The Clack Graphical Router: Visualizing Network Software

Dan Wendlandt*
Carnegie Mellon University

Martin Casado†
Stanford University

Paul Tarjan‡
Stanford University

Nick McKeown§
Stanford University

Abstract

We present Clack, a graphical environment for teaching students how Internet routers work and other core networking concepts. Clack is a router written as a Java Applet, and routes live network traffic in real-time. Students can look inside the router to see how packets are processed, and watch the dynamics of the queues. They can modify and enhance the router, making it process packets as they wish. Clack provides multiple views of the operational router including the full network topology, the router's software components, and a packet-level view of the traffic as it passes through the router. Clack's detailed visual interface to the software internals of a functioning router, as well as its ability to modify and observe live Internet traffic, provide a unqiue environment to aid in networking education.

Over the last two years, Clack has been used in the classroom at six universities. Feedback from the students through anonymous, formal evaluations has been positive. In this paper we describe the goals and design of Clack as well as our experiences using it in the classroom.

CR Categories: K.3.2 [Computers and Education]: Computer Information Science Education—;

Keywords: software visualization,education,networking,router design

1 Introduction

Teachers often use animations and simulators to explain networking concepts, such as packet-switching, routing, and congestion-control. These are usually simple toys, designed to illustrate a single concept and, when done well, they can be powerful visualization tools. Our goal is a little different: We want to enable students to peak inside a working Internet router while it is processing live Internet traffic in real-time. This allows students to understand the steps involved in routing (e.g. looking up addresses, exchanging routing tables, choosing an output port, and discarding errored packets) and understand how real networking equipment works. We'd like them to see how buffers grow with congestion, how TCP controls congestion, why packets are dropped, and how exceptions are processed. We want students to see inside routers because routers determine much of the Internet's behavior. It's inside routers where packets are processed, queueing delay happens, routes are decided, packets are dropped, and access control is implemented. By seeing what happens inside the router, students gain an appreciation of how the network works as a whole.

Normally, however, routers are opaque. Very little is instrumented inside routers (for example, routers typically don't report the occupancy of their buffers), and because router software is proprietary, it is often hard to understand what is happening.

In recent years, learning environments have been created to study networking traffic [Zhao and Mayo 2002; Kurose and Ross

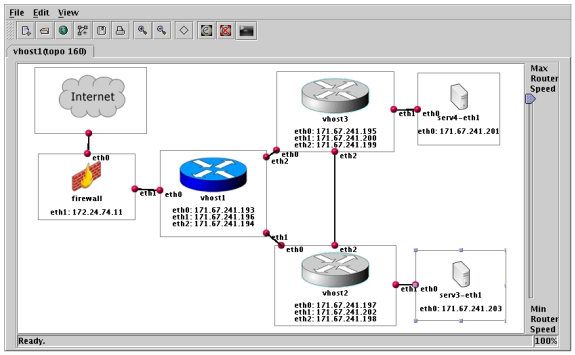

Figure 1: Network level view of three router topology in Clack.

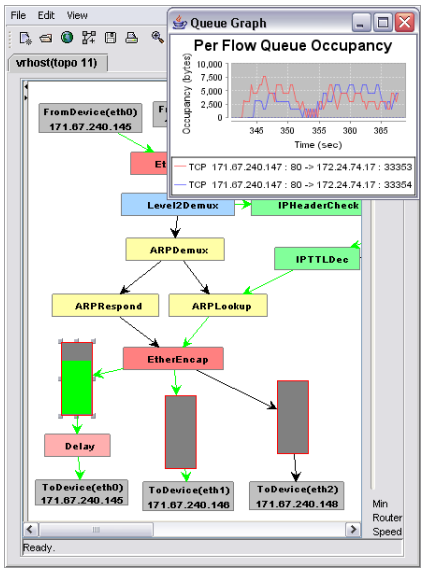

Figure 2: Component view of a Clack router with real-time buffer occupancy graph.

a; Michael J. Jipping and Porter 2003], where students interact directly with simulated, captured, or live Internet traffic. These tools strive to tie common Internet functions familiar to students, such as web requests, with network concepts, such as TCP behavior during packet-loss. But most classroom tools provide only a partial view of how networking software works. Particularly, they are usually limited to network processing at end-hosts. Hence, students are given little or no insight into how software in Internet routers is organized and configured or how routers process and affect different classes of traffic.

Furthermore, tools that visualize packets and protocol behavior are generally passive and do not give users control over the behavior of the router so they can control network behavior and visualize the consequences.

In this paper, we present the Clack graphical router, a visu-

*e-mail: dwendlan@cs.cmu.edu
†e-mail:casado@cs.stanford.edu
‡e-mail:ptarjan@cs.stanford.edu
§e-mail:nickm@stanford.edu

alization tool for demonstrating network concepts in the classroom.

The goal of Clack is to provide a graphical, web-accessible interface for students to explore the software structure of a functional router and to access a rich set of protocol visualization tools.

Clack is a Internet router (written in Java) that runs in user-space, and processes real traffic just like a commercial Internet router. Clack is a component of the Virtual Network System (VNS) developed at Stanford University for classroom use, and now used by more than 2,000 students nationwide. VNS builds network topologies, consisting of Ethernet links and nodes; a node can be a router, a switch or anything that processes packets and has multiple interfaces. VNS manages many topologies at once. Typically, each student has his/her own private topology. The nodes in the topology are user-level processes (e.g. a router written in C, C++, or Java); Clack is an example of a VNS router written in Java.

One important feature of VNS is that every topology is connected to the public Internet, and routes real traffic. Each node has real IP addresses, and processes real Ethernet packets. Another important feature of VNS is that such a user-level process can run on the VNS server that manages the topology, or it can run remotely on any machine in the Internet and "bind" to a node in VNS. This creates some interesting possibilities: A user-space Clack router running on a machine at MIT can route real Internet traffic passing through a VNS topology at Stanford. Students can create topologies with hundreds of routers, where each router is a different user-space router running elsewhere. Using our VNS server at Stanford University, we can support thousands of students around the globe on independent network topologies.

A demo of Clack is available at [Clack]. If you visit the Clack website and load the Clack applet, you will be running a router on your computer that is routing traffic in our lab at Stanford. If you click on your router, you can look inside it and see how it processes packets, how packets are queued, and when packets are dropped.

Clack was inspired by Click [Morris et al. 1999], a high-speed modular router written in C++ for both research and production use. Clack, because it is written in Java, runs on any platform without recompilation, and is easily extensible by undergraduate students with relatively little programming experience. A Clack router consists of a set of packet-processing blocks that can be (graphically) connected, added, removed, queried, and modified while the router is running. Students can add and delete blocks and configure parameters, and then use inspection tools to see the effect on network traffic. For example, they can add a block that mis-sequences packets, to see the effect on congestion control algorithms.

In this paper we describe Clack's three main characteristics:

1. The student can *visualize* the inside of a router. Clack shows each block, how it performs, and how blocks interact.

2. The student can *modify* the function of the router by adding new blocks, or modifying existing ones. For example, a student could modify Clack to be a firewall or NAT device, or decrease the size of the router's packet buffers.

3. The student can *affect traffic* by dropping, altering or reordering it and see the impact of on protocol behavior in real-time. This is in contrast to popular, passive environments for showing network traffic. The ability to affect traffic is useful to understand protocol behavior during variations in the network, such as queuing delay or loss.

Clack was designed to be simple to use and remotely accessible. It does not require students to have administrator access nor install complex software packages.

In this paper, we describe Clack and our experiences with it in the classroom. We first explain the design of Clack in Section 2 before describing its three levels of visualization in Section 3. Sections 4 and 5 describe features that enable instructors to easily create visualizations of many different types of network behavior. In Section 6 we describe and discuss our experiences with Clack in the classroom. Finally, we present previous work in the area of visualizations for networking education and briefly outline our plans for future development.

2 Clack Design

2.1 Overview

When Clack is first loaded, it displays the virtual network topology that it is connected to (see figure 1). The network topology may include one or more routers, application servers (such as web and ftp servers) and a connection to the Internet via a firewall. The network topology is being emulated by a virtual networking environment hosted remotely and described in section 2.4.

If the user sends network traffic (such as a web request) to the IP addresses assigned to the topology, the traffic will be routed over the Internet, to our lab at Stanford, into a virtual topology consisting of routers and network servers. A user may, for example, download a web page from a web server on the topology and have the traffic pass through their router.

The heart of Clack is the router view (section 2.3). From the network view, the user can click on a router to "zoom in" and see its internal state and software composition (figures 2, 3, and 4). The router contains components connected as a directed graph through which packets flow. Each component performs some portion of the packet processing functionality, such as making forwarding decisions, buffering packets, or implementing a transport protocol like TCP.

The router components provide component-specific views that describe the components and show its state and configuration. Some components provide the ability to view properties of packets or network flows passing through the router in real time. For example, it is possible to view buffer queue occupancey per flow for multiple protocols (figure 10) or how a small buffer affects the TCP congestion window (figure 7).

In the next two sections we discuss the goals behind the design of Clack and present its visualization features in more detail.

2.2 Design Goals

In building Clack we sought to create a general purpose tool for visualizing router software structure as well as the behavior of protocols handled by the router. Clack was designed to aid in the following:

1. **In-Class Demonstrations:** Instructors go beyond the blackboard and slides to demonstrate networking concepts in a clear, realistic, and interesting way.

2. **Hands-on Learning:** Students can perform visual networking "experiments" in lab exercises or benefit from interactive and exciting homework assignments.

3. **Network Programming:** When programming complex network assignments, students can see desired functional-

ity in operation with Clack, and utilize the visualizations to debug their own code.

While designing a graphical tool to support these goals, our own experience teaching networks and feedback from our initial use of Clack in the classroom led to the following core design goals:

Enable network processing transparency: Clack should clearly show the different modules of software that implement network functionality and describe their associated state in addition to how they handle different types of network traffic.
Capable of abstracting unneeded complexity: Real networks and protocols contain significant complexity at many different logical levels, ranging from network topology to individual bits in packet headers. Allowing instructors to decide what functionality is exposed is critical to keep students from being overwhelmed with unnecessary information.
Powerful control over network characteristics and traffic: Interesting networking properties are the result of the current traffic flowing through a network and the network conditions, such as topology, link characteristics, and router queues. The value of a visualization tool for networking education is closely tied to its ability to control these factors.
Lightweight and simple to use: Giving each student his/her own virtual network of routers requires that Clack and the virtualization subsystem be highly scalable. Additionally, as an instructor may choose to use Clack for only one or two assignments/labs in the duration of a course, the overhead of accessing and learning to use Clack must be minimal.
Easily extensible framework: The community of networking educators will undoubtedly think of new and innovative ways to extend Clack to meet additional needs in their own courses. The core visualization capabilities already provided by Clack should allow new network code to be plugged-in by third-parties without requiring any GUI programming. Integrating new analysis and graphical function should require minimal knowledge of the software as a whole.

These goals present our focus for the remainder of the paper. Before exploring these topics in detail in the later sections, we now cover the basics of Clack design.

2.3 A Modular Router Design

Clack's packet processing functionality is implemented as a set of individual *components* connected together by wires. Packets are passed from component to component, each of which may generate a response packet, drop the packet, or send it to another component to which it is directly connected.

This modularized design is adopted from the Click modular router [Morris et al. 1999], a high-performance router for Linux. As in Click, an individual component generally implements only a small portion of router functionality, keeping component design simple and easy to understand. Components perform packet processing and can keep state as configured manually by the user or automatically via a protocol.

Components have one or more "ports," that provide channels for passing packets between components. Each port has a direction, "In" or "Out" and any component may have multiple ports which accept traffic from multiple sources or feed traffic to multiple destinations. For example, one of our basic router components receives packets from the link layer and sends them out one of two ports, one for IP traffic and another for ARP traffic.

Each component in a Clack router is implemented as a single Java class. Similarly, packets in Clack are Java objects with a type derived from a generic packet class. Packet classes provide

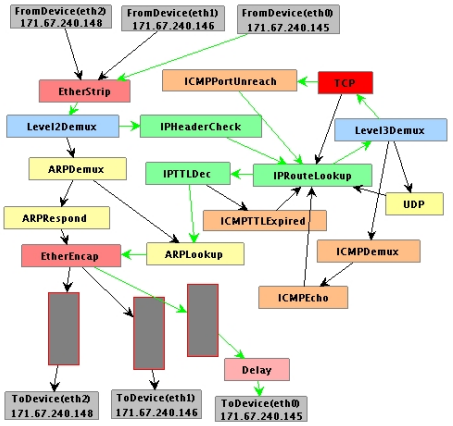

Figure 3: The components of a functional Clack router issuing an ICMP "port unreachable" packet in response to the TCP component receiving a packet at an unopen port. The highlighted links indicate the path of the arriving and departing packets.

getter/setter methods to simplify packet processing and modification for component writers. Packets are processed by a router by simply passing a reference to the packet object from one component to another via ports until a packet is either dropped or handed off to an output interface component to be sent to another host in the virtual network.

To demonstrate the use of Clack, we first implemented a base set of components comprising a simple router. This includes ARP functionality, basic IP forwarding and ICMP handling for generating echo, TTL-expired, and port-unreachable messages. In our implementation, queues are explicit components to provide greater transparency and configurability. In addition, we have developed TCP and UDP stacks which are simplified yet interoperable with real-world network stacks (figure 3).

Clack utilizes two open-source libraries: JGraph [JGraph] for visualizing network and router level object graphs and JFreeChart [JFreeChart] for real-time charting capabilities.

2.4 Gaining Access to Internet Traffic

When a Clack applet is loaded, it initiates a TCP connection back to a server that will host its virtual network, from which it can send and receive Internet traffic. This ability to access live Internet traffic is provided by the Virtual Network System(VNS) [Casado and McKeown 2005].

VNS emulates virtual network topologies reachable from the Internet. In a simple example, for each Clack instance that is loaded from the web, VNS could create a network topology with one router (which would be the Clack instance) connected as a gateway to two standard application servers (which are real Linux machines, such as a web server, that VNS multiplexes into each virtual network). If ten Clack instances were running, VNS would be emulating ten completely independent, isolated topologies. For each topology, VNS assigns global IP space to be used by the routers and application servers. Therefore, any traffic sent to those IPs (such as a web-request), would be sent to the topology and thus be processed by the Clack instance.

VNS sends all traffic received on a given topology to Clack over the TCP connection, specifying which router interface the packet was received on. Similarly, to send packets on the Internet, Clack sends the desired packets to VNS and specifies the interface to send it out on. VNS treats all packets as raw Ethernet frames, therefore the Clack router can arbitrarily modify the

IP header allowing it to acts as a full router. Note that providing direct access to link-layer traffic typically requires the user to have administrator privileges. We avoid this by running VNS as a privileged process on our remote machine and sending all the traffic which we receive on to the Clack applet over the established TCP connection.

In addition to emulating simple single router topologies, VNS can emulate topologies of arbitrary complexity. As described in section 6.2, we use topologies with three routers to teach the implementation of a simple routing protocol.

3 Visualizing Network Behavior

The major contribution of Clack is enabeling students to discern "what" is happening within the router software. That is, students are able to recognize the flow of a single packet or stream of packets through the router and determine whether a packet reached an output interface, was used to generate a reply, or was dropped. It is also important that students understand the "why" behind the router's action. To aid in this, Clack provides introspection into the router software and associated state at each component during runtime.

To provide transparency with respect to the router and protocol behavior, Clack provides a variety of mechanisms to convey information to students in an intuitive way without exposing unnecessarily complex network details. Because the optimal level of visual abstraction depends on the specific behavior being analyzed, Clack offers the user three distinct vantage points, the network level view, the router level view and the packet level view.

3.1 Network Level View

The highest level of abstraction available in Clack is the network view, which displays the topology of routers and hosts, with their interfaces connected by links (figure 1). While individual portions of router functionality are not accessible at this level of abstraction, the network view is critical in allowing students to gain a "big picture" understanding of the network topology, which orients them for understanding the operation of the network at higher-detail granularities.

Within the network view, links light-up to indicate traffic as it flows from the Internet and through the topology, allowing users to see how traffic is crossing the network. Students can also modify the status of links in the network, enabling or disabling them in order to test the network's reaction to failures.

As described in the following section, to analyze the details of router behavior, the user "zooms in" to the modular router view of Clack, where one may inspect an individual component's state and behavior. The primary exception to this paradigm is the use of the network view to simultaneously view the contents of each router's routing table state (see figure 9). These routing table views update in real-time and highlight table entries as they are used to forward packets, allowing a user to debug and understand routing in complex networks while still maintaining the overall topology within the view.

3.2 Router Level View

By clicking on a router icon in the network view, the user "zooms into" the router view, which displays a single window of box-like components connected by directed and visible wires (figure 4). As packets flow from one component to another, the wires carrying these packets change color, giving a visual indication of the packet-processing flow through the router and helping students understand the dynamics of router software. Conceptually, each wire is a procedure call between two different modules of the routing software, with the semantics of the ports defining the interface between components in a manner similar to object-oriented programming. Users have the ability to control the overall speed with which components pass packets to each other, allowing them to slow down packets to more easily see the sequential behavior of individual packets flowing through the router [1].

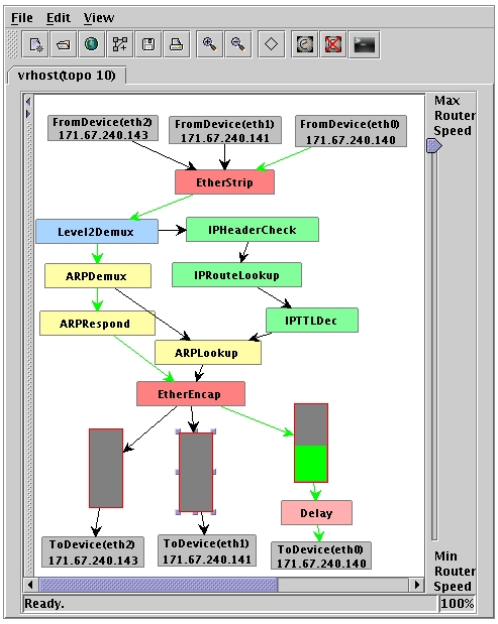

Figure 4: Router level view of Clack during an HTTP file download. The highlighted edges indicate the traffic path through the router.

Clack routers can either be built component by component or automatically loaded from configuration files (section 5.2). When empty, a router consists only of the components that represent input/output channels to the virtual network through the router's interfaces. Building or modifying a router consists of adding, deleting, moving, or editing individual components within the router view and connecting their ports to create a graph that performs the desired packet-processing.

Most components are represented on screen as simple boxes labeled with the component name. To promote a more intuitive understanding of the interrelations between different types of components and each component's position within the OSI layer model, components are colored depending on their membership in certain functional groups (e.g. all ARP-related components).

Within Clack it is also possible to make components that update their appearance in real-time depending on the component's behavior or state. For example, our simple Clack router has queues that fill and drain in real-time (see figure 4). Additionally, components can flash colors to indicate to the user that a particular event happened and that closer inspection is warranted. For example, a component such as the routing table would flash red and log an error message to indicate that a packet destination did not match any route in the table.

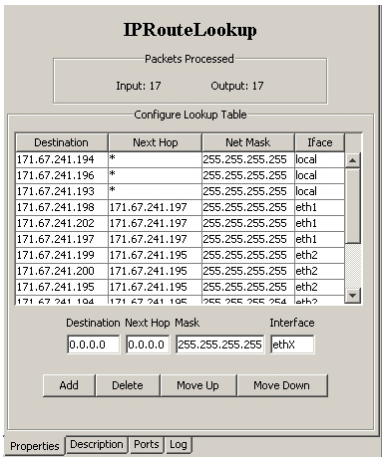

Figure 5: An example property view window used by students to explore the internal state and operation of a routing table component. The configuration pane is shown here, while the tabs at the bottom of the window are used for accessing other panes of the property view.

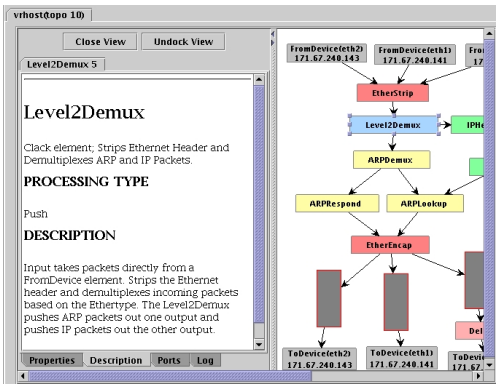

Figure 6: Documentation pane describing the *Level2Demux* component

3.2.1 Detailed Exploration of Routing Software Design and Function

The number of components within the router view is such that an actual component block on screen is too small for displaying detailed information about the component's state and configuration. Clack provides "properties views", which are similar to the properties dialog windows commonly used in modern file managers and other applications, to solve this problem (see figure 5). Properties views are pop-up windows that contain several tabbed panes of detailed information, the exact content and packaging of which is tailored to the individual component. This allow users to "drill down" into the component.

Standard Property Views: All component properties views share a few standard features, including a tabbed pane that provides a description of each of the component's ports and displays the neighboring component and port to which each local port is currently connected. This describes to students the basic software interface exported by this component, and indicates which other router components provide functionality needed by this component. For example, the routing table component's port for outgoing IP packets with valid routes connects to the IP TTL component's port for IP packets needing a TTL decrement and new header checksum [2].

Another standard feature is an embedded HTML page describing in detail the internal state and algorithms used by the component and how each port relates to this behavior (figure 6). Properties views also display statistics about the component, including general values such as packet in/out counts as well as information unique to a component's specific behavior (e.g. the total number of packets dropped by a queue).

Finally, Clack also offers a textual logging console within each component's property view for particular cases when numeric precision is necessary, viewing an entire flow of events as a sequence is beneficial, or messages will be viewed only well after they have occurred (as is the case when a component signals an error).

[1] Albeit at the potential cost of causing a network timeout if a slowed packet is mistakenly assumed to be lost by a network end-point

[2] This information is also accessible directly from the router view, using a "mouse-over" of the wire connecting the two ports

Extended Property Views: Property view windows are also important for exposing the current state of components, such as the ARP cache or IP routing table (seen in figure 5). The ability to see and edit this state helps students understand the router's decision process. As a result, the properties windows also serve as the main mechanism for modifying the internal state of a component. Property views can also provide more in-depth visualizations related to that component, such as graphical plots relating to the component's operation. These visualizations benefit from the wealth of core and third-party graphics libraries available for Java.

Creating router components with a sole purpose of supporting such visualizations has proven to be a simple yet powerful tool for visualizing properties of the traffic flowing through a router. For example, we implemented a component which is placed inline with the router's IP forwarding path and creates real-time graphs of per-flow TCP sequence numbers vs. time to demonstrate congestion control and retransmission behavior (figure 7).

3.3 Packet Level View

The third and most detailed of the different views provided by Clack is a packet analyzer modeled after Ethereal [Combs]. Due to Ethereal's popularity both in and out of the classroom, many students are already familiar with the user interface. This view contains three graphical sub-panes: the packet summary pane, the header information pane, and the packet contents pane (figure 8). The packet summary pane contains a list of all packets captured in sequential order with basic information to inform the user about the general content of each captured packet. The packets may be sorted based on packet type (e.g. FTP) or which interface it came in on. This allows the user to follow specific flows even when there is a significant amount of traffic through the router.

When a user clicks on a packet summary, the lower two views provide more detailed information about that packet. The header information pane offers an expandable tree of all major packet headers (link, network, and transport level) and provides easy to read labeled fields showing the value of each header field. The third pane shows a byte-by-byte view of the packet in hexadecimal, along with the same bytes interpreted as ASCII. To help students draw the connection between protocol parameters seen in the header information pane and the actual byte content of the packet, selecting any value in the header pane highlights the corresponding bytes in the packet contents pane. Similarly, if a student wants to know what a specific byte in the packet means,

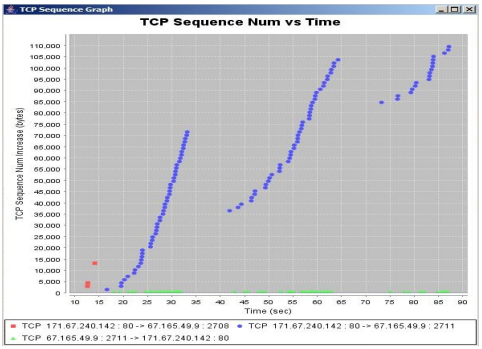

Figure 7: A real-time graph showing the evolution of the TCP congestion window for live TCP flows. This graph is used to illustrate the TCP congestion avoidance algorithm.

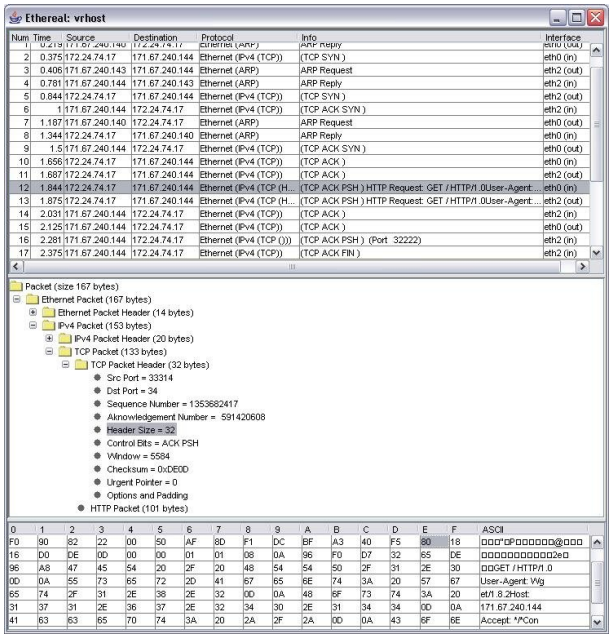

Figure 8: Packet level view of a web transfer using the Ethereal component.

they can just click on it and the expandable tree will select the corresponding information.

The packet analyzer can be used to sniff all traffic sent to or from a particular router in a Clack network or as a component placed in the router graph to sniff only the traffic sent through it. In both instances, the packet analyzer provides the packet level details that the network and router level views abstract away. If a higher level view shows traffic traversing a link or wire and the student does not already understand what packets that traffic represents, the student may use the packet analyzer to find out. This provides powerful control over the amount of detail students are confronted with when examining a router or network topology.

Similar to the "Ethereal Labs" discussed in Section 7, Clack's packet analyzer can be used to study packet features of end-to-end protocols like TCP. Additionally, Clack can explore infrastructure level protocols, like routing update messages after a link failure, that would be inaccessible to normal Ethereal users. Such inspection capabilities have also proven very useful for students in courses using Clack as a network programming platform.

4 Controlling the Network Environment

Demonstrating network concepts requires precise control over the network environment. This includes both the ability to generate different types or patterns of network traffic and to control key network parameters such as delay and loss.

4.1 Traffic Generation:

The Clack toolkit provides instructors with several options for generating traffic to be processed by Clack routers: traffic can flow either between an outside machine and a host within a Clack virtual topology, or between two hosts inside a topology. We outline two simple mechanisms for traffic generation below.

Physical Hosts within a Clack Topology:
The simplest mechanism for generating traffic through a Clack router is to use an application, such as ping or traceroute, running on the same client machines that is running Clack. While routers themselves can respond to these limited types of messages, generating interesting transport-level traffic require more sophisticated applications running within a topology. The underlying VNS layer supports the integration of physical host machines running applications, like web or audio streaming servers, into virtual topologies, giving a simple means of providing TCP and UDP download flows through the router.

Clack Applications & Redirectors:
Since Clack includes simplified TCP and UDP networking stacks, we also provide the ability for users to write "Clack applications" that run on top of these networking stacks. These applications have all the capabilities of regular Java threads (including access to regular Java sockets) and additionally have access to a simplified socket interface for sending and receiving traffic via the router's network stack. For example, we have implemented several applications, including an HTTP-GET requesting agent and a very simple web server for visualization of the local TCP stack components of a Clack router.

Clack applications also provide the ability to easily install a traffic "redirector" on a router, which transparently proxies traffic TCP or UDP to and from a real Internet host. That host can provide the actual application functionality, but appear as if a service is provided by a host within the Clack network. This allows instructors to leverage arbitrary application services (e.g., DNS) without requiring these applications to be installed on physical machines co-located with the VNS server.

4.2 Controlling Network Characteristics and Irregularities

Clack supports the creation of arbitrarily complex virtual network topologies, which is particularly important when visualizing highly topology dependent topics like routing protocols. Additionally, the ability to disable and enable network links allows students to see how routing protocols react to network failures.

Giving instructors the ability to dynamically tweak network link characteristics, such as relative link rate, packet-loss, and packet reordering, allows them to demonstrate how end-to-end protocols are impacted by such network-level factors. Such properties are difficult to demonstrate using other network visualization tools that simply "sniff" live Internet traffic on a physical network provide no similar control over network conditions. Clack uses "auxiliary components" placed in the forwarding path of the router to emulate the link level behavior. For example, we use a packet throttling component that slows

down all packets passing through it by a configurable rate. This creates a bottleneck link resulting in network congestion at output queues, which can demonstrate the interplay between queue occupancy and TCP congestion control.

5 Customizing Clack

Adopting Clack for use in a remote educational institution is easy, as the VNS infrastructure itself is hosted and administered at Stanford. Instructors (or students) interested in using Clack email us the number and type of topologies required and receive, in return, a hyper-link per topology for the students to click on to access a pre-configured Clack router. An instructor may extend Clack to add demonstration or project specific functionality. Any extensions or changes in the default configuration can then be serialized and made accessible to all students.

In this section we describe the facilities provided by Clack to handle extensions of new network functionality or visualization capabilities and serialization of router configuration state.

5.1 Extending Clack

A major consideration in Clack's design was providing a simple interface for third-party developers (such as instructors or course staff) to add new capabilities. Adding new router or analysis functionality is done by creating a new sub-class of the *ClackComponent* object that defines this newly added component's input and output ports and implements the specific packet processing functionality desired. A major benefit of Clack's modular design is that adding new functionality does not require understanding how the other parts of a Clack router operate. Property views, including basic statistics, port connectivity information, and console logs (See 3.2.1), are automatically created for a component with no work required from the developer. With a small amount of knowledge about Clack API's and Java GUI programming, the developer can allow users to view and modify component specific state or add new features such as real-time graphs and other visualizations. Developers can also easily write new Clack applications and extend the packet analyzer to parse new protocols.

5.2 Clack Configuration

Clack's support for serializing and loading configurations allows an instructor to configure the routers and network once, and then replicate this configuration over hundreds of identical topologies for individual students.

After a network of routers is built, Clack serializes both graphical position and network configuration information to an XML file so the identical set-up can be loaded at a later time, even on a different virtual network. The file also indicates whether each router will be displayed in its own graphical window for the user to view and edit or if it will be run in the background as a "supporting router." Supporting routers may be configured to participate in a routing protocol and/or generate traffic as described in section 4.

For more information about Clack's features related to extensibility and configuration, please see [Wendlandt 2005].

6 Clack in the Classroom

At present, Clack has been used in networking courses at six universities to aid in understanding the basic design and function of network routers. In spring 2006, we also piloted two new uses of Clack, including a routing protocol programming project and a "lab exercise" for non-majors analyzing different real-world transport protocols. In this section we discuss our experiences using Clack, both here at Stanford and remotely, and describe our ongoing efforts to broaden its utility in the classroom.

6.1 Building a Router

Clack's principle use has been to aid students in understanding the design and function of software routers. The VNS project at Stanford hosts a popular introductory networking assignment in which students implement Internet routers in C. To help in the design and development of their routers, we make Clack available to each student so they can see, at a high-level, a component abstraction of the functional software blocks and how they operate on real traffic.

Students use property views and other available visualizations to explore the state kept by each component. They can see how modifying state, such as the routing table, impacts how traffic is processed within the router and whether that traffic successfully reaches its destination.

In addition to demonstrating a sound software construction, the students use Clack and its inline protocol analyzer to understand fundamentals of packet processing in routers: decapsulation and encapsulation, protocol demultiplexing, the IP forwarding path, ARP processing and lookups, as well as common ICMP functionality. Students using Clack can probe their graphical routers with pings and traceroutes to see what functionality the router must implement in order to correctly handle these packets and can refer back to it throughout the course of their development to clarify how their router should be performing.

The software router assignment in C has been used at six universities, including our own, over the past four years. Over the last year, students at five of those institutions have had access to Clack while building their routers. The introduction of Clack reduced the number of questions posted to the course mailing list appreciably. Furthermore, the assignment code submitted by students during our use of the visualizations had noticeably cleaner module decompositions which, in many cases, directly mimicked the components as displayed by Clack.

At the completion of the projects we collected student feedback using formal, anonymous evaluations taken online. The evaluations asked students in what ways Clack was helpful in understanding and building network routers and whether it should continue to be part of the assignment. The students responses have been very positive. All the responses indicated that Clack was helpful and should continue to be offered along with the assignment.

The particular aspects of the visualization that students found useful varied by response. Many students commented on the ability to view packets as they travel between components and illicit responses. Particularly beneficial was Clack's support for slowing the router so that per-component processing was easy to follow. The ability to see a modular decomposition of the software was also widely commented on. As part of our future work, we plan to continue to investigate which features of Clack are most heavily relied on by students when building their own routers.

6.2 Programming & Visualization: Implementing RIP Routing

In spring 2006, an undergraduate networking class at Simon Fraser University used Clack as a programming platform for students to implement and visualize a RIP-like[RFC1058] distance vector routing protocol.

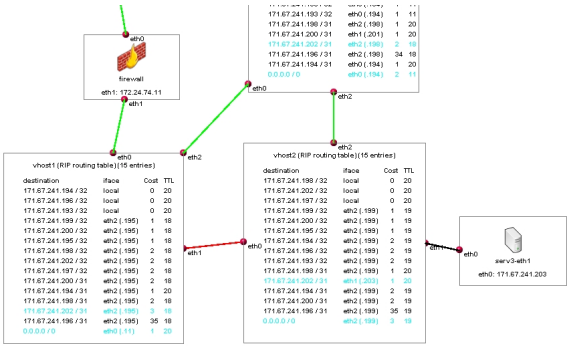

Figure 9: Close-up view of a portion of the network view used in the RIP assignment. We can see the routing tables of two routers with recently used entries highlighted. The direct network link between the two visible routers is disabled (colored red) so packets to and from the application server are routed through the third, partially visible router.

Like many distributed protocols, understanding and debugging a distance vector routing protocol is greatly aided by visualization support. Students were given a skeleton component within which to implement their RIP functionality and could use Clack visualizations such as the topology-level view of routing tables (section 3.1 and figure 9) and the packet analyzer (section 3.3) to debug their implementations while probing them with real Internet traffic.

Students received three configurations for Clack: a "reference configuration" with all three routers running a reference implementation of the RIP module, a "development configuration", where one of the three routers ran the student's RIP module, and a "test configuration", which had all three routers running the student's code. As a result, students could enable/disable a link or modify a path metric and ping through their network topology to compare their modules's behavior to the reference implementation in a simple and visual way.

Demonstrating the advantage of Clack's scalability, each pair working on the programming assignment received two independent Clack networks, enabling them to run the reference implementation and their own side by side to visually debug any discrepancies. While we were unable to complete a formal & anonymous evaluation, feedback from an in-class discussion indicated that while students found the assignment challenging, they also felt that Clack (particularly its ability to display topology-wide routing and forwarding information, see section 3.1) helped them gain improved insight into how distance vector protocols operate under normal and abnormal conditions. The course instructor was excited by his experience with Clack and indicated a strong desire to use it in future courses.

6.3 Interactive Lab: Network Transport Layers

We are currently working with the course staff of a "computer science for non-majors" course at Princeton University to develop a hands-on networking lab using Clack to demonstrate key properties of transport-level protocols with respect to network congestion. Clack's ability to create varied conditions for packet congestion and loss, along with its integrated graphing capabilities for analyzing the behavior of individual flows greatly facilitates the creation of such a lab. Also important to this course' instructor is Clack's simple interface, its ability to hide complexity by showing just a "bare bones" router, and the use of comman Internet clients, such as web browsers, to generate network traffic for analysis.

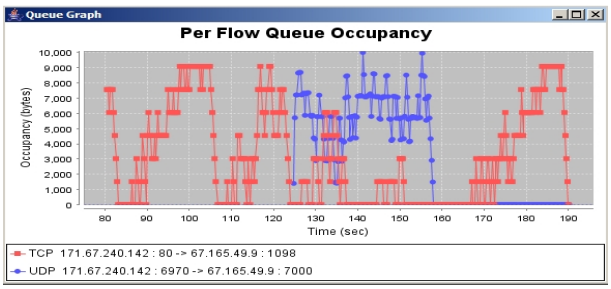

Figure 10: Real-time graph of a TCP and UDP flow competing on at bottleneck link. The UDP audio stream starts at time 125 sec and significantly limits the throughput of the TCP flow until the audio stream ends at 160 sec.

In this lab students use their web browsers to download files from a webserver in the topology to create TCP flows through their router, which has one of its interfaces configured to be the bottleneck link on the path from the webserver to the user's computer. Students use a multimedia application such as Apple's Quicktime to access a streaming audio/video server also within the topology using the Real Time Streaming Protocol (RTSP) over UDP. The current lab design offers the following core demonstrations:

1. Observe TCP congestion control with a single flow under varying queue sizes. A graph of queue occupancy will exhibit the classic "TCP sawtooth" shape, a feature often cited in networking courses because it is indicative of TCP's specific congestion control algorithm (visible in figure 10).

2. Create a UDP audio flow through this same congested buffer to illustrate the interaction of TCP's loss-based congestion control mechanism with UDP, which does not react as conservatively to loss. The queue occupancy graph (figure 10) will show the UDP flow consume nearly all queue capacity, largely starving the TCP flow until the UDP traffic halts.

3. Use audio streaming over RTSP to demonstrate that, unlike reliable protocols such as TCP, real-time protocols degrade quality instead of retransmitting lost packets. We install a configurable packet loss component in the forwarding path of the router that allows students to click a button to drop the next packet passed through the component. After seeing how a TCP download reduces it rate and retransmits after a loss, students listen to a song streamed through their router and choose when to drop packets. Single packet drops result in a small blip in the audio, while several successive drops result in more significant quality degradation.

Feedback from course staff leading the lab exercises emphasized that students were excited to be using live Internet traffic and easily grasp the new concepts because of the easy to undergrstand real-time graphs. Since the students where non-majors, staff also noted that a more simplified router-view with fewer components would be better for students since the lab exercise did not focus on router internals.

7 Related Work

Network Packet Analyzer Labs: Graphical packet analyzers, such as Ethereal [Combs], are a popular method of providing

hand-on access to real network traffic. Packet analyzers allow students to capture network traffic on a local computer (or use previously capture traffic), and view packet header fields and data within a GUI that parses and formats the packets. The Visual TCP/UDP Animator [Zhao and Mayo 2002] has capabilities similar to Ethereal for TCP and UDP traffic, and adds in some TCP specific visualizations, such as a "time-line view" of a TCP connection.

While useful for simple demonstrations, packet analyzer labs have three notable limitations when used for networking education. First, they must either be used in a lab environment, or in another network location where other "real-world" network traffic will not create a noisy and confusing packet trace. Second, protocol analyzers can typically get a packet-level view of end-host behavior, but cannot observer router processing behavior. Third, even when a packet analyzer is being used to explore end-to-end protocols, it can only observe traffic and can't modify link properties or queuing behavior that allows instructors to create many of the scenarios they wish to demonstrate.

Network Simulators: Network simulators such as ns-2 [VINT-NS-2] or OpNet [OpNet] provide a synthetic environment for simulating network protocols in multi-node topologies. They allow users to specify topology configuration including, link and router properties as well as traffic types and patterns. Most simulators have a visualization components [Deborah Estrin and Yu 2000], some of which have been used to create scripts designed for classroom use [VINT-NAM].

Generally, these environments were developed to aid in research and focus on correctness rather than providing an intuitive educational interface. Further, because they rely on discrete event simulation, they often don't have native support for visualizing real Internet traffic or modifying device configurations in real time.

Specific Network Animations: In step with the large number of animations used to teach popular algorithms to computer science students, over the years networking educators have created animations to demonstrate many core networking concepts, such as encapsulation, queuing mechanisms, or routing protocols. Predominantly these are stand-alone visualizations to demonstrate a single networking concept[Kurose and Ross b; Holliday 2003; White 2001; VITELS].

Visualization of Network Programming Assignments: Some special-purpose programming environments allow students to implement and visualize the behavior of network code [Rohit Goyal and Durresi 1998; McDonald 1991; Pilu Crescenzi and Innocenti 2005]. Primarily these tools focus on endhost concepts such as link-level access control or transport-level congestion control and operate over idealized protocols.

8 Conclusion

Clack fills a gap in network-education visualization environments by combining a detailed view of the software of an operational router along with real network traffic and the ability to drop, modify and inspect packets. It is easy to adopt and integrate into an existing classroom setting because it is web accessible as a Java applet and doesn't require administrator privilieges to gain access to raw network traffic.

We've successfully used Clack at six universities to aid in the development of software routers and demonstrate networking concepts. Feedback both from the professors and students has been very encouraging.

Clack and associated curriculum is publicly available and fully supported by our group. A live demo is available online at http://www.clackrouter.net along with the full source code, developer documentation and other resources to help instructors integrate Clack into their classroom.

We are working to expand Clack and provide more course materials for use in networking classes. Our plans include using Clack as an in-class demonstration tool, a graphical environment for "lab experiments", and a visualization tool for students developing Internet software as part of networking courses.

9 Acknowledgements

This work was funded by the NSF, grant 02-082. Both Dan Wendlandt and Martin Casado received funding from the Dept. of Homeland Security Scholar & Fellow programs. Dan Wendlandt also received support from Achievement Rewards for College Scientists (ARCS). We would like to thank Jon Effrat for significant help with Clack user-interface issues. Additional thanks to the many students and staff that have provided invaluable feedback for improving both Clack and VNS and their accompanying assignments.

References

CASADO, M., AND MCKEOWN, N. 2005. The virtual network system. In *Proceedings of the ACM SIGCSE Conference*.

CLACK. Clack graphical router project. http://www.clackrouter.net.

COMBS, G. Ethereal network analyzer. www.ethereal.com.

DEBORAH ESTRIN, MARK HANDLEY, J. H. S. M. Y. X., AND YU, H. 2000. Network visualization with nam: the vint network animator. In *IEEE Computer 33(11):63-68*.

HOLLIDAY, M. A. 2003. Animation of computer networking concepts. In *Journal on Educational Resources in Computing*.

JFREECHART. Jfreechart: A free java chart library. http://www.jfree.org/jfreechart/index.php.

JGRAPH. Jgraph: Java graph visualization and layout. http://www.jgraph.com.

KUROSE, J. F., AND ROSS, K. W. Ethereal labs. http://gaia.cs.umass.edu/ethereal-labs.

KUROSE, J. F., AND ROSS, K. W. Kurose and ross student resources - applets. http://wps.aw.com/aw_kurose_network_3/0,9212,1406346-,00.html.

MCDONALD, C. 1991. A network specification language and execution environment for undergraduate teaching. In *SIG Computer Science Education Bulletin*.

MICHAEL J. JIPPING, AGATA BUGAJ, L. M., AND PORTER, D. 2003. Using java to teach networking concepts with a programmable network sniffer. In *SIG Computer Science Education Bulletin*.

MORRIS, R., KOHLER, E., JANNOTTI, J., AND KAASHOEK, M. F. 1999. The click modular router. In *Symposium on Operating Systems Principles*, 217–231.

OPNET. Opnet modeler software. http://www.opnet.com/products/modeler/home.html.

PILU CRESCENZI, G. G., AND INNOCENTI, G. 2005. Netpride: An integrated environment for developing and visualizing computer network protocol. In *Innovation and Technology in Computer Science Education*.

RFC1058. Request for comments (rfc) 1058: Routing information protocol. http://www.ietf.org/rfc/rfc1058.txt?number=1058.

ROHIT GOYAL, STEVE LAI, R. J., AND DURRESI, A. 1998. Laboratories for data communications and computer networking. In *Proc. of Frontiers In Education Conference, p. 1113-1118*.

VINT-NAM. Using ns and nam in education. http://www.isi.edu/nsnam/ns/edu/index.html.

VINT-NS-2. The network simulator. http://www.isi.edu/nsnam/ns/.

VITELS. Virtual internet and telecommunications laboratory of switzerland. http://www.vitels.ch.

WENDLANDT, D. 2005. *Clack: A Graphical Router Toolkit for Networking Education*. Undergraduate honors thesis, Stanford University, Computer Science Dept.

WHITE, C. M. 2001. Visualization tools to support data communications and computer network courses. In *Journal of Computer Sciences in Colleges, Vol. 17 Issue 1*.

ZHAO, C., AND MAYO, J. 2002. A tcp/udp protocol visualization tool: Visual tcp/udp animator (vta). In *Intl. Conference on Engineering Education*.