Clack: A Graphical Router Toolkit for Networking Education

Daniel Wendlandt
Department of Computer Science
Stanford University
Stanford, CA
danwent@cs.stanford.edu

May 16, 2005

Abstract

We present the Clack Graphical Router Toolkit as an extensible platform for teaching students about network infrastructure and protocols. Clack is a web-accessible Java application that graphically displays the operation of a software router made up of modular components. The components in a Clack router visually update in real-time while handling live Internet traffic and also support on-the-fly reconfiguration. Instructors using Clack will benefit from a software design focused on extensibility and ease of use, making it simple to create routers and surrounding network environments conducive to teaching a variety of networking concepts. In this thesis, we outline the potential for Clack by describing the toolkits design and providing in-depth examples of its use as a classroom demonstration device, hands-on learning tool, and network-programming platform.

Acknowledgments

Special thanks to the members of the High Performance Networking Group at Stanford University for their help in developing VNS and Clack. Particularly, Prof. Nick McKeown for advising this work and Jon Effrat for helping with development. The greatest thanks to Martin Casado for mentorship concerning Clack and too many other things to mention here. I would also like to thank my close friends for helping me out during all the ups and downs with Clack in the past year.

Contents

Chapter 1

Introduction

As the applications running on the Internet and other networks continue to become more central to the daily functioning of business, government and society as a whole, instructors recognize the increasing importance of students gaining an understanding of the underlying network infrastructure and protocols. Students need familiarity with how routers and end-host networks stacks operate to understand key topics ranging from queuing delay, routing protocols, and TCP congestion control to intrusion detection and denial-of-service. Unfortunately, in today's networked systems this functionality is often buried deep within expensive hardware or complex kernel code, making it difficult for students to explore these topics in a meaningful way.

In most introductory networking courses, there are a limited number of tools to give students interactive experience dealing with real Internet traffic below the socket layer and no existing mechanism for students to get a similar level of exposure to router internals. Students primarily learn about networking infrastructure and protocols by reading texts and working through pencil-and-paper examples.

A recent trend in networking education has recognized the value of hands-on and visual mechanisms for teaching about networks. Common approaches, as outlined in Chapter 2 include the use of packet analyzers, simulators, physical network labs, stand-alone visualizations, and network programming projects. These approaches are often either limited in their ability to demonstrate real Internet behavior in a compelling way or provide realism but at the cost of significant overhead for instructors and students.

The Virtual Network System(VNS)[19], a teaching tool recently developed at Stanford, provides a compelling option by reducing the overhead for students to access network traffic. VNS simplifies the process of students developing a router or similar piece of network infrastructure by giving each student a lightweight virtual network that is fully connected to the Internet. Building a user-space routing in this environment provides the realism of dealing with Internet protocols and the excitement of handling live traffic with the advantages of lower system overhead and reduced complexity for students who do not have to deal with a complex kernel programming environment.

However, while VNS allows students to more easily build or modify a software router, it does little to support student analysis of the key network protocol or infrastructure topics mentioned above. Thus, our motivation in developing Clack is to make the compelling realism and flexibility provided by VNS more accessible by creating a graphical router. We believe that in many cases the dynamic and complex nature of network protocols is best demonstrated visually, and the Clack toolkit provides a flexible platform for graphically analyzing protocol behavior.

A Clack router has a modular design that displays a router as a collection of interconnected blocks, each which implements a simple piece of packet processing functionality and exposes its internal state and operation to the user in an intuitive way. As packets flow through a router, a user can see their state change in real-time. Furthermore, users can reconfigure routers on-the-fly in order to demonstrate how different configurations affect traffic flowing through the router.

Importantly, Clack provides this capability in an extremely low-overhead and easy-to-access way. For example, an instructor could email a link to each student in an introductory networking class. When a student clicks on this link their browser will open and begin running the Clack Java applet, which displays the student's router within its own virtual network. Students can ping this router, since it is connected to the Internet, and even download files through it by accessing a web server located within their virtual network. As packets from the download flow through the router, wires connecting each processing block light up so that students can see each action the router performs in real-time. If the instructor wishes for students to explore the behavior of the router buffers in more detail, students can double-click on an individual buffer to see a real-time graph of queue occupancy that details the behavior of TCP congestion control through a drop-tail queue.

While this simple example falls far short of demonstrating all a Clack router is capable of, it gives a taste of why we believe the simplicity and flexibility provided by Clack can provide substantial benefit to the networking education community.

Chapter 2 of this thesis discusses a wide array of related work to understand how Clack fits into the collection of networking education tools. Chapter 3 takes a more detailed approach to describing Clack's modular design and implementation. This provides a foundation for chapters 4 and 5, which discuss the two core design goals of Clack: network processing transparency and a flexible design that is easily configured and extended by instructors. We then look at several concrete examples of how the Clack toolkit may be used by educators before moving to the final chapter, which discusses design constraints, limitations and future work.

Chapter 2

The Clack Toolkit and Networking Education

The importance of teaching networking at the infrastructure and protocol level has grown along with the Internet, and this research area has enjoyed a significant amount of exploration by educators around the world. This chapter provides an in-depth look at other educational efforts related to our development of the Clack toolkit. The chapter ends by discussing the advantages and disadvantages of Clack compared to related teaching tools.

2.1 Related Work in Networking Education

In the area of network infrastructure and protocol education, prior efforts to provide an interactive supplements to textbooks and paper assignments have explored several avenues. Some instructors use packet analyzers or simulators to teach about major Internet protocols and physical infrastructure labs or emulator software to teach about routers and other network infrastructure. Visual tutorials are another tool frequently employed to explain major networking concepts. Additionally, instructors have developed programming assignments that allow students to implement simplified functionality similar to that contained within the TCP/IP networking layers. We discuss each of these approaches below.

2.1.1 Network Packet Analyzer Labs

A growing trend in networking education is the use of network packet analyzers, particularly the graphical application Ethereal[1], to provide hands-on networking lab assignments or tutorials for students. Packet analyzers allow students to capture network traffic on a local computer's network interface and view these packets via a GUI that parses and formats the packets, making each of the encapsulated header and data fields simple to view.

Ethereal is even being integrated into leading networking textbooks such as *Computer Networking: A Top-Down Approach Featuring the Internet* (J.F.Kurose & Keith W. Ross, 3rd edition) and *Communication Networks* (Alberto Leon-Garcia & Indra Widjaja, 2nd edition). For example, the Kurose & Ross text provides a set of pre-written labs for students to access from their home or lab machines[2], while many other university networking courses have leveraged Ethereal to create their own laboratory projects[3]. With its focus on packet and header data, Ethereal labs are primarily used to explore the specific nature of certain protocols by analyzing one or more packets in a sequential stream. These hands-on labs are popular because of both their simplicity and the appeal of working with real Internet traffic.

2.1.2 Network Simulators

The Network Simulator (NS-2) [4] and its graphical companion Network Animator (NAM) [5] provide a different vantage point to students, allowing them to see beyond the per-packet view of a network analyzer like Ethereal. NS-2 allows users to built networks, specify link and router properties, and generate traffic to flow over these topologies. A simulator has the benefit of offering a high degree of control to the simulation creator and can thus be attractive for in-class demonstrations. With this use in mind, a small repository of pre-built demonstration scripts is available for educators on the NAM web site. The majority of these demonstrations, written in the Tcl language, display variations of TCP behavior, queuing policies, or multi-cast [6]. In addition, a tool called NAM-Editor was developed to create scripts implementing a subset of NS-2 functionality via a graphical interface, potentially removing the need for Tcl scripting [7, 8].

2.1.3 Hands-on Learning for Networking and Router Configuration

Recognizing the importance of teaching about Internet infrastructure, the National Science Foundation (NSF) joined with industry partners to support the creation of Internet Teaching Laboratories at approximately 30 US universities. Using donated equipment, this project set up physical labs with PCs and Cisco hardware and funded the creation of curriculum using these resources[9]. These labs, usually a collection of small networks each consisting of several end-hosts connected by switches and routers, received high marks from students and faculty for their ability to provide real-world networking skills. In most cases, the curriculum in these labs focused on configuring the network devices to set-up working networks that remained isolated for the Internet. While physical labs offer many benefits, the costs of building and running these labs are prohibitive for many smaller universities.

A related collection of educational tools, frequently sold as commercial products to prepare users for Cisco Certification tests, emulate the behavior of a simple network with routers and end-hosts without using any real hardware. Such tools allow students to create small network topologies and enter router commands into an emulated router terminal in order to configure interfaces, set-up routing tables, or perform more advanced network configuration. Users can perform basic commands like ping and traceroute in order to verify correct configuration, and in some cases significant logic is embedded in within the application to help correct network configuration errors made the by user[10].

2.1.4 Networking Education Visualizations

An additional category includes many efforts in the broad realm of networking visualizations. Previous work in this category indicates that many core networking concepts, like encapsulation, queuing mechanisms, or routing protocols, lend themselves well to graphical and interactive presentation. Predominantly, these visualizations are highly targeted to the specific networking concepts they demonstrate and were not created in an open or extensible way. Some, however, are implemented as Java applets, including a group of visualizations paired with the Kurose & Ross text web site [11, 12, 13]. Another visualization tool, GIDEN, is notable in that it provides a debugger-like ability to step through and analyze complex distributed network algorithms[14]. Additionally, the Virtual Internet and Telecommunications Laboratory of Switzerland combines visualizations and router emulation with in-depth web tutorials in eight distance-learning modules on networking and security topics[15].

2.1.5 Heuristic Programming Assignments

While introductory networking courses commonly have programming assignments consisting of server and client applications that utilize network sockets, some instructors have recognized the need for a programming assignments teaching concepts about lower-level protocols. Many of these solutions are developed and used on a per-school basis. For example, the University of Washington created Fishnet[16], a Java-based library for teaching protocol and transport principles. While Fishnet uses its own virtual network, assignments at other universities use real Internet traffic by having students build upon the unreliable UDP layer[17]. Still others combine the use of programming and simulation, completely removing students from having to deal with real network constraints. As an added benefit, these simulated environments often provide support for the visualization of the network protocol implemented by the student[18]. The majority of these projects are focused on teaching networking principles without attempting to replicate full network stacks capable of inter-operating with Internet hosts.

Recent work at Stanford to create and use the Virtual Network System(VNS)[19] provides a different and compelling option for instructors looking to create network programming assignments below the socket level. VNS is distinguished in its ability to allow students to gain experience building network infrastructure systems capable of inter-operating with other generic Internet hosts. In an introductory networking course, students have created a basic Internet router capable of ARP processing, IP-forwarding, and basic ICMP handling in one assignment, and built a simplified but inter-operable version of TCP in the following assignment. As described below in section 3.1, VNS allows these routers to process traffic as if the routers were full-fledged Internet connected hosts.

In a graduate-level Stanford course, VNS was used in combination with NetFPGA[20] hardware to allow students working in development teams to build a combination software and hardware router that supports basic router functionality as well as simplified OSPF routing and a command-line interface. Feedback from both of these Stanford courses has shown these assignments to be both significantly challenging and extremely rewarding. This success has led to VNS and the simple router assignment being used remotely by several other universities [21].

2.2 The Clack Toolkit's Place in Networking Education

We believe that the Clack Graphical Router Toolkit fills an unmet need in the networking community for an intuitive tool that can visually demonstrate router and protocol concepts in a compelling, low-overhead and easy-to-use way. Central to this goal is Clack's use of real Internet traffic, which we believe will generate a more significant level of interest among students compared to the related use of simulations or visualizations. In this section, we outline at a high level how Clack compares to the different networking educational approaches mentioned above.

Packet analyzers like Ethereal provide a key packet-level view for students learning about Internet protocols. Recognizing this value, we are currently creating an analyzer component aimed at replicating this capability inside of Clack. While our packet analyzer will not contain the significant breadth of protocol analyzers that exists in a widely-used tool like Ethereal, we believe Clack provides added benefit compared to production tools by giving instructors more control over the surrounding network environment without the significant overhead of setting up a physical lab.

Clack also provides opportunities in the realm of in-class demonstrations and hands-on tutorials. Compared to a graphical network simulator like NAM, Clack's use of live network traffic can potentially produce more compelling demonstrations because students know they are seeing actual network behavior, not simply a simulation. Additionally, Clack can offer a higher degree of granularity, since no part of the packet processing is abstracted away via a simulation. On the down-side, simulators may provide more deterministic results since they eliminate the potential for variations due to the use of live network traffic. Simulators also often allow users to control packet processing speed, a task that is difficult in Clack because of network timeouts (see section 7.2).

For smaller universities, Clack can eliminate the need to construct physical networking labs and provides easy mechanisms for configuring and deploying a variety of different network environments, depending on educational needs. Clack also has a low barrier-of-entry for students, since it is available via a browser and has an intuitive graphical interface designed for students new to networking. However, since Clack focuses on building conceptual understanding, not developing specific router or network configuration skills, it is not right for those looking to develop industry specific skills.

Clack's use as a visualization tool is compelling because to our knowledge there have been no full-scale visualizations of a router capable of processing live Internet traffic. This provides students with an opportunity to get an intuitive sense of what a router does as a whole, and a chance to peer into the operation of individual components. Additionally, since Clack is not simply displaying a simulation of network behavior, a Clack router is likely to be more highly configurable and extensible if an instructor wishes to modify an existing demonstration. While Clack is a general framework capable of incorporating any type of visualization that can be programmed in Java, it is possible that the network and router level views supported by the Clack toolkit will not align well with the goals of visualization. In this case, a stand-alone visualization would be preferable.

The modular design and simple programmatic interfaces of Clack components make assignments to program additional network level functionality very reasonable for students with little or no systems programming experience. The obvious downside of this approach is that the Java-based environment provided by the toolkit does not provide students with realistic systems-level experience. However, we feel that Clack can nevertheless be compelling for instructors who want students to be able to focus solely on building the data-structures and algorithms used in major networking concepts.

Clack's ability to easily create and duplicate interesting network topologies with specific traffic loads and characteristics benefits all applications mentioned above. Additionally, Clack's sole focus on education results in the potential for more features that illuminate key educational concepts than are available with current tools that are primarily focused on production use.

Chapter 3

Architecture and Design of the Clack Toolkit

Having presented Clack in comparison to other tools within networking education, we now turn our focus to the architecture of the Clack Router Toolkit. This chapter describes Clack's underpinnings in VNS and outlines the software design and high-level features offered by Clack to serve as the foundation for more detailed topics in later chapters.

3.1 The Virtual Network System

The Virtual Network System(VNS)[19] provides the base of network virtualization upon which the Clack toolkit is built. A VNS server can host hundreds of virtual network topologies, which are made up of both physical and virtual hosts connected by virtual links. The VNS infrastructure, including servers and physical hosts, is operated and administered out of Stanford University for use by both local and remote courses. Physical hosts are real machines residing within the VNS lab that are virtualized into each topology. Virtual hosts are not physical machines but rather represent generic network processing entities with interfaces connected to other virtual or physical hosts in the topology. Af-

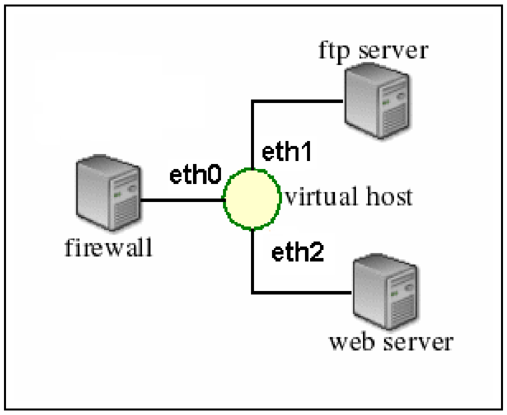


Figure 3.1: A simple VNS topology with a single virtual host. One virtual host interface is connected to the VNS firewall that connects the topology to the Internet. The other two virtual host interfaces are connected to two physical hosts, one acting as a web server and the other acting as an FTP-server.

ter connecting to the VNS server, a client like a Clack router can take on the role of a virtual host by requesting that the server forward the client traffic seen by a particular virtual host within a virtual topology. The client may then inspect, discard, modify, or inject new traffic into the network using any of the virtual host's interfaces, giving it the full capabilities of a real network host. Packets extracted from or injected into the network are full Ethernet frames, with the virtual host performing all packet-level processing.

The most significant contribution of VNS is that hosts on these topologies are given routable IPs and are connected to the Internet. Thus, the traffic processed by virtual and physical hosts represents real Internet protocols, allowing a student to open a command prompt, ping her router, and watch the router respond. Experience with VNS in the classroom has demonstrated that this use of real Internet traffic is central to a student's excitement about VNS-based projects. VNS has also proven straight-forward to use by instructors and students and is highly scalable, giving it a strong advantage over more heavy-weight and difficult-to-configure solutions such as setting up a physical labs or using virtual machines like User-Mode-Linux[22].

For more information about VNS visit http://yuba.stanford.edu/vns/.

3.2 What is Clack?

The Clack Router Toolkit is a Java library and accompanying graphical application that provides a platform for using VNS to achieve a variety of network educational goals. This section focuses on describing the design of the router and supporting application, while later chapters describe in more detail the educational capabilities built into the system.

3.2.1 Modular Design

A Clack router is a collection of interconnected packet processing blocks that we refer to as "components." This design logic borrows heavily from the Click Router, a high-performance modular router for Linux created at MIT[23]. 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. For example, the *LookupIPRoute* component contains a simple routing table and tags packets with an identifier based on the packet's correct out-bound interface.

Components have one or more "ports," that provide channels for passing packets between different components. Each port has a direction, "In" or "Out," and semantics that are specified by the component creator. For example, the *LookupIPRoute* component has an "In" port for accepting packets that require a routing table lookup and an "Out" port for packets that have been tagged with the outgoing interface. Other *LookupIPRoute* "Out" ports carry packets sent to local IP addresses or packets for which no routing entry could be found.

Another distinction is that ports may be classified as either "Push" or "Pull." With a "Push" port, data can be sent to or from a component immediately, but in the case of "Pull" ports, the recipient only receives a packet when it asks its neighboring port for a packet. This "Pull" behavior is designed to allow for various types of packet buffering within the router.

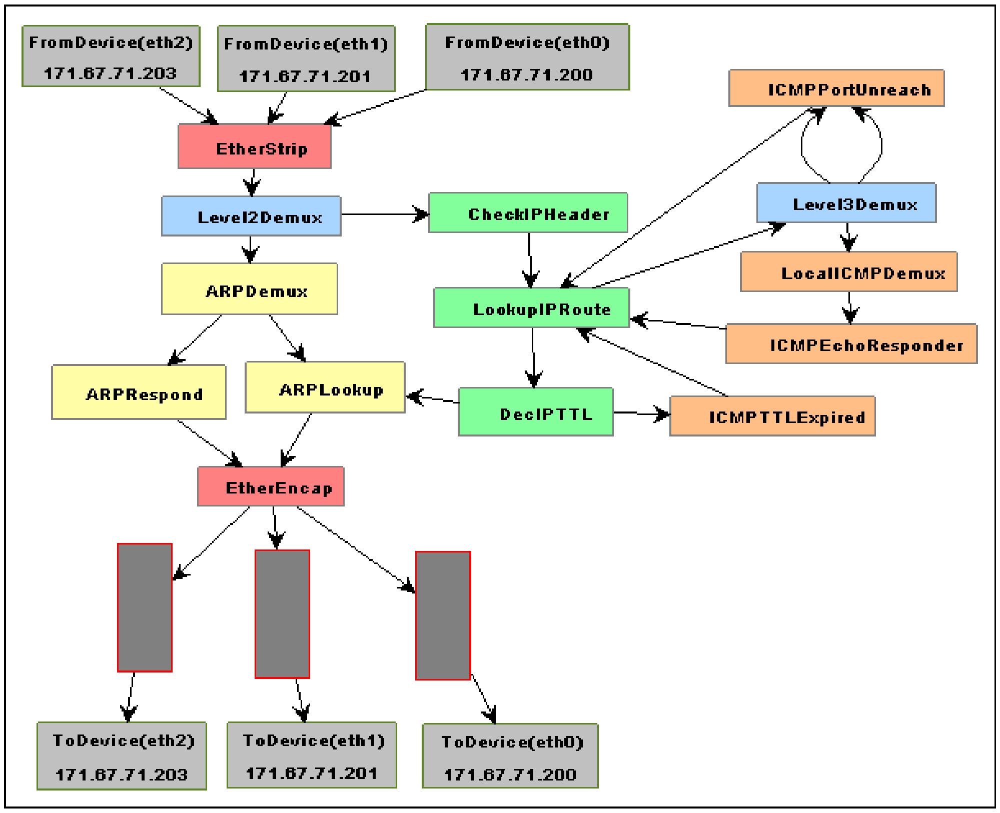


Figure 3.2: A simple configuration for a router with three interfaces. Packets enter at *FromDevice* components at the top and exit the router at *ToDevice* components near the bottom. Components processing at layers higher in the standard OSI model are positioned further to the right and components are logically grouped by color.

Component ports are connected via unidirectional connectors called "wires." Wires begin at an "Out" port and end at an "In" port and only ports of the same "Push" or "Pull" classification can be connected. Clack routers are simply a set of components connected by wires that implement the desired packet processing functionality.

3.2.2 Router Packet Processing

A router component initiates packet processing because of one of two possible events. First, a Clack client may recognize that the VNS server has sent it a packet destined to one of its virtual interfaces within the topology. In this case the main router processing loop pushes the packet to an input port on a *FromDevice* component representing the correct input interface. The packet may continue to be processed by being pushed out ports of successive components until it is discarded or reaches a queuing component. The second method for initiating packet processing allows components to register a call-back function with the router, which will be called once per processing loop. This "polling" behavior allows internal components to initiate packet processing by removing buffered packets from queues or generating new traffic.

3.2.3 Component and Router Implementation

Packets themselves are Java objects, inheriting from the generic *VNSPacket* class. Packet processing is simplified by the use of specialized protocol-specific packet classes with accessor functions for header information and data.

Each component is implemented as a single Java class that inherits from the abstract *ClackComponent* class, which provides the basic mechanism for interoperability among all components. Ports are implemented within the *ClackPort* class that handles the actual transfer of packets between components. Each component object has a collection of *ClackPort* objects representing the description and connectivity of each logical port.

All components with push input ports implement the *void acceptPacket(VNSPacket packet, int port)* function to act as the starting point for packet processing. Since each port number has semantics defined by the component writer, the method knows how to handle packets passed to any port. Packets are pushed out of a component using the *void pushOut(VNSPacket packet)* on a *ClackPort* object, which sends the packet to the connected component for further processing.

For a pull input, components call the *VNSPacket pullIn()* method of the *ClackPort* class, which queries the neighboring component connected on that port to see if it has a packet to pass on. Components with output pull ports provide the *VNSPacket acceptPullRequest()* method to answer these queries. In most cases, a *pullIn()* call is initiated by the call-back function registered with the router. For call-backs, components must register with the router and implement the void *poll()* method of the *ClackComponent* interface.

3.2.4 Current Router Functionality

To demonstrate the use of Clack, we have already 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 simplified but inter-operable TCP and UDP stacks, including a supporting socket interface that supports the creation of networking applications to run on top of these network layers.

We are still actively developing additional components and applications that bring new capabilities to Clack. See Appendix A for a complete table of currently implemented components.

3.2.5 Graphical Interface

The major contribution of the Clack work is the graphical representation of the modular routers described above, which includes the ability to observe, configure, build, and extend these routers. This section describes the basics of the Clack graphical interface design, with later sections going into greater depth about how different aspects of the Clack framework can be leveraged to help students improve their understanding of networks.

The Clack graphical application provides three distinct levels of views, traversed by the user using a zoom-in and zoom-out paradigm. The first is the network-wide view (see figure 3.3), which displays a topology of routers and hosts connected by links. Properties of these links can be modified, and links light-up to indicate traffic as it flows from the Internet and through the topology.

Users then choose a host on that topology to examine, zooming into the router view of Clack (see figure 3.2). The router view displays a graph of box-

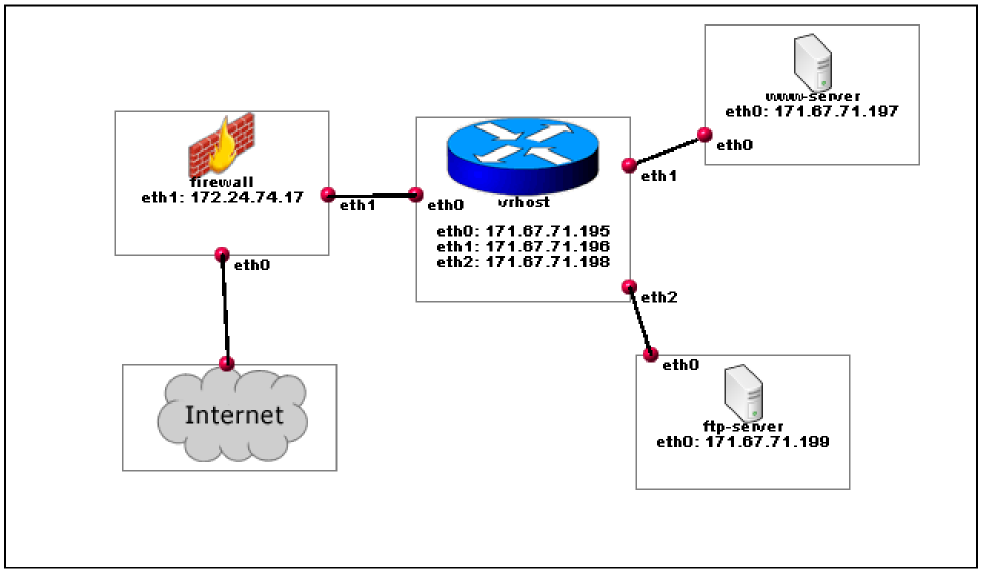


Figure 3.3: The network-wide view of Clack for a simple topology with one Clack router and two physical hosts.

like components connected by wires within a single window. As packets flow from one component to another, the wires carrying these packets light-up, giving a visual indication of the packet-processing flow through the router.

Certain components may be significantly complex such that it makes sense to allow Clack to zoom-in again so that the entire window can be taken up with information describing the state or internal processing behavior of this component. We call such instances hierarchical components and discuss them in greater depth in sections 4.5 and 6.3.1 .

Clack routers can either be built component by component or automatically loaded from configuration files. When empty, a router consists only of the *FromDevice* and *ToDevice* components that represent the router's interfaces as defined by the virtual network topology. Leveraging notions used in popular diagramming software, building a router consists of placing components within the router view and connecting specific ports to create packet-processing functionality. The ability to load from configuration files gives flexibility to educators using the tool for demonstrations or labs and assists students who do not want to continually build a router every time they open Clack.

The graph-drawing functionality of Clack is based on the open-source JGraph libraries[24]. JGraph offers a documented and debugged implementation base that supports complex graph rendering and manipulation. These well-documented libraries and interfaces will be of benefit to anyone looking to make significant graphical changes to Clack. However, as described in chapter 5, we have made significant efforts to provide programming interfaces so that nearly all users will be able to extend Clack without needing to learn about JGraph.

Chapter 4

Design Goal - Network Processing Transparency

With an understanding of Clack's basic design we now take a more detailed look at our two main design goals: transparency of network processing and flexibility in an instructor's ability to extend and configure the toolkit. In this chapter we look at the first goal of transparency. In pursuing this goal, we seek to provide educators with a variety of mechanisms to convey information to students in an intuitive way, while also making sure these features do not introduce unnecessary complexity for students if they are not used.

Our notion of transparency within the Clack toolkit has a dual focus. First, students must be able to tell "what" is happening within the router. That is, students should be able to recognize the flow of a packet or stream of packets through the router, following the packet until it reaches an output interface, is used to generate a reply, or is dropped. However, even more important for educational purposes is that students understand the "why" behind the router's action. That is, students must be able to connect that behavior with its underlying cause rooted in protocol or router state and decision-making logic. The following section introduces the variety of transparency mechanisms Clack provides to help students understand what is happening inside their routers.

4.1 Dynamic Components

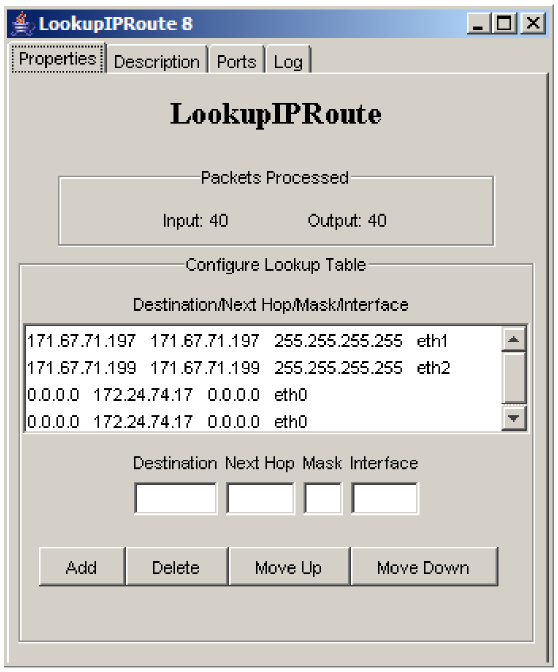
Graphically, most components are represented by simple boxes labeled with the component name. To promote a more intuitive understanding of the interrelations between different types of components and each component's position within the OSI layer model, components are colored depending on their membership in certain functional groups. For example, the three components that perform ARP functionality are in the same functional group.

However, 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 display occupancies updating in real-time (see figure 6.1).

4.2 Component Properties Views

The amount of data easily visible within the small router blocks, however, is insufficient for displaying detailed information about the component state and behavior. Similar to the pop-up properties windows commonly used in most operating systems to provide detailed information about a file, Clack provides "properties views" for each component. 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.

All component properties windows share a few standard features. One feature is a tab that provides descriptions of each of the component's ports and displays which component each port is currently connected to. Another standard feature is an embedded HTML page describing in detail the internal behavior of the component and the role of each of its ports. Properties views also display statistics about the component, including general statistics such as packet



Input: 40 Output: 40

```
171.67.71.197  171.67.71.197  255.255.255.255  eth1
171.67.71.199  171.67.71.199  255.255.255.255  eth2
0.0.0.0  172.24.74.17  0.0.0.0  eth0
0.0.0.0  172.24.74.17  0.0.0.0  eth0
```

Figure 4.1: An example property view: The view exposes the internal routing table of the *LookupIPRoute* component and allows users to modify component properties. The view also includes the standard description, ports and log tabs.

in/out counts as well as statistics unique to a component's specific behavior.

Property view windows are also important for exposing the current state of components, such as the ARP cache or an IP routing table. The ability to see and edit this state is critical to helping students understand why the router is making certain decisions. As a result, the properties windows also serve as the main mechanism for modifying the internal state of a component.

4.3 Component Logs

In some cases, it may be overly complex or sub-optimal to expose component behavior via graphical property windows. Instead, a textual log can be beneficial, particularly in the case that numeric precision is necessary or viewing an entire flow of events as a sequence is beneficial. For these purposes, we provide each component with a text logging console that is also accessible via the properties window. In a simple case, this output could explain to a user why the component has chosen to drop a packet, helping students diagnose an error in their router. In a more complex example, students could use the log of a *TCPRetranmitter* component to analyze the timeouts and retransmissions experienced by a particular TCP flow.

4.4 Pluggable Monitoring Components

Another mechanism for providing students with the ability to view router behavior is to design components with the sole purpose of analyzing the packets passed through them. As reporting media, these components can use any of the mechanisms described above, including as dynamic components, properties views, component console logs, a separate graphical window, or even an external file.

As our main monitoring component, we are creating a packet analyzer component with a graphical window that strives to build on student's familiarity with the commonly-used Ethereal tool. While this and many of our monitoring components are very general, the potential for significantly more specific monitoring elements exists. For example, we implemented a component called *TCPMonitor*, which is placed in-line with the router's IP forwarding path. This component keeps information about all TCP flows passing through the router and keeps a running estimate of each end-host's congestion window by measuring the amount of data each has outstanding. Combined with real-time graphing capabilities, this allows students to view different behaviors of real TCP streams, including slow-start and the congestion window's response to packet-loss.

4.5 Hierarchical Components

In some cases, the pursuit of transparency can lead to level of detail that causes an explosion in the number of different components inside the router window. As a means of providing detail without creating unnecessary complexity, Clack supports the creation of hierarchical components. Hierarchical components expose a component's internal state and decision-making at a degree of detail degree of detail significantly higher than is possible within a dynamic component or properties view. Hierarchical components may include graphs of components, similar to a full Clack router, or may simply provide more information and configuration capabilities to the user. Clack's TCP-stack is implemented as a hierarchical component and is described in section 6.3.1.

Chapter 5

Design Goal - Extensibility and Configurability

This chapter considers the second design priority: creating a flexible graphical network toolkit that provides instructors with an extensible and easily configurable teaching platform. This goal recognizes that the value of an educational tool depends significantly on the ease with which instructors can mold the tool to their specific needs. This necessitates not only a router and application platform that is easy to configure and extend, but also network characteristics and traffic loads that instructors can tailor to fit their specific teaching needs. Additionally, we consider Clack's capabilities to support serializing and replicating router and network configurations to promote ease-of-use.

5.1 Extending Current Packet Processing or Analysis Functionality

Adding new processing and analysis functionality to Clack will be important in applying the Clack toolkit to different areas within the field of networking education.

5.1.1 Adding New Router Components

We hope that networking educators will have the desire to create new router components either for the purposes of implementing packet-processing functionality or offering an improved ability to modify or inspect various properties of the router. In designing Clack, we focused on making this task simple enough that students could be asked to implement a component as part of an assignment.

New components are created by sub-classing the *ClackComponent* class, which transparently provides everything needed to support plugging into a graphical Clack router. Component implementers are left with only two responsibilities: defining the port interfaces and performing packet processing (see appendix B.1 for an example component implementation). Clack is designed so that new components plugged into a router easily integrate with the graphical capabilities that the toolkit provides. For example, every component automatically has a properties window that provides default packet in/out statistics, information about the components ports and connections, and its console log. Component creators can easily write to their console log using the *void log(String s)* function.

5.1.2 Extending a Router with Additional Graphical Features

While the introduction of new components is very simple for the common case, advanced users are not limited in their ability to create very highly sophisticated graphical components. Extenders familiar with Java's graphical Swing library can easily augment the default properties window class to provide more detailed information about component statistics or internal state, or can even create a hierarchical component. Creating a dynamic component that redraws itself in real-time is similarly a straight-forward exercise in Swing programming, since Clack allows each component to easily replace the default component renderer with its own class. The ability to add dynamically updating graphical functionality is supported by a simple listener architecture that allows components to inform interested graphical entities when packet-level events occur.

5.1.3 Leveraging Pre-built Java Libraries

A significant advantage to developing Clack in Java is the wealth of pre-built and freely available libraries available to provide additional capabilities when building monitoring and hierarchical components or adding features to properties windows. In our basic implementation, we frequently leverage the JFreeChart library[25] to provide real-time graphs of features like TCP Windows size or queue occupancy. Similarly, the use of the JPCAP libraries[26] resulted in a very simple implementation of packet capturing functionality.

5.2 Configuring Network Characteristics and Traffic Loads

The ease with which an instructor can configure the network environment, including link characteristics, neighboring hosts, and traffic loads, directly impacts the overall flexibility and value of Clack as a network teaching tool.

5.2.1 Using Components to Simulate Link Characteristics and Network Irregularities

Network link considerations, such as relative link speed, packet-loss, and packet reordering can be simulated through the use of router components. Being able to tweak these components in real-time and show the reaction of protocols is an important teaching mechanism. For example, we use a *Delay* component to simulate a bottle-neck link when demonstrating the interplay between queue occupancy and TCP congestion control.

5.2.2 Clack Traffic Generation Mechanisms

The Clack toolkit provides instructors with several options for generating traffic to be processed by Clack routers. Leveraging the connection of topologies to the Internet, traffic can flow either between an outside machine and a host within a Clack virtual topology, or simply between to hosts inside a topology. We outline four mechanisms for traffic generation below.

Leveraging 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 the Clack client. While routers can respond to these limited types of messages, significant amounts of 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 HTTP or FTP servers into virtual topologies, giving a simple means of providing TCP download flows through the router.

Adding Application-level Servers and Clients

Since Clack provides a 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 are implemented as Java threads and have access to a simplified socket interface for sending and receiving traffic via the router's network stack (see Appendix B.2 for an example application).

While it is certainly possible to have students themselves program applications to run on top of their routers, our primary desire in creating this capability was to give instructors a flexible mechanism for generating a variety of traffic loads. For example, we have currently implemented several applications, including an HTTP-GET requesting agent and a very simple web server as a means of generating traffic for users of our TCP hierarchical component.

Since Clack applications are full-fledged Java threads the potential for highly sophisticated applications exists and any number of conceivable host types could be created. For example, to demonstrate the effects of UDP's lack of congestion control, a UDP and TCP flow could be sent through a single router configured to analyze the streams.

Leveraging Application-Level Redirectors

While a great deal of potential exists for writing applications to run on top of Clack host network stacks, the work involved with developing and debugging such applications could be significant, especially in the case of a network service that runs a complex protocol. Because of this, Clack provides the ability to easily install a traffic "redirector" on a router, which transparently proxies traffic to and from another Internet host that provides the actual application functionality.

This scheme is implemented using Clack applications running atop TCP or UDP sockets that act as proxies, communicating with the remote host via Java sockets that operate outside of the Clack virtual network. Redirectors are installed by specifying a five-tuple consisting of *(transport protocol, Clack router IPaddress, Clack router port, proxied host IP address, proxied host port)* . This capability can be very useful in leveraging pre-built server applications, making it appear that a type of server exists inside a Clack topology. Section 6.4.2 outlines the use of a redirector to proxy a DNS server for the purposes of performing DNS hijacking within a Clack router.

Programmatic Traffic Generation

The previous three mechanisms all utilized application level traffic generation, either with real Internet applications or those running on Clack sockets. While these options will be sufficient for most needs, they may prove to be either to cumbersome or not significantly fine-grained for some traffic generation goals. For these reason, those using Clack also have the option of directly creating traffic with the packet-level classes used by router components. We call this option "programmatic traffic generation."

Using this type of traffic generation can simply be a matter of convenience: it may be simpler to create components that generate a certain traffic characteristic than it is to create an equivalent application. However, programmatic generation can also give the ability to create malicious traffic content, patterns or timing not able to be created via a correctly functioning socket interface and network stack. In the case of a network security course, one could imagine generating both legitimate and malicious traffic that must be monitored by a student-programmed intrusion detection system.

Furthermore, Clack provides the ability to use programmatic packet creation to control and analyze all data entering or leaving a single router, allowing the crafting of highly tuned traffic behavior for use in demonstrations, labs, or router tests.

5.3 Automatic Saving and Loading of Router Configurations

This and past chapters demonstrate the significant configuration supported by Clack, both at the network and individual router level. A significant asset of the Clack Toolkit is its ability to provide an automated mechanism for serializing both router and network configurations, so they can be reloaded on demand and even deployed across hundreds of identical topologies for individual user by students.

5.3.1 Serializing Router Configurations

After a router has been built, Clack allows the user to save its configuration in an XML file so the identical router can be loaded at a later time. Saving a router configuration records the existence of each component within the router, as well as its port connections to other components. Beyond saving the type, graphical location and interconnections between components, router serialization also allows components to save configuration information that will be needed when the components are recreated. This could include the size of a router buffer or the rule-sets for a firewall component. However, some component state, such as routing tables, may be specific to a given virtual network and would render this configuration file useless for other network topologies. For these reason, saving a router does not serialize this transient state unless specifically requested by the user.

5.3.2 Configuring Entire Network Topologies

The configuration of an entire Clack network topology, including surrounding hosts and applications running on all routers can all be set from a single file. This master configuration file can specify a router configuration to be installed on each host in a Clack topology, and select whether that router will be displayed in a graphical window for the user or be run in the background as a supporting router. These supporting routers can also be automatically configured to generate traffic through the use of Clack applications or programmatic traffic generation.

Chapter 6

Educational Uses of the Clack Router Toolkit

Having covered the main design goals of Clack in the previous two chapters, this portion of the thesis gives the reader a concrete description of how Clack can add educational value as a classroom demonstration device, hands-on learning tool, and network-programming platform. These descriptions aim to give instructors an idea of the potential of Clack and, as such, some of these examples seek to teach fairly sophisticated networking concepts. However, we feel strongly that the toolkit can be utilized to benefit a variety of students. With a few of these examples, we also hope to show the versatility of Clack by demonstrating concepts often taught in network security courses.

6.1 In-class Demonstration: Queue Dynamics

Analyzing the queuing discipline employed by a router's packet buffers and its interaction with transport protocols helps students understand root causes for latency and packet loss within a network. This section describes an in-class demonstration of these concepts, assuming that the instructor is running Clack on a demonstration computer hooked-up to a projector in the classroom.

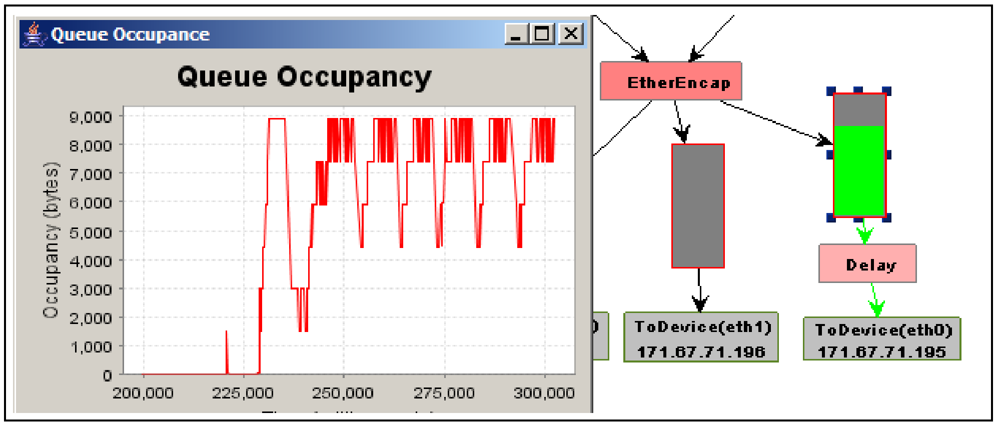


Figure 6.1: Clack's queues dynamically update as they fill with packets. Next to the queue is a real-time graph of queue occupancy over time.

The most basic type of queuing supported by Clack is a simple drop-tail queue with static size. In a simple set-up, the instructor can access a large file hosted on a web-server on the same network as the Clack router, causing the packets to flow through this router en-route to the client.

The instructor can begin by showing students the network topology containing the Clack router and web-server, in order to give students a context for understanding how the interfaces within the Clack router fit into the larger picture of the network. The instructor can then zoom into the individual router, and students will see wires along the basic IP-forwarding path of the router light up as packets flow to and from the web-server. By placing a delay block in front of the out-bound queue of data packets, the instructor can ensure that this egress is the bottleneck for packets being sent from the web-server back to the demonstration computer, meaning that the buffer will fill and overflow at this point. Students can see the queue slowly fill up in real-time until the queue flashes red, indicating that a packet has been dropped. This drop results in the queue's occupancy lowering significantly, before building up again.

To explain the dynamics of the TCP traffic flow, the instructor can place a *TCPMonitor* component in the forwarding path, which estimates the window sizes of TCP flows going through the router by keeping track of the total number of bytes outstanding in a flow. Both the properties windows of the queue and the TCPMonitor support real-time graphing, vividly demonstrating the saw-tooth behavior of TCP congestion control and its relation to queue occupancy.

6.2 Guided Tutorials: Building a Router

We believe that guided tutorials can be effective tools in helping students take a more hands-on approach to working with Clack. Tutorials could be used to reinforce concepts introduced in lecture or textbook reading or accompanied with questions and integrated into homework assignments. This tutorial on building a simple IP router leverages the fact that Clack can scale to provide each student with their own router and topology to work and learn that whatever pace suits them best.

Our example tutorial uses the modular design of Clack to help students develop an understanding about how a router works as they gradually build and test their router block by block. When loading their router, students will see the simple network introduced in figure 3.3. However, pinging either of the servers will not work, since they have not built their intermediate router yet. When they zoom into their router, they are presented with a router graph that is empty except for the input and output interfaces. Hooking the packet analyzer up to eth0 allows them to see the traffic entering their router, which is an ARP request from the VNS firewall. The tutorial then informs students about the operation of ARP and has them add the components necessary to strip an Ethernet header, handle ARP operations, encapsulate a packet in Ethernet, and put it in an output queue on the correct interface. As students add components, they can examine ports and HTML descriptions inside the properties windows to get a clear idea of the exact behavior of each component.

With ARP implemented, pinging a connected web-server will now result in dropped IP packets from the component de-multiplexing ARP and IP packets. Students can then be guided through the creation of an IP-forwarding path and the correct configuration of routing tables. Now able to ping the connected servers, students can also examine the ARP cache component to see that it has updated with replies from neighboring interfaces. This tutorial process can continue, with students enabling their routers to respond to local ICMP echo requests and send ICMP port unreachable messages, thereby allowing students to analyze the implementation of the ping and traceroute commands commonly used in networking courses.

We envision an online repository of Clack tutorials and accompanying configurations that will allow instructors to easily leverage the efforts of others in the networking educational community.

6.3 Hands-on Network Laboratory: Analyzing Protocols and Configuring Devices

Because of the significant and growing interest to incorporate a laboratory component into networking courses, this section provides several examples of how Clack can provide a low-overhead and highly flexible solution for such scenarios.

6.3.1 TCP Inspection

A common network laboratory assignment has students examine TCP through the use of a packet analyzer like Ethereal. Clack could support similar exercises through the use of its own packet analyzer, but also offers the capability to go much deeper in its analysis through the use of the *TCPMonitor* component and the hierarchical component built to demonstrate the internal operation of the Clack TCP stack. In this example, we focus on describing the capabilities of the hierarchical TCP component.

Double-clicking on a router's TCP component zooms into a full-window view of a hierarchical component describing the operation of the router's TCP stack. Since viewing TCP behavior requires an application running on the router to use a socket, the view includes the ability to launch any application registered with Clack. All current TCP flows terminated at the router appear as an entry in a graphical list, labeled by their local and remote addresses and ports.

When one of these entries is selected, the remainder of the hierarchical component updates to display the current state and behavior of this particular flow. This includes a sub-graph of interconnected components, similar to a full Clack router, with each component implementing a portion of TCP behavior (A list of these components is in Appendix A.4). As in the main router graph, packet passed between these internal TCP components can be visualized by flashing wires as data travels to and from the *TCPSocket* component. Additionally a "TCP-Dashboard" updates dynamically to display a wealth of information related to the flow's internal state and behavior. The dashboard depicts statistical information like average and instant data-rate, estimated round-trip-time, as well as current Transmission Control Black(TCB) data like the state of the TCP connection, contents of its retransmission list, and the congestion window size.

This ability to look at TCB state and flow processing goes well beyond the capabilities offered by simple packet analysis with a tool like Ethereal. In addition, the ability to adjust parameters like the retransmission time-out or congestion control behavior provides a more interactive mechanism to explore these topics compared to other tools that use real Internet traffic.

6.3.2 Analyzing the OSPF Routing Protocol

In the previous case, we examined how Clack is capable of providing better insight into a protocol that students could have analyzed from any networked computer. In this example we look at how Clack's ability to create and visualize more complex network topologies can be utilized to explore concepts such as routing protocols that are usually hidden at the core of large networks.

While we do not currently have OSPF implemented for Clack, a simplified but inter-operable version of the protocol has been implemented within a VNS client router written in C. Thus, we feel it would be reasonable to port a simplified OSPF to Clack, accompanied by a graphical component to expose internal state.

The network view of Clack gives users the ability to break links within their network topology, and see how the routing protocol responds. Students can view these changes from multiple levels. First, with a packet analyzer they can see the actual protocol messages as they traverse the wires, helping students get a concrete idea of what data is transferred by the routing protocols and what events prompt this communication. Secondly, students would also be able to view the internal state of the protocol via a hierarchical component and gain additional insight into the algorithm and how it populates routing tables.

6.3.3 Configuring a Firewall

In a laboratory exercise aimed at teaching students about network security, a Clack firewall component could be developed and used in combination with the packet analyzer to identify and block malicious traffic.

In this experiment, malicious traffic could be worm payloads, network discovery probes, or denial-of-service activity and could be generated and mixed with legitimate traffic in several ways. Instructional staff could create Clack applications or the programmatic traffic generation capabilities outlined in section 5.2.2 to generate different types of malicious payloads. In an alternative and more complex setup, this traffic could be generated by a tool, like nmap[27] or hping[28], operating outside VNS and controlled by students testing their own set-ups.

After creating a default firewall setup configured with a well-guarded end-host subnet and a more open DMZ containing web and other servers, students would utilize the packet analyzer to identify the different types of malicious traffic still reaching their hosts and create rules to block it without impeding legitimate traffic. Because Clack presents a virtual setting, students can safely fire-walk and attempt to evade the firewall rules using security tools in an exercise to educate students about both sides of the network security game.

6.4 Heuristic Network Programming

Adding new network processing functionality to Clack is greatly simplified by its modular design which allows instructors to give students well-encapsulated projects that plug into a working router. Additionally, the use of Java further simplifies the task by offering students a language and associated libraries that they are already likely to be familiar with from previous course-work. While Clack does not provide a realistic environment for those looking to gain systems-level development experience, it does provide a strong pedagogical mechanism to help students conceptualize the behavior of network devices through actual implementation. In this way, Clack has an aim similar other heuristic programming platforms like Nachos, albeit with a desired lower barrier of entry resulting from a simpler programming interface. In this section we outline two different programming assignments implemented within Clack. Both consist of a single Clack component containing relatively few lines of actual code to demonstrate the simplicity with which a student can insert new functionality into a Clack router.

6.4.1 Creating a Random Early Detection (RED) Queue

The design and benefits of a RED queuing policy are a topic discussed in many introductory networking courses. In this example, we have students program a RED queue and then analyze its behavior through the Clack GUI. Having described the operation of a RED queue in lecture, an instructor can provide students with the shell of *REDQueue* component, with the Clack boilerplate code and interface already defined, leaving only the actual queue and drop functionality to be implemented by students. An example implementation is given in Appendix B.1, with an estimated 25 lines of code to be implemented by students.

As is, the queue would be ready to plug into the Clack GUI, but ideally the students would be given a pre-written component property window that allows them to modify the many RED parameters dynamically and see operation of the queue in relation to these chosen values. The properties window requires no work on the part of the student once they have implemented the interface getter and setter methods of the *REDQueue* component.

Following the implementation, students can test their routers in Clack and answer a variety of questions posed to them concerning the behavior of the *REDQueue* and its behavior compared to Clack's standard *ByteQueue*. Clack traffic generation mechanisms can be used to put specified traffic loads across the queue on demand, demonstrating the value of RED as well as its weakness in handling traffic, such as UDP flows, that does not reduce its rate in response to packet-loss.

6.4.2 Security Lesson: DNS Request Hijacking

Recognizing the significant control exerted by network infrastructure is an important lesson for network security students, yet it is rarely able to be demonstrated because of the expensive and closed nature of common routers. This example has students implement a DNS-hijacking attack, in which a router meant to forward a UDP DNS packet maliciously drops the packet and replies to the querying host with a DNS-reply containing a false address. This programming assignment effectively demonstrates the capability of routers to perform an active attack due to the lack of a cryptographically secure mechanism like DNSSEC.

Configuring a network topology for this assignment requires two virtual hosts: one is the malicious router, operated by the student while the other is a virtual host with a UDP port 53 redirector to a real DNS server, resulting in a DNS-server inside the topology that is reachable only via the malicious router. Students can easily verify their work by plugging their DNS-hijacking component into the forwarding path of their router and using a command-line DNS utility like "dig" or "nslookup" to specify the resolving DNS server. For a more effective demonstration, instructors could have students temporarily change their DNS server to the topology host, and then use a web-browser to surf to a web site address that will be poisoned by the router.

Both of the programming examples presented here are quite simple, with the goal of demonstrating that protocol-level programming with Clack need not be complex or require a significant programming background. An advantage of the component-based design is that students can easily be provided with a specific subset of router functionality and asked to implement the remainder. The addition of more complex components, implementing functionality like NAT, intrusion detection, or weighted-fair-queuing, would also be reasonable for students with previous exposure to Java programming.

Chapter 7

Design Constraints, Limitations, and Future Work

In the final chapter of this thesis, we wish to discuss some of the constraints and limitations we faced while designing Clack, with a particular focus on how these factors suggest future areas of improvement for the Clack toolkit. This chapter first examines the challenges of providing beginning networking students with useful feedback to facilitate the correct construction and configuration of routers. Additionally we look at difficulties that arise as side-effects of two major benefits offered by Clack: its access to real Internet traffic and its ease-of-use resulting from remote hosting.

7.1 Diagnosis and Feedback for Novice Networking Students

The expectation that Clack will be used by students with little background in networking presents another challenge to create mechanisms that provide error-checking and trouble-shooting feedback to students building, configuring or testing routers. While the major design goal of transparency discussed in chapter 4 helps in this capacity, it will not necessarily be able to help students who do not even know where to begin debugging.

7.1.1 Static Typing of Component Ports

Incorrectly connecting components is one of the most obvious pitfalls when building a router. When connecting two ports, Clack provides checks to make sure that port direction and method correctly match, or the connection is disallowed. To go beyond these simple checks, Clack also performs a static type-check algorithm on the ports with the goal of ferreting out difficult-to-detect bugs in the interconnection of components. Each port is typed according to the actual packet classes Clack uses to pass data between components. Using the notion of sub-types, a port of type *VNSPacket* could, for example, packets of any class that is a sub-type of *VNSPacket*, but a port of type *VNSTCPPacket* could only handle that specific packet type. Most ports on packet processing components can be strongly typed to a specific packet type, but in the case of generic components like queues or monitors, ports can assume only the generic *VNSPacket* type.

When checking for errors, Clack ensures that no port will ever handle a packet it is not typed to process. For many connections, this is a simple matter of checking that the source port is a sub-type of the target port. However, analogous to the need to cast objects that are removed from generic collections in a language like Java, some connections will have a source port of a more generic type than the target port. In this case the algorithm works backwards from the input ports of the source component to find all possible packet types that could arrive at that port according to the type system. If any of these types lead to a violation the connection is not allowed and the location, type conflict and root cause of the error are reported to the user (see figure 7.1).

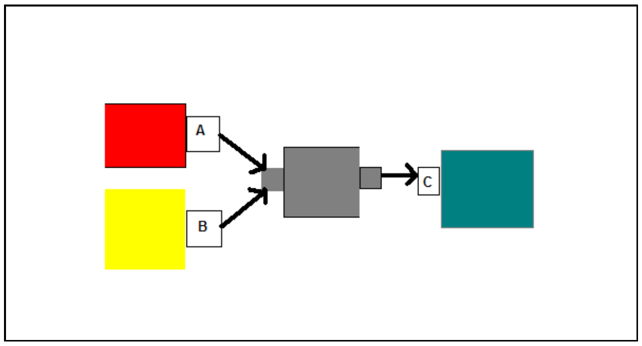


Figure 7.1: An example of static port checking. If the center component has two generically typed ports (grey), then we must check the components from which it receives packets. In this case, types A and B must be a sub-type of C for the router to be correctly typed.

7.1.2 Component Flow and Configuration Checks

Static router analysis can also be useful in performing basic facts checks about how the user has configured their router. One type of analysis in this realm seeks to verify basic information about the reachability of different components in the router graph. For example, if a router is supposed to implement IP-forwarding with the LookupIPRoute component, a simple test can verify that there exists a path between each pair of input and output interfaces that includes the LookupIPRoute component. These types of flow path checks can be more detailed to the point of verifying the entire contents of the router.

Another angle to verify routers involves querying the state of components to assure correct configuration. Examining the routing table of the LookupIPRoute component to confirm that routes for all interfaces exist, and that a default route also exists demonstrates this tactic.

Importantly, failed tests can be accompanied with detailed explanations and suggestions that are capable of giving the user significantly more specific feedback about how to fix the problem than when a user only knows that certain network tests, like a ”ping”, fail.

7.1.3 Component Error Logs

When a component reaches an error condition in their packet processing, it can provide information to its console log with the *void error(String s)* function call. The difference between this mechanism and simply writing to the log is that calling *error()* causes the component to flash red on the screen, informing the user to consult the log view a description of the error. Ports use this same reporting mechanism if a component's use of the port deviates from the behavior specified by its direction, method or packet type.

7.1.4 Programmatic Traffic Generation Tests

Another mechanism for detecting errors within a router is to leverage programmatic traffic generation, described in section 5.2.2 to test a router's reaction to actual packet inputs. This approach is more complex for the test-writer than the previous methods described, but its power and flexibility can result in both better diagnosis or the problem and more useful error reporting to the student. This mechanism allows a test-writer to inject arbitrary Ethernet packets into any of the router's interfaces and receive the packets the router sends from its interfaces.

A simple test may have as its goal to verify that a router supports all of the necessary functionality for a user to ping it. This test would first send the router an ARP request on its Internet facing interface, and would look to receive a correct reply on that same interface. This would be followed by the ICMP echo request, with the test verifying that a reply is sent. Such tests are often simple to write using a state-machine paradigm. These tests can be a major asset to students because they offer the opportunity to explain why a test failed and suggest corrective action.

7.2 Negative Impacts of Using Internet Traffic

A Clack router's ability to integrate with real systems on the Internet provides many benefits to users of the toolkit, but also presents several challenges for an application designed to be used with students new to the topic of networking.

7.2.1 Speed of Clack Packet Processing

First, while a Clack router processes packets orders of magnitude slower than an operational router, the speed of live network traffic through Clack can still be too fast to be followed visually, potentially leaving beginning networking students unable to recognize the "what" and "why" of the many packet processing decisions made by the router.

Recognizing that different processing speeds would be desirable for the variety of educational uses we aim to support, we created a mechanism for slowing down the router by specifying a wait time between each component's processing of a packet. This feature is highly desirable if students wish to get a better understanding of exactly how the packets are flowing through the router, or if they need other graphical components displaying state or statistical information to update more slowly.

However, the use of this mechanism with real Internet systems introduces a significant complication: network timeouts. Slowing down traffic processing significantly may cause remote systems communicating with or through the router to assume their packets have been dropped, causing the system to either fail or retransmit. As a result, the speed of processing affects the actual behavior of the network traffic being observed, often leading to undesirable effects.

This effect represents a natural trade-off between processing real-time Internet traffic with VNS and simulating packet processing within a closed system on a single machine. In considering future work, we plan to develop a system capable of giving users full control over the speed of packet processing within the router, without the threat of network timeouts. We envision a type of graphical network debugger which allows users to control the speed and even stop, start and step through router packet processing.

One approach we plan on exploring is to remove the dependency on external Internet systems so that all behavior can be slowed down simultaneously. Instead of having external systems sending traffic to a Clack router via the Internet, traffic would be generated either by Clack applications or any of the mechanisms outlined in section 5.2.2. In this scenario, Clack would subject all routers to a similar notion of time, so that slowing or stopping one Clack router would affect all other systems on the network similarly. This possibility is discussed in greater detail below in section 7.3.2.

7.2.2 Complexity of Dealing with Internet Protocols

Another concern resulting from the use of real-network traffic is the threat of exposing too much of the complexity inherent in real networks and routers to student with relatively little networking background. For example, a Clack router must consider issues like network timeouts or recalculating the IP-header checksum when forwarding packets. Compared to a visualization or simulation designed solely to illustrate a specific network concept, presenting the student with an entire router may result in an overload of information that obscures the specific teaching goal.

Clack's modular design was developed with this concern in mind, giving component creators the ability to expose or abstract as much detail as they wish. We incorporated the concept of hierarchical components to allow a single high-level component to represent many smaller components. While we currently use this only for TCP, it could also be used to provide a single component for entire classes of components such as ICMP, ARP or IP-layer processing.

This approach, however, is limited in its flexibility. For example, an instructor who wants to hide certain functionality not already coalesced into a single component must write code and redistribute the Java archive (JAR) file to students. In later iterations of Clack, we are considering an automatic "wrapping" mechanism that will graphically allow users to dynamically create hierarchical components. Users could zoom-in on these hierarchical components to reveal a sub-graph of components if desired, but would be presented with less router complexity within the main router view.

7.3 Remote Hosting of VNS

The hosting of VNS at Stanford frees other schools from having to supply their own network lab infrastructure, but this reliance on a remote system also gives rise to concerns about network connection latency and its impact on the behavior of a Clack router.

7.3.1 Dealing with Latency between Client and VNS Servers

The remote hosting of VNS may cause concerns for two reasons. First, latency due to geographic distance between the Clack application and the VNS-server may lead to varied behavior depending on the geography of the user. In the example of pinging a Clack router from a user at an east-coast university, the packet will cross the country four times, incurring a propagation latency of approximately 200 ms. This time is in addition to already significant processing or queuing time within a Clack router, which ranges from about 100 milliseconds to longer times if the user has chosen to slow their router processing. Importantly, this problem is exacerbated when traversing a topology includes multiple router hops, all of which result in a round-trip between the host running the Clack router and the VNS-server. As a result, certain applications may incur timeouts or other error behavior on a more frequent basis. These latencies also create large round-trip times that limit the throughput of TCP streams through the routers.

A second concern about latency is that significant variations in delay on the wide area link between the client computer and Stanford could lead to different results in Clack due to subtle and unpredictable timing issues. This is less likely at locations connected to Stanford via Internet2, but may be a concern for other universities with a less reliable connection.

7.3.2 Potential for Non-VNS Router Topologies

Concerns about the latency of links between a Clack router client and the VNS-server, along with the inability to adequately slow network traffic mentioned in 7.2.1, suggest that there may be cause for Clack to support running network topologies internally, thereby entirely removing any dependency on VNS. This approach adds capabilities to Clack that move it closer to the simulation and visualization realm, while still allowing users to leverage networking and graphical portions of the Clack code base.

The main disadvantage of such an approach is obvious: a router running in one of these topologies would be unable to be contacted over the Internet. Yet the predictability, elimination of latency issues, and ability to control packet processing speed may be worth this trade-off in some cases, particularly when Clack can sufficiently create the desired traffic loads through the use of applications, redirection, or programmatic generation.

We are also considering a hybrid system, in which a Clack topology is still connected to the Internet, yet tests to see if the destination of a packet is running locally before sending a packet back to the VNS-server.

7.4 Additional Future Work

The creation of VNS and the more recent development of the Clack Router Toolkit is part of a larger effort to create a center for teaching Internet infrastructure at Stanford. As such, both VNS and the Clack toolkit are still under active development and we are preparing Clack for its initial 1.0 release, after which we hope to work with several networking courses to gain more insight into how Clack can best assist instructors.

During this time we plan to perform substantial user testing to identify stumbling blocks for users new to Clack. In performing this user testing, we recognize two main types of user: course staff and students. The former group can be assumed to have significant networking experience and must be able to create demonstrations and assignments with as little VNS or Clack-specific overhead as possible. For the student group, less network experience can be assumed and all details of the underlying systems must be hidden.

We recognize that Clack will only achieve its potential if it becomes a tool both actively used and improved by a community of networking educators. With this in mind, in the following months we plan to open-source the Clack toolkit and create and support an online information portal and public repository for the collection of components, graphical add-ons, demonstrations, tutorials, and assignments related to Clack and VNS. As a first step we plan to create web-accessible demonstrations with Clack tied to sections in major networking text-books.

Chapter 8

Conclusion

This thesis describes the first iteration of the Clack Graphical Router Toolkit, highlighting its design goals of network processing transparency and flexible use through simple yet powerful extensibility and configuration. Built atop VNS, we feel that Clack's use of live network traffic and a compelling and intuitive graphical interface will make it an effective tool for in-class demonstrations, hands-on lab projects and programming assignments in networking education.

Appendix A

Current Clack Component Index

A.1 Basic Router Components

name	**description**
FromDevice	Reading incoming packet from a network interface
EtherStrip	Removes Ethernet header from a frame
Level2Demux	Demultiplexes incoming ARP and IP packets
ARPDemux	Demultiplexes incoming ARP requests and replies
ARPLookup	Contains ARP cache for hardware address lookups
ARPRespond	Responds to incoming ARP requests for local interfaces
CheckIPHeader	Verifies the contents of an IP Header
LookupIPRouter	Performs longest-prefix-match lookup in an IP routing table
DecIPTTL	Attempts to decrement the Time-To-Live field in the IP header
Level3Demux	Strips the IP header and demultiplexes TCP, UDP and ICMP packets
LocalICMPDemux	Demultiplexes local ICMP packets based on Type field
ICMPEchoResponder	Responds to local ICMP echo requests
ICMPTTLExpired	Generates ICMP TTL-expired packets
ByteQueue	A simple drop-tail FIFO queue measuring its size in bytes
ToDevice	Sends outgoing packets to a network interface

A.2 Advanced Functionality Components

name	**description**
REDQueue	Packet buffer implementing RED algorithm
TCP	TCP-stack and socket implementation
UDP	UDP-stack and socket implementation
UDPSource	Generates UDP packets at a specified rate
DNSHijacker	Injects false DNS responses for DNS requests being forwarded

A.3 Control and Auxiliary Components

name	**description**
Capture	Writes packets passed through it to a file in PCAP format
Counter	A dynamic component that counts packets passed through it
Delay	Slows packets passed through it, creating a bottleneck link
Loss	Allows a user to specify when the next packet through it should be dropped
TCPMonitor	Analyzes TCP-flows being forwarded through the router
Tee	Duplicates a packet stream and sends it out two output ports

A.4 TCP Components

The TCP component is a hierarchical component with sub-components implementing different functionality.

name	**description**
OrderPackets	Manages the sequencing of incoming TCP segments
ProcessAck	Updates the TCB state concerning what data has been acknowledged
ProcessSegment	Handles incoming data segments and sends them to the socket
ReceiveWindowCheck	Enforces the receive window on incoming packets
Retransmitter	Stores packets for retransmission and issues timeouts
SendWindowCheck	Manages the congestion window and coordinates the sending of data
SetChecksum	Sets checksum on outgoing data segments
SockBuffer	Used to buffer incoming and outgoing data in the socket
TCPSocket	Interfaces with the application to handle data sending and receiving data
VerifyChecksum	Validates the checksum for incoming TCP segments

Appendix B

Clack Code Examples

B.1 Clack Components Example: REDQueue

Below is the code for a simple component that implements the functionality of a RED queue. To take advantage of the Clack GUI, students could be provided with a GUI class that sets queue parameters via the component's properties window.

```
public class REDQueue extends ClackComponent {

   public static final int PORT_HEAD = 0, PORT_TAIL = 1, NUM_PORTS = 2;
   private double minThresh, maxThresh, maxProb, avgLen, curLen, maxLen, alpha;
   private ArrayList packets;
   private Random rand;

   public REDQueue(Router router, String name){
       this(router, name, 10000, 5000, 8000, .7, .4); // defaults
   }

   public REDQueue(Router router, String name, double mLen,
         double minTh, double maxTh, double mProb,  double a){
       super(router, name);
       minThresh = minTh;
       maxThresh = maxTh;
       maxProb = mProb;
       maxLen = mLen;
       alpha = a;
       avgLen = curLen = 0;
       packets = new ArrayList();
       rand = new Random();
       rand.setSeed(System.currentTimeMillis());
```

```
          setupPorts(NUM_PORTS);
    }

    public void setupPorts(int numPorts){
         super.setupPorts(numPorts);
         m_ports[PORT_HEAD] = new ClackPort(this, PORT_HEAD, "head of RED Queue",
                 ClackPort.METHOD_PULL, ClackPort.DIR_OUT, VNSPacket.class);
         m_ports[PORT_TAIL]     = new ClackPort(this, PORT_TAIL,"tail of RED Queue",
                 ClackPort.METHOD_PUSH, ClackPort.DIR_IN, VNSPacket.class);
     }

    public void acceptPacket(VNSPacket packet, int port_number){
        int size = packet.getByteBuffer().capacity();

        if(curLen + size > maxLen) {return; // must drop}
        if(avgLen > maxThresh) { return; // Threshold drop }
        if(avgLen > minThresh && probDropTest()) {return; // Early RED drop}

        packets.add(packet);
        curLen += size;
        recalculateAvgLen();
     }

     public VNSPacket handlePullRequest(int port_number) {
        if(packets.size() > 0){
            VNSPacket packet = (VNSPacket)packets.remove(0);
            curLen -= packet.getByteBuffer().capacity();
            recalculateAvgLen();
            return packet;
         }
         return null;
      }

     private void recalculateAvgLen() { avgLen = (1 - alpha) * avgLen + alpha * curLen; }

     private boolean probDropTest() {
        double slope = (maxProb) / (maxThresh - minThresh);
        double probability = (avgLen - minThresh) * slope;
        return (probability > rand.nextDouble());
     }

    public boolean isModifying() { return false; }

}
```

B.2 Clack Application Example: Mini Webserver

This examples demonstrates a very simple Clack application that acts as a webserver using Clack's TCP-stack. The webserver replies with the same content regardless of what a connecting client sends.

```
public class MiniWebServer extends ClackApplication {

   public void application_main(String args[]) {
     try {
         InetAddress localAddr = InetAddress.getByName(args[0]);
         TCPSocket socket = createTCPSocket();
         String content =  "<html><h1> Hello from Clack! </h1></html>\r\n\r\n";
         ByteBuffer page = ByteBuffer.wrap(content.getBytes());

        // prepare to serve content
        socket.bind(localAddr, 80);
        socket.listen();

       while (true) {
              // blocks until connection is received
             ClackSocket clientSocket = socket.accept();
             clientSocket.send(page);
             clientSocket.close();
       }
     }catch (Exception e){ e.printStackTrace(); }
   }

}
```

Bibliography

[1] Ethereal Network Analyzer www.ethereal.com

[2] Ethereal Labs http://gaia.cs.umass.edu/ethereal-labs

[3] Bruce Mechtly and Jack Decker ”Using ethereal and TCPportconnect in undergraduate networking labs” in *J. Comput. Small Coll.*, Vol. 19 No. 1, 2003

[4] The Network Simulator http://www.isi.edu/nsnam/ns/

[5] The Network Animator http://www.isi.edu/nsnam/nam/index.html

[6] Using ns and nam in Education http://www.isi.edu/nsnam/ns/edu/index.html

[7] NAM Editor Presentation http://www.isi.edu/nsnam/nam/nam-editor.ps

[8] Extended NAM Editor http://www.grid.unina.it/grid/ExtendedNamEditor

[9] CAIDA Internet Teaching Laboratory Workshop http://www.caida.org/outreach/itl/workshops/0106/materials.xml

[10] RouterSim http://www.routersim.com

[11] Kurose and Ross Student Resources - Applets http://wps.aw.com/aw_kurose_network_3/0,9212,1406346-,00.html

[12] A Web-Based Introduction to Computer Networks for Non-Majors http://cs.wcu.edu/ holliday/cware/index.html

[13] Curt M. White ”Visualization tools to suport data communications and computer network courses” in *Journal of Computer Sciences in Colleges*, Vol. 17 Issue 1, 2001

[14] GIDEN: a graphical environment for network optimization http://giden.nwu.edu/

[15] Virtual Internet and Telecommunications Laboratory of Switzerland http://www.vitels.ch/

[16] Introduction to Fishnet http://www.cs.washington.edu/education/courses/461/05wi/fishnet-intro.pdf

[17] SRMP Assignment: Simple, Reliable, Sequenced Message Transport Protocol http://www.cse.ucsd.edu/classes/wi05/cse123a/Prog2.pdf

[18] Laboratories for Data Communications and Computer Networking http://www.cse.ohio-state.edu/ jain/cise/index.html

[19] The Virtual Network System http://yuba.stanford.edu/vns/

[20] The NetFPGA Project http://yuba.stanford.edu/NetFPGA/

[21] Martin Casado, Nick McKeown The Virtual Network System in *SIGCSE*, 2005

[22] User-Mode Linux http://user-mode-linux.sourceforge.net/

[23] The Click Modular Router Project http://pdos.csail.mit.edu/click/

[24] JGraph: Java Graph Visualization and Layout http://www.jgraph.com

[25] JFreeChart http://www.jfree.org/jfreechart/

[26] JPCAP: A network packet capture library for applications written in Java http://jpcap.sourceforge.net

[27] Network Mapper http://www.insecure.org/nmap/

[28] HPing http://www.hping.org