout to present the topology to the user for acceptance or regeneration.

Nam's ability to create graphical scenarios is appropriate for small scenarios with a few nodes and links. We have been pleased with the results of using nam to produce scripts for these cases while starting with scripts directly for larger, more complex, or automated simulations. For our target audience of protocol designers, learning Tcl syntax requires minimal effort, and it is more than offset by the finer control and ability to use looping constructs in place of repeated manual point-and-click operations.

Nam development is ongoing, with a number of incremental improvements under consideration or planned. For example, we plan to improve scenario-editing capabilities and add support for entering mobile node tracks.

Two major focuses of future work remain:

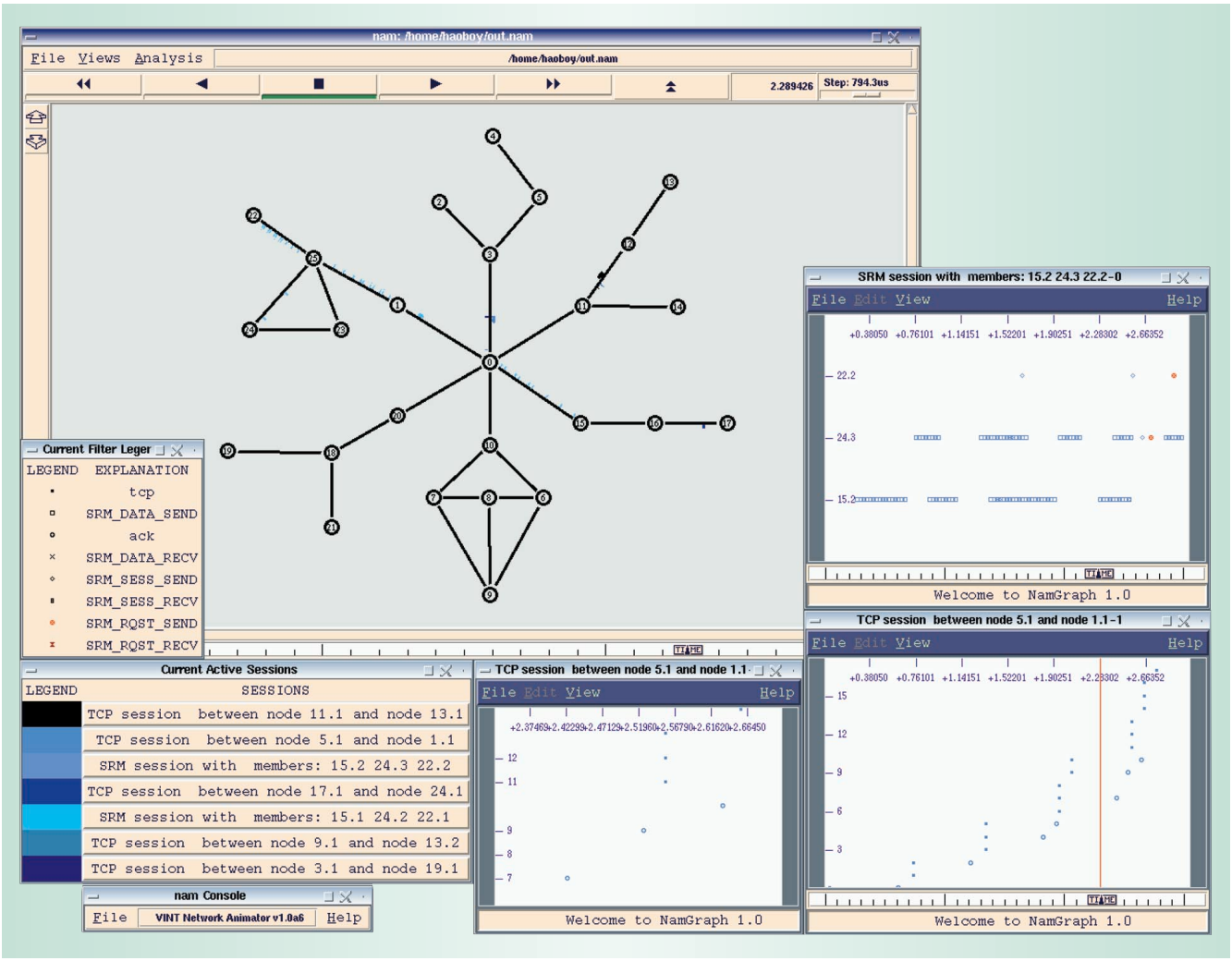

Figure 6. Nam provides time-event graphs, which aid specific investigations. The bottom center and bottom right windows show TCP time-sequence number graphs. The top right window shows a plot of SRM events against time.

- We would like to make nam easier to extend, providing better internal APIs to allow users to add custom controls to the output and to control object rendering. For example, one application would allow users to interactively control node colors to indicate application groups or characteristics.
- We are just beginning to understand how to visualize large-scale networks (more than 100 nodes), and we intend to devote more work to this area.

Network protocol visualization is easy to dismiss because its contributions to protocol development are indirect. More than a tool for fancy demos, visualization through nam can substantially ease protocol debugging and help developers understand dynamic behavior. For these reasons, a growing number of researchers are using nam in their work. ✷

Acknowledgments

This research is supported by the US Defense Advanced Research Projects Agency through the VINT project at Lawrence Berkeley Laboratory under DARPA order E243, at the University of Southern California's Information Sciences Institute under DARPA grant ABT63-96-C-0054, and at Xerox PARC under DARPA grant DABT63-96-C-0105. Steve McCanne wrote the original version of nam in 1990 at Lawrence Berkeley National Laboratory. Marylou Orayani made substantial contributions to nam as part of her work at Berkeley in 1995 and 1996. Since 1997, the VINT research project has maintained and enhanced nam. We especially thank Elan Amir, Lee Breslau, Kevin Fall, Sally Floyd, Ahmed Helmy, Polly Huang, Scott Shenker, and Christos Papadopoulos for their input into nam and this article.

References

1. L. Breslau et al., "Advances in Network Simulation," *Computer*, May 2000, pp. 59-67.
2. T.J. Shepard, *TCP Packet Trace Analysis,* Tech. Report 494, Laboratory of Computer Science, Massachusetts Inst. of Technology, Cambridge, Mass., 1991.
3. L.S. Brakmo, S.W. O'Malley, and L.L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *Proc. ACM SIGCOMM 93,* ACM Press, New York, pp. 24-35.
4. S. Floyd et al., "A Reliable Multicast Framework for Lightweight Sessions and Application-Level Framing," *ACM/IEEE Trans. Networking,* Dec. 1997, pp. 784-803.
5. G.G. Robertson, S.K. Card, and J.D. MacKinlay, "Information Visualization Using 3D Interactive Animation," *Comm. ACM*, Apr. 1993, pp. 56-71.
6. X.W. Huang, R. Sharma, and S. Keshav, "The Entrapid Protocol Development Environment," *Proc. IEEE Infocom,* 1999, pp. 1107-1115.
7. R. Bagrodia et al., "Parsec: A Parallel Simulation Environment for Complex Systems," *Computer*, Oct. 1998, pp. 77-85.
8. The CMU Monarch Project, "The CMU Monarch Project's ad-hockey Visualization Tool for ns Scenario and Trace Files," Carnegie Mellon Univ., Pittsburgh, Pa., 1998.
9. T. Fruchterman and E. Reingold, "Graph Drawing by Force-Directed Placement," *Software Practice and Experience*, Nov. 1991, pp. 1129-1164.
10. K. Calvert, M. Doar, and E.W. Zegura, "Modeling Internet Topology," *IEEE Comm. Magazine*, June 1997, pp. 160-163.

Deborah Estrin *is a professor of computer science at the University of California, Los Angeles. Her networking research has included multicast protocols and network simulation and visualization and currently focuses on networking small devices and sensor networking. She has a PhD in computer science from MIT. Contact her at destrin@cs.ucla.edu.*

Mark Handley *is a computer scientist at the AT&T Center for Internet Research at ICSI (ACIRI), where his research examines congestion control and reliable multicast. He has a PhD in computer science from University College, London. Contact him at mjh@aciri.org.*

John Heidemann *is a computer scientist at the Information Sciences Institute, University of Southern California. His research interests include network protocols, simulation and traffic modeling, and embedded networking. He has a PhD in computer science from UCLA. Contact him at johnh@isi.edu.*

Steven McCanne *is CTO of FastForward Networks. His research interests include reliable multicast, Mbone multimedia tools, and layered video coding. He has a PhD in computer science from the University of California, Berkeley. Contact him at mccanne@ffnet.com.*

Ya Xu *is a member of the technical staff at Cisco Systems. His research interests include networking protocols and network simulation. He has an MS in computer science from the University of Southern California, where he is a PhD candidate. Contact him at yaxu@isi.edu.*

Haobo Yu *is a computer scientist at Packet Design. His research interests include Web cache consistency protocols and application-level routing. He has a PhD in computer science from the University of Southern California. Contact him at haoboy@packetdesign.com.*