

2008

A framework for graphical analysis of a home-network router using DTrace

Christopher S. Nelson
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Nelson, Christopher S., "A framework for graphical analysis of a home-network router using DTrace" (2008). *Master's Theses*. 3622.
DOI: <https://doi.org/10.31979/etd.hry2-b982>
https://scholarworks.sjsu.edu/etd_theses/3622

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

**A FRAMEWORK FOR GRAPHICAL ANALYSIS OF A
HOME-NETWORK ROUTER USING DTRACE**

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Christopher S. Nelson

December 2008

UMI Number: 1463414

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1463414

Copyright 2009 by ProQuest LLC.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

© 2008

Christopher S. Nelson

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Thesis Titled


A FRAMEWORK FOR GRAPHICAL ANALYSIS OF A

HOME-NETWORK ROUTER USING DTRACE

by

Christopher S. Nelson


APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING



Dr. Rod Fatoohi, Department of Computer Engineering Date 10/14/08

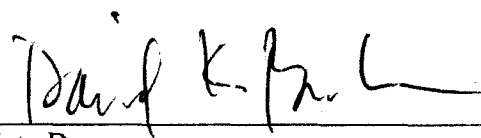


Dr. Frank Lin, Department of Computer Engineering Date 10/14/08



Dr. Xiao Su, Department of Computer Engineering Date 10/13/08

APPROVED FOR THE UNIVERSITY



Associate Dean Date 10/28/08

ABSTRACT

A FRAMEWORK FOR GRAPHICAL ANALYSIS OF A HOME-NETWORK ROUTER USING DTRACE

by Christopher S. Nelson

Simple network routers used in homes and small offices typically lack tools for monitoring and analysis that would be useful to the normally novice users of these products. Sophisticated network simulation applications require too much effort for typical users to consider, but including simple tools in the router management software would enable common users to more quickly and completely understand reasons for performance problems.

DTrace provides the opportunity to gather performance data from the router itself, and if presented in an easily understood graphical format, common users will be empowered to understand and address problems quickly and without need for additional support. This thesis addresses the development of a framework—utilizing DTrace, Java, AJAX, and remote procedure calls (RPCs) for client-to-server communication—for creating graphical analysis tools for analyzing common home-network routers. A reference implementation and test results that validate the framework architecture are also provided.

ACKNOWLEDGEMENTS

This thesis represents the capstone of my graduate studies and the fulfillment of the thesis requirement for a Master of Science in Computer Engineering at San José State University. Although this thesis represents the compilation of my own efforts, I would like to acknowledge and extend my sincere gratitude to the following persons for their valuable time and assistance—without whom the completion of this thesis would not have been possible:

1. Dr. Rod Fatoohi, Professor in the Computer Engineering department at San José State University, for his support and guidance in the organization and development of this thesis while acting as my faculty advisor.
2. Dr. Frank Lin, Professor in the Computer Engineering department at San José State University, for his support and guidance in the finalization of this thesis as a member of my department thesis review committee.
3. Dr. Xiao Su, Associate Professor in the Computer Engineering department at San José State University, for her support and guidance in the finalization of this thesis as a member of my department thesis review committee.
4. Dr. Lee Chang, Professor and Graduate Advisor for the Computer Engineering department at San José State University, for his guidance through the requirements of a thesis project at San José State University.
5. My friends and family, for patience through the many hours I was occupied with this project and the support and encouragement to stick with it and get it finished.

DEDICATION

To my beautiful wife, Lisa. Thanks for the support, encouragement, and time to complete this project and reach this life milestone.

TABLE OF CONTENTS

LIST OF FIGURES.....	xi
LIST OF TABLES.....	xiii
LIST OF ACRONYMS.....	xiv
GLOSSARY.....	xvi
I. INTRODUCTION.....	1
Project Goal and Objectives.....	2
Overview of this Document.....	3
II. TECHNICAL AND MARKET BACKGROUND.....	4
Technology Trends.....	4
Market Research.....	5
III. ARCHITECTURE AND DESIGN.....	8
General Architecture.....	8
Browser-Based User Interface.....	8
Front-to-Back-End Communication.....	9
Server-Side Design.....	10
Introduction to DTrace.....	11
IV. PLATFORM PREPARATION.....	13
Choosing an Operating System.....	13
Choosing a Hardware Platform.....	14
Installing and Configuring the Operating System.....	16
Supporting Wireless.....	18
Choosing Routing Software.....	19
Installing and Configuring the Routing Software.....	20
Choosing a Web Server.....	21
Installing and Configuring the Web Server.....	22
Choosing a Web Application Framework.....	23
Using the Web Application Framework.....	24
Choosing a Programming Language.....	26

V.	IMPLEMENTATION.....	27
	Developing DTrace Scripts to Gather Data.....	27
	The Back End: Incorporating DTrace with Application Code.....	34
	The Back End: Other Server-Side Code.....	42
	The Front End: Developing the User Interface.....	48
	RPCs: Tying the Front and Back Ends Together.....	57
	Deploying the Complete Web Application.....	66
VI.	TESTING.....	72
	Testing on a Virtual System.....	72
	Testing on a Real System.....	82
VII.	SUGGESTIONS FOR FUTURE DEVELOPMENT.....	89
	OpenSolaris on MIPS.....	89
	Wireless Support on OpenSolaris.....	90
	DTrace in Linux.....	90
	Cleaning Up the User Interface.....	91
	Additional Features.....	92
	Final Integration.....	93
VIII.	CONCLUSIONS AND RECOMMENDATIONS.....	94
	REFERENCES.....	96
	APPENDICES.....	99
	APPENDIX A: PROJECT REQUIREMENTS.....	100
	Project Deliverables.....	100
	Functional Requirements.....	100
	Non-Functional Requirements.....	102
	Requirements Analysis.....	103
	Hardware, Software, and Skill-Set Requirements.....	104
	APPENDIX B: PROJECT SCHEDULE.....	107
	Initial Schedule.....	107
	Final Schedule.....	109
	APPENDIX C: DEVELOPING WITH THE NETBEANS IDE.....	111
	Creating a Web Application Project.....	111
	Using GWT4NB.....	114
	Creating an RPC.....	115
	Using Additional Java Libraries.....	116

Building and Hosting the Web Application Locally.....	117
Building the Web Application for Deployment.....	119
APPENDIX D: SOURCE CODE – GENERAL FILES.....	121
index.jsp.....	121
web.xml.....	121
dgrp.css.....	122
license.txt.....	127
context.xml.....	131
gwt.properties.....	131
APPENDIX E: SOURCE CODE – PACKAGE org.dgrp.....	132
DtraceGraphicalRouterProject.gwt.xml.....	132
APPENDIX F: SOURCE CODE – PACKAGE org.dgrp.client.....	133
About.java.....	133
Analysis.java.....	135
AnalysisMenu.java.....	138
BandwidthInfo.java.....	151
BandwidthMonitor.java.....	153
BandwidthMonitorAsync.java.....	153
DGRPEntryPoint.java.....	154
GetVersionInfo.java.....	157
GetVersionInfoAsync.java.....	157
ImagePanel.java.....	158
Settings.java.....	170
Sidebar.java.....	171
SidebarItem.java.....	173
TopologyInfo.java.....	174
Version.java.....	175
VersionContents.java.....	180
Welcome.java.....	181
APPENDIX G: SOURCE CODE – PACKAGE org.dgrp.server.....	182
BandwidthMonitorImpl.java.....	182
DGRPLogger.java.....	184
GetVersionInfoImpl.java.....	185
APPENDIX H: SOURCE CODE – PACKAGE org.dgrp.server.dtraceservices	192
DTraceCountDataBytesService.java.....	192
DtraceCountPacketsService.java.....	196
count_data_bytes.d.....	199

count_packets.d.....	200
SOFT COPY SOURCE CODE.....	in pocket

LIST OF FIGURES

Figures	Page
1. Linksys Browser Interface	6
2. High-Level Framework Diagram	9
3. DTrace Extracting Data	28
4. Utilizing DTrace Scripts from Java Classes	36
5. Basic Interface Layout	51
6. Application Interface Analysis Menu	55
7. Graphics Controlled by the ImagePanel Class	56
8. RPCs	58
9. Results of a Successful Version RPC	63
10. Tomcat Web Application Manager	71
11. Creating Virtual Network Interfaces	76
12. DGRP Welcome Screen	77
13. DGRP Settings Screen	78
14. DGRP Version Screen	79
15. DGRP About Screen	80
16. Snoop Capture	81
17. DGRP Analysis Screen	82

18.	DGRP On Real Hardware	84
19.	Test Laptop IP Address	85
20.	Ordering According to Activity	86
21.	DTrace Script Output	87
22.	DGRP Analysis Matches DTrace Scripts	88
23.	House-of-Quality Diagram	104
24.	Original Project Schedule	108
25.	Final Project Schedule	110
26.	Creating a New Web Application Project in NetBeans	112
27.	Choosing a Web Server in NetBeans	113
28.	Choosing the GWT Framework in NetBeans	114
29.	The GWT4NB Plugin in NetBeans	115
30.	Creating an RPC in NetBeans	116
31.	Using Additional Java Libraries	117
32.	Local Deployment of a Web Application from NetBeans	118
33.	NetBeans Hosting a Web Application Locally	119
34.	A Web Application Ready for Deployment	120

LIST OF TABLES

<u>Tables</u>	<u>Page</u>
1. Common Home-Network Router Architectures	15
2. Project Deliverables	100
3. Hardware Requirements	105
4. Software Requirements	105
5. Skill-Set Requirements	106

LIST OF ACRONYMS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CD	Compact Disc
DGRP	DTrace Graphical Router Project
DNS	Domain Name System
DVD	Digital Video Disc
GUI	Graphical User Interface
GWT	Google Web Toolkit
GWT4NB	Google Web Toolkit for NetBeans
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
JAR	Java Archive
JDK	Java Development Kit
JSP	JavaServer Pages
MIPS	Multi-Instruction Processing System

NIC	Network Interface Card
OS	Operating System
QFD	Quality Function Design (or Deployment)
RIP	Routing Information Protocol
RPC	Remote Procedure Call
SPARC	Scalable Processor Architecture
SSID	Service Set Identifier
SXCE	Solaris Express Community Edition
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
XML	Extensible Markup Language

GLOSSARY

back-end	An acronym for “server-side. The server software in the client-server model.
client-server model	The concept in computing in which one application or system (the client) makes requests of another application or system (the server)—which services those requests upon receipt according to the server software.
DTrace	The dynamic tracing facility introduced in the Solaris operating system that allows querying thousands of software “probes” for real-time information while the software is running. For significant information on DTrace, refer to the OpenSolaris community web site (<i>OpenSolaris Community: DTrace</i> , 2008).
front-end	Opposite of “back-end.” An acronym for “client-side.” The client software in the client-server model.
Google Web Toolkit	An open-source framework for the development of AJAX web applications using Java source code. It also includes functionality for remote procedure calls.
GWT4NB	A plug-in developed for easy integration of the NetBeans IDE and the Google Web Toolkit.
home-network router	The multi-function device found in many homes that often serves as a cable or ADSL modem, gateway, wireless access point, and router for the computing devices on the home network.

Java DTrace library	A Java library of classes that provides access to the DTrace facility on the Solaris operating system from within Java classes (as opposed to via the command-line or scripts in a shell).
NetBeans IDE	An open-source integrated development environment distributed by Sun Microsystems that simplifies the development, debugging, packaging, compilation, and distribution of software applications.
OpenSolaris	The open-source version of the Solaris operating system originally developed by Sun Microsystems. For information about OpenSolaris, refer to the OpenSolaris web site (<i>OpenSolaris</i> , n.d.).
remote procedure call	The concept in distributed computing in which an application or system makes a call to (and often waits for a response from) a piece of software functionality (e.g., method, function, etc.) that may physically reside on a system somewhere remote from the initiating system.
web application	A software application most often designed to be run in a web browser. This uses the client-server model, as the client-side software executes in the browser, and the server-side code executes on the web server.
web server	The server-side software that provides the services for clients to access—most often through a web browser. Web servers host web applications that clients load and execute in a browser.

I. INTRODUCTION

Many home and non-enterprise network administrators (to use the term loosely), lack a set of tools that would enable them to do basic network monitoring and analysis of problems. Troubleshooting of a “slow Internet connection” frequently involves steps like rebooting the client computer(s) and/or the router and then—if that did not fix the problem—calling the service provider for additional help.

Meanwhile, as a separate issue, the real-time, in-production analysis capabilities of Dtrace—the dynamic tracing functionality introduced in the Solaris 10 operating system—have only begun to be realized. Apple Inc. has implemented similar analysis tools in their recently released “Leopard” operating system and introduced a graphical user-interface (GUI) front-end called “Instruments,” and ports of DTrace to Linux have begun (but have substantial work remaining before becoming pervasive). Still, relatively little has been done thus far (except by Apple and a couple of newer OpenSolaris projects) to provide graphical front-ends for these tools that would make them helpful to the common user.

The intersection of these issues is at the point of a possible solution for that inexperienced network “administrator.” By using the capabilities of DTrace at the heart of the network—on the router itself, and if the data is presented in such a way (graphically) that the user could understand it quickly and easily, one could get a much

clearer picture of the true source of the performance problem—whether that be an overactive client computer, an issue on the router itself, delays on the Internet Service Provider (ISP) side, etc.

Today's home-network routers are typically Linux-based (or use a proprietary operating system) and operate on Multi-Instruction Processing System (MIPS) processors—an architecture to which Solaris has apparently not been ported. Given a proven framework and reference implementation, future applications could include the development of productized Solaris-based routers that include this type of analysis tools (Solaris would need to be ported to MIPS to use today's hardware) or the development of similar graphical front-ends for the Linux version(s) of DTrace (after their development is complete) and integration of these into products similar to what is already available in today's market.

Project Goal and Objectives

Thus the general goal of this project is to develop a framework for and reference implementation of a graphical interface for analyzing a Solaris-based network router using DTrace. The high-level objectives for the project are as follows:

1. Develop a software architecture to graphically present the analysis data made available by DTrace in an interface similar to the graphical user interfaces commonly available in today's home-network routers.
2. Using the aforementioned architecture, develop a core set of analysis points in the graphical interface that form a model to be followed in the development of future analysis points.

3. Thoroughly document the design, architecture, and reference implementation code, and make these available to the open-source community in an effort to promote future development in this area.

Overview of this Document

This thesis reports on the project in chapters: II. Technical and Market Background; III. Architecture and Design; IV. Platform Preparation; V. Implementation; VI. Testing; VII. Suggestions for Future Development; and lastly, VIII. Conclusions and Recommendations. Chapter II, immediately following, discusses the need for this type of project in relation to the current technology market. Project requirements, schedules, and implementation source code are provided in the appendices for reference.

II. TECHNICAL AND MARKET BACKGROUND

Technology Trends

It is no secret that home computing has moved from nearly nonexistent to pervasive in the last two decades, that—in the latter part of that same time—access to the Internet has become a necessary part of life for most people in the developed countries of the world, and that the number of devices connecting to the Internet is growing at an ever-increasing rate. Many of these devices connect from home or other small sub-networks across the world through devices providing routing, network switching, access point, and sometimes modem capabilities, and the setup and maintenance of these devices is still too cumbersome for the average user.

As the Home Gateway Initiative—“an industry body that offers an active dialogue between telecoms operators, vendors, and manufacturers, and defines technical specifications for home gateways (Home Gateway Initiative, 2007, para. 1)”—states in a white-paper describing the growing need for such an organization,

“Multiple devices wish to share the broadband connection [to the Internet]. Games consoles, PC’s, telephones and IPTV settops all want a broadband connection, so the consumer needs to be able to share that connection between all devices, simultaneously.... Clearly networking is complex to manage for both the customer and for the service provider who is often the first point of contact when a customer encounters a problem. (Home Gateway Initiative – Vision, 2007, para. 4)”

As the number of devices in the home connecting to the Internet continues to increase,

the need for removing that network-management complexity will increase as well. The device manufacturer who can produce a device that is simple to implement and to debug when things go wrong, or the service provider who can provide a service offering to manage the complexity for the customer will have a business advantage as the number and complexity of home networks continues to rise.

Market Research

In the home-network router market, a number of recognizable companies exist. A quick search on the web sites of Best Buy, Circuit City, or Fry's will reveal products from Netgear, D-Link, Belkin, Linksys, Apple, 2Wire, and many others. Nearly all of these provide a browser-based interface for management like that in Figure 1, and few—if any—debugging tools are provided in the interfaces to help determine the cause of problems. The “Help” link from a 2Wire home router, for example, points the user to the 2Wire support web page—offering basic troubleshooting tips but nothing specific to the user's network or situation.

Outside of the home-network market, however, efforts are being made to ease the pain of troubleshooting computing devices. Sun introduced their new Dynamic Tracing (DTrace) technology in their Solaris 10 operating system; the DTrace manual explains that “DTrace enables you to explore your system to understand how it works, track down performance problems across many layers of software, or locate the cause of aberrant behavior” (DTrace – Introduction, 2007, para. 1). Since then, a few OpenSolaris projects

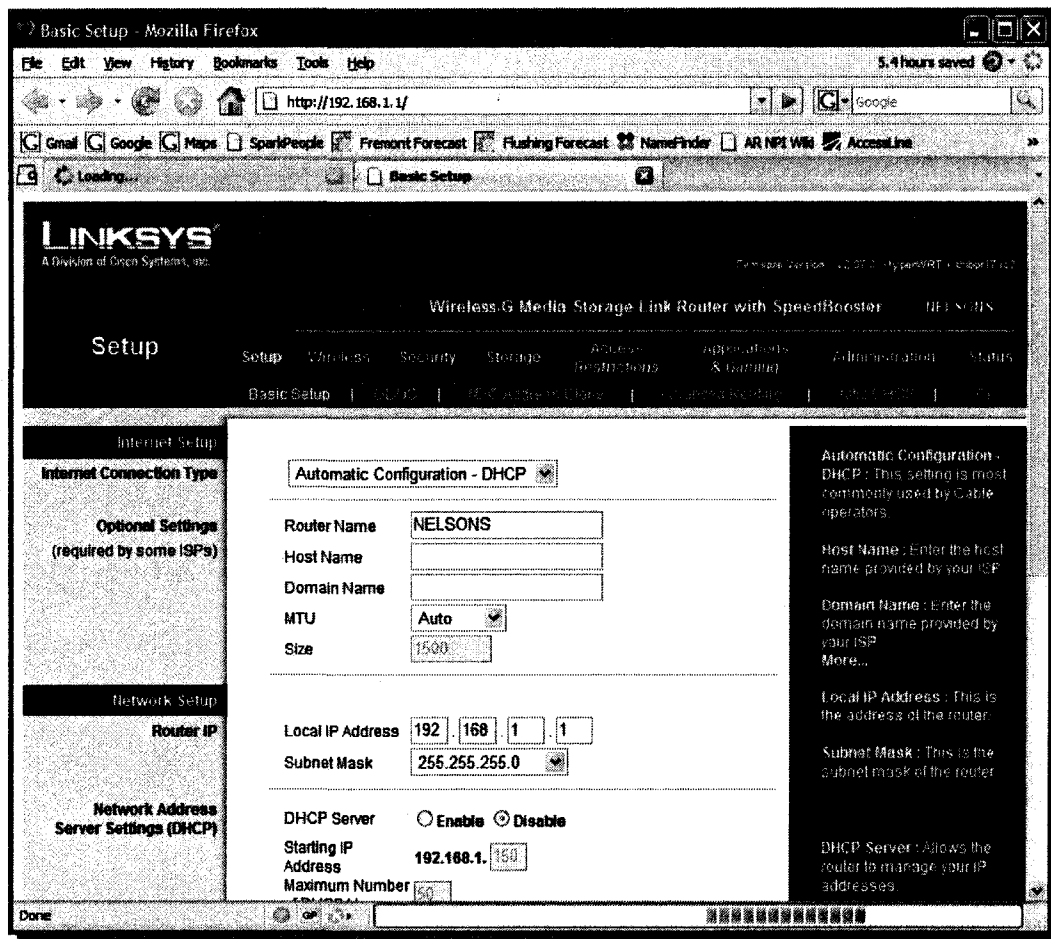


Figure 1. Linksys Browser Interface

The Linksys browser interface is similar to those offered by most of the home-network router providers.

—like *Chime* (*OpenSolaris Project: Chime Visualization Tool for DTrace, 2008*)—have taken up the effort to make graphical front-ends for DTrace, and one has been created for integrating with the Sun Studio and NetBeans Integrated Development Environments (IDEs) (NetBeans DTrace GUI Plugin, n.d.). Apple has also taken interest in DTrace, and in their latest operating system, they implemented a graphical tool called *Instruments* that

utilizes their own version of DTrace. Work is ongoing in the Linux community as well to port DTrace to the various distributions of Linux, and other variations—like SystemTap—are also developing (SystemTap, n.d.). The development of these tools is in apparent recognition of the need for easier debugging of problems in increasingly complex computing devices. It seems reasonable to apply these tools to the growing complexity of home network devices just the same.

The current collection of home-network routers are primarily specialized hardware making use of a MIPS processor and relatively little memory. The author was unable to find any recent version of Solaris that has been compiled for the MIPS architecture, so loading Solaris onto the existing products' hardware directly is not directly possible. An implementation using DTrace on a router (and thus—at this time—requiring Solaris instead of Linux) would have to be made to use some other hardware until such a time as Solaris is available for the MIPS architecture.

With this background information in-hand and the tools described in the current state, the author set out to create a framework that would enable the use of DTrace to collect useful data from a home-network router and present it to users in an easy-to-understand graphical format that would enable them to understand problems in their home network. It was also necessary to develop an implementation of the framework and to test that implementation in order to prove the framework architecturally sound. The following chapters describe the framework, the reference implementation developed, and the testing performed to validate both.

III. ARCHITECTURE AND DESIGN

General Architecture

The general architecture for this framework is depicted in Figure 2. A web application provides a browser-based user interface as the front end of the application—communicating with the back end (i.e., software on the router itself) via remote procedure calls. This architecture is further discussed in the following sections which detail the design of this framework and a reference implementation that proves its effectiveness and functionality.

Browser-Based User Interface

As discussed previously in the *Market Research* section (p. 5), most of today's home network routers provide a user interface via a web browser (see Figure 1). In order to easily integrate the functionality of this project into routers like those in today's market, the logical choice for user interface is to also develop it in a browser-based fashion. With this as a design assumption, the user interface for this project is designed as a web application. The choice of web server, application language, etc.—though indicated in part in Figure 2—is actually implementation-specific and is thus discussed further in the following two chapters. The primary functions of the front-end user interface are to accept user input and properly format and display data: The front end should not rely on

the server side to provide formatting or other control over the user interface display.

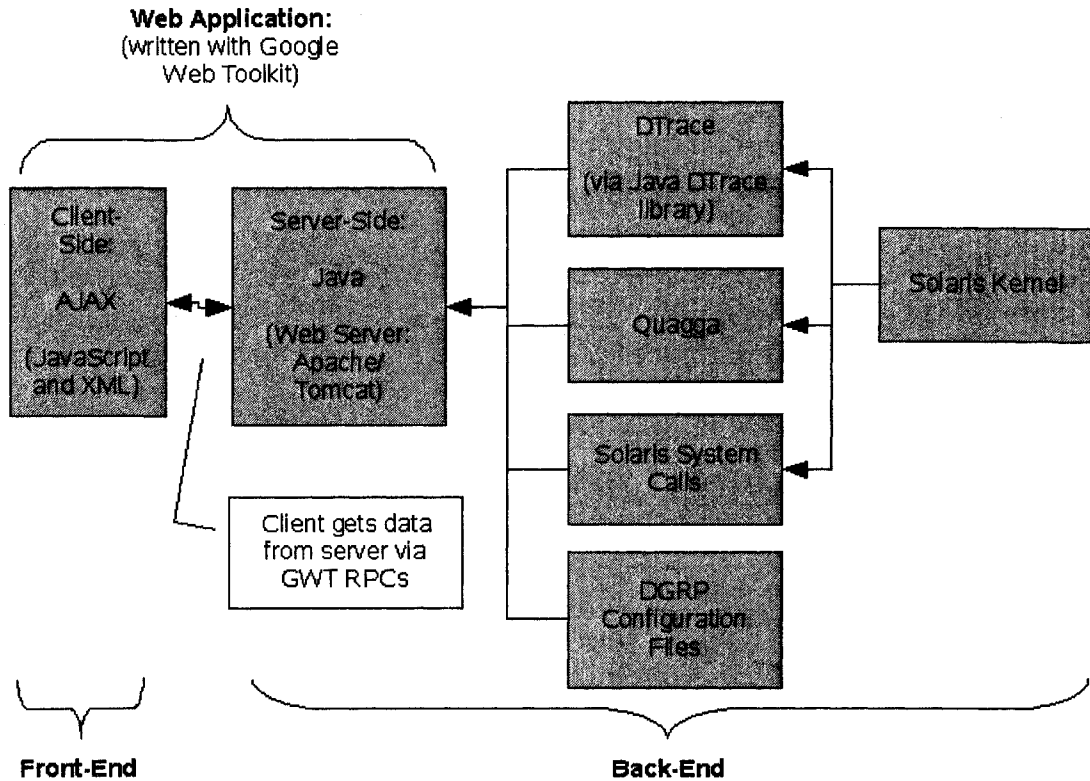


Figure 2. High-Level Framework Diagram

The general architecture of this application includes a web application pulling data from DTrace, Quagga, and the Solaris OS directly

Front-to-Back-End Communication

Communication between the front end (i.e., browser-based user interface) and the back end (server-side) of this program is achieved using the concept of remote procedure calls. RPCs, in summary, allow a software program to use a piece of software

functionality (e.g., method, procedure, or function—depending on the programming language) as if it was resident with the program on whatever hardware on which it is running, though the implementing side of that piece of code is often on a different (and perhaps distant) piece of hardware. (For further information on the concept of RPCs, the reader is encouraged to reference the many helpful articles available on the Internet and elsewhere.) For this application, the client-side code running in the browser interacts with data-providing code running on the router via RPCs. The choice of what RPC package is used and whether the RPCs should be synchronous (i.e., blocking calls), asynchronous (i.e., non-blocking calls), or some combination of the two is an implementation-specific discussion and is thus addressed in the following chapter.

Server-Side Design

In this design, the “server” is actually the router itself—from which the user interface is served as a web application. So, in addition to performing the functions of a router, the operating system and software running on the router hardware must also act as a web server for the user interface and—as already briefly discussed in the previous sections—provide data to client software (running in the user's browser) in response to requests in the form of remote procedure calls. While this data may be pulled from the operating system (OS) or from software running on top of the OS, the most interesting pieces of data in the context of this application are those provided by DTrace—and most specifically the DTrace Network Providers. (For a brief introduction to DTrace and the

DTrace Network Providers, refer to the next section.) While this design and reference implementation rely on DTrace as it is currently (at the time of this writing) made available by the Solaris OS, other operating systems or implementations of DTrace could be used to provide a similar back end in future implementations: For further discussion on this topic, refer to Chapter VII. *Suggestions for Future Development* (p. 89).

Introduction to DTrace

As it is described in the formal DTrace documentation, “DTrace helps you understand a software system by enabling you to dynamically modify the operating system kernel and user processes to record additional data that you specify at locations of interest, called *probes*. (DTrace – Introduction, 2007, para. 2)” Probes are little pieces of code included in operating systems and applications that have implemented them for DTrace to use for collecting data when asked; as the formal documentation says, they are like “programmable sensors scattered all over your [operating] system in interesting places. (DTrace – Introduction, 2007, para. 2)” The modules of the operating systems or applications that provide these probes to the DTrace facility are aptly named *providers*. For example, the Solaris operating system makes available *send* and *receive* probes in the *ip Provider* (Gregg, B., 2008); the probes fire (i.e., DTrace can collect data) each time the kernel sends or receives a packet, and relevant data (e.g., source and destination address, packet header flags, number of data bytes included, etc.) can be collected for analysis. Additional network-related providers (e.g., TCP, User-Datagram Protocol (UDP), etc.),

collectively known as the “Network Providers” are being added to the Solaris operating system for future use by DTrace (DTrace Network Providers, 2008). For details on what providers and probes are implemented in a given operating system or application, the user is directed to the DTrace-relevant documentation for that OS or application.

Multiple interfaces exist to the DTrace facility. For example, a program can be written in DTrace's D programming language to extract data from specified probes and format the text-based output, or a programmer could make use of the Java DTrace library currently available for the Solaris OS (Java DTrace API, 2007). The analysis that can be performed by DTrace is limited only by the providers and probes already implemented in a specific application or OS and the imagination of the programmer utilizing the data made available by those providers and probes.

This very brief and high-level introduction to DTrace only touches on the very basics, and the reader is encouraged to refer to the available DTrace documentation for significant additional detail.

IV. PLATFORM PREPARATION

The sections in this chapter describe the choice, installation, and configuration of a number of available pieces of hardware and software that create the underlying platform on which the reference implementation was developed and tested. The following chapter describes in detail the actual implementation.

Choosing an Operating System

That DTrace is the source of the data used for this application is an underlying assumption and a premise of the entire project, thus when making one of the first implementation choices—the choice of which operating system to use—the list of options is limited to only those which already provide the DTrace facility. At the time of commencing this project, only two operating systems had incorporated DTrace functionality—Sun Microsystems' Solaris 10 OS and its subsequent open-source derivatives and Apple's Mac OS X 10.5 (Leopard), though work had begun on porting DTrace to FreeBSD, and—as of the time of this writing—recent information suggests that FreeBSD's DTrace facility is ready for use—at least in its initial form (DTrace for FreeBSD, 2008).

Choosing between Solaris and OS X for this project iteration was a simple decision: OpenSolaris—a freely available, open-source derivative of Sun's Solaris 10—

includes the latest and greatest features of DTrace, is actively supported by the open-source community, is regularly updated (biweekly or less frequently—depending on the release chosen) (OpenSolaris Download Center, 2008), and works on at least SPARC and x86 platforms with the possibility of porting it to other hardware as well, while OS X is limited to Apple's hardware only, is not free, and may not include the full DTrace functionality (refer to Leventhal, A., 2008, for a discussion of Apple's DTrace implementation and its limitations). This simple comparison led the author to choose OpenSolaris as the operating system for this iteration of this project. Installing and using this operating system is discussed in the *Installing and Configuring the Operating System* section (p. 16).

Choosing a Hardware Platform

With the operating system decision made, the choice of hardware platform is next to be determined. As the long-term goal of this project is to integrate with existing management software on today's common home-network routers, the ideal hardware platform for development is the hardware on which those home-network routers are currently built. As briefly discussed in Chapter II. *Technical and Market Background* (p. 4), research shows that many of today's home-network routers are built on the MIPS architecture with a relatively small memory footprint. The OpenWrt Community—an “open source project to create a free embedded operating system for network devices (OpenWrt, 2008, para. 3)” —tracks the hardware specifications of many home network

routers. The following table is a sampling of data adapted from OpenWrt's extensive *Table of Hardware* (*Table of Hardware*, 2008).

Table 1. Common Home-Network Router Architectures

Brand	Model	Processor	Architecture	Memory
D-Link	DSL-G604T	TI AR7	MIPS	16 MB
Linksys	WAG354G	TI AR7	MIPS	16 MB
Linksys	WRTSL54GS	Broadcom 4704	MIPS	32 MB
Netgear	WGT624	Atheros 2312	MIPS	16 MB
Belkin	F5D8230-4	Realtek 8651B	MIPS-like	16 MB

As shown, many of the big-name home-network router makers use the MIPS architecture for today's products. While other architectures exist, most of the mainstream products appeared to be based on MIPS at the time of commencing this project. With that in mind, it made sense to investigate whether Solaris could be made to run on MIPS. Unfortunately, at the time, the author was unable to find any current existing work toward porting the Solaris operating system to MIPS. Outdated work existed (see *Rational Apex Embedded Solaris to MIPS Family Release Note for Tornado*, 2001) for an example of work over 10 years old in which a previous version of Solaris was made to run on MIPS, but nothing was found that indicated Solaris 10 (or later) had run successfully on MIPS or was even close to being able to do so. In fact, an e-mail conversation from recent years—archived on the OpenSolaris website—discusses in some detail the idea of porting Solaris to MIPS but clearly states that that work is not in progress (*MIPS port of*

opensolaris, 2005). To be certain, the author contacted the initiator of the OpenSolaris e-mail conversation who confirmed that her queries had been met with answers and discussion but no progress—either at the time of the original conversation or in the months since (T. Snyder, personal communication, February 4, 2008).

Without the pre-existing capability to run Solaris on MIPS, a decision was necessary whether to begin work on porting Solaris to MIPS to get the functionality of DTrace on a currently available home-network router or to defer the Solaris-to-MIPS port and concentrate on the other major portions of the project using a different hardware platform for this iteration. For the development of this framework, the choice was made to use a different hardware platform and to encourage the future porting of Solaris to MIPS as a follow-on to this project. For more information on this step, refer to Chapter VII. *Suggestions for Future Development* (p. 89). With MIPS no longer an option for this project, the choice of a hardware platform for development was quite simple: Choose a common platform that is readily available to developers and on which Solaris can already run. The choice of an x86 platform met these simple criteria, so the author developed on an Intel Centrino-based laptop and tested on a number simple Intel and AMD-based servers. For further information on testing, refer to Chapter VI. *Testing* (p. 72).

Installing and Configuring the Operating System

There are many ways to install the OpenSolaris operating system—from compact discs (CDs) or digital video discs (DVDs), from iso images, or via a network connection,

for example, so this section will document the basic settings and adaptations necessary during the installation of the OpenSolaris operating system for the purposes of this project iteration—using a DVD installation as an example. Detailed installation instructions are available from the OpenSolaris web site (*OpenSolaris*, 2008), so only a summary of the common steps will be provided here. While development of this project iteration commenced on OpenSolaris's Solaris Express Community Edition build 82 and later moved to build 96, the installation and modifications process was the same for both versions.

1. Download a DVD image of the required build, burn the image to a DVD, and boot the development system from the DVD.
2. Select the appropriate boot option from the GRUB menu (the default for a local install or one of the tty options for a console install, and select from the following menu whether to use a windowed or text-based installation process.
3. Using the menus provided, when prompted, complete the appropriate language, networking, time, and password configurations.
4. Ensure that the installation takes place from the DVD, and accept the license agreement when prompted.
5. Use the default install, or select custom install to configure disk partition information and which parts of the OS should be installed.
6. When the install completes and the system reboots, be sure to eject the DVD so that the system will boot from the new operating system on the hard drive.

For the purposes of this project, very little post-installation configuration of the operating system is required. After the reboot, log in using the super-user *root* and the password set during installation. From this point, the additional installation and configuration steps described in the latter sections of this chapter could be followed, but there is one useful

step the author preferred to do after the OS installation is complete: Enabling remote login of the *root* user via secure shell (SSH) proved to be very convenient for the author, as he frequently used a terminal emulation software to open a remote console on the router. While this is considered a security risk in normal deployments, it proved to be very useful in development. To enable remote login by *root* via SSH, edit the */etc/ssh/ssd_config* file, and change the line

```
PermitRootLogin no
```

to

```
PermitRootLogin yes
```

and restart the SSH service using the command

```
svcadm restart ssh
```

Supporting Wireless

Wireless support in a home-network router takes two forms: First, a wireless NIC must be identified that is supported by the hardware platform and operating system of choice—OpenSolaris in the case of this project iteration; second, software must be available to make the router act as an access point—not just a member of a wireless network but rather the point at which others access the wireless network (e.g., the owner and perhaps broadcaster of the service set identifier (SSID), etc.).

For the first task, the author investigated a few different Universal Serial Bus (USB) NICs on the development platform running OpenSolaris—namely the Belkin

F5D7050 4001 model, the AirLink101 AWLL3028 model, and the Linksys WUSB54G model. Using the information available from the subset of the OpenSolaris community working toward the development of the *zyd* wireless NIC driver (*Wireless Network Driver for ZyDAS*, 2008), the author determined that the Belkin and AirLink101 NICs were not yet supported by OpenSolaris. Using the Linksys NIC, however, the author was able to install the necessary drivers and configure the interface such that it joined an existing wireless network.

But while the author was able to make a wireless NIC work with OpenSolaris, he was unable to find any access point software available for OpenSolaris at the outset of this project, and a recent discussion on the OpenSolaris community forums confirmed this research, so it was decided that wireless support would be out of the scope of this reference implementation (*Thread: solaris as a wireless access point*, 2007). The topic is, however, discussed in Chapter VII. *Suggestions for Future Development* (p. 89).

Choosing Routing Software

Because of the long-term goal of this project—that the software will be integrated into the management software of today's home-network routers, the choice of routing software for this initial implementation is not of long-term significance. In other words, because the routing functionality is already part of home-network routers, this iteration of this project need only find a solution that will provide that functionality until this software can be integrated back into the software on the true home-network routers.

Given the decision to use Solaris for this iteration's operating system, the routing software must function on Solaris. Early in the project, the author's university advisor recommended the consideration of a routing software called *Zebra*—one he knew was once available for use on the Solaris OS. Simple research into the recent history of Zebra showed that it had been forked and that development for Solaris had continued under the name *Quagga* (*OpenSolaris Project: Quagga Routing Protocol Suite Integration*, 2007). In fact, Quagga is now pre-installed in OpenSolaris, so the use of it is very straightforward—as described in the next section.

Installing and Configuring the Routing Software

As noted previously, Quagga is pre-installed in OpenSolaris, so only a few steps are required to enable and use it—as listed below. For the complete installation and configuration documentation, refer to the OpenSolaris Quagga web site (*OpenSolaris Project: Quagga Routing Protocol Suite Integration*, 2007).

1. Disable the other routing services available on Solaris:

```
# svcadm disable route:default
# svcadm disable ripng:default
```

2. Enable the Routing Information Protocol (RIP) using Quagga (and its dependencies):

```
# svcadm enable -r rip:quagga
```

3. Verify the Quagga RIP service is online using either *svcadm* or *routeadm*:

```
nv96-vbox$ svcs -l rip:quagga
fmri          svc:/network/routing/rip:quagga
name         Quagga: ripd, RIPv1/2 IPv4 routing protocol daemon.
```

```

enabled      true
state        online
next_state   none
state_time   Wed Aug 27 16:53:29 2008
logfile      /var/svc/log/network-routing-rip:quagga.log
restarter    svc:/system/svc/restarter:default
contract_id  104
dependency   require_all/none svc:/system/filesystem/usr:default
(online)
dependency   optional_all/refresh svc:/network/ipv4-forwarding
(disabled)
dependency   require_all/refresh svc:/network/routing-setup
(online)
dependency   optional_all/restart
svc:/network/routing/zebra:quagga (online)
nv96-vbox$
nv96-vbox$
nv96-vbox$ routeadm

```

Configuration Option	Current Configuration	Current System State
IPv4 routing	enabled	enabled
IPv6 routing	disabled	disabled
IPv4 forwarding	disabled	disabled
IPv6 forwarding	disabled	disabled
Routing services	"route:default ripng:default"	

Routing daemons:

	STATE	FMRI
routing:ipv4	disabled	svc:/network/routing/legacy-
routing:ipv6	disabled	svc:/network/routing/legacy-
	online	svc:/network/routing/zebra:quagga
	online	svc:/network/routing/rip:quagga
	disabled	svc:/network/routing/ripng:default
	disabled	svc:/network/routing/ripng:quagga
	disabled	svc:/network/routing/ospf:quagga
	disabled	svc:/network/routing/ospf6:quagga
	disabled	svc:/network/routing/bgp:quagga
	online	svc:/network/routing/ndp:default
	disabled	svc:/network/routing/rdisc:default
	disabled	svc:/network/routing/route:default

nv96-vbox\$

Choosing a Web Server

The choice of web server was not one based primarily on the comparison of available options, rather it was based on the author's experience with previous web application development. The author was most familiar with the Apache Tomcat server—produced by The Apache Software Foundation (*The Apache Software Foundation*, 2008), and its simple integration with the author's primary Integrated Development Environment, the *NetBeans IDE*, made for simple development. Similar to the choice of routing software in this initial iteration, the choice of web server is of little long-term consequence: When a future iteration integrates this software with existing home-network router management software, the web server used will be that already in use by the existing management software.

Installing and Configuring the Web Server

Like Quagga, the Apache and Apache Tomcat web server suite come pre-installed in the OpenSolaris operating system. Little preparatory work is necessary to be ready to deploy basic web applications, though the Apache Foundation provides detailed installation and configuration instructions on their web site (*Apache Tomcat 6.0*, 2008). The following steps are those necessary to prepare Apache Tomcat for use with this project iteration's software (as described later in this chapter):

1. In the directory */var/apache/tomcat/conf/*, copy *server.xml-example* to *server.xml* as in the command

```
cp /var/apache/tomcat/conf/server.xml-example /var/apache/tomcat/conf/server.xml
```

2. In the directory */etc/apache*, copy *httpd.conf-example* to *httpd.conf* as in the command

```
cp /etc/apache/httpd.conf-example /etc/apache/httpd.conf
```

3. Edit the file */etc/apache/httpd.conf* to remove the “#” from the start of the line (i.e., uncomment the line)

```
#include /etc/apache/tomcat.conf
```

4. In the file */var/apache/tomcat/conf/tomcat-users.xml*, add *manager* to the roles for user *tomcat*.

5. Restart the apache daemon using the series of commands

```
/etc/rc3.d/S50apache stop; sleep 1; /etc/rc3.d/S50apache start
```

A few remaining steps specific to the deployment of this project's web application are discussed later in this chapter in the section *Deploying the Complete Web Application* (p. 66). Note that the installation and configuration described here do not make use of the authentication, authorization, or other security-related capabilities of the Apache software. The assumption exists that the web server used in future iterations when this software is integrated with an existing home-network router's management software will already be configured to address these security needs.

Choosing a Web Application Framework

While web applications can be developed entirely from scratch using any number of programming languages, there are a number of development frameworks and toolkits available to ease the development and maintenance burden for today's increasingly complex web applications. A large list of available web application frameworks is

available on Wikipedia (*List of web application frameworks*, 2008). Examples of these frameworks include *Struts* from the Apache Foundation (*Struts*, 2008), *Stripes* (*Stripes Home*, 2008), *Ruby on Rails* for the Ruby programming language (*Ruby on Rails*, 2008), and several based on the JavaScript client-side programming language—including the *Google Web Toolkit* from Google. The Google Web Toolkit (GWT) is an open-source project that promises to “ease [the development and maintenance] burden by allowing developers to quickly build and maintain complex yet highly performant JavaScript front-end applications in the Java programming language. (*Google Web Toolkit*, 2008, para. 1)” Given the author's experience with the Java programming language, this was a quickly a leading candidate among the available toolkits for this project iteration. After some experimentation with sample web applications provided by the GWT community (*Building a Sample Application*, 2008), the author determined that the Google Web Toolkit would suffice for this project iteration. As discussed in Chapter VII. *Suggestions for Future Development* (p. 89), the authors of future iterations of the project may want to explore other available and perhaps more fully featured toolkits and frameworks for greater flexibility in the user interface.

Using the Web Application Framework

The community developing the Google Web Toolkit provides significant documentation on the use of GWT in web application development (*Google Web Toolkit*, 2008). The reader is encouraged to refer to that documentation for details, though a

summary is provided here. The GWT is provided in a package that includes documentation, samples, and a collection of Java Archive (JAR) files. These JAR files include Java classes that can be used by application developers while writing their code along with tools for compiling, hosting, and debugging applications. These tools can be used via a command-line interface on a console or through a graphical IDE such as *Eclipse* (*Eclipse – an open development platform*, 2008) or *NetBeans* (*NetBeans*, 2008). The author used NetBeans and an open-source plugin called *GWT4NB* (*gwt4nb Project Home*, n.d.) that allowed for easy integration of Netbeans with the compiler and other tools included in GWT.

It is important to understand what GWT does with the code written by developers: GWT produces AJAX code—collections of JavaScript and Extensible Markup Language (XML) files that are used by a web browser in the rendering of a web application—from Java code written by the application developer. As mentioned before, GWT includes Java (note, not JavaScript) classes that are made available for the developer to use; these classes provide either identical or very similar functionality to most of the classes in the standard Java Development Kit (JDK). This allows developers to code web applications in the Java language and use GWT to translate (i.e., compile) that Java code into equivalent JavaScript and XML. A complete Application Programming Interface (API) and simple sample code is provided for developers to reference in the documentation provided in the GWT package..

One other significant function provided by GWT worthy of summarizing here is

that of the Remote Procedure Call. GWT's RPC functionality allows web applications running in a browser to asynchronously make requests of the server without reloading the entire page in the browser. It is important to understand the two sides of this communication: The client-side code—the AJAX translated from GWT Java classes and running in the browser—requests a function to be performed by the server and listens without blocking for a response; the server-side code is true Java (from the JDK—not GWT's Java classes) and can perform any function made possible by the Java language before returning to the client. Examples of how this works in the context of this project are included later in this chapter in the *RPCs: Tying the Front and Back Ends Together* section (p. 57).

Choosing a Programming Language

The choice of programming language for this iteration of this project was based primarily on the author's previous experience and comfort with the Java language as a development language for web applications. The use of Java as the source language in the Google Web Toolkit framework also contributed to this decision. While Java is the language with which most of this iteration's code is written, future iterations need not be tied to Java—especially if using a different web application framework.

V. IMPLEMENTATION

The previous chapter described the choice, installation, configuration, and use of hardware and software that collectively formed the platform on which the reference implementation was developed and tested. The sections in this chapter describe in detail the original work done by the author to implement the framework described in Chapter III. *Architecture and Design* (p. 8).

Developing DTrace Scripts to Gather Data

A brief introduction to DTrace is provided in Chapter III. *Architecture and Design* (p. 8), and for detail beyond that introduction, the reader is encouraged to reference significant DTrace documentation from Sun Microsystems (*BigAdmin System Administration Portal. DTrace*, 2008) and the OpenSolaris community at (*OpenSolaris Community: DTrace*, 2007). This section will describe two of the most important DTrace scripts written to gather data relevant to this project. These scripts provide the foundational data that the web application processes and presents to the user. Each of these scripts uses DTrace's ip Provider (Gregg, 2008) to extract useful data from the IP headers of packets sent and received by the router's operating system's kernel's IP stack; this is conceptually depicted in Figure 3. Note that these scripts are written so that they can be executed from a console on the router during development and testing and also be

used—without modification—by the web application's server-side Java code. All of the DTrace scripts written for this project are available in the appendices of this document and should be understandable given the descriptions in this section, the available DTrace documentation, and the comments in the scripts themselves.

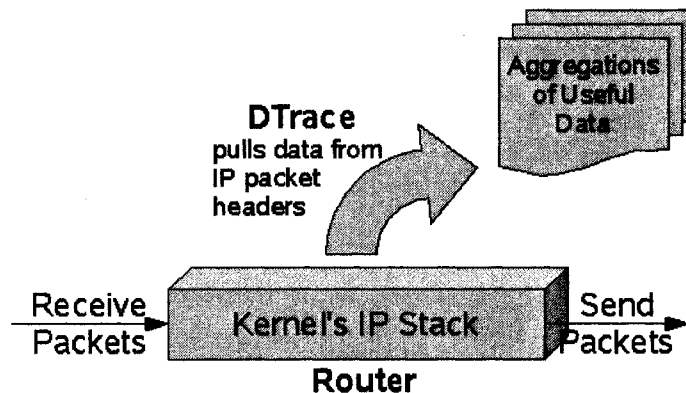


Figure 3. DTrace Extracting Data

The DTrace scripts activate probes in the kernel's IP stack to collect useful data into aggregations.

The first of the two important DTrace scripts in this project iteration which will be described in detail here is named *count_data_bytes.d*. The “.d” extension in the file name indicates that this is a script written in DTrace's D programming language. The first line in this DTrace script

```
#!/usr/sbin/dtrace -s
```

is similar to the first line in many shell scripting languages: When executing this file, this indicates to the operating system which program should be used to process it; in this case, a program called *dtrace* in the directory */usr/sbin* is used and is passed one parameter, *-s*

(which indicates to *dtrace* that the remainder of the file should be interpreted as D code).

The next line

```
#pragma D option defaultargs
```

indicates to DTrace that it should use default values for any parameters referenced in the D code that were not explicitly defined when the script was executed. When executed from a console, parameters are defined by adding additional items after the script name when executing it—as in the following example:

```
$ ./count_data_bytes.d param1 param2
```

How parameters can be defined when using the DTrace script with Java code is discussed in the following section, *The Back End: Incorporating DTrace with Application Code* (p. 34).

The next several lines of code in *count_data_bytes.d*

```
BEGIN /* Special probe upon script startup */
{
    givenSubnet = $$1; /* subnet either given or set
                       as empty string */

    printf("\n\n-----" +
           "-----\n");
    printf("Counting data bytes sent and received by IP" +
           " address...\n");
    printf("-----" +
           "-----\n");
}
```

define the actions to be taken when the *BEGIN* probe—a special probe that DTrace triggers when the script begins execution—fires. In this script, two things occur in the *BEGIN* probe: A variable *givenSubnet* is created and set to the value of the first parameter given to this script or—because of the *defaultargs* setting—set to the default

value of an empty string, and an informational heading is printed describing what this script is doing.

Following the definition of actions for the *BEGIN* probe, actions are defined for two additional probes—*send* and *receive* in the *ip* provider. These probes fire whenever the network stack in the OS kernel sends or receives—respectively—an Internet Protocol packet. (For a full discussion of the *ip* provider, the reader is encouraged to refer to the DTrace *ip* provider web site (Gregg, 2008.)

```
ip:::send /* Probe for sent packets (by destination address) */
{
    @snd[args[2]->ip_daddr] = sum(args[2]->ip_plength);
    @tot[args[2]->ip_daddr] = sum(args[2]->ip_plength);
}

ip:::receive /* Probe for received packets (by source address) */
{
    @rcv[args[2]->ip_saddr] = sum(args[2]->ip_plength);
    @tot[args[2]->ip_saddr] = sum(args[2]->ip_plength);
}
```

Whenever the *send* probe fires, two *aggregations* are updated: *@snd* and *@tot*. In DTrace, an aggregation is something like an array in other programming languages—indexed by something called a *tuple*. In this case, the tuple is the *ip_daddr*—the destination IP address—of the structure provided in *args[2]*. The *ip* provider provides a series of structures containing information in an array called *args* whenever a probe fires; it is from these structures that DTrace scripts can obtain and analyze data. *args[2]* contains a simple structure of the type *ipinfo_t*—which is defined as follows (Gregg, B., 2008):

```
typedef struct ipinfo {
    uint8_t ip_ver; /* IP version (4, 6) */
```

```

uint16_t ip_plength;          /* payload length */
string ip_saddr;             /* source address */
string ip_daddr;            /* destination address */
} ipinfo_t;

```

Thus, when the *send* probe fires, the entries in aggregations *@snd* and *@tot* for the destination IP address provided in the *args[2]* structure are updated with the value in the payload length field of the same structure according to DTraces's *sum* function. The *sum* function adds to an existing value whatever new value is provided to it. So, in summary, the actions in the *send* probe add the payload length of a packet to two aggregations which are indexed by destination IP address. The *receive* probe actions work very much the same way, though the aggregations are indexed by the source IP address, *ip_saddr*, and the aggregations updated are *@rcv* and *@tot*. The observant reader may notice that the *@tot* aggregation is updated in both the *send* and *receive* probe actions—thus its values are a sum of the number of data bytes sent to **and** received from each IP address, whereas the *@snd* and *@rcv* aggregations track only the data bytes sent to and received from—respectively—each IP address.

There is no code in the script to cause it to terminate on its own, so it will continue to run and to count the data bytes sent and received until the user stops it manually (such as with Cntl-C on the console). When the script is terminated, a special *END* probe is triggered—similar to the *BEGIN* probe which fired at the start of the script.

```

END /* Special probe upon script termination */
{
    printf("\n\n-----" +
           "-----\n");
    printf("Printing results...\n");
    printf("-----" +
           "-----\n");
}

```

```

printf("\nData bytes sent to:\n");
printa("  %15s %8u\n", @snd);

printf("\nData bytes received from:\n");
printa("  %15s %8u\n", @rcv);

printf("\nTotal data bytes received from and sent to:\n");
printa("  %15s %8u\n", @tot);
}

```

These actions in the *END* probe print an informational header and then use DTrace's *printa* function to print each aggregation, *@snd*, *@rcv*, and *@tot*, according to the formatting specified. For a complete discussion of formatting output from DTrace, the reader is encouraged to reference the online DTrace manual's description of output formatting (*Output Formatting*, 2007). Essentially, these statements provide a formatted printing of each aggregation's tuples and corresponding values—sorted by the value.

Using the *count_data_bytes.d* script from a console on a very quiet system may provide output similar to this:

```

nv96-vbox$ ./count_data_bytes.d
dtrace: script './count_data_bytes.d' matched 21 probes
CPU      ID          FUNCTION:NAME
  0       1                :BEGIN

-----
Counting data bytes sent and received by IP address...
-----

^C
  0       2                :END

-----
Printing results...
-----

Data bytes sent to:
    10.0.1.50      160
    10.0.2.50      160
    10.0.0.50      320
    10.0.3.50      320

```

```
Data bytes received from:
    10.0.1.50      160
    10.0.2.50      160
    10.0.0.50      320
    10.0.3.50      320
```

```
Total data bytes received from and sent to:
    10.0.1.50      320
    10.0.2.50      320
    10.0.0.50      640
    10.0.3.50      640
```

```
nv96-vbox$
```

Similar to *count_data_bytes.d*, the DTrace script *count_packets.d* uses the *send* and *receive* probes from the ip provider but simply counts the number of packets sent and received rather than the number of data bytes in each packet. The *BEGIN* and *END* probes are very similar and can be understood from the explanations of those in *count_data_bytes.d*, and the actions of the *send* and *receive* probes use DTrace's *count* function instead of *sum*—as shown here:

```
ip:::send /* Probe for sent packets (by destination address) */
{
    @snd[args[2]->ip_daddr] = count();
    @tot[args[2]->ip_daddr] = count();
}

ip:::receive /* Probe for received packets (by source address) */
{
    @rcv[args[2]->ip_saddr] = count();
    @tot[args[2]->ip_saddr] = count();
}
```

Output from this script on a quiet system may look something like this:

```
nv96-vbox$ ./count_packets.d
dtrace: script './count_packets.d' matched 21 probes
CPU      ID          FUNCTION:NAME
  0       1              :BEGIN
```

```
Counting packets sent and received by IP address...
```

```
-----  
^C  
  0      2                               :END
```

```
-----  
Printing results...
```

```
-----  
Packets sent to:  
  10.0.2.50      4  
  10.0.3.50      4  
  10.0.0.50      8  
  10.0.1.50     12
```

```
Packets received from:  
  10.0.2.50      4  
  10.0.3.50      4  
  10.0.0.50      8  
  10.0.1.50     12
```

```
Total packets received from and sent to:  
  10.0.2.50      8  
  10.0.3.50      8  
  10.0.0.50     16  
  10.0.1.50     24
```

```
nv96-vbox$
```

How these scripts are used from within the web application's server-side Java code is discussed in the following section.

The Back End: Incorporating DTrace with Application Code

With working DTrace scripts written, the next step in implementation is to incorporate those scripts with application code. In the case of this project iteration, the application code is written in Java, so this means using DTrace from a Java class. Fortunately, the OpenSolaris community has developed a Java DTrace library that provides this capability: By including the Java Archive *dtrace.jar* that includes all of the

Java DTrace functionality from the OpenSolaris community, this project's code can utilize DTrace's data-collecting features in its server-side (i.e., back-end) code. Complete documentation for the use of Java DTrace is provided online (*Java DTrace API*, 2007), but the discussion of some of this project's back-end code here will provide an overview of how to use this library as well. This interaction is conceptualized in Figure 4.

The Java class *DtraceCountDataBytesService* in the package *org.dgrp.server.dtraceservices* utilizes the DTrace *count_data_bytes.d* script described in the previous section to provide the data collected by the script to Java classes via a typical Java method interface. This Java class will be described in detail in this section, and the reader is encouraged to refer to the full set of source code in the appendices of this document for the details of other Java classes written for this project iteration. The public interface to this Java class includes no constructor but does include the following methods:

```
public void startService(String subnet);
public boolean isRunning();
public void stopService();
public String[] getBusiestIPsByDataBytes();
```

Each will be described line-by-line along with the other portions of code that make up this Java class.

As is typical in Java code, this Java class is assigned to a package—in this case named *org.dgrp.server.dtraceservices*. The acronym *DGRP* is short for *DTrace Graphical Router Project*; *server* indicates that this is server-side code; and *dtraceservices* is a simple package for those Java classes which provide DTrace-related

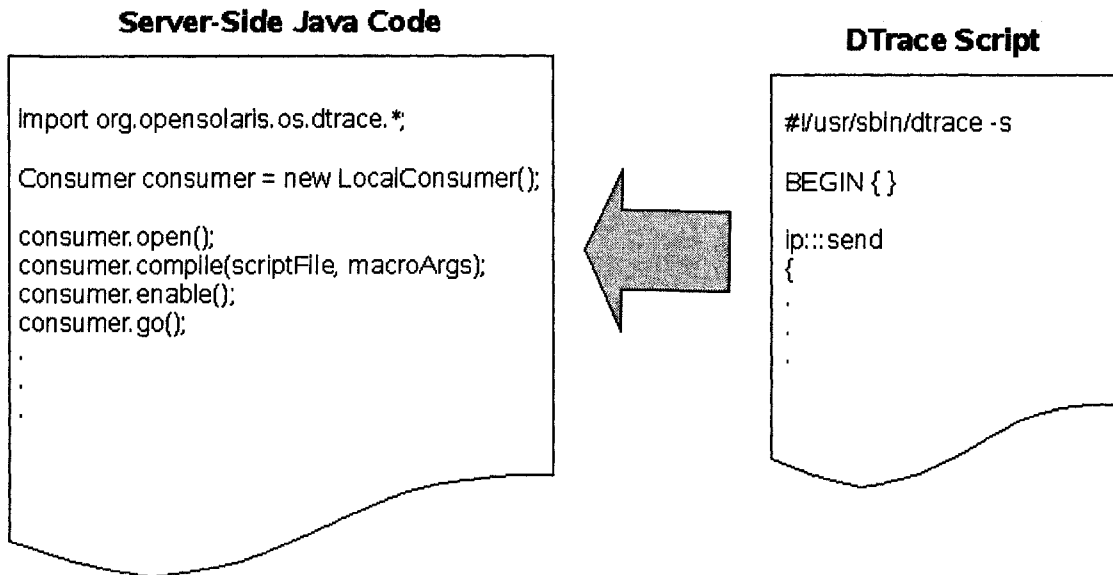


Figure 4. Utilizing DTrace Scripts from Java Classes

The OpenSolaris community provides a Java library that enables Java classes to compile and use standard DTrace scripts.

data (as opposed to some other server-side code which has nothing to do with DTrace and will be discussed in the following section). Following the assignment of this class to a package, several common *import* statements are used to include necessary Java classes from other libraries: *java.io.File*, *java.net.URL*, and *java.util.** are all from the standard Java Development Kit; *org.dgrp.server.DGRPLogger* is a simple class written for this project that provides a rudimentary logging facility for the server-side code; and *org.opensolaris.dtrace.** is from the Java DTrace library written by the OpenSolaris community. These lines of source code are as follows:

```

package org.dgrp.server.dtraceservices;

import java.io.File;

```

```

import java.net.URL;
import java.util.*;
import org.dgrp.server.DGRPLogger;
import org.opensolaris.os.dtrace.*;

```

Following these opening lines of code is the start of the actual class definition—
including a number of private class variables as seen here:

```

public class DTraceCountDataBytesService {

    private URL url = DTraceCountDataBytesService.class.
        GetResource(
            "/org/dgrp/server/dtraceservices/count_data_bytes.d");

    private Consumer consumer;
    private boolean isRunning = false;

```

The variable *url* provides the location of the DTrace script used by this Java class—
count_data_bytes.d in this case—which is also included in the same package. The
variable *consumer* is a DTrace consumer—an object which collects and can provide data
from DTrace according to the interfaces in the Java DTrace library; this variable is used
extensively throughout this Java class as will be apparent in the following lines of source
code. The final private variable, *isRunning*, is a simple boolean variable that is used to
provide a client of this service an indicator of whether or not this service has been started
(i.e., whether or not an instance of this Java class has an active DTrace consumer that is
collecting data).

The first of the public methods in this class is simple enough to be described all at
once. The source is provided here:

```

public void startService(String subnet) {
    try {
        DGRPLogger.log("Entering DTraceCountDataBytesService.
            startService()...\n");
        File scriptFile = new File(url.toURI());

```



```

String macroArgs = new String(subnet);
DGRPLogger.log("Creating DTrace consumer.\n");
consumer = new LocalConsumer();
DGRPLogger.log("Opening DTrace consumer.\n");
consumer.open();
DGRPLogger.log("Compiling DTrace script.\n");
consumer.compile(scriptFile, macroArgs);
DGRPLogger.log("Enabling DTrace consumer.\n");
consumer.enable();
DGRPLogger.log("Starting DTrace consumer.\n");
consumer.go();
isRunning = true;
DGRPLogger.log("Leaving DtraceCoutnDataBytesService.
startService().\n");
}
catch (Exception e) {
    e.printStackTrace();
}
}

```

First, the several *DGRPLogger.log()* function calls make use of the simple logging facility mentioned already in this section to provide some basic log messages. Second, the reader will notice that the private variable *url* is used to create a File object, named *scriptFile*, that provides access to the relevant DTrace script to be used by *consumer*. Third, the string parameter *subnet* is indirectly passed to the the consumer as a parameter of the DTrace script. Finally, the reader can see in this code the typical series of method calls used when starting a DTrace consumer: the creation of a *LocalConsumer* object, *open()*, *compile()*—to which a DTrace script file and its parameters are passed as parameters, *enable()*, and finally *go()*—which starts the consumer collecting data according to the DTrace script. Following the creation, compilation, enabling, and starting of the consumer, the private class variable *isRunning* is updated to indicate that this object's consumer is indeed running, and—of course—the requisite try-catch code is included to manage exceptions thrown during the execution of any of the method calls

(though some additional intelligence in the try-catch code regarding actions for specific exceptions would be a recommended modification in future refactoring of this code.

The next two public methods in the *DTraceCountDataBytesService* Java class are even more straightforward. First, *isRunning()* simply returns the value of the private variable *isRunning*. Second, *stopService()* does essentially the opposite of the *startService()* method just discussed: It stops and closes the DTrace consumer and sets the *isRunning* variable to indicate that the service is no longer running. The code for both methods is provided here:

```
public boolean isRunning() {
    return isRunning;
}

public void stopService() {
    consumer.stop();
    consumer.close();
    isRunning = false;
}
```

The last of the public methods in this class, *getBusiestIPsByDataBytes()*, returns a string array of all of the IP addresses which the DTrace consumer has added to the *@tot* aggregation—sorted according to the number of total data bytes sent to or received from each IP address. (For details about the *@tot* aggregation or other code internal to the DTrace script, refer to the previous section; the sorting mechanism will be discussed later in this section.) The first several lines provide a simple log message and then check to ensure that it is relevant to call this method by ensuring the service is running:

```
public String[] getBusiestIPsByDataBytes() {
    DGRPLogger.log("Entering getBusiestIPsByDataBytes()...\n");

    if (!isRunning()) { //consumer not running, data not available
```

```

        DGRPLogger.log("Consumer not running; returning null from
            getBusiestIPsByDataBytes().\n");
        return null;
    }

```

Next, a series of variables are created and used to get the current *@tot* aggregation from the DTrace consumer and store it in a local *Aggregation* object for processing:

```

    final String totAgg = "tot";
    List ipAddr = new ArrayList();
    Set<String> aggSet = new HashSet();
    aggSet.add(totAgg);
    Aggregation aggregation;
    try {
        DGRPLogger.log("Getting aggregation from consumer...\n");
        aggregation = consumer.getAggregate(aggSet).
            getAggregation(totAgg);
    } catch (Exception e) {
        //consumer is probably not running, return null
        return null;
    }

```

With the aggregation is successfully retrieved from the DTrace consumer, it is first checked for being empty—in which case the method returns immediately rather than attempting to process it:

```

    if (aggregation.equals(null)) {
        return null;
    }

```

If the aggregation is not empty, the code continues to process it. First, a *List* object is created that contains the records from the aggregation. Each record includes a tuple (in this case, a string representation of an IP address) and a value (in this case the number of total bytes sent to and received from the corresponding IP address). The records in the list are sorted according to a custom sorting algorithm, and then the IP addresses from the sorted list's records are put—in order—into a string array for returning. The code for all of these steps—not including the sorting algorithm—is provided here:

```

else { //aggregation exists
    DGRPLogger.log("Aggregation existed...\n");
    List list = aggregation.getRecords();
    Collections.sort(list, new AggRecordComparator());
    Iterator iterator = list.iterator();
    while (iterator.hasNext()) {
        AggregationRecord aggRec = (AggregationRecord)
            iterator.next();
        String ip = (String) aggRec.getTuple().
            iterator().next().getValue();
        ipAdrs.add(ip);
        DGRPLogger.log("Adding IP: " + ip);
        long val = (long) aggRec.getValue().
            getValue().longValue();
        DGRPLogger.log(" (value is " + val + ")\n");
    }
}

String[] ipAdrsStrings = (String[]) ipAdrs.toArray(new
    String[0]);
DGRPLogger.log("Returning from getBusiestIPsByDataBytes()...\n");
return ipAdrsStrings;
} //end of method
} //end of class

```

The sorting algorithm used to sort the aggregation's records is an implementation of the JDK's *Comparator* interface and defines the *compare()* method such that records with a larger value (i.e., number of data bytes) will come before those with a smaller value in the sorted list of aggregation records. The code for this *Comparator* implementation is provided here:

```

class AggRecordComparator implements Comparator {
    public int compare(Object obj1, Object obj2) {
        DGRPLogger.log("Using AggRecordComparator.compare...\n");
        AggregationRecord aggRec1 = (AggregationRecord) obj1;
        AggregationRecord aggRec2 = (AggregationRecord) obj2;
        long val1 = aggRec1.getValue().getValue().longValue();
        long val2 = aggRec2.getValue().getValue().longValue();
        if (val1 < val2)
            return 1;
        else if (val1 == val2)
            return 0;
        else
            return -1;
    }
}

```

```
}
```

Other Java classes in this project that provide DTrace services by utilizing a DTrace script written in the D language follow a pattern similar to *DTraceCountDataBytesService*, and the reader is encouraged to review them in the appendices. Other server-side code that is not DTrace-related is described in the next section.

The Back End: Other Server-Side Code

The previous two sections described the DTrace scripts and DTrace-related Java classes written for this project—all of which are packaged in the *org.dgrp.server.dtraceservices* Java package. This section will describe the other server-side Java classes in the *org.dgrp.server* Java package.

The *DGRPLogger* class—mentioned briefly in the previous section—is a simple logging facility designed to output simple strings to a log file on the server (i.e., the router). The code is simple: There is no constructor; the log file is set in a private string variable; and there is one public method for outputting log messages—*log()*. The code is provided here:

```
package org.dgrp.server;
import java.io.*;

public class DGRPLogger {
    private static String logfile = "/var/tmp/dgrplog.txt";

    public static void log(String string) {
        try {
```

```

        BufferedWriter out = new BufferedWriter(new
            FileWriter(logfile, true));
        out.write(string);
        out.close();
    } catch (IOException e) { //ignore
    }
}
}

```

Also in the *org.dgrp.server* package is the *GetVersionInfoImpl* class—the server-side class in a set of classes that follow a strict pattern provided by the Google Web Toolkit for Remote Procedure Calls. How the various client-side and server-side pieces of the different RPCs fit together will be discussed in the *RPCs: Tying the Front and Back Ends Together* section (p. 57), so the following comments will deal only with explaining what the server-side code in this class does—not how it interacts with the client-side classes.

First, necessary package and import statements are made and the class is defined:

```

package org.dgrp.server;

import java.io.*;
import java.util.*;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import org.dgrp.client.GetVersionInfo;
import org.dgrp.client.VersionContents;
import java.net.URL;

public class GetVersionInfoImpl extends RemoteServiceServlet
    implements GetVersionInfo {

```

Refer to the *RPCs* section (p. 57) for details of *RemoteServiceServlet* statements and the *org.dgrp.client.GetVersionInfo* class, and refer to the *Front End* section (p. 48) for details of the *org.dgrp.client.VersionContents* class. Following these opening lines of code, a single public method is defined—*getVersionInfo()*. The complete source of that method is provided here with explanations following:

```

public VersionContents getVersionInfo() {

    DGRPLogger.log("Entering getVersionInfo()...\n");

    VersionContents ver = new VersionContents();

    InputStream in = null;
    Properties props = new Properties();

    try {
        in = getClass().getResourceAsStream
            ("/appinfo.properties");
        props.load(in);

        //Solaris info
        ver.solarisRelease = getSolarisRelease();
        ver.solarisInstallDate = getSolarisInstallDate();
        ver.solarisArch = System.getProperty("os.arch");
        ver.solarisUptime = getSolarisUptime();

        //Quagga info
        ver.quaggaVersion = getQuaggaVersion();
        ver.quaggaInsDate = getQuaggaInstallDate();

        //This software info
        ver.dgrpAuthor = props.getProperty("program.AUTHOR");
        ver.dgrpBuildDate = props.
            getProperty("program.BUILDDATE");
        ver.dgrpBuildNumber = props.
            getProperty("program.BUILDNUM");
        ver.dgrpDescription = props.
            getProperty("program.DESCRPTION");
        ver.dgrpVersion = props.
            getProperty("program.VERSION");

        //Java info
        ver.javaVMName = System.getProperty("java.vm.name");
        ver.javaVMVendor = System.
            getProperty("java.vm.vendor");
        ver.javaVMVersion = System.
            getProperty("java.vm.version");
        ver.javaVendor = System.getProperty("java.vendor");
        ver.javaVersion = System.getProperty("java.version");

        //Browser info
        ver.browserInfo = null; //determined client-side

        //Web-Server info
        ver.tomcatVersion = getTomcatVersion();
        ver.apacheVersion = getApacheVersion();
    }
}

```

```

        //Remove this test
        ver.removeThis = removeThisMethod();

        in.close();
    }
    catch (IOException e) {
        e.printStackTrace();
    }

    DGRPLogger.log("Returning from getVersionInfo().\n");
    return ver;
}

```

First, an object of the client-side class *VersionContents* is created that will be populated with all of the version info retrieved from the server and will then be returned at the end of the method. The rest of the method—up to the point of returning—is a series of method calls used to populate bits of version information in the *VersionContents* object. Rather than describing each line of code in the remainder of the class, the three major forms of retrieving version information will be described by example, and the reader is encouraged to review the full source of this class in the appendices for further detail. The first of the three methods by which version information is retrieved is via the *props.getProperty()* method. *props* is created near the start of the class code and is an object of the *Properties* class that refers to the file *appinfo.properties*—which is modified by the build process when the whole web application is compiled, built, and packaged for deployment. This file follows the format of a properties file according to the *Properties* class specification in the JDK, thus *props.getProperty()* can retrieve values from name-value pairs in this file by passing the name as a parameter to the method—as seen in lines like the following:

```

ver.dgrpAuthor = props.getProperty("program.AUTHOR");

```



```
ver.dgrpBuildDate = props.getProperty("program.BUILDDATE");  
ver.dgrpBuildNumber = props.getProperty("program.BUILDNUM");
```

The second of the three methods is the retrieval of system properties from the Java Virtual Machine (JVM) via the *System.getProperty()* method call. This is similar to the properties method discussed already except that the source of these properties is the JVM itself rather than a properties file. Examples of this method can be seen in lines like these:

```
ver.javaVMName = System.getProperty("java.vm.name");  
ver.javaVMVendor = System.getProperty("java.vm.vendor");  
ver.javaVMVersion = System.getProperty("java.vm.version");
```

The last of the three methods for retrieving version information from the server is through a series of private methods also defined in this class. These methods all follow a similar pattern: Either open a file or execute a command and then extract the relevant text from the output for return and eventual placement into the VersionContents object. Examples of these method-calls can be seen here:

```
ver.solarisRelease = getSolarisRelease();  
ver.solarisInstallDate = getSolarisInstallDate();  
ver.solarisUptime = getSolarisUptime();
```

The code that defines each of these methods is very basic, so the reader is encouraged to simply review it in its entirety in the appendices of this document.

Finally, the *BandwidthMonitorImpl* Java class in the *org.dgrp.server* package defines the server-side part of another RPC—the RPC by which the client-side code requests information from the DTrace services running on the server (i.e., router). As will become obvious to the reader in the next section, the client-side code does not interact directly with the classes in the *org.dgrp.server.dtraceservices* package; rather, the

server-side code of the RPCs starts, stops, and gets updates from the DTrace services.

(This whole interaction will be described more fully in the *RPCs: Tying the Front and Back Ends Together* section, p. 57.)

Looking more closely at the code, the reader will see that the package and import statements are as expected and that the class definition is similar to other server-side classes in other RPCs:

```
package org.dgrp.server;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.util.Random;
import org.dgrp.client.BandwidthInfo;
import org.dgrp.client.BandwidthMonitor;
import org.dgrp.server.dtraceservices.*;

public class BandwidthMonitorImpl extends RemoteServiceServlet
    implements BandwidthMonitor {

    private DTraceCountDataBytesService countDataBytesService;
    private DTraceCountPacketsService countPacketsService;
```

The reader will also notice that there are two private objects created—one of each of the DTrace service classes in the *org.dgrp.server.dtraceservices* package previously described. The remaining methods in this class utilize these objects so that the client-side code need not have any knowledge of them; the client-side code need only be concerned with the interfaces defined for each RPC. For each service, there are start, stop, and other relevant methods defined—as in the examples here:

```
public void startServiceCountDataBytes(String subnet) {
    DGRPLogger.log("Entering BandwidthMonitorImpl.
        startServiceCountDataBytes()...\n");
    countDataBytesService = new DTraceCountDataBytesService();
    countDataBytesService.startService(subnet);
}

public void stopServiceCountDataBytes() {
    DGRPLogger.log("Entering BandwidthMonitorImpl.
```

```

        stopServiceCountDataBytes()...\n");
        countDataBytesService.stopService();
    }

    public String[] getRefreshedIPs() {
        DGRPLogger.log("Entering BandwidthMonitorImpl.
            getRefreshedIPs()...\n");
        return countDataBytesService.getBusiestIPsByDataBytes();
    }

```

While these examples (and most of the methods in this class in this iteration of the project) do little more than call and return methods directly from the DTrace services classes, this design offers the flexibility to implement more sophisticated wrapper methods or to change the implementation of the DTrace services classes without necessitating an alteration to the RPC interface on which the client-side code depends. For the source of all of the methods in this Java class, the reader is encouraged to refer to the appendices.

The Front End: Developing the User Interface

While the previous two sections described server-side or back-end code in the *org.dgrp.server* and *org.dgrp.server.dtraceservices* packages, this section will discuss the front-end, client-side code in the *org.dgrp.client* package. Many of the Java classes in this package utilize the various widgets provided by the Google Web Toolkit to create the graphical interface through which the user interacts with this web application. Other classes are used in the process of analyzing and processing the data provided by the server-side code, and still others are responsible for the client-side portion of the RPCs that communicate between the front and back end. One of the key classes used for

creating the graphical interface, *DGRPEntryPoint*, will be described in detail, and the reader is encouraged to view the complete source code for this class in the appendices and to refer to the available GWT documentation to understand the rest of the code in similar classes (*Google Web Toolkit, 2007*). The analysis classes will be described in detail, and the classes related to the RPCs will be covered in the next section, *RPCs*:

Tying the Front and Back Ends Together.

The graphical interface for this iteration of this project is created entirely by the use of GWT widgets. Examples of the creation, placement, and modification of these widgets can be found in this project source code and in the examples included with the GWT (see *Using the Application Framework*). Various types of panels—one of the GWT widgets—make up the conceptual map that lays out the graphics in the interface. Panels are included within panels, and the base panel is defined in the *DGRPEntryPoint* class.

DGRPEntryPoint imports a number of necessary widget classes from the *com.google.gwt.user.client* package and is defined to implement the *EntryPoint* and *HistoryListener* interfaces as shown here:

```
package org.dgrp.client;

import org.dgrp.client.SidebarItem.SidebarItemInfo;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.History;
import com.google.gwt.user.client.HistoryListener;
import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.HasAlignment;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.VerticalPanel;

public class DGRPEntryPoint implements EntryPoint,
    HistoryListener {
```

As the entry point Java class, GWT configures the web application when compiling and building it to start the loading and display of the application from this class; all other graphics are initiated from this class. As an implementation of *HistoryListener*, this class enables browser history to work correctly with the AJAX application. The class then creates a number of object instances and defines the *onHistoryChanged()* function according to the *HistoryListener* interface:

```
public DGRPEntryPoint() {
    }
    protected Sidebar list = new Sidebar();
    private SidebarItemInfo curInfo;
    private SidebarItem curItem;
    private HTML description = new HTML();
    private DockPanel panel = new DockPanel();
    private DockPanel mainPanel;

    public void onHistoryChanged(String token) {
        SidebarItemInfo info = list.find(token);
        if (info == null) {
            showInfo();
            return;
        }
        show(info, false);
    }
}
```

The next method defined, *onModuleLoad()*, is responsible for the layout of the panels on the browser that make up the graphical interface. Since this is in the entry point Java class, this method is called almost immediately when a user points the browser to the web application's URL. Utilizing this method and the others called from it, the client-side code lays out an interface that looks like that in Figure 5.

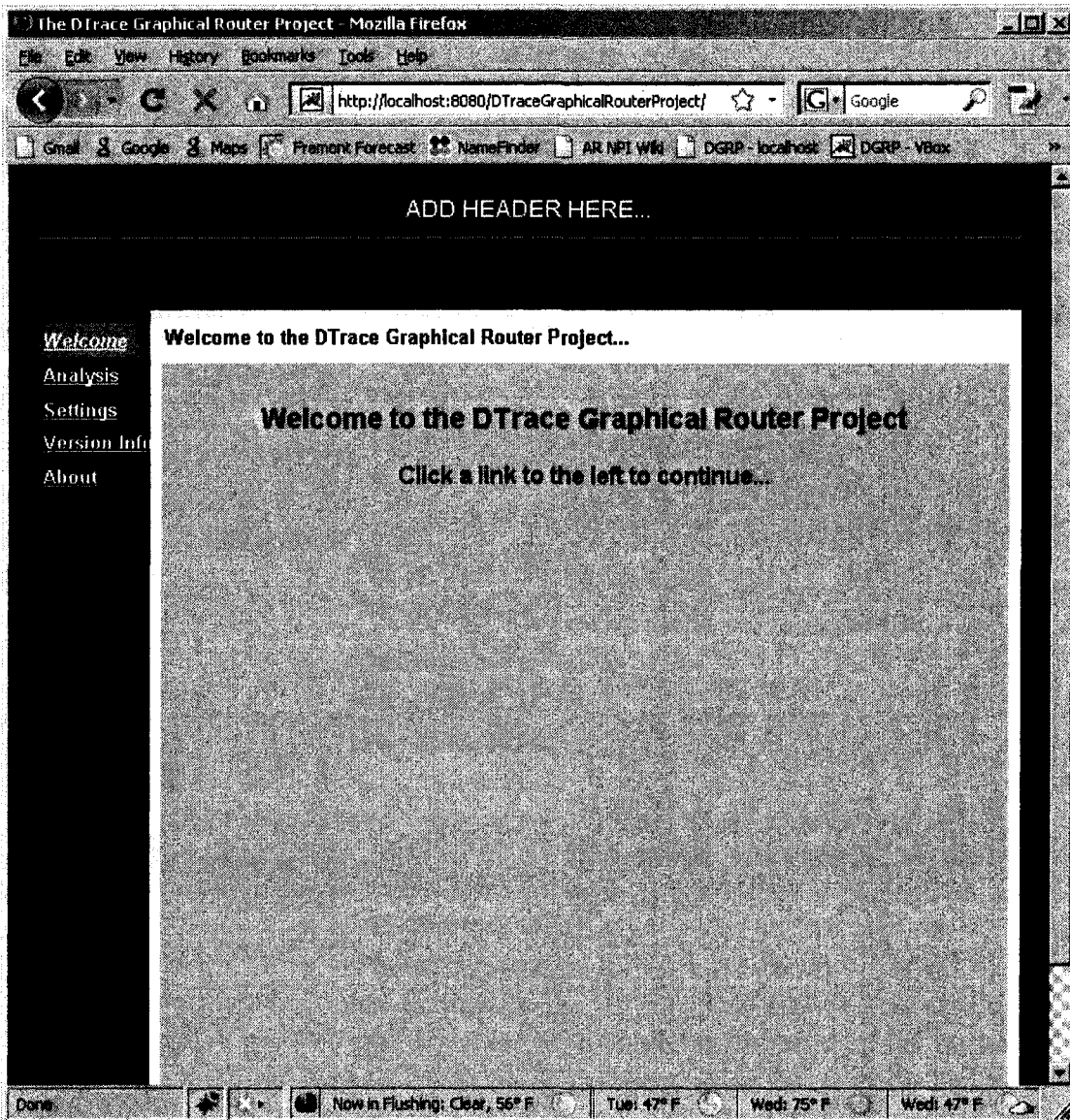


Figure 5. Basic Interface Layout

This is an example of the layout created by the entry point Java class.

The first step in this process is adding items to the *Sidebar* object named *list*. This occurs in the following method call:

```
loadSidebarItems();
```

This method is defined as follows:

```
protected void loadSidebarItems() {  
    list.addItem(Welcome.init());  
    list.addItem(Analysis.init());  
    list.addItem(Settings.init());  
    list.addItem(Version.init());  
    list.addItem(About.init());  
}
```

These *addItem()* method calls create instances of each of the classes shown: *Welcome*, *Analysis*, *Settings*, *Version*, and *About*—all of which are extensions of the abstract *SidebarItem* class. This is responsible for creating the links in the left-hand side of the interface shown in Figure 5. The next several lines of code create other panels, set their styles, and add them in the appropriate order:

```
mainPanel = new DockPanel();  
mainPanel.setStyleName("dgrp-MainPanel");  
  
VerticalPanel vp = new VerticalPanel();  
vp.setWidth("100%");  
vp.add(description);  
vp.add(mainPanel);  
  
description.setStyleName("dgrp-Heading");  
  
panel.add(list, DockPanel.WEST);  
panel.add(vp, DockPanel.CENTER);  
  
panel.setCellVerticalAlignment(list, HasAlignment.ALIGN_TOP);  
panel.setCellWidth(vp, "100%");  
panel.setCellHeight(vp, "100%");  
  
History.addHistoryListener(this);  
RootPanel.get().add(panel);
```

Each call to *setStyleName()* assigns one of the Cascading Style Sheets (CSS) styles included in the project to the object. Working essentially backwards through the other lines, the main *DockPanel* widget, *panel*, is added to GWT's default *RootPanel*; *panel*

includes the sidebar menu object, *list*, on the left and a *VerticalPanel*, *vp*, which in turn includes an *HTML* object, *description*, at the top and then *mainPanel* beneath that. In Figure 5, *description* can be seen as the space containing the text, “Welcome to the DTrace Graphical Router Project...,” while *mainPanel* contains the repeat of that welcome with the additional text, “Click a link to the left to continue...” By clicking a link in the sidebar, the user invokes the next method defined in this class—*show()*. The *show()* method quite simply tells the sidebar object to highlight the selected choice, updates the text in *description*, removes the current widget from *mainPanel*, and loads the selected item into *mainPanel* instead. The code for this is relatively straightforward:

```
public void show(SidebarItemInfo info, boolean affectHistory) {
    if (info == curInfo) {
        return;
    }
    curInfo = info;

    if (curItem != null) {
        curItem.onHide();
        mainPanel.remove(curItem);
    }

    curItem = info.getInstance();
    list.setItemSelection(info.getName());
    description.setHTML(info.getDescription());

    if (affectHistory) {
        History.newItem(info.getName());
    }

    mainPanel.add(curItem, DockPanel.CENTER);
    mainPanel.setCellWidth(curItem, "100%");
    mainPanel.setCellHeight(curItem, "100%");
    mainPanel.setCellVerticalAlignment(curItem,
        DockPanel.ALIGN_TOP);
    curItem.onShow();
}
```

Given the explanation of *DGRPEnterPoint* and the source code available in the

appendices, the reader should be well-equipped to understand the *Sidebar*, *SidebarItem*, *About*, *Welcome*, and *Settings* classes as well.

Two other classes in the *org.dgrp.client* package, *AnalysisMenu* and *ImagePanel*, are also similar to the classes just discussed. They involve code primarily responsible for the creation, layout, and modification of widgets, though there are a couple of things in each worth special mention here. First, *AnalysisMenu* creates a menu of choices for the user when the *Analysis* sidebar options is clicked. The menu is like that in Figure 6, and each selection in the menu corresponds to a command that triggers other code to execute. An example of a command that presents the user with an informational window warning that the selected feature is not yet implemented is shown in the following code:

```
Command notSupported = new Command () {
    public void execute() {
        Window.alert("This feature is not yet supported.");
    }
};
```

By passing this *Command* object as a parameter in the creation of a *MenuItem*, as in

```
menu_general_int_status = new MenuItem(notsup +
    "Interface Status", true, notSupported);
```

an option in the menu is created that will execute the *notSupported* command when selected. Similar to the *AnalysysMenu* class, *ImagePanel* is used by the *Analysis* class when it is selected in the sidebar. *ImagePanel* controls the layout of a number of other graphics used by the *Analysis* class—several of which can be seen in Figure 7.

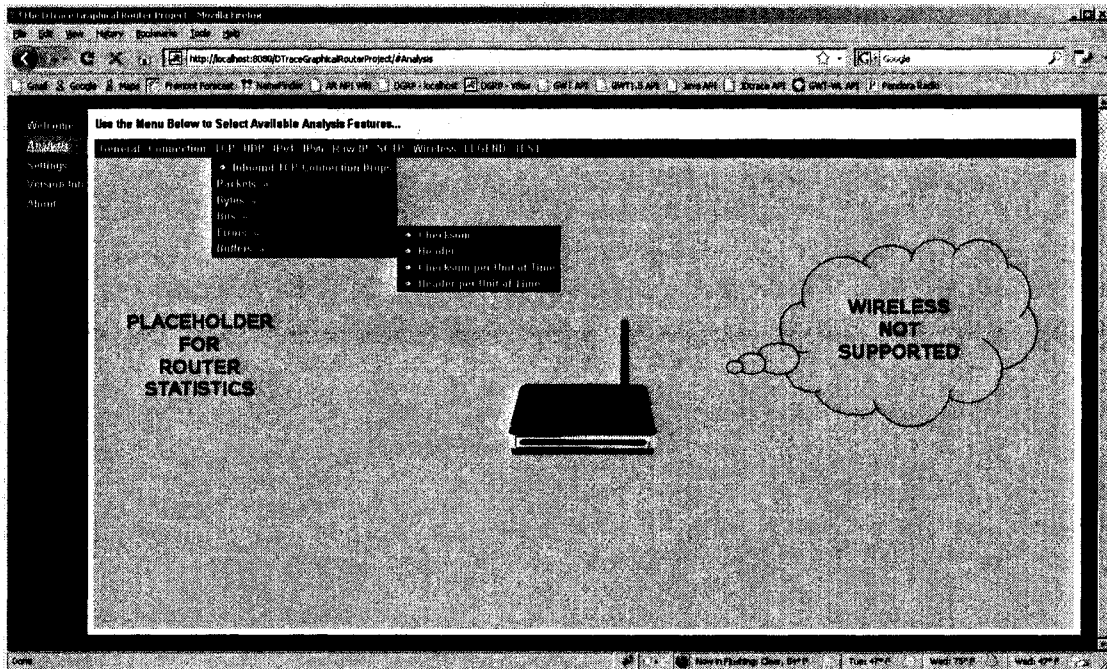


Figure 6. Application Interface Analysis Menu

The *AnalysisMenu* class creates a menu of choices during analysis.

ImagePanel also provides a number of methods that allow other code to control the visibility of or otherwise edit the graphics in this class. For example, the following method, *hideLaptop()*, is used to hide the laptop graphics seen in Figure 7:

```
public void hideLaptop(int position) {
    switch (position) {
        case 0:
            laptop0.setUrl("images/placeholder.png");
            laptop0.setWidth("131px");
            laptop0.setHeight("104px");
            laptop0.setStyleName("dgrp-Images-Image");
            break;
        case 1:
            laptop1.setUrl("images/placeholder.png");
            laptop1.setWidth("131px");
            laptop1.setHeight("104px");
            laptop1.setStyleName("dgrp-Images-Image");
            break;
        case 2:
            laptop2.setUrl("images/placeholder.png");
    }
}
```

```

        laptop2.setWidth("131px");
        laptop2.setHeight("104px");
        laptop2.setStyleName("dgrp-Images-Image");
        break;
    case 3:
        laptop3.setUrl("images/placeholder.png");
        laptop3.setWidth("131px");
        laptop3.setHeight("104px");
        laptop3.setStyleName("dgrp-Images-Image");
        break;
    default:
        break; //ignore others for now
}
}

```

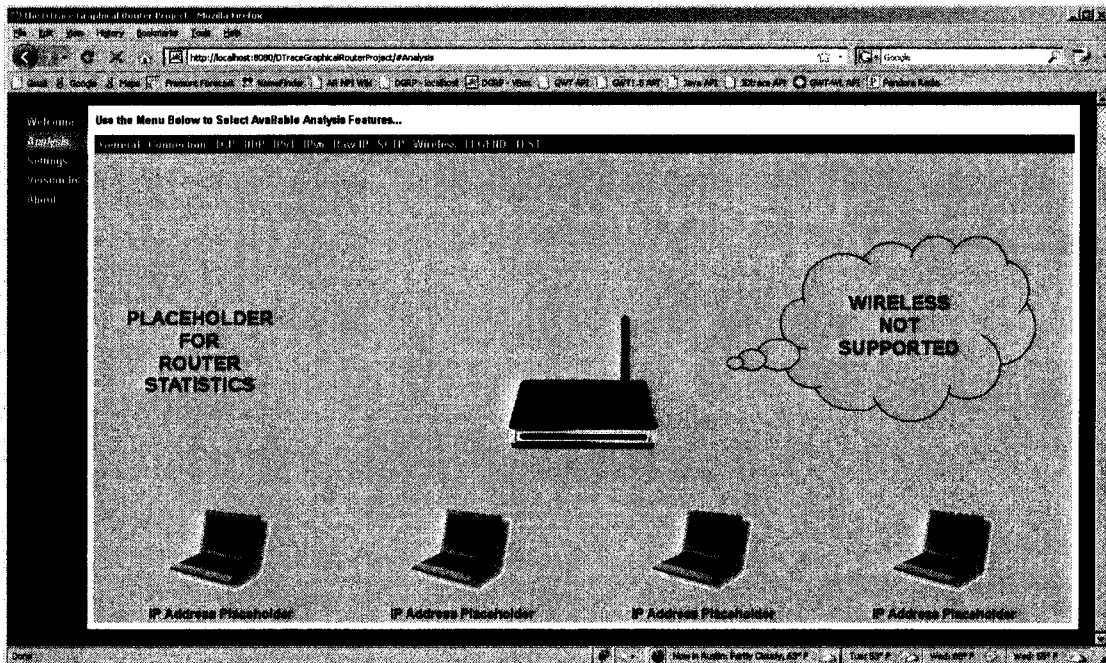


Figure 7. Graphics Controlled by the ImagePanel Class

The *ImagePanel* class provides controls for many of the graphics used by the *Analysis* class.

The rest of the code in *AnalysisMenu* and *ImagePanel* should be understandable given the detailed description of *DGRPEnterPoint* and the complete source code in the appendices.

Of the remaining classes in *org.dgrp.client*, three are simple and should be

understood by the reader without any special explanation: *TopologyInfo* objects are used to determine the placement and keep track of which IP addresses appear on the analysis graphics (where the “IP Address Placeholder” text appears in Figure 7); *VersionContents* contains a number of public string objects that are used to pass information regarding software versions from the server to the client when the *Version* link is chosen from the sidebar; and *BandwidthInfo* objects are used to communicate how much available bandwidth is being used by a given connection. The remaining classes in this package either directly utilize or are a necessary part of Remote Procedure Calls and will thus be described in the next section.

RPCs: Tying the Front and Back Ends Together

Much has been discussed in the previous sections about creating DTrace scripts, utilizing those scripts from server-side Java code, and creating the client-side graphical interface, but the real power of this software comes from the tying together of these pieces: By enabling the front-end code to get information from the back-end code and act accordingly, the web application is enabled to provide useful and current information. This is achieved through Remote Procedure Calls; see Figure 8. Two of the sidebar choices not yet discussed—*Version* and *Analysis*—will be described in detail here along with the corresponding RPCs through which each class is able to get useful information from the server-side code.

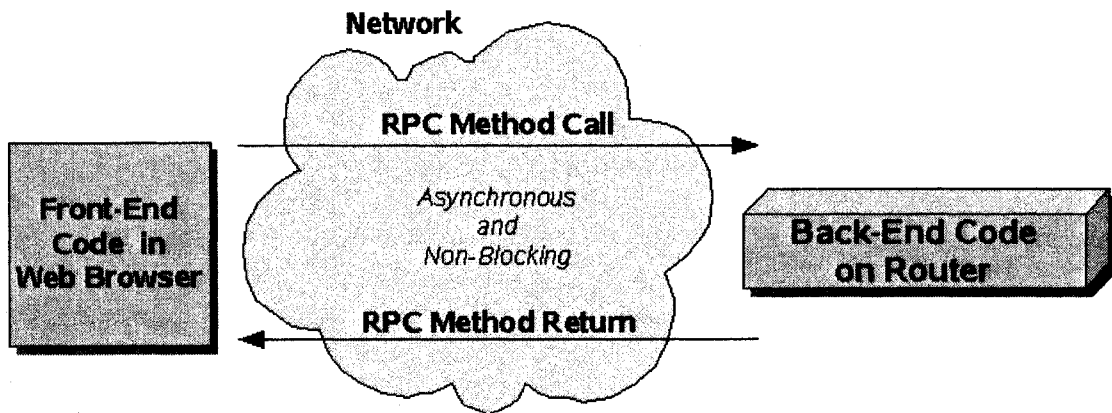


Figure 8. RPCs

RPCs provide a way of allowing code interaction between the client-side code and the server-side code through asynchronous method calls.

As the simpler example, the *Version* class will be described first. The goal of the *Version* link in the interface sidebar is simple: Provide the user with version information relevant to this web application. Of course, much of the relevant software is outside the control of this web application, so its versions must be retrieved from the server (i.e., router). In the code, like other *SidebarItem* classes, necessary package and import statements are included, and an *init()* function is defined:

```
package org.dgrp.client;

import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;

public class Version extends SidebarItem {

    private HTML verInfo = new HTML(
        "<div class='dgrp-About-Prose'>" +
        "Retrieving version information from the server..." +
```

```

        "</div>",
        true);

    public static SidebarItemInfo init() {
        return new SidebarItemInfo("Version Info",
            "Version Information for the DTrace Graphical Router
            Project...") {
            public SidebarItem createInstance() {
                return new Version();
            }
        };
    }
}

```

As seen, a placeholder *HTML* object is also created and used in the constructor to display an initial message to the user—as seen here:

```

public Version() {
    initWidget(verInfo);
}

```

The constructor then creates an asynchronous callback object; this will be used to react to the return of the RPC once it is made. It is important to remember that the GWT RPC implementation is *asynchronous*, thus when an RPC call is made, the code continues to execute without blocking until the RPC returns—at which point the code in the callback object will be executed according to the success (execute *onSuccess()*) or failure (execute *onFailure()*) of the RPC. In the case of *Version*, *onSuccess()* is defined to update the *HTML* object with the version information returned from the server—as seen partially here:

```

final AsyncCallback callback = new AsyncCallback() {
    public void onSuccess(Object result) {

        VersionContents verResults = (VersionContents) result;

        verInfo.setHTML(
            .
            .
            .
        );
    }
}

```

```

    }

    public void onFailure(Throwable caught) {
        verInfo.setHTML(
            "<div class='dgrp-About-Prose'>" +
            "Failed to retrieve version information from the " +
            "server.</div>"
        );
    }
};

```

In the case of an RPC failure, the *HTML* object is updated to display an appropriate failure message. The result of a successful RPC call can be seen in Figure 9. Following the definition of the callback object, the RPC call can actually be made—as in the following method call

```
getService().getVersionInfo(callback);
```

where *getService()* is defined as

```

public static GetVersionInfoAsync getService(){
    GetVersionInfoAsync service = (GetVersionInfoAsync)
        GWT.create(GetVersionInfo.class);
    ServiceDefTarget endpoint = (ServiceDefTarget) service;
    String moduleRelativeURL = GWT.getModuleBaseURL() +
        "getversioninfo";
    endpoint.setServiceEntryPoint(moduleRelativeURL);
    return service;
}

```

This method refers to the GWT-prescribed configuration of this RPC as a servlet in the web application's *web.xml* file—as seen here:

```

<servlet>
    <servlet-name>GetVersionInfo</servlet-name>
    <servlet-class>
        org.dgrp.server.GetVersionInfoImpl
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>GetVersionInfo</servlet-name>
    <url-pattern>
        /org.dgrp.DTraceGraphicalRouterProject/getversioninfo

```

```
        </url-pattern>
    </servlet-mapping>
```

One last useful point that can be taken from the *Version* class is seen in the following

method:

```
    public static native String getBrowserInfo() /*- {
        return $wnd.navigator.userAgent;
    }-*/;
```

This method shows an example of how raw JavaScript can be used from within client-side GWT code. The exact syntax is required, but this makes it possible to do things with JavaScript that the GWT cannot do, though the need for this was quite sparse in the course of this project iteration.

For the RPC in the *Version* class to work correctly, two other classes must also be defined. Recall that the RPC called the method *getVersionInfo()*. This prototype for this method is in the class, *GetVersionInfo*—as seen in the code here:

```
package org.dgrp.client;
import com.google.gwt.user.client.rpc.RemoteService;

public interface GetVersionInfo extends RemoteService {
    public VersionContents getVersionInfo();
}
```

According to the GWT RPC implementation, another—almost identical—class must also be defined: In this case, that class is *GetVersionInfoAsync*:

```
package org.dgrp.client;
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface GetVersionInfoAsync {
    public void getVersionInfo(AsyncCallback callback);
}
```

Finally, the actual implementation of the *getVersionInfo()* method is defined in the class *GetVersionInfoImpl* in the server-side package *org.dgrp.server*. For a discussion of this

class, refer to the *The Back End: Other Server-Side Code* section (p. 42). So, in order for the *Version* class to use an RPC, an asynchronous callback object must be created and passed to the RPC call—which uses a servlet configured in `web.xml` to call a method prototyped in *GetVersionInfo* and *GetVersionInfoAsync* and actually implemented in the server-side class *GetVersionInfoImpl*. Note that these names were not arbitrary but were chosen according to the requirements of the GWT RPC (*Remote Procedure Calls*, 2008). Note also that the GWT4NB plugin to the NetBeans IDE automatically configures the `web.xml` file and creates templates for the necessary Java classes—both on the client and server side—greatly simplifying the creation of RPCs.

The RPC used by the *Analysis* class is very similar in concept to that used by the *Version* class, though the interface classes highlight one important point—that a single RPC implementation can accommodate multiple method definitions—as seen here from the *BandwidthMonitor* and *BandwidthMonitorImpl* interfaces:

```
public interface BandwidthMonitor extends RemoteService{
    public void startServiceCountPackets(String s);
    public void startServiceCountDataBytes(String s);
    public void stopServiceCountPackets();
    public void stopServiceCountDataBytes();
    public BandwidthInfo getBandwidthInUse(String s);
    public BandwidthInfo getRandomBandwidthInUse(String s);
    public String[] getRefreshedIPs();
}
```

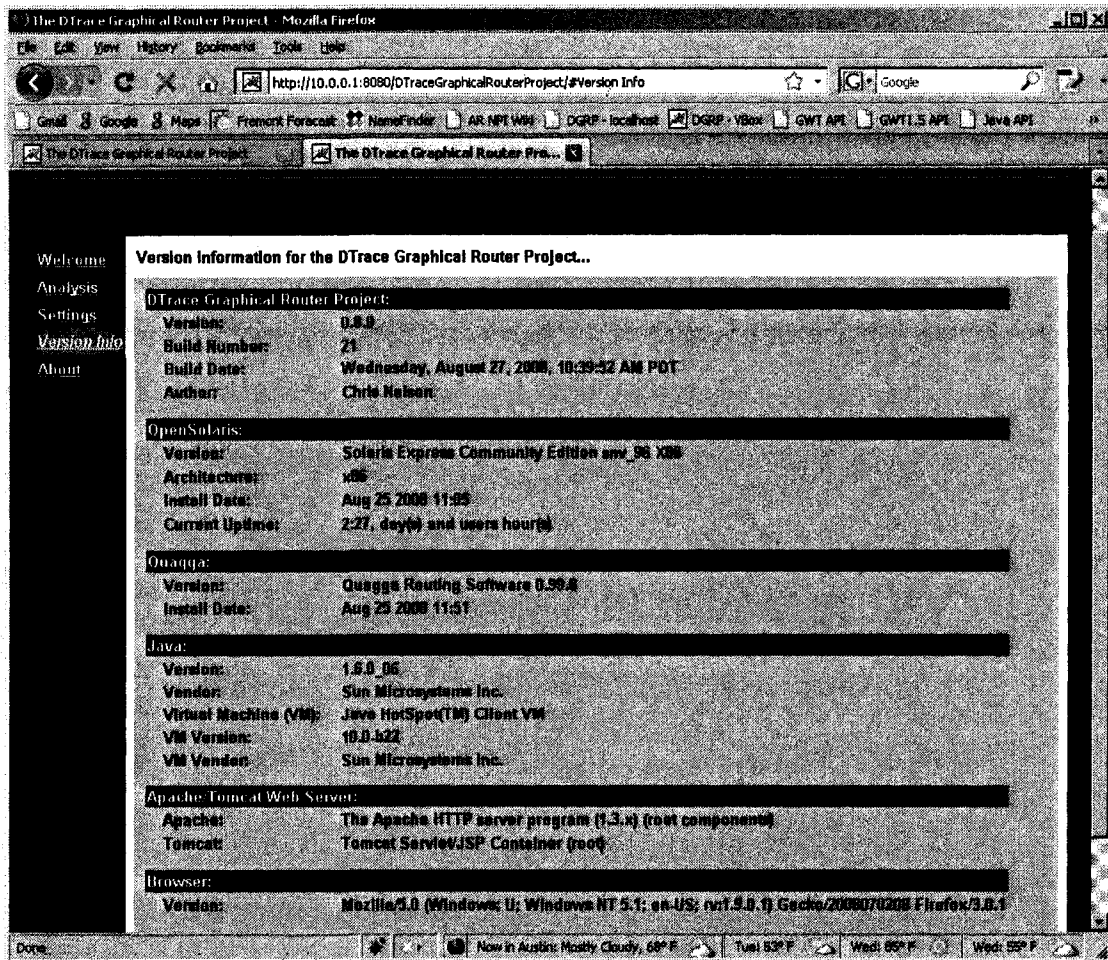


Figure 9. Results of a Successful Version RPC

In the Version class, a successful RPC return provides version information from the server.

```
public interface BandwidthMonitorAsync {
    public void startServiceCountPackets(String s, AsyncCallback
        AsyncCallback);
    public void startServiceCountDataBytes(String s, AsyncCallback
        AsyncCallback);
    public void stopServiceCountPackets(AsyncCallback
        AsyncCallback);
    public void stopServiceCountDataBytes(AsyncCallback
        AsyncCallback);
    public void getBandwidthInUse(String s, AsyncCallback
```

```

        callback);
    public void getRandomBandwidthInUse(String s, AsyncCallback
        callback);
    public void getRefreshedIPs(AsyncCallback callback);
}

```

The implementation of these methods in the *BandwidthMonitorImpl* class in *org.dgrp.server* has already been described in the *The Back End: Other Server-Side Code* section (p. 42). Outside of the use of these RPCs in the *Analysis* class—which the reader can now undoubtedly understand by reviewing the full source code in the appendices, a few other parts of the class are worth describing here.

First, GWT's *Timer* class is used to make RPC calls on a regular and repeated interval—as seen here:

```

bwMonitorService.getRefreshedIPs(ipCallback);
Timer ipRefresh = new Timer() {
    public void run() {
        bwMonitorService.getRefreshedIPs(ipCallback);
    }
};
ipRefresh.scheduleRepeating(10000);

```

In this code, the RPC *getRefreshedIPs()* is called once, and a timer is then created which will trigger the same RPC to be called again every 10,000 milliseconds—or 10 seconds. The definition of the *ipCallback* parameter describes what this client-side code will do with the result of this RPC:

```

final AsyncCallback ipCallback = new AsyncCallback() {
    public void onSuccess(Object result) {
        processIPUpdates(result);
    }
    public void onFailure(Throwable caught) {
        //ignore for now
    }
};

```

Looking then to the definition of the *processIPUpdates()* function,

```

private void processIPUpdates(Object result) {
    String[] newAddrs = (String[]) result;
    for (int i=0; i<MAX_NODES; i++) {
        imgPanel.hideLaptop(i);
        imgPanel.hideLaptopPipe(i);
        imgPanel.setLaptopIPAddrLabel(i, null);
    }
    topoInfo = new TopologyInfo(MAX_NODES);
    for (int i=0; i<newAddrs.length; i++) {
        try {
            topoInfo.setAddress(i, newAddrs[i]);
            imgPanel.setLaptopIPAddrLabel(i, newAddrs[i]);
            imgPanel.showLaptop(i);
        } catch (Exception e) {
            //ignore for now
        }
    }
}

```

it can be seen that the graphical interface is updated once every 10 seconds to show laptop graphics on *imgPanel*—up to a maximum number (*MAX_NODES*)—and to display an IP address label for each each laptop graphic according to the array of strings that is returned by the server-side RPC code for *getRefreshedIPs()*. Similarly, the following code shows a once-per-second update to the pipe graphics displayed for each laptop:

```

Timer pipeUpdate = new Timer() {
    public void run() {
        for (int i=0; i<=topoInfo.getMaxNodes(); i++) {
            if (!(topoInfo.getAddress(i).equals(null))) {
                bwMonitorService.getBandwidthInUse(
                    topoInfo.getAddress(i), pipeCallback);
            }
        }
    }
};
pipeUpdate.scheduleRepeating(1000);

final AsyncCallback pipeCallback = new AsyncCallback() {
    public void onSuccess(Object result) {
        processPipeUpdates(result);
    }
}

```

```

        public void onFailure(Throwable caught) {
            //ignore for now
        }
    };

    private void processPipeUpdates(Object result) {
        BandwidthInfo bwInfo = (BandwidthInfo) result;
        try {
            imgPanel.showLaptopPipe(topoInfo.findPosition(
                bwInfo.getIPAddress()), bwInfo.getBandwidthInUse());
        } catch (Exception e) {
            //skip addresses not currently tracked
        }
    }
}

```

With this explanation of the *Version* and *Analysis* classes, it should be clear to the reader how client-side code can interact with server-side code via RPCs to retrieve information and act accordingly. With all of these pieces together, the final step in implementation is actually creating and deploying the complete web application—which is discussed in the following section.

Deploying the Complete Web Application

With all of the pieces discussed in this and the previous chapters in place, a few final things are necessary to bring them all together in a complete web application. Note that—where noted—the NetBeans IDE provided the author a simplified process that may be more complicated in a different development environment. These items are provided in the following list in no particular order:

1. To version-control the web application, code was added to the *build.xml* file in the NetBeans project directory:

```

<target name="-pre-dist">
  <buildnumber file="buildnumber.properties"/>
  <propertyfile file="appinfo.properties"
    comment="Everything can be manually updated except
    buildnum and builddate.">
    <entry key="program.PROGNAME" default="${main.class}" />
    <entry key="program.AUTHOR" default="" />
    <entry key="program.COMPANY" default="" />
    <entry key="program.COPYRIGHT" default="now" type="date"
      pattern="yyyy" />
    <entry key="program.DESCRPTION" default="" />
    <entry key="program.VERSION" default="1.0.0" />
    <entry key="program.BUILDNUM" value="${build.number}" />
    <entry key="program.BUILDDATE" type="date" value="now"
      pattern="EEEE, MMMM dd, yyyy, hh:mm:ss a z" />
  </propertyfile>
  <copy file="appinfo.properties"
    todir="${build.classes.dir}"/>
</target>

```

A new file, *appinfo.properties*, was also created in the same directory:

```

program.PROGNAME=The DTrace Graphical Router Project
program.BUILDNUM=22
program.AUTHOR=Chris Nelson
program.BUILDDATE=Tuesday, September 16, 2008, 07\:49\:55 PM PDT
program.DESCRPTION=See the About page in the web application.
program.COPYRIGHT=2008
program.VERSION=0.9.0
program.COMPANY=San Jose State University

```

With this in place, the `program.BUILDNUM` and `program.BUILDDATE` properties are updated automatically with each build of the web application (*HOWTO: use ANT with JAVA to dynamically create build numbers, 2007*).

2. An `index.jsp` file is included in the web application and is the default page loaded when a user points a browser to the web application root address. This page includes only a simple Hypertext Markup Language (HTML) header, a pointer to the web application's CSS file, and the necessary JavaScript entry to load the

GWT AJAX code:

```
<script language="javascript" src="org.dgrp.DTraceGraphicalRouter  
Project/org.dgrp.DTraceGraphicalRouterProject.nocache.js">  
</script>
```

3. All of the images used in the web application are included in an *images* directory.

4. Code run by a web application served by Apache Tomcat's web server on the Solaris operating system executes as user nobody by default. Because of the detail of information that DTrace can provide, Solaris—by default—only allows the root super-user to utilize the full set of DTrace probes. To give user nobody permission to use all of DTrace's functionality, the following line was added to the file */etc/user_attr*:

```
nobody::::defaultpriv=basic,dtrace_kernel
```

5. NetBeans provides the ability to build and package a web application into a Java Archive .war file that can be immediately deployed on a web server. The author used this function regularly.

6. In the author's opinion, the simplest way to deploy a pre-packaged .war file containing a web application on an Apache Tomcat web server is through the *Tomcat Web Application Manager* interface; see Figure 10. If the edits in the *Configuring the Web Server* section (p. 22) were made, this interface can be

loaded in a web browser at `http://<server IP address>/manager/html`. The `.war` file can be directly uploaded from that interface and will automatically be deployed. In the case of this project, the DGRP web application can then be accessed at `http://<router IP address>/DTraceGraphicalRouterProject`. There are a couple of possible sub-steps necessary in this process:

- a. The directory `/var/apache/tomcat/webapps`—where the `.war` file will be placed during deployment—may not allow writing of files by default, so write permissions may need to be added before uploading a `.war` file.
- b. Tomcat 5.5.26, the version included in OpenSolaris SXCE build 94, is missing a library, `commons-io`, in the directory `/usr/apache/tomcat/server/webapps/manager/WEB-INF/lib`. This is fixed in future versions, but for this iteration, it was necessary to obtain a copy of that library, place it in the specified directory, and restart the apache daemon to enable uploading a `.war` file for deployment.
- c. In order to use the DTrace library from a web application, it is necessary to add the appropriate path to Java's library path. The following series of commands stops the apache daemon, sets an environmental variable appropriately, and restarts the daemon so that it can access the DTrace library

as needed:

```
/etc/rc3.d/S50apache stop  
export JAVA_OPTS=-Djava.library.path=/usr/lib  
/etc/rc3.d/S50apache start
```

manager Mozilla Firefox

http://10.0.0.1:8080/manager/html

Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message: OK

[List Applications](#) [HTML Manager Help](#) [Manager Help](#) [Server Status](#)

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/TraceGraphicsRouterProject		true	0	Start Stop Reload Undeploy
/admin	Tomcat Administration Application	true	0	Start Stop Reload Undeploy
/balancer	Tomcat Simple Load Balancer Example App	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

Deploy directory or WAR file located on server

Context Path (optional):

XML Configuration file URL:

WAR or Directory URL:

WAR file to deploy

Select WAR file to upload

Done

Now in Fremont: Partly Cloudy, 66° F Thu: 56° F Fri: 70° F Fri: 57° F

Figure 10. Tomcat Web Application Manager

The Tomcat Web Application Manager allows for simple deployment and management of web applications.

VI. TESTING

The purpose of testing in the development of this framework and its reference implementation is to ensure that the design proposed here does indeed work—that a web application can dynamically provide network information via remote procedure calls from a router utilizing DTrace to collect this data. During this development, two forms of testing have been performed: The first used a virtual installation of the OpenSolaris operating system sending and receiving network traffic through virtual network interfaces to virtual interfaces on the host operating system; the second used real hardware with other systems physically connected via real network interfaces. These two forms of testing are discussed in this chapter. For each form of testing, one important assumption is made: The accuracy of DTrace is already proven and is thus out of scope for this testing, so these tests will not attempt to validate the data shown in the browser interface by comparing it to what could be captured by independent network analyzers.

Testing on a Virtual System

Virtualization technology has seen rapid improvement in recent years, and software from companies like VMWare and others now offers stable and generally very usable methods by which one or more “guest operating systems” can exist on a “host operating system.” The virtualization software makes it appear to the guest OS that it is

actually running native on the system hardware—but without requiring a true installation of the OS onto the system memory (e.g., hard drive) in place of the original OS. One of these virtualization software programs is *VirtualBox*, an open-source software distributed by Sun Microsystems (VirtualBox, 2008). VirtualBox offers the capability to run a large variety of guest operating systems on many different host operating systems. For the purpose of this project, the author was able to utilize VirtualBox to install OpenSolaris as a guest operating system on a laptop running Microsoft Windows XP as the host OS.

The installation of VirtualBox itself is simple and follows the pattern of most software application installations. Complete installation and user-guide instructions are available on the VirtualBox download web site (*Download VirtualBox*, 2008). Once VirtualBox was installed, the addition of OpenSolaris as a guest operating system was also quite straightforward. The author downloaded a single DVD disc image (in .iso format) of OpenSolaris (build nv_96) from the OpenSolaris download web site (*OpenSolaris Download Center*, 2008), mounted it as a virtual DVD-ROM for the OpenSolaris guest OS in VirtualBox, and “powered on” the OpenSolaris OS to begin installation just as if a real DVD had been inserted into real hardware. For additional information about the OpenSolaris installation process, refer to the *Installing and Configuring the Operating System* section (p. 16) in Chapter IV. *Platform Preparation*. Two important notes should be made about the setup of OpenSolaris in VirtualBox on the author's development laptop. First, although the VirtualBox documentation indicated that OpenSolaris should function with only 512MB of system memory allocated for it, the

author found that the installation failed unless 1GB of system memory was allocated; second, the author also installed the “Solaris guest additions” provided with the VirtualBox software for making the transition between host and guest operating systems more seamless (for the mouse and keyboard, etc.) during development and testing. After the installation was complete, configuration and the deployment of the web application followed the steps outlined in the following sections from Chapter IV. *Platform Preparation* and Chapter V. *Implementation* with the additions to be described:

1. *Installing and Configuring the Operating System* (p. 16)
2. *Installing and Configuring the Routing Software* (p. 20)
3. *Installing and Configuring the Web Server* (p. 22)
4. *Deploying the Complete Web Application* (p. 66)

The additions to this process included the special configuration of virtual network interfaces on the host and guest operating systems. The VirtualBox software provides this functionality. The author first created four virtual interfaces on the host (Windows XP) OS using the following commands in a console:

```
vboxmanage createhostif "VirtualBox if1"  
vboxmanage createhostif "VirtualBox if2"  
vboxmanage createhostif "VirtualBox if3"  
vboxmanage createhostif "VirtualBox if4"
```

Next, each interface was assigned an IP address on a different subnet:

VirtualBox if1:	10.0.0.50	(netmask: 255.255.255.0)
VirtualBox if2:	10.0.1.50	(netmask 255.255.255.0)
VirtualBox if3:	10.0.2.50	(netmask 255.255.255.0)
VirtualBox if4:	10.0.3.50	(netmask 255.255.255.0)

Then, in the VirtualBox settings for the guest OS, four virtual interfaces were created for

the the guest OS, and each was paired with one of the virtual interfaces on the host OS—
as shown in Figure 11. After powering on the OpenSolaris guest OS, each interface was
assigned an IP address on the same subnet as its paired interface:

```
ifconfig e1000g0 plumb; ifconfig e1000g0 10.0.0.1 netmask  
255.255.255.0 up  
ifconfig e1000g1 plumb; ifconfig e1000g0 10.0.1.1 netmask  
255.255.255.0 up  
ifconfig e1000g2 plumb; ifconfig e1000g0 10.0.2.1 netmask  
255.255.255.0 up  
ifconfig e1000g3 plumb; ifconfig e1000g0 10.0.3.1 netmask  
255.255.255.0 up
```

In this configuration, the guest OS has four virtual physical ports—each configured on a
different subnet, and there is exactly one other system active on each subnet—the
corresponding virtual port on the host OS.

With everything set up, some simple tests were performed. First, the author
verified that the simple features worked. The following several screen shots in Figures
12, 13, 14, and 15 show the output as expected from all of the screens except for
Analysis.

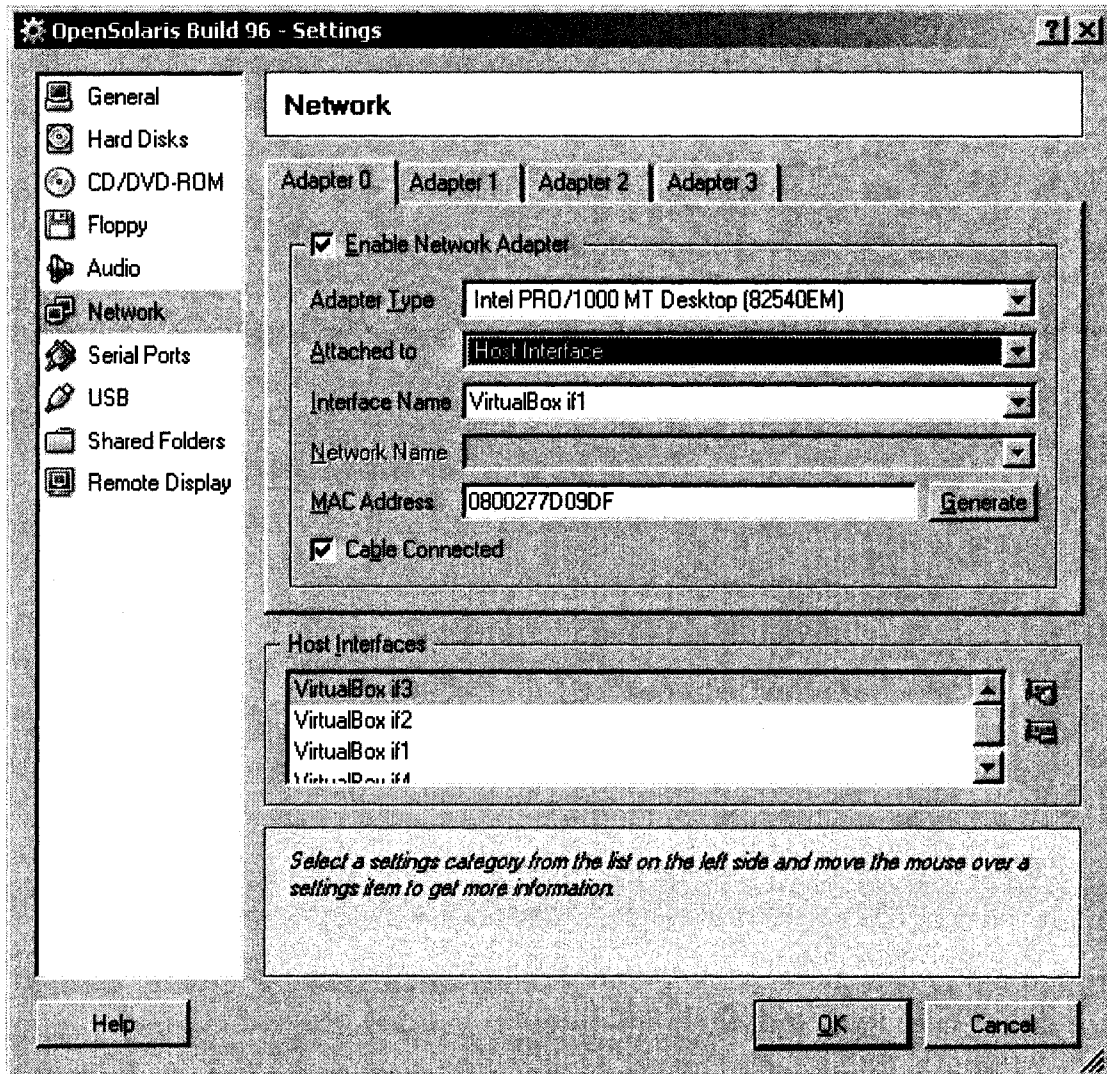


Figure 11. Creating Virtual Network Interfaces

VirtualBox allows the creating of virtual network interfaces on both the host and guest operating systems and the pairing of them to allow network traffic between the two.

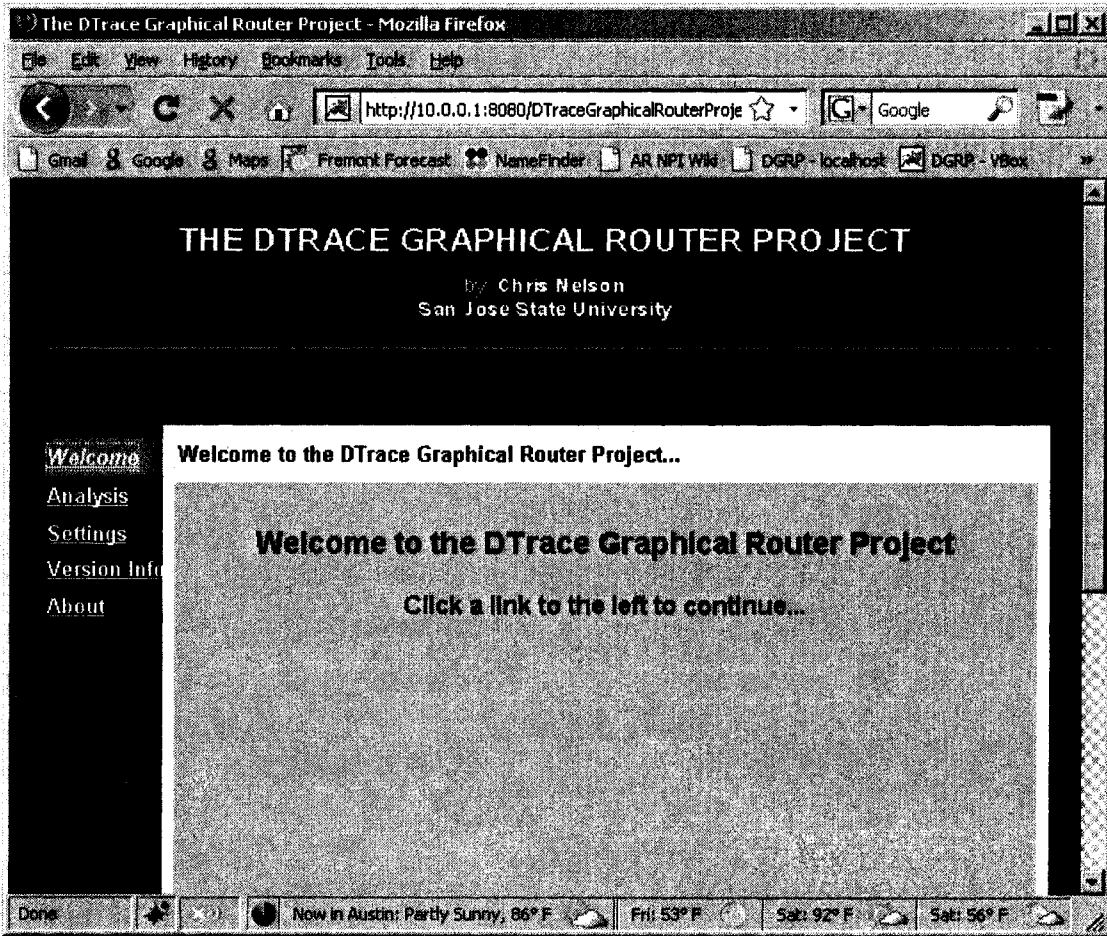


Figure 12. DGRP Welcome Screen

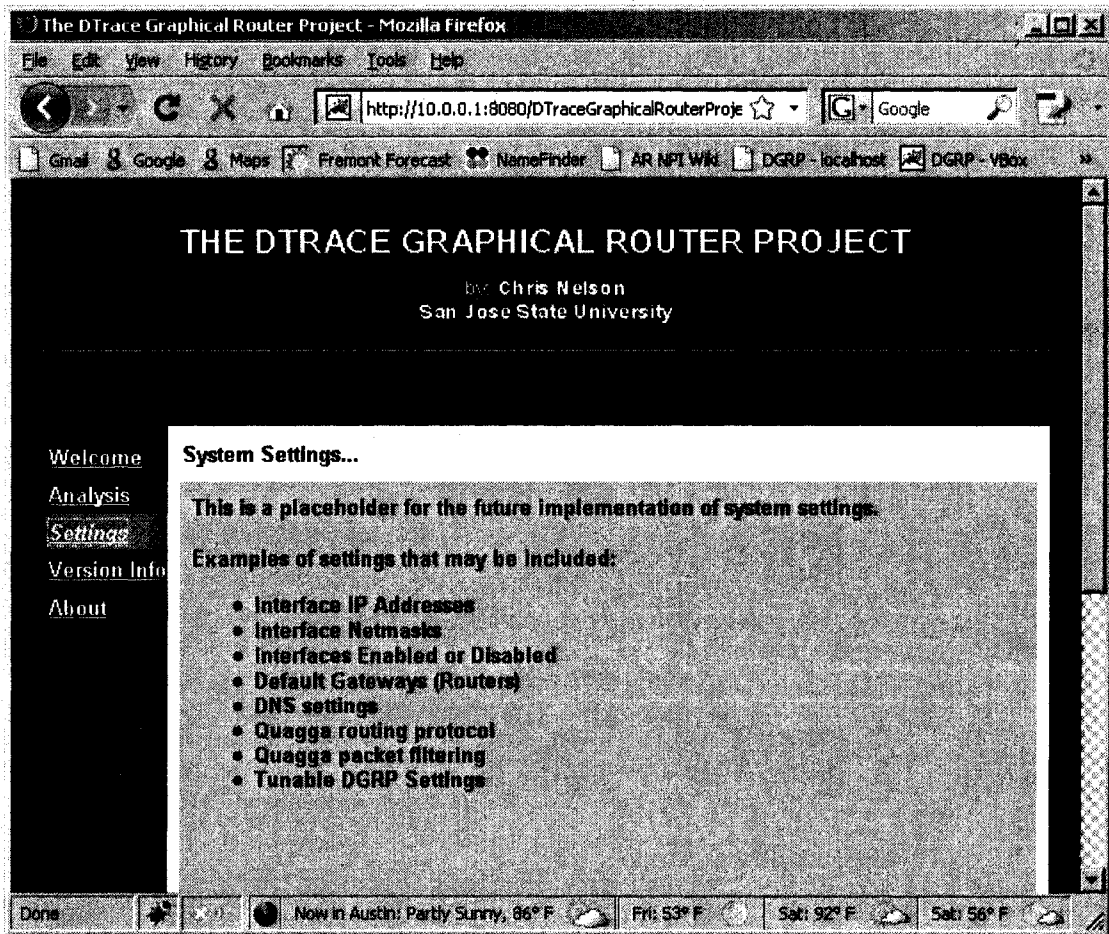


Figure 13. DGRP Settings Screen

The DTrace Graphical Router Project - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://10.0.0.1:8080/DTraceGraphicalRouterProje

Gmail Google Maps Fremont Forecast NameFinder AR NPT Wiki DGRP - localhost DGRP - VBox

San Jose State University

[Welcome](#)
[Analysis](#)
[Settings](#)
[Version Info](#)
[About](#)

Version Information for the DTrace Graphical Router Project...

DTrace Graphical Router Project:
Version: 0.9.0
Build Number: 26
Build Date: Monday, September 22, 2008, 11:00:52 PM PDT
Author: Chris Nelson

OpenSolaris:
Version: Solaris Express Community Edition env_96 X86
Architecture: x86
Install Date: Aug 25 2008 11:05
Current Uptime: 58 day(s) and 2 users hour(s)

Quagga:
Version: Quagga Routing Software 0.99.8
Install Date: Aug 25 2008 11:51

Java:
Version: 1.6.0_06
Vendor: Sun Microsystems Inc.
Virtual Machine (VM): Java HotSpot(TM) Client VM
VM Version: 10.0-b22
VM Vendor: Sun Microsystems Inc.

Apache/Tomcat Web Server:
Apache: The Apache HTTP server program (1.3.x) (root component)
Tomcat: Tomcat Servlet/JSP Container (root)

Browser:
Version: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1) Gecko/2008070208 Firefox/3.0.1

Done Now in Austin: Partly Sunny, 86° F Fri 53° F Sat 92° F Sat 56° F

Figure 14. DGRP Version Screen

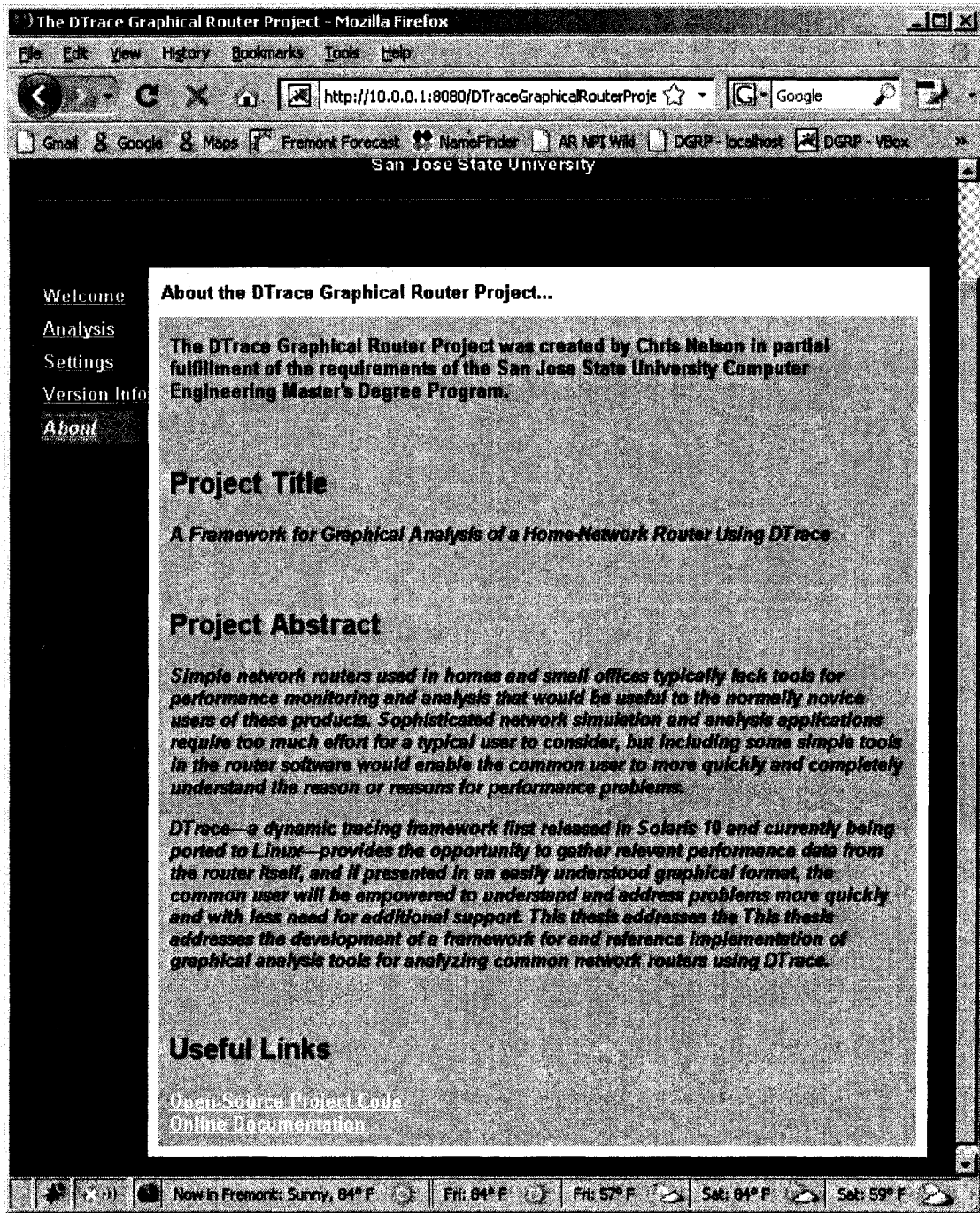


Figure 15. DGRP About Screen

With these basic functions proven working, the author moved on to the Analysis page. By just loading the DGRP web application and performing no extra tasks, one should see some activity: The expected activity would include just basic ARP traffic on three of the subnets and some additional traffic on the subnet over which the RPC calls are occurring for the web application itself. In fact, this is exactly what was seen. Figure 16 is a snapshot of the minimal traffic observed with the Solaris *snoop* command on the 10.0.3.0 subnet, and Figure 17 shows the output on the DGRP Analysis screen (with the IP packet and data byte features turned on).

```

root@nv96-vbox ~/Desktop
File Edit View Terminal Tabs Help
inet 10.0.1.1 netmask fffffff0 broadcast 10.0.1.255
ether 8:0:27:d0:25:dd
e1000g2: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 4
inet 10.0.2.1 netmask fffffff0 broadcast 10.0.2.255
ether 8:0:27:c7:99:ff
e1000g3: flags=201000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,CoS> mtu 1500 index 5
inet 10.0.3.1 netmask fffffff0 broadcast 10.0.3.255
ether 8:0:27:23:6e:db
lo0: flags=2002000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv6,VIRTUAL> mtu 8252 index 1
inet6 ::1/128
nv96-vbox$ ifconfig e1000g0 down; sleep 1; ifconfig e1000g0 up
nv96-vbox$ ping 10.0.0.50
no answer from 10.0.0.50
nv96-vbox$ ping 10.0.0.50
^C
nv96-vbox$ snoop -d e1000g3
Using device e1000g3 (promiscuous mode)
 10.0.3.50 -> 10.0.3.255 NBT Datagram Service Type=17 Source=US-CHRISNE-01[20]
nv96-vbox-g3 -> (broadcast) ARP C Who is 10.0.3.1, nv96-vbox-g3 ?
 10.0.3.50 -> 10.0.3.255 NBT NS Query Request for USMPSTVM001[0], Success
 10.0.3.50 -> 10.0.3.255 NBT NS Query Request for USMPSTVM001[0], Success
 10.0.3.50 -> 10.0.3.255 NBT NS Query Request for USMPSTVM001[0], Success
nv96-vbox-g3 -> (broadcast) ARP C Who is 10.0.3.1, nv96-vbox-g3 ?

```

Figure 16. Snoop Capture

Solaris's *snoop* command can capture traffic received on an interface.

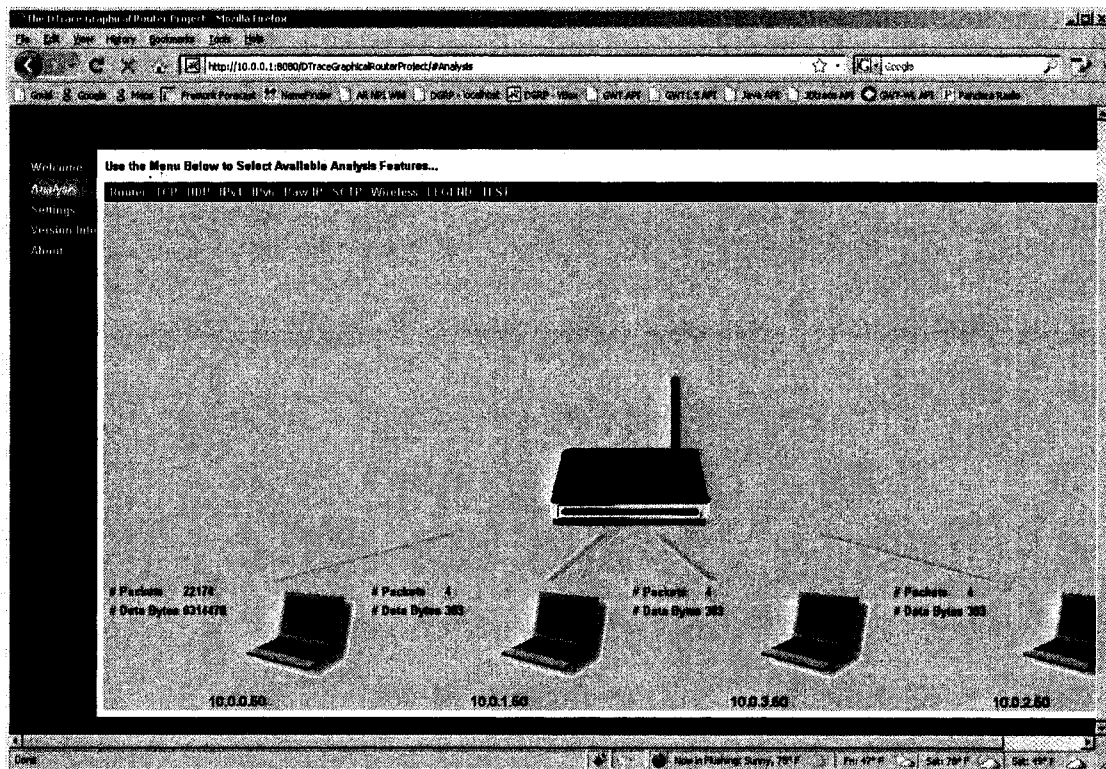


Figure 17. DGRP Analysis Screen

DGRP's Analysis screen shows network information from the busiest nodes on the connected subnets.

With this confirmation of functionality in place, though the software running on the virtual OpenSolaris system is exactly the same as that running on a real piece of hardware, the author turned his attention to repeating these tests on real hardware.

Testing on a Real System

The setup for testing on real hardware was very similar to that of the virtual hardware. OpenSolaris nv_96 was installed on a simple x86 server, and the same

configuration steps—minus the setup of virtual interfaces—as used in the virtual testing were followed. The OpenSolaris system had one physical interface—which was configured with IP address 10.4.32.184. The network to which the OpenSolaris system was attached was very large and active, but the 10.4.32.0 subnet was kept relatively quiet during this testing to allow for the observation of expected network traffic. Figure 18 shows the DGRP Analysis screen as it observed normal network activity. The author's laptop was connected to the network using IP address 129.150.192.18—as seen in Figure 19—and was running the DGRP application in a browser; this correlated with the laptop's IP address showing as the most active system on the DGRP application.

To validate the dynamic updating of the web application—including the reordering of the systems shown according to how busy they are, the author copied two large files from another system on the network (assigned IP address 10.4.32.180) to the router. As expected, the screen updated with the now-busiest system showing in the far-left position and the author's laptop in the second-to-the-left position—as shown in Figure 20.

Finally, to validate the data shown in the browser interface with that of a typical DTrace script, the author refreshed the web application (to reset the counters) and—as close to simultaneously as possible—started the equivalent DTrace script in a console window to observe if the counters would match. As seen in Figure 21 and Figure 22, the counters for the relatively quiet systems do indeed match, and the slight difference in the data for the busiest system (the author's laptop) can easily be accounted for in the

difference of start and stop time in the browser and script tests.

To conclude, simple testing—both on virtual and real hardware—shows that the reference implementation of this framework does indeed correctly utilize DTrace to capture information about the network traffic processed by the OpenSolaris router and display it in a graphical interface for the user. In future iteration of this project, as the number of features built on this framework grows larger, a more structured test plan should be developed and executed to ensure bugs in the user interface or data processing logic are identified and fixed.

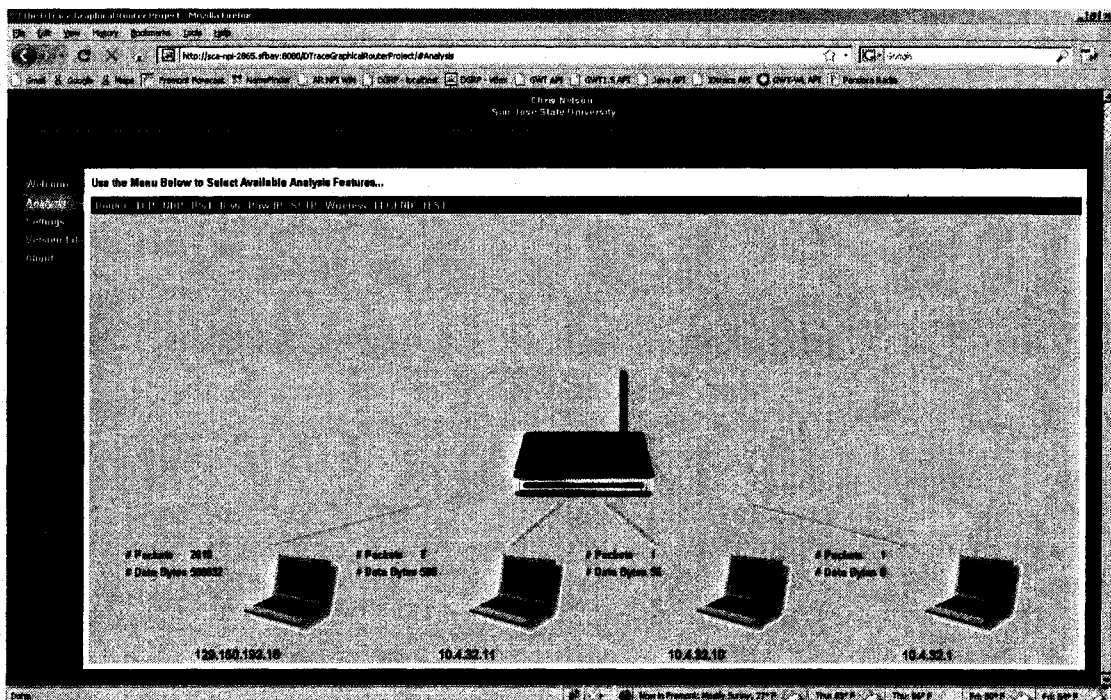


Figure 18. DGRP On Real Hardware

The author's laptop—running the DGRP web application—was the most active system on the network under normal conditions.

```
Command Prompt
Ethernet adapter VirtualBox IF2:

Media State . . . . . : Media disconnected
Description . . . . . : VirtualBox TAP Adapter #2
Physical Address. . . . . : 00-FF-74-0C-86-00

Ethernet adapter VirtualBox IF1:

Media State . . . . . : Media disconnected
Description . . . . . : VirtualBox TAP Adapter
Physical Address. . . . . : 00-FF-82-44-35-00

Ethernet adapter VPN Connection:

Connection specific DNS Suffix . . : sun.com
Description . . . . . : Cisco Systems VPN Adapter
Physical Address. . . . . : 00-05-90-3C-78-00
Dhcp Enabled. . . . . : No
IP Address. . . . . : 129.150.192.18
Subnet Mask . . . . . : 255.255.252.0
Default Gateway . . . . . : 129.150.192.1
DNS Servers . . . . . : 129.80.114.1
                          129.80.114.2

D:\Documents and Settings\Chris\>
```

Figure 19. Test Laptop IP Address

The author's laptop was assigned IP address 129.150.192.18 during the testing.

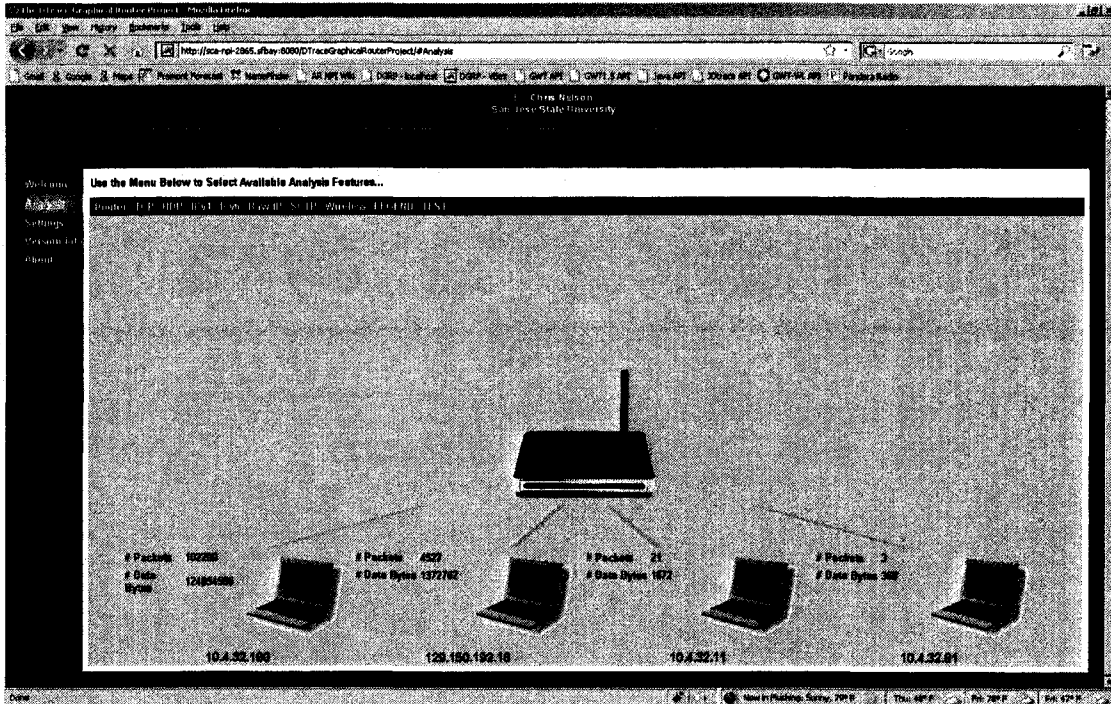


Figure 20. Ordering According to Activity

The DGRP web application reorders the systems according to how busy each is—assuming that the busiest systems are those of most interest to the users.

```
sca-npi-2865.sfbay.sun.com - PuTTY
-----
Total data bytes received from and sent to:
-----
Data received from IP(s):
  10.4.12.10      21150
-----
Data sent to IP(s):
  10.4.12.10      8
  10.4.12.10      54
  10.4.12.10      100
  10.100.100.10  24000
-----
Total data bytes received from and sent to:
  10.4.12.10      8
  10.4.12.10      54
  10.4.12.10      100
  10.100.100.10  24000
```

Figure 21. DTrace Script Output

Typical DTrace scripts are run in a console.

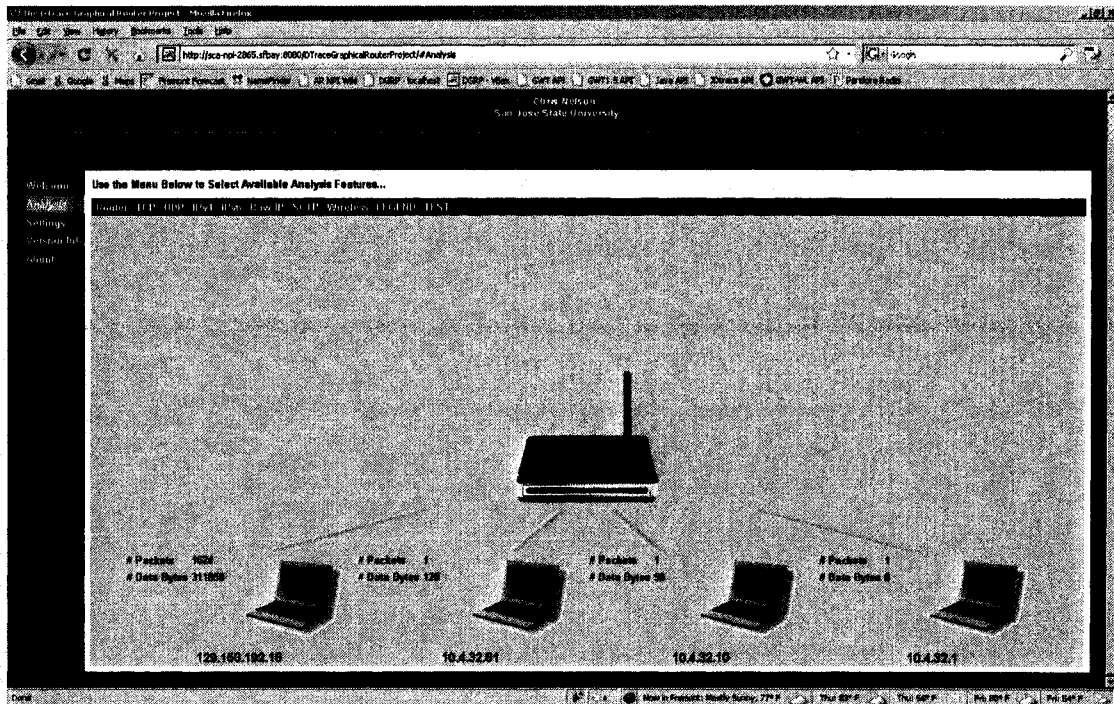


Figure 22. DGRP Analysis Matches DTrace Scripts

The DGRP application relies on DTrace scripts in the background, and the data displayed in the browser interface matches that of a typical script on a console.

VII. SUGGESTIONS FOR FUTURE DEVELOPMENT

While significant work has been completed in the development of this framework and the reference implementation provided in this iteration of this project, the long-term goals has not yet been realized: Integrating the type of tool discussed in this paper into a true home-network router like those sold to consumers today will require some additional work. The major remaining pieces of work are discussed in the following sections; some are dependent upon each other, and some present an implementation choice that must be made work in that area continues.

OpenSolaris on MIPS

Porting the OpenSolaris operating system to the MIPS architecture would enable the use of DTrace on the hardware platforms already in use in today's home-network routers. This type of port is obviously non-trivial, but some support already exists for the idea within the open-source OpenSolaris community (*MIPS port of opensolaris*, 2005). Significant positive aspects of this port exist: The full functionality of DTrace would be available, and the full framework and reference implementation developed for this project would be directly applicable and immediately usable. However, the negative parts include the amount of work required not only to port Solaris to MIPS but to reduce the memory footprint to a size appropriate for the typically small memory sizes available in

common home-network routers; in addition, the management interface functionality that already exists for Linux on today's routers would need to be either ported or re-developed (or at least recompiled) for Solaris as well. The effort to port OpenSolaris to MIPS would probably be a good fit for a small group of computer Engineering graduate students—as a knowledge of both hardware architecture and its interaction with software is required.

Wireless Support on OpenSolaris

As discussed earlier in this document, the support for wireless networking on OpenSolaris is immature at best. Yet even with a small number of wireless chipsets supported, this application could work in a wireless setting using OpenSolaris—save for one missing piece of functionality: Code must be written to allow OpenSolaris to act as a wireless access point. Today's common home-network routers actually server a number of roles: They are routers, gateways, and wireless access points—at least. At the outset of developing this framework, the author was unable to find any software available for using a Solaris system as a wireless access point, so that functionality would need to be developed by—likely—a pair of software or computer engineering graduate students in order to make OpenSolaris a truly viable option for the long-term solution.

DTrace in Linux

The choice to develop DTrace for Linux is really the second of two options—the first of which is the porting of OpenSolaris to the MIPS architecture. While OpenSolaris

on MIPS provides the ability to use DTrace and this framework on the hardware platform in use by today's home-network routers, adding DTrace to Linux would enable the use of DTrace and this framework on both the hardware platform and operating system of today's routers. On the positive side, all of the work to get the operating system working on the hardware platform (including wireless support) is already complete, and this framework—and even the code from the reference implementation—would be quite easy to use in the Linux OS. The downside to this choice is the non-trivial effort required to implement DTrace in (at least) the Linux kernel and to develop the Java DTrace library for Linux necessary for the server-side code to utilize DTrace after it is implemented. Since both the DTrace implementation in OpenSolaris and the Java DTrace library are open-source, there is plenty of reference code available for this effort. Still, this task would probably be appropriate for a small group of computer and software Engineering graduate students.

Cleaning Up the User Interface

The team behind the Google Web Toolkit makes a true statement about the development of user interfaces: “As developers, we tend to be more interested in elegant algorithms and clever optimizations, but remember that the user's opinion of our application will be formed *almost entirely* on the interface's appearance and how well it works. Don't neglect it! (*Add Styling*, 2008, last para.)” Indeed, during the course of developing this framework and its reference implementation, the author placed relatively

little emphasis on the appearance of the graphical interface. Significant improvements could be made to it through the judicious modification of the cascading style sheet, the use of better clip art images, the rearrangement or integration of additional GWT widgets or even those from add-on GWT widget libraries like those from the *GWT Widget Library* project (*GWT Widget Library*, 2008) or *GWT-Ext* (*GWT-Ext*, 2008). If necessary, the Google Web Toolkit could even be replaced by a different web application framework that provides greater flexibility and capability for developing good-looking applications. Note, however, that GWT provides not only the widgets and automated AJAX code development but also the facility for Remote Procedure Calls, so choosing another web application framework would necessitate finding other ways to provide those functions as well. This effort is almost entirely software-related, so a single software engineer or perhaps a pair would be appropriate for this task.

Additional Features

The reference implementation documented in this paper provides a few of the most basic and arguable most useful data points that a tool of this nature could provide, but an almost endless list of possibly useful features remains. A list of many potential future features is provided in the *Functional Requirements* section (p. 100) of the *Requirements* appendix, though future developers are encouraged to consider other ideas as well. Development of additional features requires a firm understanding of the entire architecture presented in this paper, so it is most likely applicable for computer

engineering graduate students, though software engineering students with a good background in networking would also be appropriate for this task.

Final Integration

This effort is really the last step in achieving the goal set out in the introduction to this project. It is dependent up on the other suggestions in this chapter—either porting OpenSolaris to MIPS and adding wireless support or adding DTrace to Linux, improving the user interface, and adding additional features. When these are complete, a single graduate student—either a computer or software engineer—could pull these pieces together to actually produce a home-network router that provides a user with a graphical tool that utilized DTrace for analyzing his or her home-network. Depending on how well the dependencies are completed, this final integration effort could range from somewhat trivial to a larger amount of work; it could even potentially be combined with the previous suggestion of developing more analysis features.

VIII. CONCLUSIONS AND RECOMMENDATIONS

The technology of DTrace offers a way to solve the need for simple, user-friendly tools for home-network analysis. By collecting the detailed network traffic information offered by DTrace and presenting it in an easy-to-understand graphical format, common users can quickly identify and address problems in their networks without the need to understand the many details of computer networking. The framework developed in this project and its reference implementation provide a clear picture of how to piece together DTrace with the other necessary technologies to make this type of easy-to-understand graphical tool a reality.

In this project, several technologies have been pulled together to form a framework and reference implementation for a DTrace-based graphical network analysis tool. Because DTrace is only available at this time in the Solaris and Mac operating systems, and because Mac OS X is not free and is only available on specific hardware, the free, open-source OpenSolaris operating system was chosen for this implementation. While many home-network routers use the MIPS architecture, because Solaris is not currently available for MIPS, the x86 platform was chosen for this implementation. On top of OpenSolaris running on x86, the Apache Tomcat web server was used, and the web application—both client-side and server-side code—was written in the Java programming language utilizing the Google Web Toolkit to translate the client-side code into fast

Asynchronous Java and XML (AJAX) code that can be executed in a browser without reloading web pages. Asynchronous remote procedure calls (RPCs) were used to provide the means of communication between the front-end (client-side) code executing in the user's browser and the back-end (server-side) code executing on the router.

This framework—a client-side web application pulling data through RPCs from server-side code utilizing Dtrace—is well defined in this document and is clearly validated by the reference implementation also documented here. Even so, the ultimate goal of integrating this functionality with a real home-network router (running on its MIPS architecture and integrating into its web application management interface) has not yet been achieved. To achieve this, there remain a few additional significant pieces of work that must be completed. These pieces—including the porting of OpenSolaris to MIPS or developing DTrace in the Linux kernel—may form the basis of future graduate work that enhances what has been already completed in this project; for a complete discussion of these pieces of suggested future development, refer to Chapter VII.

Suggested Future Development (p. 89).

In conclusion, a framework to meet the stated goal has been developed and proven by a reference implementation, and a roadmap is provided for future development that shows the path between this iteration and the achievement of the final goal.

REFERENCES

- Add Styling.* (2008). Retrieved September 2, 2008, from <http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=GettingStartedStyle>
- Apache Tomcat 6.0.* (2008). Retrieved February 2008, from <http://tomcat.apache.org/tomcat-6.0-doc/index.html>
- BigAdmin System Administration Portal. DTrace.* (2008). Retrieved February 2008, from <http://www.sun.com/bigadmin/content/dtrace/>
- Building a Sample Application.* (2008). Retrieved February 2008, from <http://code.google.com/webtoolkit/gettingstarted.html#Sample>
- Download VirtualBox.* (2008). Retrieved August 17, 2008, from <http://www.virtualbox.org/wiki/Downloads>
- DTrace.* (2007). Retrieved March 1, 2008, from <http://wikis.sun.com/display/DTrace/>
- DTrace for FreeBSD.* (2008). Retrieved April 17, 2008, from <http://www.bsdcn.org/2008/schedule/events/66.en.html>
- DTrace – Introduction.* (2007). Retrieved March 1, 2008, from <http://wikis.sun.com/display/DTrace/Introduction>
- DTrace Network Providers.* (2008, August 29). Retrieved September 1, 2008, from <http://opensolaris.org/os/community/dtrace/NetworkProvider/>
- Eclipse – an open development platform.* (2008). Retrieved September 2008, from <http://www.eclipse.org/>
- Google Web Toolkit.* (2008). Retrieved March 15, 2008, from <http://code.google.com/webtoolkit/>
- Gregg, B. ip Provider.* (2008, July 28). Retrieved August 25, 2008, from <http://wikis.sun.com/display/DTrace/ip+provider>
- GWT-Ext.* (2008). Retrieved September 2008, from <http://gwt-ext.com/>

- GWT Widget Library*. (2008). Retrieved September 2008, from <http://sourceforge.net/projects/gwt-widget>
- gwt4nb Project Home*. (n.d.). Retrieved March 2008, from <https://gwt4nb.dev.java.net/>
- Home Gateway Initiative*. (2007). Retrieved April 17, 2008, from <http://www.homegatewayinitiative.org/publis/HGI2007%20flyer.pdf>
- Home Gateway Initiative – Vision*. (2007). Retrieved April 17, 2008, from http://www.homegatewayinitiative.org/public/docs/HGI_white_paper.pdf
- HOWTO: use ANT with JAVA to dynamically create build numbers*. (2007, April 16). Retrieved March 2008, from <http://stoken-tips-and-tricks.blogspot.com/2007/04/howto-use-ant-with-java-to-dynamically.html>
- Java DTrace API*. (2007, May 27). Retrieved May 15, 2008, from http://opensolaris.org/os/project/dtrace-chime/java_dtrace_api
- Leventhal, A. *Mac OS X and the missing probes*. (2008, January 18). Retrieved April 17, 2008, from http://blogs.sun.com/ahl/entry/mac_os_x_and_the
- List of web application frameworks*. (2008). Retrieved March 2008, from http://en.wikipedia.org/wiki/List_of_web_application_frameworks
- MIPS port of opensolaris*. (2005, December 3). Retrieved May 2, 2008, from <http://opensolaris.org/jive/thread.jspa?messageID=51805>
- NetBeans*. (2008). Retrieved February 2008, from <http://www.netbeans.org/>
- NetBeans DTrace GUI Plug-in*. (n.d.). Retrieved March 12, 2008, from http://www.netbeans.org/kb/dtracegui_plugin/NetBeans_DTrace_GUI_Plugin.html
- OpenSolaris*. (n.d.). Retrieved March 1, 2008, from <http://www.opensolaris.org>
- OpenSolaris Community: DTrace*. (2007, October 23). Retrieved March 1, 2008, from <http://opensolaris.org/os/community/dtrace/>
- OpenSolaris Download Center*. (2008). Retrieved March 1, 2008, from <http://www.opensolaris.org/os/downloads/>
- OpenSolaris Project: Chime Visualization Tool for DTrace*. (2008). Retrieved April 17, 2008, from <http://opensolaris.org/os/project/dtrace-chime/>

OpenSolaris Project: Quagga Routing Protocol Suite Integration. (2007). Retrieved May 2, 2008, from <http://opensolaris.org/os/project/quagga/>

OpenWrt. (2008). Retrieved April 17, 2008, from <http://wiki.openwrt.org/>

Output Formatting. (2007). Retrieved May 2008, from <http://wikis.sun.com/display/DTrace/Output+Formatting>

Rational Apex Embedded Solaris to MIPS Family Release Note for Tornado. (2001). Retrieved April 17, 2008, from ftp://ftp.software.ibm.com/software/rational/docs/apex/400b_vxw/vxworks_relnote_mips/vxworks_release_note.html

Remote Procedure Calls. (2008). Retrieved March 2008, from <http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideRemoteProcedureCalls>

Ruby on Rails. (2008). Retrieved August 2008, from <http://www.rubyonrails.com/>

Solaris Dynamic Tracing Guide. (2005). Retrieved March 1, 2008 from <http://docs.sun.com/app/docs/doc/817-6223>

Stripes Home. (2008). Retrieved August 2008, from <http://www.stripesframework.org/>

Struts. (2008). Retrieved August 2008, from <http://struts.apache.org/>

SystemTap. (n.d.). Retrieved April 17, 2008, from <http://sourceware.org/systemtap/>

Table of Hardware. (2008). Retrieved March 2008, from <http://wiki.openwrt.org/TableOfHardware>

The Apache Software Foundation. (2008). Retrieved February 2008, from <http://www.apache.org/>

Thread: solaris as a wireless access point. (2007). Retrieved March 2008, from <http://www.opensolaris.org/jive/thread.jspa?threadID=52197&tstart=45>

VirtualBox. (2008). Retrieved August 25, 2008, from <http://www.virtualbox.org/>

Wireless Network Driver for ZyDAS ZD1211 802.11b/g USB Chipset (zyd). (2008). Retrieved February, 2008, from <http://www.opensolaris.org/os/community/laptop/wireless/zyd/>

APPENDICES

APPENDIX A. REQUIREMENTS

Project Deliverables

The deliverables for this project are as described in Table 2:

Table 2. Project Deliverables

Deliverable	Description
Proven Framework and Reference Implementation	A working implementation of the graphical analysis interface—including capability to provide a basic set of datapoints—must be completed.
Complete Source Code (hard and soft copy)	The source code for all components of the reference implementation will be provided with the final report—both in hard and soft copy.
Portable Presentation and Demonstration	A digital presentation of the project and a demonstration of its functionality (likely as a screencast) will be created and made available in soft copy with the final report.
Formal Presentation	A formal presentation of the project and a live demonstration will be given to faculty members of the university and other graduate students.
Thesis Report	A formal report (this document)—formatted according to university requirements and thoroughly documenting the project background, design, implementation, and suggestions for future work—will be submitted to the university for approval and binding.

Functional Requirements

The primary customers of this framework and reference implementation project

will be the makers of home-network routers who would see a business advantage by including a tool of this sort with their product(s) in order to make analysis of the home network and debugging of common problems easier for the typical customer. Other customers include the open-source community who may be encouraged to continue the effort to port DTrace to Linux and/or work to port Solaris to the MIPS architecture—either way enabling this tool to be used on routers like those currently in production.

As a framework, the functional requirements for this project are somewhat loose. The following list (in no particular order) provides a set of features and a subset of the data-points that may be of interest in future implementations of this framework. As described in the discussion of project deliverables, this framework and reference implementation should make available some—but not all—of this data in order to prove the usefulness of the design.

1. Indication of current version information (of thesis and other relevant software)
2. Indication of basic network interface settings (e.g., Internet Protocol (IP) address, netmask)
3. Indication of other relevant system settings (e.g., Domain Name Service (DNS) servers, gateway)
4. Indication of a new connection (in the case of Transmission Control Protocol (TCP)) or a new address from which data is received or to which data should be sent
5. Indication of connection termination (explicit in the case of TCP or after a period

of no traffic for other “connections”)

6. Connection speed (theoretical maximum and actual)
7. Bandwidth (theoretical maximum and actual being used)
8. Response time (maximum, minimum, average)
9. Number of packets/frames received or sent (total or per period of time)
10. Number of bits/bytes received or sent (total or per period of time)
11. Number of checksum errors (total or per period of time)
12. Some indication of overall connection quality (likely as an aggregation of several data points)
13. User-tunable parameters for data points
14. Indication of router's CPU and memory (real and virtual) utilization
15. Indication of network buffer overflows
16. Ability to modify system and interface settings (e.g., IP addresses, DNS servers, etc.)
17. Ability to easily integrate into the existing interfaces provided in common home-network routers (e.g., exist as a browser-based application)

Non-Functional Requirements

As a framework project, performance, compliance, security, and similar nonfunctional requirements are—for the most part—not applicable. It is worth mentioning that much of the DTrace functionality requires super-user privileges on the

Solaris operating system , so some form of authentication and authorization would probably be required to ensure an appropriate user is the one using the graphical interface, but this form of authentication already exists in most of the existing routers' interfaces and is outside the scope of this project.

In general, this reference implementation software should perform quickly enough and be stable enough to make obvious its usefulness. The interface should not crash in normal operating conditions, and if errors do occur, they should be handled gracefully with proper notification given to the user to enable him or her to take necessary action.

Requirements Analysis

As mentioned previously, the functional requirements for this framework project are rather loose, so a an in-depth requirements analysis—in the form of a multi-level quality-function-design (QFD) analysis or some other format—is not applicable. Nonetheless, a single-level house of quality is provided in Figure 23 to offer some basic correlation between the assumed customer requirements and the initial technical requirements.

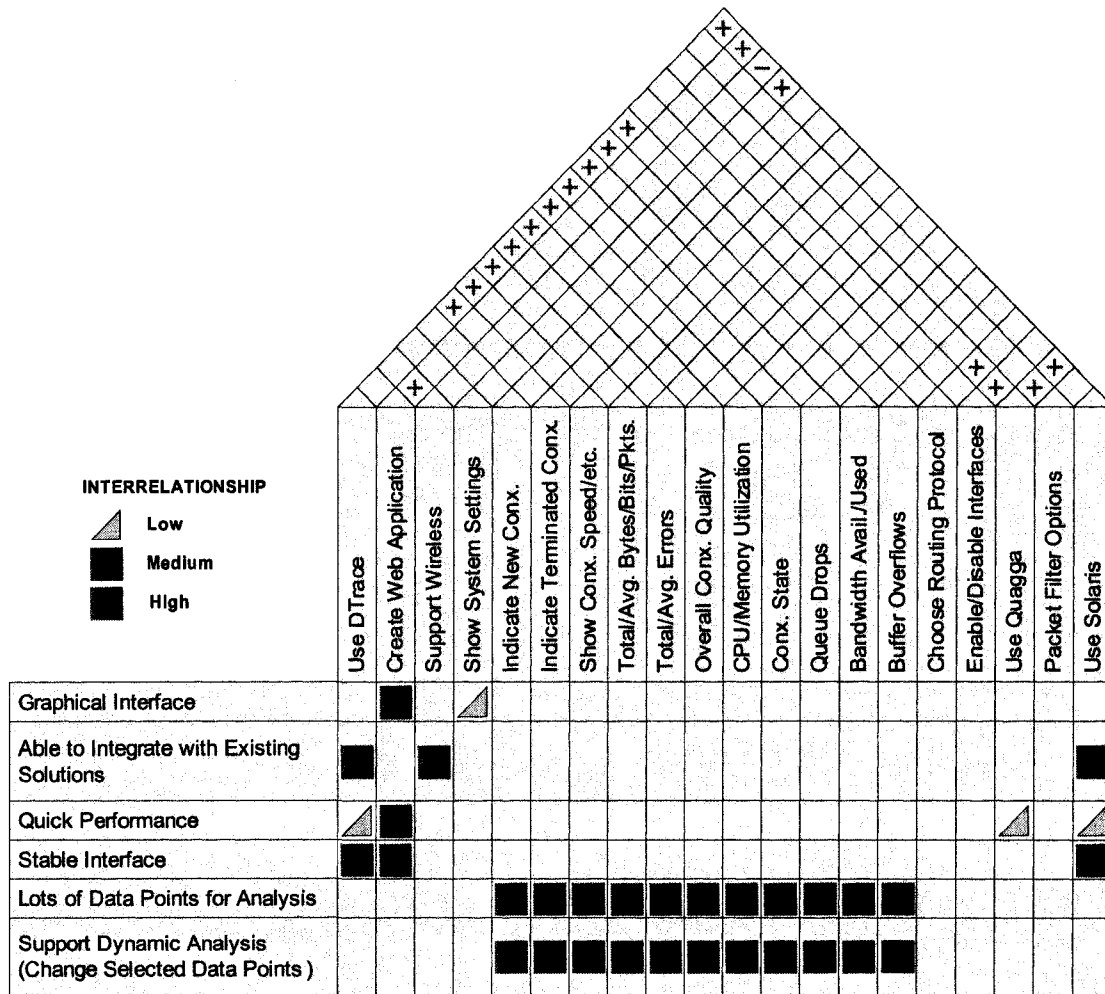


Figure 23. House-of-Quality Diagram

This House-of-Quality diagram shows some correlation between the basic customer requirements and the initial technical requirements

Hardware, Software, and Skill-Set Requirements

Tables 3, 4, and 5 provide lists of the high-level hardware, software, and skill-set requirements for the completion of this project. Brief comments are provided about the choice of some of the components, and these choices are discussed in more detail in the

Architecture and Design (p. 8) and *Implementation* (p. 27) chapters of this document.

Table 3. Hardware Requirements

Component	Comments
X86 System to Serve as Router	Any basic x86 or x64 system will work to run Solaris. (A SPARC system would work too.)
Multi-port PCI-E Ethernet Network Interface Card (NIC)	Needed to provide additional Ethernet ports for multiple connections to the router
Monitors, Keyboards, Mice, Power Cables, Ethernet Cables	Available as needed during development and testing
Two or Three Other Computers for Testing	Needed to connect to router and generate/receive traffic during testing
Laptop for Portable Development Environment	Developer will be mobile and will load snapshots of code onto the router from development laptop at various intervals for testing

Table 4. Software Requirements

Component	Comments
Operating System for Router (must provide DTrace functionality)	Using OpenSolaris Solaris Express Community Edition (SXCE). New builds are provided bi-weekly, though development of this project will likely sync only as needed for major bug fixes or feature enhancements
Individual Development Environment	Using NetBeans IDE
Web Server	Using Apache Tomcat Web Server to serve web application for browser-based interface
Routing Software	Using Quagga—a fork of Zebra (supports Solaris)

Table 5. Skill-Set Requirements

Skill-Set	Comments
Knowledge of DTrace	DTrace scripting relies on knowledge of C programming language and Unix-style shell scripting
Basic Administrative Knowledge of the Chosen Operating System	Using OpenSolaris
Java Programming Skills	Used in JavaServer Pages (JSPs), servlets, and other web-application coding
Web-Application Development Skills	
Knowledge of AJAX	Used for fast updating of graphics in the browser
Knowledge of Networking Basics	
Willingness to Learn	As in many architecture projects, the need for additional skills will arise during the course of the project.

APPENDIX B. PROJECT SCHEDULE

Initial Schedule

Final Schedule

ID	Task Name	Start	Finish	Duration	% Complete	Q1 08		Q2 08			Q3 08			Q4 08			
						Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	PHASE 1	1/30/2008	3/10/2008	41d	100%	[Gantt bar]											
2	Initial meeting with requested advisor	1/30/2008	1/30/2008	1d	100%	[Gantt bar]											
3	Submit informal proposal to advisor	2/4/2008	2/6/2008	3d	100%	[Gantt bar]											
4	Secure advisor's commitment	2/8/2008	2/8/2008	1d	100%	[Gantt bar]											
5	Determine hardware platform	2/8/2008	2/21/2008	14d	100%	[Gantt bar]											
6	Determine software routing application	2/8/2008	2/21/2008	14d	100%	[Gantt bar]											
7	Advisor meeting: Project scope	2/13/2008	2/13/2008	1d	100%	[Gantt bar]											
8	Submit signed abstract	2/13/2008	2/14/2008	2d	100%	[Gantt bar]											
9	Register project with SJSU	2/14/2008	2/14/2008	1d	100%	[Gantt bar]											
10	Acquire necessary hardware	2/20/2008	2/29/2008	10d	100%	[Gantt bar]											
11	Set up development environment	3/1/2008	3/10/2008	10d	100%	[Gantt bar]											
12	Advisor meeting	3/5/2008	3/5/2008	1d	100%	[Gantt bar]											
13	Determine wireless feasibility	3/5/2008	3/9/2008	5d	100%	[Gantt bar]											
14	PHASE 2	3/7/2008	5/3/2008	58d	100%	[Gantt bar]											
15	Complete Dtrace education	3/7/2008	3/20/2008	14d	100%	[Gantt bar]											
16	Develop Dtrace data-point list	3/14/2008	3/27/2008	14d	100%	[Gantt bar]											
17	Advisor meeting	3/19/2008	3/19/2008	1d	100%	[Gantt bar]											
18	Determine SW languages for app.	4/3/2008	4/9/2008	7d	100%	[Gantt bar]											
19	Advisor meeting	4/16/2008	4/16/2008	1d	100%	[Gantt bar]											
20	Complete SW language education	4/9/2008	4/22/2008	14d	100%	[Gantt bar]											
21	Design initial architecture	4/20/2008	5/3/2008	14d	100%	[Gantt bar]											
22	PHASE 3	4/18/2008	6/30/2008	135d	100%	[Gantt bar]											
23	Submit thesis project plan to SJSU	4/18/2008	5/1/2008	14d	100%	[Gantt bar]											
24	Advisor meeting	4/30/2008	4/30/2008	1d	100%	[Gantt bar]											
25	Complete CMPE299A Enrollment	5/8/2008	5/8/2008	0d	100%	[Gantt bar]											
26	Develop Dtrace baseline scripts	5/28/2008	8/25/2008	90d	100%	[Gantt bar]											
27	Identify Dtrace system calls	5/28/2008	8/25/2008	90d	100%	[Gantt bar]											
28	Develop single-data-point prototype	5/28/2008	8/25/2008	90d	100%	[Gantt bar]											
29	Develop test cases	8/24/2008	8/30/2008	7d	100%	[Gantt bar]											
30	PHASE 4	8/18/2008	9/30/2008	44d	100%	[Gantt bar]											
31	Agree with advisor on "end-state" for project	8/27/2008	8/27/2008	1d	100%	[Gantt bar]											
32	Expand prototype for additional data	8/18/2008	9/21/2008	35d	100%	[Gantt bar]											
33	Complete testing and bugfixes	9/1/2008	9/30/2008	30d	100%	[Gantt bar]											
34	Advisor meeting: Demo	9/24/2008	9/24/2008	1d	100%	[Gantt bar]											
35	PHASE 5	9/1/2008	1/16/2009	137d	100%	[Gantt bar]											
36	Apply for graduation	9/1/2008	9/1/2008	1d	100%	[Gantt bar]											
37	Complete remaining documentation in thesis format	9/3/2008	9/30/2008	26d	100%	[Gantt bar]											
38	Advisor meeting: Review thesis draft	10/1/2008	10/1/2008	1d	100%	[Gantt bar]											
39	Develop presentation	9/29/2008	10/3/2008	5d	100%	[Gantt bar]											
40	Present to department committee and student classes	10/6/2008	10/17/2008	12d	100%	[Gantt bar]											
41	Edit thesis for submission	10/20/2008	10/31/2008	12d	100%	[Gantt bar]											
42	Submit thesis to Graduate Studies	11/13/2008	11/13/2008	0d	100%	[Gantt bar]											
43	Complete CMPE299B Enrollment	12/4/2008	12/4/2008	0d	100%	[Gantt bar]											
44	Submit thesis for binding	1/16/2009	1/16/2009	0d	100%	[Gantt bar]											

Figure 25. Final Project Schedule

APPENDIX C. DEVELOPING WITH THE NETBEANS IDE

For the development of the reference implementation code in this project, the author made use of the NetBeans Integrated Development Environment. NetBeans is an open-source IDE distributed by Sun Microsystems, Inc. A significant amount of information regarding the installation and use of NetBeans is available on the NetBeans web site (*NetBeans*, 2008), but this appendix will provide a brief overview of several of the common steps the author used in the course of developing this framework and reference implementation.

Creating a Web Application Project

To create a web application project in NetBeans, select *File* → *New Project*, and choose *Web Application* as in Figure 26. Name the project appropriately, and select the desired web server for association with this project. The author used Apache Tomcat—as shown in Figure 27. Finally, to use the Google Web Toolkit web application framework, select it from the available frameworks—as shown in Figure 28; this requires installation of the GWT4NB plug-in—as described in the following section.

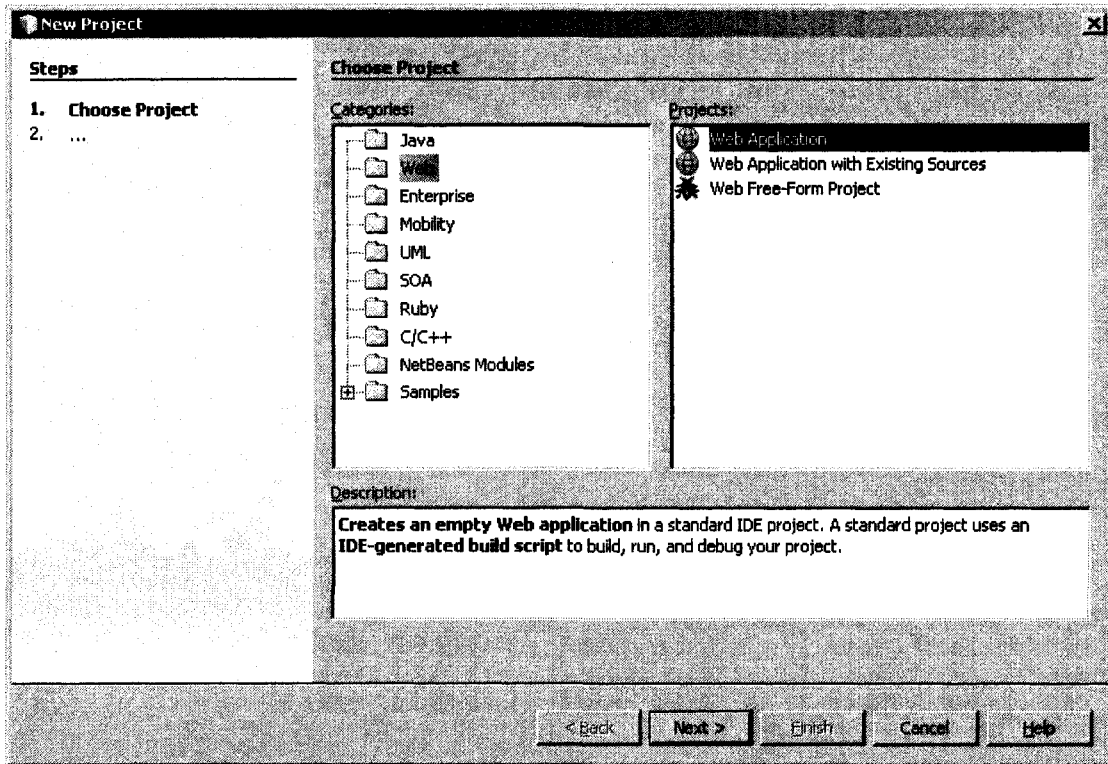


Figure 26. Creating a New Web Application Project in NetBeans

Creating a web application project in NetBeans will automatically include and configure necessary files for a web application.

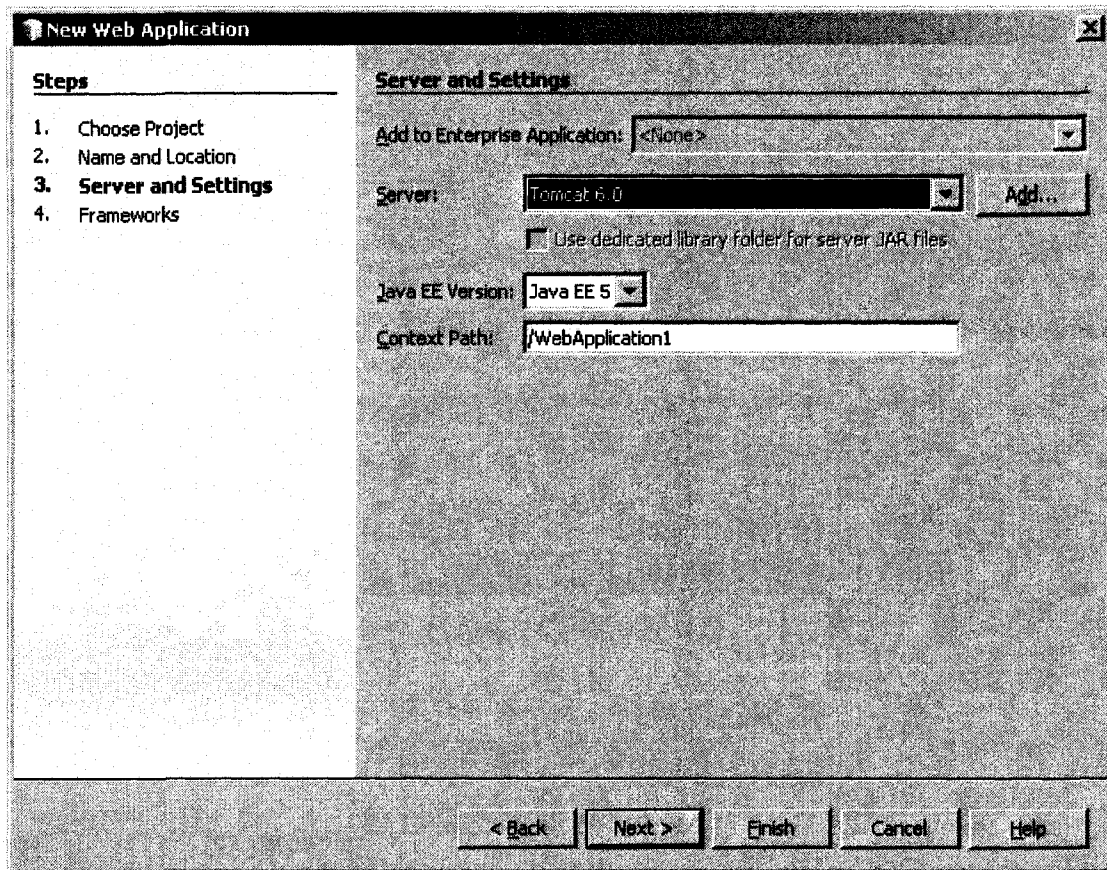


Figure 27. Choosing a Web Server in NetBeans

The author used the Apache Tomcat web server for development.

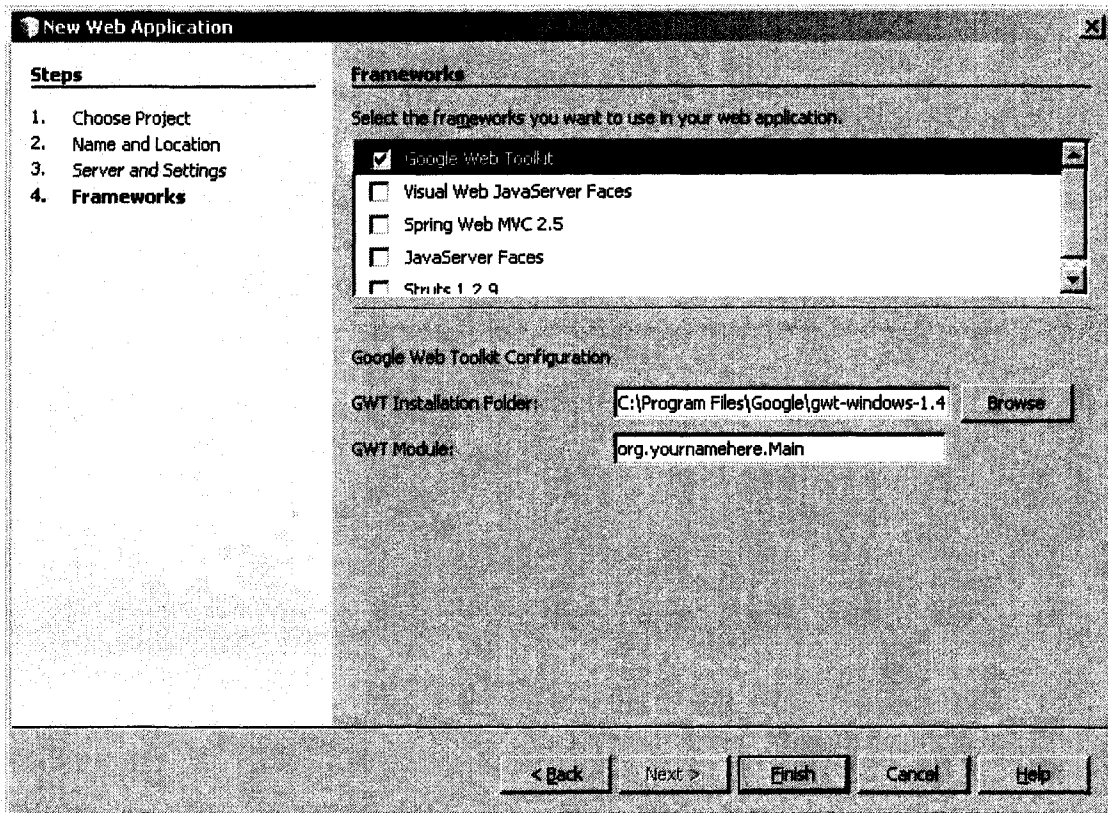


Figure 28. Choosing the GWT Framework in NetBeans

Integration between NetBeans and the Google Web Toolkit was made easy by the GWT4NB plug-in.

Using GWT4NB

To create a web application project using the Google Web Toolkit framework as described in the previous section, it is required that the GWT4NB plug-in be installed. This is handled via the plug-in wizard in NetBeans—as shown in Figure 29.

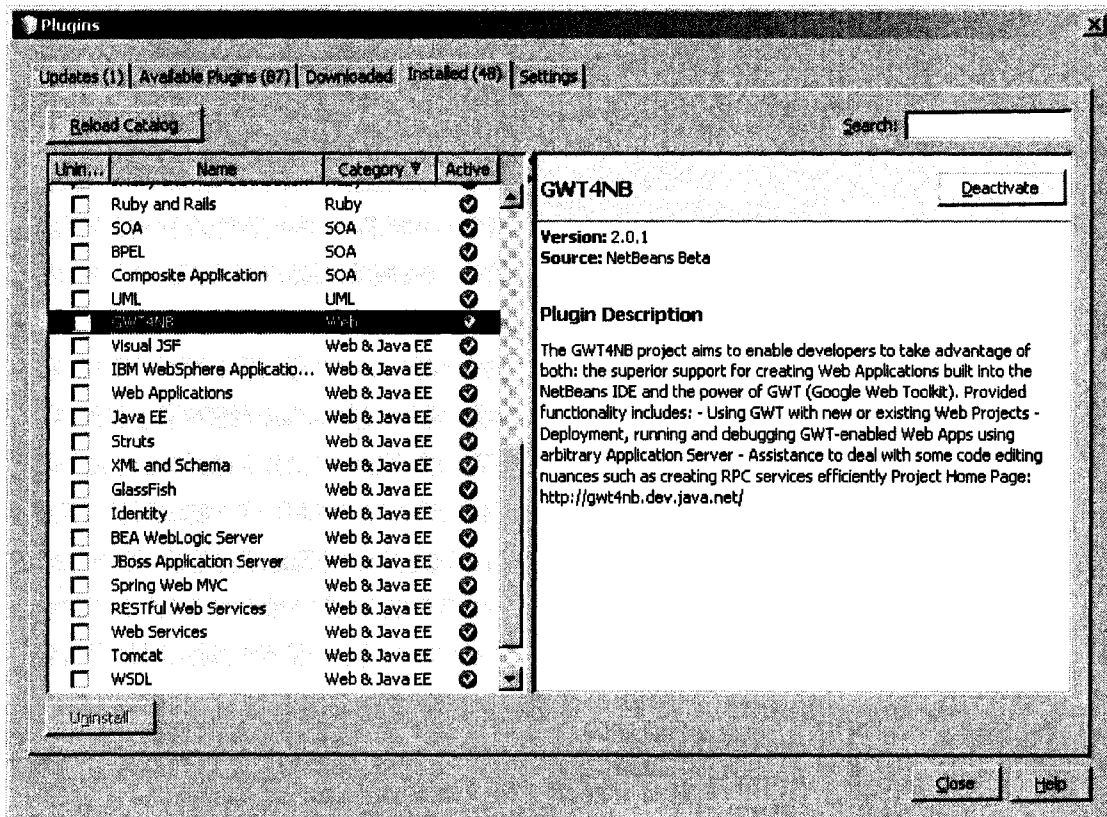


Figure 29. The GWT4NB Plugin in NetBeans

With the GWT4NB plugin installed in NetBeans, development of web applications using the GWT framework is significantly easier.

Creating an RPC

To create a remote procedure call in a web application project using the GWT4NB plugin in NetBeans, select *File* → *New File*, and choose the *GWT RPC Service* from the available options—as shown in Figure 30. According to configuration options selected on the following wizard screen, the necessary modifications will be made to files like

web.xml, and the necessary front and back-end Java classes will be created with placeholders for code development.

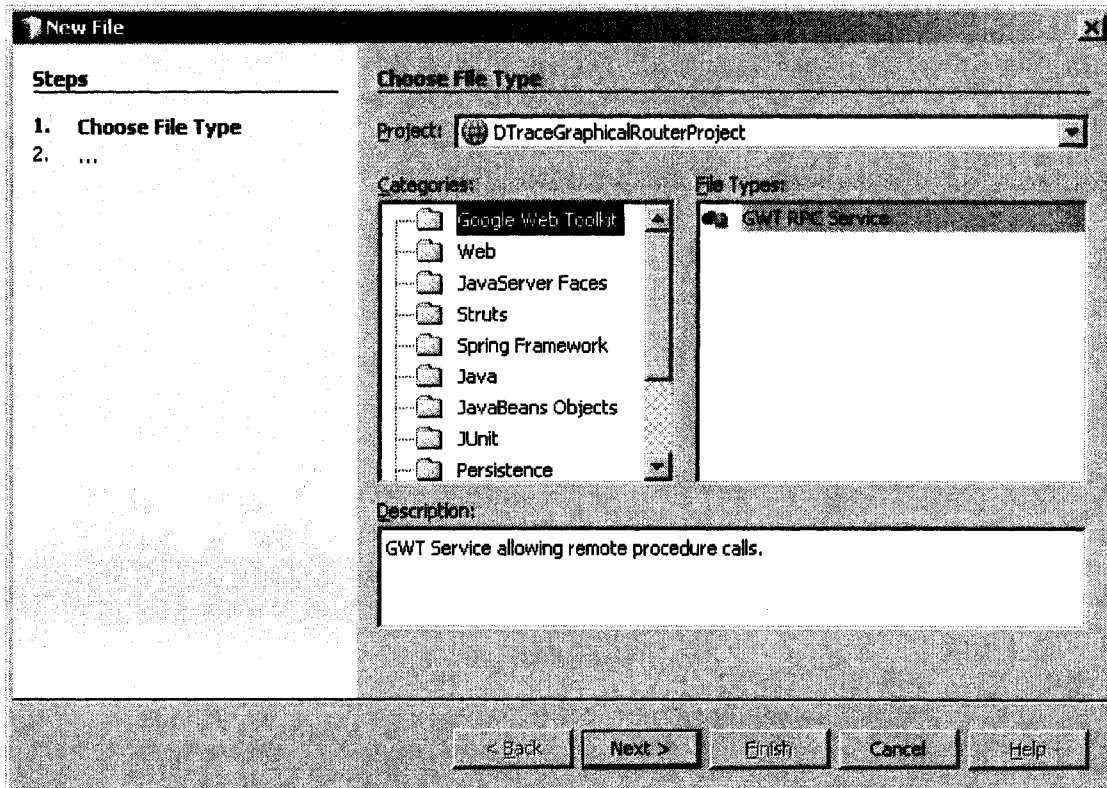


Figure 30. Creating an RPC in NetBeans

Creating an RPC is made rather simple in NetBeans with the GWT4NB plugin.

Using Additional Java Libraries

To use additional Java libraries—often packaged in JavaArchive .jar files, add them using the project properties wizard—as seen in Figure 31.

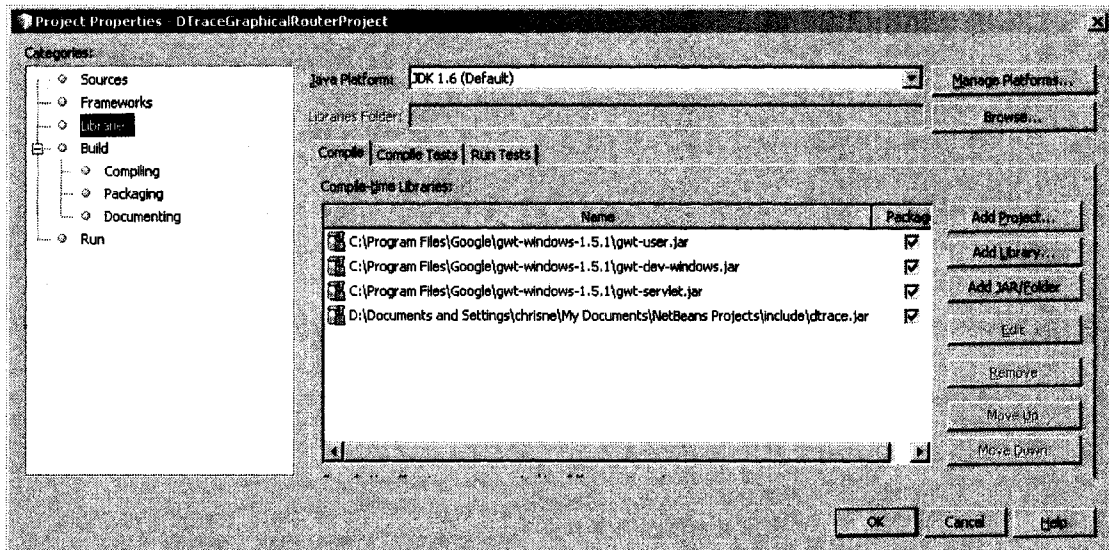


Figure 31. Using Additional Java Libraries

To use additional Java libraries in a NetBeans project, add them in the project properties wizard.

Building and Hosting the Web Application Locally

For basic verification of functionality or simple debugging, the author frequently found it useful to build the web application and host it locally on the development machine using the Apache Tomcat web server (as specified in the Project creation, see the *Creating a Web Application Project* section in this appendix, p. 111). To do this in NetBeans, select *Run* → *Run Main Project*. Output like that in Figure 32 will be displayed in the NetBeans console, and the web application will launch in a local browser window—like in Figure 33.

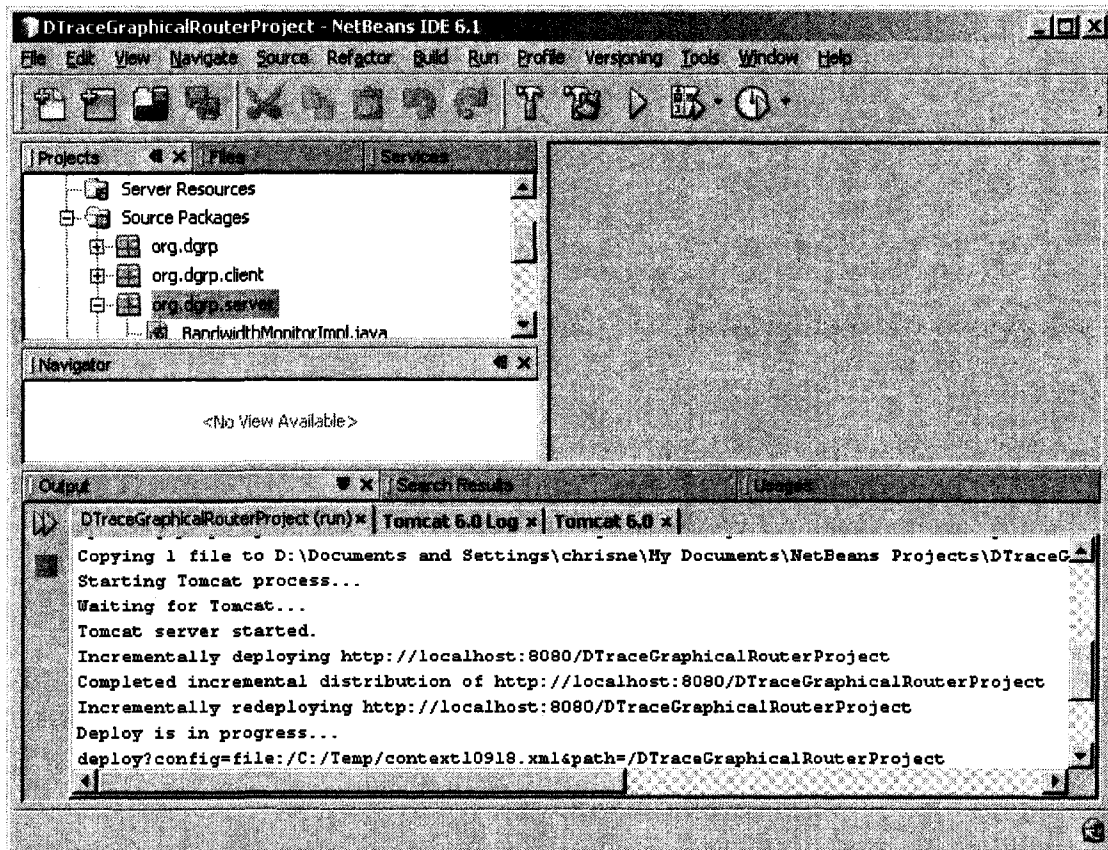


Figure 32. Local Deployment of a Web Application from NetBeans

NetBeans can build and host a web application on the development machine using a web server specified during the creation of the project.

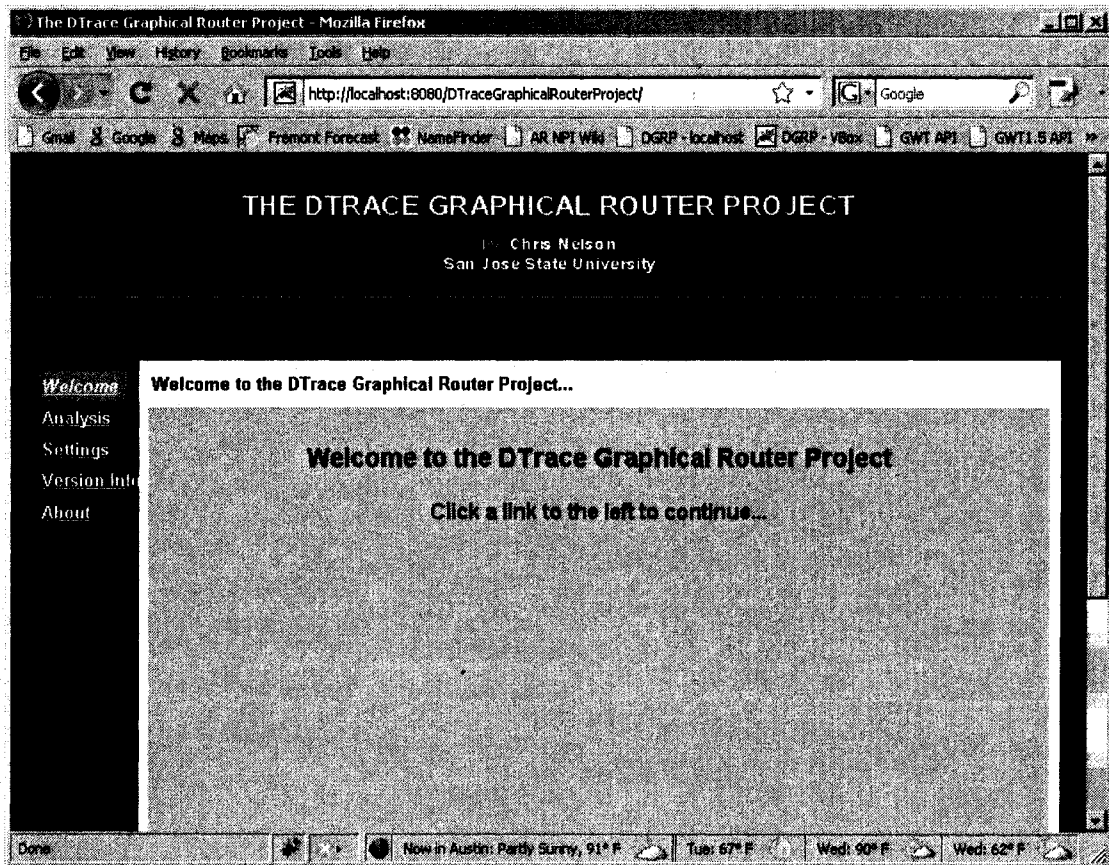


Figure 33. NetBeans Hosting a Web Application Locally

NetBeans can build and host a web application on the local machine using a web server specified during the creation of the project.

Building the Web Application for Deployment

When ready to deploy the complete web application, NetBeans can build and create a JavaArchive .war file that can be deployed using a web server like Apache Tomcat. To do this, select *Build* → *Clean and Build Main Project*. This will produce a .war file in the project's *dist* directory—as seen in Figure 34.

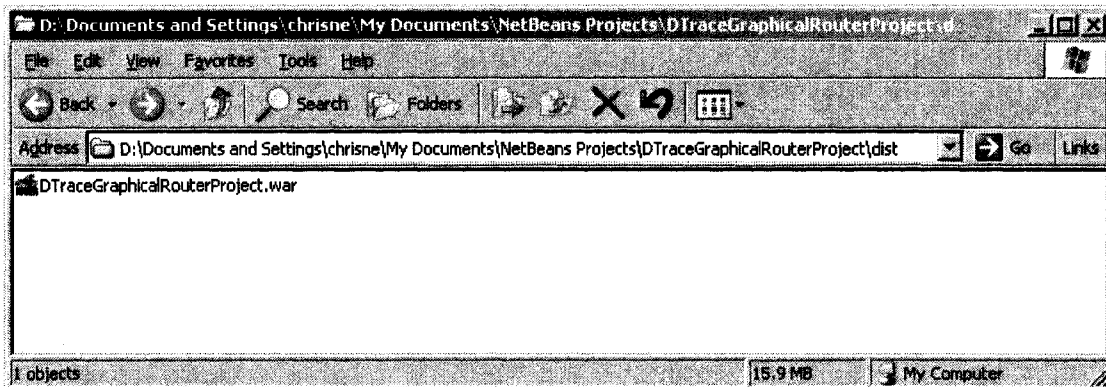


Figure 34. A Web Application Ready for Deployment

NetBeans can build a web application and package it in a .war file for deployment on a web server.

APPENDIX D. SOURCE CODE – GENERAL FILES

The complete source code for the reference implementation of this framework is provided in this and other appendices to this document for the reader's easy reference. For the simplest viewing experience or to use the code without copying and pasting it into a new source file, the reader is encouraged to review the soft-copy files available on the CD-ROM included with this document.

index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta name='gwt:module'
content='org.dgrp.DTraceGraphicalRouterProject=org.dgrp.DTraceGra
phicalRouterProject'>
    <title>The DTrace Graphical Router Project</title>
    <link rel="stylesheet" href="dgrp.css">
  </head>
  <body>
    <iframe src="javascript:''" id='__gwt_historyFrame'
style='width:0;height:0;border:0'></iframe>
    <center></center>
    <hr>
    <br><br>
    <script language="javascript"
src="org.dgrp.DTraceGraphicalRouterProject/org.dgrp.DTraceGraphical
RouterProject.nocache.js"></script>
  </body>
</html>
```

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>GetVersionInfo</servlet-name>
    <servlet-
class>org.dgrp.server.GetVersionInfoImpl</servlet-class>
    </servlet>
    <servlet>
    <servlet-name>BandwidthMonitor</servlet-name>
    <servlet-
class>org.dgrp.server.BandwidthMonitorImpl</servlet-class>
    </servlet>
    <servlet-mapping>
    <servlet-name>GetVersionInfo</servlet-name>
    <url-
pattern>/org.dgrp.DTraceGraphicalRouterProject/getversioninfo</ur
l-pattern>
    </servlet-mapping>
    <servlet-mapping>
    <servlet-name>BandwidthMonitor</servlet-name>
    <url-
pattern>/org.dgrp.DTraceGraphicalRouterProject/bandwidthmonitor</
url-pattern>
    </servlet-mapping>
    <session-config>
    <session-timeout>
      30
    </session-timeout>
    </session-config>
    <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
  </web-app>

```

dgrp.css

```

body {
  background-color: black;
  color: white;
  font-family: Arial, sans-serif;
  font-weight: bold;
  font-size: medium;
  margin: 20px 20px 20px 20px;
}

```

```
code {
}

a {
    color: white;
}

a:visited {
    color: white;
}

.gwt-BorderedPanel {
}

.gwt-Button {
}

.gwt-Canvas {
}

.gwt-CheckBox {
}

.gwt-DialogBox {
}

.gwt-DialogBox .Caption {
}

.gwt-FileUpload {
}

.gwt-Frame {
}

.gwt-HorizontalSplitter .Bar {
}

.gwt-VerticalSplitter .Bar {
}

.gwt-HTML {
    font-size: smaller;
}

.gwt-Hyperlink {
}

.gwt-Image {
}
```

```

.gwt-Label {
}

.gwt-ListBox {
}

.gwt-MenuBar {
  background-color: #444444;
  color: white;
  border: 1px solid white;
  cursor: default;
}

.gwt-MenuBar .gwt-MenuItem {
  padding: 1px 4px 1px 4px;
  font-size: smaller;
  cursor: default;
  color: white;
}

.gwt-MenuBar .gwt-MenuItem-selected {
  background-color: #222222;
  color: white;
}

.gwt-PasswordTextBox {
}

.gwt-RadioButton {
}

.gwt-TabPanel {
}

.gwt-TabPanelBottom {
}

.gwt-TabBar {
}

.gwt-TabBar .gwt-TabBarFirst {
}

.gwt-TabBar .gwt-TabBarRest {
}

.gwt-TabBar .gwt-TabBarItem {
}

.gwt-TabBar .gwt-TabBarItem-selected {
}

```

```

}

.gwt-TextArea {
}

.gwt-TextBox {
}

.gwt-Tree {
}

.gwt-Tree .gwt-TreeItem {
}

.gwt-Tree .gwt-TreeItem-selected {
}

.gwt-StackPanel {
}

.gwt-StackPanel .gwt-StackPanelItem {
}

.gwt-StackPanel .gwt-StackPanelItem-selected {
}

/*
-----
* Styling added for the DTrace Graphical Router Project
*/

.dgrp-MainPanel {
border: 8px solid white;
background-color: #cccccc;
color: black;
width: 100%;
height: 35em;
}

.dgrp-Heading {
background-color: white;
color: black;
padding: 10px 10px 2px 10px;
font-size: small;
}

.dgrp-Sidebar-List {
margin-top: 8px;
margin-bottom: 8px;
font-size: smaller;
}

```



```

}

.dgrp-Sidebar-List .dgrp-Sidebar-Item {
    width: 100%;
    padding: 0.3em;
    padding-right: 16px;
    cursor: pointer;
    cursor: hand;
}

.dgrp-Sidebar-List .dgrp-Sidebar-Item-Selected {
    background-color: #999999;
    color: black;
    font-weight: bold;
    font-style: italic;
}

.dgrp-Images-Image {
    margin: 10px;
}

.dgrp-Images-Wireless {
    margin-left: 75px;
}

.dgrp-Images-RouterStats {
    margin-right: 75px;
}

.dgrp-Images-Laptop0Pipe {
    margin-left: 100px;
}

.dgrp-Images-Laptop1Pipe {
    margin: 0px;
}

.dgrp-Images-Laptop2Pipe {
    margin: 0px;
}

.dgrp-Images-Laptop3Pipe {
    margin-right: 100px;
}

.dgrp-Images-Button {
}

.dgrp-Layouts {
}

```

```
.dgrp-Layouts-Label {  
}  
  
.dgrp-Layouts-Scroller {  
}  
  
.dgrp-Popups-Popup {  
}  
  
.dgrp-About-Prose {  
    margin: 8px;  
}  
  
.dgrp-Stat-Table {  
    font-size: small;  
}
```

license.txt

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code,

documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You

a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of

Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted

against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/DTraceGraphicalRouterProject"/>
```

gwt.properties

```
# The name of the module to compile
gwt.module=org.dgrp.DTraceGraphicalRouterProject
# Path of the GWT installation directory. Use Internet-Standard of
forward slashes for this path
gwt.install.dir=C:/Program Files/Google/gwt-windows-1.4.62
# Folder within the web app context path where the output
# of the GWT module compilation will be stored.
gwt.output.dir=/org.dgrp.DTraceGraphicalRouterProject
# Script output style: OBF[USCATED], PRETTY, or DETAILED
gwt.compiler.output.style=OBF
# The level of logging detail: ERROR, WARN, INFO, TRACE, DEBUG,
gwt.compiler.logLevel=WARN
# Script output style: OBF[USCATED], PRETTY, or DETAILED
gwt.shell.output.style=OBF
# The level of logging detail: ERROR, WARN, INFO, TRACE, DEBUG,
gwt.shell.logLevel=WARN
```

APPENDIX E. SOURCE CODE – PACKAGE org.dgrp

The complete source code for the reference implementation of this framework is provided in this and other appendices to this document for the reader's easy reference. For the simplest viewing experience or to use the code without copying and pasting it into a new source file, the reader is encouraged to review the soft-copy files available on the CD-ROM included with this document.

DTraceGraphicalRouterProject.gwt.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module>
  <inherits name="com.google.gwt.user.User"/>
  <entry-point class="org.dgrp.client.DGRPEnterPoint"/>
  <!-- Do not define servlets here, use web.xml -->
</module>
```

APPENDIX F. SOURCE CODE – PACKAGE org.dgrp.client

The complete source code for the reference implementation of this framework is provided in this and other appendices to this document for the reader's easy reference. For the simplest viewing experience or to use the code without copying and pasting it into a new source file, the reader is encouraged to review the soft-copy files available on the CD-ROM included with this document.

About.java

```
package org.dgrp.client;

import com.google.gwt.user.client.ui.HTML;

/**
 * About page.
 */
public class About extends SidebarItem {

    public static SidebarItemInfo init() {
        return new SidebarItemInfo("About", "About the DTrace
Graphical Router Project...") {
            public SidebarItem createInstance() {
                return new About();
            }
        };
    }

    public About() {
        initWidget(new HTML(
            "<div class='dgrp-About-Prose'>" +
            "<p>The DTrace Graphical Router Project was created by
Chris Nelson " +
            "in partial fulfillment of the requirements of the San Jose
State " +
            "University Computer Engineering Master's Degree Program.</
p><br>" +
            "<h2>Project Title</b></center></h2>" +
```



```

        "<p><i>A Framework for Graphical Analysis of a Home-Network
" +
        "Router Using DTrace</i></p><br>" +
        "<h2>Project Abstract</h2>" +
        "<p><i>Simple network routers used in homes and small
offices " +
        "typically lack tools for performance monitoring and
analysis that " +
        "would be useful to the normally novice users of these
products. " +
        "Sophisticated network simulation and analysis applications
require too " +
        "much effort for a typical user to consider, but including
some simple " +
        "tools in the router software would enable the common user
to more " +
        "quickly and completely understand the reason or reasons
for " +
        "performance problems.</p>" +
        "<p>DTrace—a dynamic tracing framework first released in
Solaris 10 " +
        "and currently being ported to Linux—provides the
opportunity to " +
        "gather relevant performance data from the router itself,
and if " +
        "presented in an easily understood graphical format, the
common user " +
        "will be empowered to understand and address problems more
quickly and " +
        "with less need for additional support. This thesis
addresses the " +
        "This thesis addresses the development of a framework for
and " +
        "reference implementation of graphical analysis tools for
analyzing " +
        "common network routers using DTrace.</i></p><br>" +
        "<h2>Useful Links</h2>" +
        "<a href=\"http://unknown\">Open-Source Project
Code</a><br>" +
        "<a href=\"http://unknown\">Online Documentation</a>" +
        "</div>",
        true));
    }

    public void onShow() {
    }
}

```

Analysis.java

```
package org.dgrp.client;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.Timer;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;
import com.google.gwt.user.client.ui.VerticalPanel;

public class Analysis extends SidebarItem {

    public static SidebarItemInfo init() {
        return new SidebarItemInfo("Analysis",
            "Use the Menu Below to Select Available Analysis
Features...") {

            public SidebarItem createInstance() {
                return new Analysis();
            }
        };
    }

    private ImagePanel imgPanel;
    private AnalysisMenu analysisMenu;
    private TopologyInfo topoInfo;
    private BandwidthMonitorAsync bwMonitorService;
    private String subnet = "10.0.0.0";

    private final int MAX_NODES = 4;

    public Analysis() {

        imgPanel = new ImagePanel();
        analysisMenu = new AnalysisMenu(imgPanel);
        VerticalPanel vp = new VerticalPanel();
        vp.setWidth("100%");
        vp.setHeight("100%");
        vp.add(analysisMenu);
        vp.add(imgPanel);
        initWidget(vp);

        // Create asynchronous callbacks to handle results
        final AsyncCallback ipCallback = new AsyncCallback() {
            public void onSuccess(Object result) {
                processIPUpdates(result);
            }
            public void onFailure(Throwable caught) {
                //ignore for now
            }
        };
    }
};
```

```

final AsyncCallback statCallback = new AsyncCallback() {
    public void onSuccess(Object result) {
        processStatUpdates(result);
    }

    public void onFailure(Throwable caught) {
        //ignore for now
    }
};

final AsyncCallback emptyCallback = new AsyncCallback() {
    public void onSuccess(Object result) {
        //ignore
    }
    public void onFailure(Throwable caught) {
        //ignore
    }
};

bwMonitorService = getBandwidthMonitorService();
bwMonitorService.startServiceCountDataBytes(subnet,
emptyCallback);
bwMonitorService.startServiceCountPackets(subnet,
emptyCallback);
topoInfo = new TopologyInfo(MAX_NODES);

//Create timers to repeatedly trigger updates
bwMonitorService.getRefreshedIPs(ipCallback);
Timer ipRefresh = new Timer() {
    public void run() {
        bwMonitorService.getRefreshedIPs(ipCallback);
    }
};
ipRefresh.scheduleRepeating(10000);

Timer statUpdate = new Timer() {
    public void run() {
        for (int i=0; i<=topoInfo.getMaxNodes(); i++) {
            if (!(topoInfo.getAddress(i).equals(null))) {
                bwMonitorService.getBandwidthInUse(
                    topoInfo.getAddress(i),
statCallback);
            }
        }
    }
};
statUpdate.scheduleRepeating(1000);
}

```

```

    public static BandwidthMonitorAsync
getBandwidthMonitorService(){
        BandwidthMonitorAsync service =
(BandwidthMonitorAsync) GWT.create(BandwidthMonitor.class);
        ServiceDefTarget endpoint = (ServiceDefTarget)
service;
        String moduleRelativeURL = GWT.getModuleBaseURL() +
"bandwidthmonitor";
        endpoint.setServiceEntryPoint(moduleRelativeURL);
        return service;
    }

    private void processIPUpdates(Object result) {
String[] newAddrs = (String[]) result;
for (int i=0; i<MAX_NODES; i++) {
    imgPanel.hideLaptop(i);
    imgPanel.hideLaptopPipe(i);
    imgPanel.setLaptopIPAddrLabel(i, null);
}
topoInfo = new TopologyInfo(MAX_NODES);
for (int i=0; i<newAddrs.length; i++) {
    try {
        topoInfo.setAddress(i, newAddrs[i]);
        imgPanel.setLaptopIPAddrLabel(i, newAddrs[i]);
        imgPanel.showLaptop(i);
    } catch (Exception e) {
        //ignore for now
    }
}
}

    private void processStatUpdates(Object result) {
BandwidthInfo bwInfo = (BandwidthInfo) result;
try {
    imgPanel.showLaptopPipe(topoInfo.findPosition(
        bwInfo.getIPAddress()),
bwInfo.getBandwidthInUse());
    imgPanel.setLaptopStatValue(topoInfo.findPosition(bwI
nfo.
        getIPAddress()), imgPanel.NUM_PACKETS, "" +
        bwInfo.getTotalPacketsSentToAndReceivedFrom()
);
    imgPanel.setLaptopStatValue(topoInfo.findPosition(bwI
nfo.
        getIPAddress()), imgPanel.NUM_DATA_BYTES, "" +
        bwInfo.getTotalDataBytesSentToAndReceivedFrom
());
} catch (Exception e) {
    //skip addresses not currently tracked
}
}

```

```

        public void onShow() {
        }
    }
}

```

AnalysisMenu.java

```

package org.dgrp.client;

import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.MenuBar;
import com.google.gwt.user.client.ui.MenuItem;
import com.google.gwt.user.client.Command;
import com.google.gwt.user.client.Window;

/**
 *
 * @author chrisne
 */
public class AnalysisMenu extends Composite {

    // Declare menu bars and items
    private MenuBar menu = new MenuBar();
    private MenuBar menu_router = new MenuBar(true);
    private MenuItem routerStatsOnOff;
    private MenuItem menu_router_int_status;
    private MenuItem menu_router_int_max_bw;
    private MenuItem menu_router_int_cur_bw;
    private MenuItem menu_router_int_speed;
    private MenuBar menu_router_CPUUtil = new MenuBar(true);
    private MenuItem menu_router_CPUUtil_tot;
    private MenuItem menu_router_CPUUtil_byproc;
    private MenuBar menu_router_memUtil = new MenuBar(true);
    private MenuBar menu_router_memUtil_total = new MenuBar(true);
    private MenuItem menu_router_memUtil_total_real;
    private MenuItem menu_router_memUtil_total_virtual;
    private MenuBar menu_router_memUtil_byProc = new MenuBar(true);
    private MenuItem menu_router_memUtil_byProc_real;
    private MenuItem menu_router_memUtil_byProc_virtual;
    private MenuBar menu_TCP = new MenuBar(true);
    private MenuItem menu_TCP_inboundDrops;
    private MenuItem menu_TCP_connState;
    private MenuBar menu_TCP_packets = new MenuBar(true);
    private MenuItem menu_TCP_packets_recvd;
    private MenuItem menu_TCP_packets_sent;
    private MenuItem menu_TCP_packets_recvdByTime;
    private MenuItem menu_TCP_packets_sentByTime;
}

```

```

private MenuBar menu_TCP_bytes = new MenuBar(true);
private MenuItem menu_TCP_bytes_recvd;
private MenuItem menu_TCP_bytes_sent;
private MenuItem menu_TCP_bytes_recvdByTime;
private MenuItem menu_TCP_bytes_sentByTime;
private MenuBar menu_TCP_bits = new MenuBar(true);
private MenuItem menu_TCP_bits_recvd;
private MenuItem menu_TCP_bits_sent;
private MenuItem menu_TCP_bits_recvdByTime;
private MenuItem menu_TCP_bits_sentByTime;
private MenuBar menu_TCP_errors = new MenuBar(true);
private MenuItem menu_TCP_errors_hdr;
private MenuItem menu_TCP_errors_chksum;
private MenuItem menu_TCP_errors_hdrByTime;
private MenuItem menu_TCP_errors_chksumByTime;
private MenuBar menu_TCP_buffers = new MenuBar(true);
private MenuBar menu_TCP_buffers_send = new MenuBar(true);
private MenuItem menu_TCP_buffers_send_status;
private MenuItem menu_TCP_buffers_send_overflows;
private MenuBar menu_TCP_buffers_receive = new MenuBar(true);
private MenuItem menu_TCP_buffers_receive_status;
private MenuItem menu_TCP_buffers_receive_overflows;
private MenuBar menu_UDP = new MenuBar(true);
private MenuBar menu_UDP_packets = new MenuBar(true);
private MenuItem menu_UDP_packets_recvd;
private MenuItem menu_UDP_packets_sent;
private MenuItem menu_UDP_packets_recvdByTime;
private MenuItem menu_UDP_packets_sentByTime;
private MenuBar menu_UDP_bytes = new MenuBar(true);
private MenuItem menu_UDP_bytes_recvd;
private MenuItem menu_UDP_bytes_sent;
private MenuItem menu_UDP_bytes_recvdByTime;
private MenuItem menu_UDP_bytes_sentByTime;
private MenuBar menu_UDP_bits = new MenuBar(true);
private MenuItem menu_UDP_bits_recvd;
private MenuItem menu_UDP_bits_sent;
private MenuItem menu_UDP_bits_recvdByTime;
private MenuItem menu_UDP_bits_sentByTime;
private MenuBar menu_UDP_errors = new MenuBar(true);
private MenuItem menu_UDP_errors_hdr;
private MenuItem menu_UDP_errors_chksum;
private MenuItem menu_UDP_errors_hdrByTime;
private MenuItem menu_UDP_errors_chksumByTime;
private MenuBar menu_UDP_buffers = new MenuBar(true);
private MenuBar menu_UDP_buffers_send = new MenuBar(true);
private MenuItem menu_UDP_buffers_send_status;
private MenuItem menu_UDP_buffers_send_overflows;
private MenuBar menu_UDP_buffers_receive = new MenuBar(true);
private MenuItem menu_UDP_buffers_receive_status;
private MenuItem menu_UDP_buffers_receive_overflows;
private MenuBar menu_IPv4 = new MenuBar(true);

```

```

private MenuBar menu_IPv4_packets = new MenuBar(true);
private MenuItem menu_IPv4_packets_recvd;
private MenuItem menu_IPv4_packets_sent;
private MenuItem menu_IPv4_packets_sent_and_rcvd;
private MenuItem menu_IPv4_packets_recvdByTime;
private MenuItem menu_IPv4_packets_sentByTime;
private MenuBar menu_IPv4_bytes = new MenuBar(true);
private MenuItem menu_IPv4_bytes_recvd;
private MenuItem menu_IPv4_bytes_sent;
private MenuItem menu_IPv4_bytes_sent_and_rcvd;
private MenuItem menu_IPv4_bytes_recvdByTime;
private MenuItem menu_IPv4_bytes_sentByTime;
private MenuBar menu_IPv4_bits = new MenuBar(true);
private MenuItem menu_IPv4_bits_recvd;
private MenuItem menu_IPv4_bits_sent;
private MenuItem menu_IPv4_bits_sent_and_rcvd;
private MenuItem menu_IPv4_bits_recvdByTime;
private MenuItem menu_IPv4_bits_sentByTime;
private MenuBar menu_IPv4_errors = new MenuBar(true);
private MenuItem menu_IPv4_errors_hdr;
private MenuItem menu_IPv4_errors_chksum;
private MenuItem menu_IPv4_errors_hdrByTime;
private MenuItem menu_IPv4_errors_chksumByTime;
private MenuBar menu_IPv4_buffers = new MenuBar(true);
private MenuBar menu_IPv4_buffers_send = new MenuBar(true);
private MenuItem menu_IPv4_buffers_send_status;
private MenuItem menu_IPv4_buffers_send_overflows;
private MenuBar menu_IPv4_buffers_receive = new MenuBar(true);
private MenuItem menu_IPv4_buffers_receive_status;
private MenuItem menu_IPv4_buffers_receive_overflows;
private MenuBar menu_IPv4_responseTime = new MenuBar(true);
private MenuItem menu_IPv4_responseTime_max;
private MenuItem menu_IPv4_responseTime_min;
private MenuItem menu_IPv4_responseTime_avg;
private MenuBar menu_IPv6 = new MenuBar(true);
private MenuBar menu_rawIP = new MenuBar(true);
private MenuBar menu_SCTP = new MenuBar(true);
private MenuBar menu_wireless = new MenuBar(true);
private MenuItem wirelessOnOff;
private MenuBar menu_legend = new MenuBar(true);
private MenuBar menu_test = new MenuBar(true);
private MenuItem testItem;

//Strings for special menu characters
public static final String notsup = "&nbsp;<DGRP-NOTSUP><DGRP-SPLIT><FONT color=\"white\">&#8855;</FONT>&nbsp;&nbsp;&nbsp;<DGRP-SPLIT>";
public static final String on = "&nbsp;<DGRP-ON><DGRP-SPLIT><FONT color=\"green\">&#8730;</FONT>&nbsp;&nbsp;&nbsp;<DGRP-SPLIT>";
public static final String off = "&nbsp;<DGRP-OFF><DGRP-

```

```

SPLIT<FONT color=\"red\">&#8855;</FONT>&nbsp;&nbsp;&nbsp;<DGRP-SPLIT>;
public static final String submen = "&nbsp;&nbsp;&nbsp;&#187;";

public AnalysisMenu(final ImagePanel imgPanel) {
    /* Create the not-supported command */
    Command notSupported = new Command () {
        public void execute() {
            Window.alert("This feature is not yet implemented.");
        }
    };

    /* Create the legendInfo command */
    Command legendInfo = new Command () {
        public void execute() {
            Window.alert("The menu items in the legend perform no
action.");
        }
    };

    /* Create the test command */
    Command testCmd = new Command () {
        public void execute() {
            testItem.setHTML(changeMenuStatus(testItem.getHTML())
);
            imgPanel.showAllGraphics();
        }
    };

    /* Create the wireless command */
    Command wirelessCmd = new Command () {
        public void execute() {
            wirelessOnOff.setHTML(changeMenuStatus(wirelessOnOff.
getHTML()));
            if (imgPanel.isWirelessVisible() == true) {
                imgPanel.hideWireless();
            }
            else {
                imgPanel.showWireless();
            }
        }
    };

    /* Create the routerStats command */
    Command routerStatsCmd = new Command () {
        public void execute() {
            routerStatsOnOff.setHTML(changeMenuStatus(routerStats
OnOff.getHTML()));
            if (imgPanel.isRouterStatsVisible() == true) {
                imgPanel.hideRouterStats();
            }
            else {

```



```

        imgPanel.showRouterStats();
    }
};

/* Create the IPv4 Packets Sent and Received command */
Command IPv4PacketsSentAndRcvdCmd = new Command () {
    public void execute() {
        menu_IPv4_packets_sent_and_rcvd.setHTML(changeMenuSta
tus(menu_IPv4_packets_sent_and_rcvd.getHTML()));
        if (imgPanel.isIPv4PacketsSentAndRcvdVisible() ==
true) {
            imgPanel.hideIPv4PacketsSentAndRcvd();
        }
        else {
            imgPanel.showIPv4PacketsSentAndRcvd();
        }
    }
};

/* Create the IPv4 Data Bytes Sent and Received command */
Command IPv4BytesSentAndRcvdCmd = new Command () {
    public void execute() {
        menu_IPv4_bytes_sent_and_rcvd.setHTML(changeMenuStatu
s(menu_IPv4_bytes_sent_and_rcvd.getHTML()));
        if (imgPanel.isIPv4BytesSentAndRcvdVisible() == true)
        {
            imgPanel.hideIPv4BytesSentAndRcvd();
        }
        else {
            imgPanel.showIPv4BytesSentAndRcvd();
        }
    }
};

/* Create the "Router" menu */
menu_router.setAutoOpen(true);
menu_router_CPUUtil.setAutoOpen(true);
menu_router_memUtil.setAutoOpen(true);
menu_router_memUtil_total.setAutoOpen(true);
menu_router_memUtil_byProc.setAutoOpen(true);
routerStatsOnOff = new MenuItem(off + "Enable/Disable", true,
routerStatsCmd);
menu_router.addItem(routerStatsOnOff);
menu_router_int_status = new MenuItem(notsup + "Interface
Status", true, notSupported);
menu_router.addItem(menu_router_int_status);
menu_router_int_max_bw = new MenuItem(notsup + "Interface
Maximum Bandwidth", true,
notSupported);

```

```

        menu_router.addItem(menu_router_int_max_bw);
        menu_router_int_cur_bw = new MenuItem(notsup + "Interface
Current Bandwidth Usage", true,
        notSupported);
        menu_router.addItem(menu_router_int_cur_bw);
        menu_router_int_speed = new MenuItem(notsup + "Interface
Speed", true, notSupported);
        menu_router.addItem(menu_router_int_speed);
        menu_router_CPUUtil_tot = new MenuItem(notsup + "Total",
true, notSupported);
        menu_router_CPUUtil.addItem(menu_router_CPUUtil_tot);
        menu_router_CPUUtil_byproc = new MenuItem(notsup + "By
Process", true, notSupported);
        menu_router_CPUUtil.addItem(menu_router_CPUUtil_byproc);
        menu_router.addItem("CPU Utilization" + submenu, true,
        menu_router_CPUUtil);
        menu_router_memUtil_total_real = new MenuItem(notsup +
"Real", true, notSupported);
        menu_router_memUtil_total.addItem(menu_router_memUtil_total_r
eal);
        menu_router_memUtil_total_virtual = new MenuItem(notsup +
"Virtual", true, notSupported);
        menu_router_memUtil_total.addItem(menu_router_memUtil_total_v
irtual);
        menu_router_memUtil.addItem("Total" + submenu, true,
        menu_router_memUtil_total);
        menu_router_memUtil_byProc_real = new MenuItem(notsup +
"Real", true, notSupported);
        menu_router_memUtil_byProc.addItem(menu_router_memUtil_byProc
_real);
        menu_router_memUtil_byProc_virtual = new MenuItem(notsup +
"Virtual", true, notSupported);
        menu_router_memUtil_byProc.addItem(menu_router_memUtil_byProc
_virtual);
        menu_router_memUtil.addItem("By Process" + submenu, true,
        menu_router_memUtil_byProc);
        menu_router.addItem("Memory Utilization" + submenu, true,
        menu_router_memUtil);

```

```

/* Create the "TCP" menu */
menu_TCP.setAutoOpen(true);
menu_TCP_packets.setAutoOpen(true);
menu_TCP_bytes.setAutoOpen(true);
menu_TCP_bits.setAutoOpen(true);
menu_TCP_errors.setAutoOpen(true);
menu_TCP_buffers.setAutoOpen(true);
menu_TCP_buffers_send.setAutoOpen(true);
menu_TCP_buffers_receive.setAutoOpen(true);
menu_TCP_inboundDrops = new MenuItem(notsup + "Inbound TCP

```

```

Connection Drops", true,
    notSupported);
    menu_TCP.addItem(menu_TCP_inboundDrops);
    menu_TCP_connState = new MenuItem(notsup + "Connection
State", true,
    notSupported);
    menu_TCP.addItem(menu_TCP_connState);
    menu_TCP_packets_recvd = new MenuItem(notsup + "Received",
true, notSupported);
    menu_TCP_packets.addItem(menu_TCP_packets_recvd);
    menu_TCP_packets_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_TCP_packets.addItem(menu_TCP_packets_sent);
    menu_TCP_packets_recvdByTime = new MenuItem(notsup +
"Received per Unit of Time", true,
    notSupported);
    menu_TCP_packets.addItem(menu_TCP_packets_recvdByTime);
    menu_TCP_packets_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_TCP_packets.addItem(menu_TCP_packets_sentByTime);
    menu_TCP.addItem("Packets" + submenu, true, menu_TCP_packets);
    menu_TCP_bytes_recvd = new MenuItem(notsup + "Received",
true, notSupported);
    menu_TCP_bytes.addItem(menu_TCP_bytes_recvd);
    menu_TCP_bytes_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_TCP_bytes.addItem(menu_TCP_bytes_sent);
    menu_TCP_bytes_recvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_TCP_bytes.addItem(menu_TCP_bytes_recvdByTime);
    menu_TCP_bytes_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_TCP_bytes.addItem(menu_TCP_bytes_sentByTime);
    menu_TCP.addItem("Bytes" + submenu, true, menu_TCP_bytes);
    menu_TCP_bits_recvd = new MenuItem(notsup + "Received", true,
notSupported);
    menu_TCP_bits.addItem(menu_TCP_bits_recvd);
    menu_TCP_bits_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_TCP_bits.addItem(menu_TCP_bits_sent);
    menu_TCP_bits_recvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_TCP_bits.addItem(menu_TCP_bits_recvdByTime);
    menu_TCP_bits_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_TCP_bits.addItem(menu_TCP_bits_sentByTime);

```

```

        menu_TCP.addItem("Bits" + submenu, true, menu_TCP_bits);
        menu_TCP_errors_chksum = new MenuItem(notsup + "Checksum",
true, notSupported);
        menu_TCP_errors.addItem(menu_TCP_errors_chksum);
        menu_TCP_errors_hdr = new MenuItem(notsup + "Header", true,
notSupported);
        menu_TCP_errors.addItem(menu_TCP_errors_hdr);
        menu_TCP_errors_chksumByTime = new MenuItem(notsup +
"Checksum per Unit of Time", true,
notSupported);
        menu_TCP_errors.addItem(menu_TCP_errors_chksumByTime);
        menu_TCP_errors_hdrByTime = new MenuItem(notsup + "Header per
Unit of Time", true,
notSupported);
        menu_TCP_errors.addItem(menu_TCP_errors_hdrByTime);
        menu_TCP.addItem("Errors" + submenu, true, menu_TCP_errors);
        menu_TCP_buffers_send_status = new MenuItem(notsup +
"Status", true, notSupported);
        menu_TCP_buffers_send.addItem(menu_TCP_buffers_send_status);
        menu_TCP_buffers_send_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
        menu_TCP_buffers_send.addItem(menu_TCP_buffers_send_overflows
);
        menu_TCP_buffers.addItem("Send" + submenu, true,
menu_TCP_buffers_send);
        menu_TCP_buffers_receive_status = new MenuItem(notsup +
"Status", true, notSupported);
        menu_TCP_buffers_receive.addItem(menu_TCP_buffers_receive_sta
tus);
        menu_TCP_buffers_receive_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
        menu_TCP_buffers_receive.addItem(menu_TCP_buffers_receive_ove
rflows);
        menu_TCP_buffers.addItem("Receive" + submenu, true,
menu_TCP_buffers_receive);
        menu_TCP.addItem("Buffers" + submenu, true, menu_TCP_buffers);

/* Create the "UDP" menu */
menu_UDP.setAutoOpen(true);
menu_UDP_packets.setAutoOpen(true);
menu_UDP_bytes.setAutoOpen(true);
menu_UDP_bits.setAutoOpen(true);
menu_UDP_errors.setAutoOpen(true);
menu_UDP_buffers.setAutoOpen(true);
menu_UDP_buffers_send.setAutoOpen(true);
menu_UDP_buffers_receive.setAutoOpen(true);
menu_UDP_packets_recvd = new MenuItem(notsup + "Received",
true, notSupported);
menu_UDP_packets.addItem(menu_UDP_packets_recvd);
menu_UDP_packets_sent = new MenuItem(notsup + "Sent", true,

```

```

notSupported);
    menu_UDP_packets.addItem(menu_UDP_packets_sent);
    menu_UDP_packets_recvdByTime = new MenuItem(notsup +
"Received per Unit of Time", true,
    notSupported);
    menu_UDP_packets.addItem(menu_UDP_packets_recvdByTime);
    menu_UDP_packets_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_UDP_packets.addItem(menu_UDP_packets_sentByTime);
    menu_UDP.addItem("Packets" + submenu, true, menu_UDP_packets);
    menu_UDP_bytes_recvd = new MenuItem(notsup + "Received",
true, notSupported);
    menu_UDP_bytes.addItem(menu_UDP_bytes_recvd);
    menu_UDP_bytes_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_UDP_bytes.addItem(menu_UDP_bytes_sent);
    menu_UDP_bytes_recvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_UDP_bytes.addItem(menu_UDP_bytes_recvdByTime);
    menu_UDP_bytes_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_UDP_bytes.addItem(menu_UDP_bytes_sentByTime);
    menu_UDP.addItem("Bytes" + submenu, true, menu_UDP_bytes);
    menu_UDP_bits_recvd = new MenuItem(notsup + "Received", true,
notSupported);
    menu_UDP_bits.addItem(menu_UDP_bits_recvd);
    menu_UDP_bits_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_UDP_bits.addItem(menu_UDP_bits_sent);
    menu_UDP_bits_recvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_UDP_bits.addItem(menu_UDP_bits_recvdByTime);
    menu_UDP_bits_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_UDP_bits.addItem(menu_UDP_bits_sentByTime);
    menu_UDP.addItem("Bits" + submenu, true, menu_UDP_bits);
    menu_UDP_errors_chksum = new MenuItem(notsup + "Checksum",
true, notSupported);
    menu_UDP_errors.addItem(menu_UDP_errors_chksum);
    menu_UDP_errors_hdr = new MenuItem(notsup + "Header", true,
notSupported);
    menu_UDP_errors.addItem(menu_UDP_errors_hdr);
    menu_UDP_errors_chksumByTime = new MenuItem(notsup +
"Checksum per Unit of Time", true,
    notSupported);
    menu_UDP_errors.addItem(menu_UDP_errors_chksumByTime);

```

```

        menu_UDP_errors_hdrByTime = new MenuItem(notsup + "Header per
Unit of Time", true,
        notSupported);
        menu_UDP_errors.addItem(menu_UDP_errors_hdrByTime);
        menu_UDP.addItem("Errors" + submen, true, menu_UDP_errors);
        menu_UDP_buffers_send_status = new MenuItem(notsup +
"Status", true, notSupported);
        menu_UDP_buffers_send.addItem(menu_UDP_buffers_send_status);
        menu_UDP_buffers_send_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
        menu_UDP_buffers_send.addItem(menu_UDP_buffers_send_overflows
);
        menu_UDP_buffers.addItem("Send" + submen, true,
menu_UDP_buffers_send);
        menu_UDP_buffers_receive_status = new MenuItem(notsup +
"Status", true, notSupported);
        menu_UDP_buffers_receive.addItem(menu_UDP_buffers_receive_sta
tus);
        menu_UDP_buffers_receive_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
        menu_UDP_buffers_receive.addItem(menu_UDP_buffers_receive_ove
rflows);
        menu_UDP_buffers.addItem("Receive" + submen, true,
        menu_UDP_buffers_receive);
        menu_UDP.addItem("Buffers" + submen, true, menu_UDP_buffers);

        /* Create the "IPv4" menu */
        menu_IPv4.setAutoOpen(true);
        menu_IPv4_packets.setAutoOpen(true);
        menu_IPv4_bytes.setAutoOpen(true);
        menu_IPv4_bits.setAutoOpen(true);
        menu_IPv4_errors.setAutoOpen(true);
        menu_IPv4_buffers.setAutoOpen(true);
        menu_IPv4_buffers_send.setAutoOpen(true);
        menu_IPv4_buffers_receive.setAutoOpen(true);
        menu_IPv4_responseTime.setAutoOpen(true);
        menu_IPv4_packets_rcvd = new MenuItem(notsup + "Received",
true, notSupported);
        menu_IPv4_packets.addItem(menu_IPv4_packets_rcvd);
        menu_IPv4_packets_sent = new MenuItem(notsup + "Sent", true,
notSupported);
        menu_IPv4_packets.addItem(menu_IPv4_packets_sent);
        menu_IPv4_packets_sent_and_rcvd = new MenuItem(off + "Sent
and Received", true, IPv4PacketsSentAndRcvdCmd);
        menu_IPv4_packets.addItem(menu_IPv4_packets_sent_and_rcvd);
        menu_IPv4_packets_rcvdByTime = new MenuItem(notsup +
"Received per Unit of Time", true,
        notSupported);
        menu_IPv4_packets.addItem(menu_IPv4_packets_rcvdByTime);
        menu_IPv4_packets_sentByTime = new MenuItem(notsup + "Sent

```

```

per Unit of Time", true,
    notSupported);
    menu_IPv4_packets.addItem(menu_IPv4_packets_sentByTime);
    menu_IPv4.addItem("Packets" + submenu, true,
menu_IPv4_packets);
    menu_IPv4_bytes_rcvd = new MenuItem(notsup + "Received",
true, notSupported);
    menu_IPv4_bytes.addItem(menu_IPv4_bytes_rcvd);
    menu_IPv4_bytes_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_IPv4_bytes.addItem(menu_IPv4_bytes_sent);
    menu_IPv4_bytes_sent_and_rcvd = new MenuItem(off + "Sent and
Received", true, IPv4BytesSentAndRcvdCmd);
    menu_IPv4_bytes.addItem(menu_IPv4_bytes_sent_and_rcvd);
    menu_IPv4_bytes_rcvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_IPv4_bytes.addItem(menu_IPv4_bytes_rcvdByTime);
    menu_IPv4_bytes_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_IPv4_bytes.addItem(menu_IPv4_bytes_sentByTime);
    menu_IPv4.addItem("Bytes" + submenu, true, menu_IPv4_bytes);
    menu_IPv4_bits_rcvd = new MenuItem(notsup + "Received",
true, notSupported);
    menu_IPv4_bits.addItem(menu_IPv4_bits_rcvd);
    menu_IPv4_bits_sent = new MenuItem(notsup + "Sent", true,
notSupported);
    menu_IPv4_bits.addItem(menu_IPv4_bits_sent);
    menu_IPv4_bits_sent_and_rcvd = new MenuItem(notsup + "Sent
and Received", true, notSupported);
    menu_IPv4_bits.addItem(menu_IPv4_bits_sent_and_rcvd);
    menu_IPv4_bits_rcvdByTime = new MenuItem(notsup + "Received
per Unit of Time", true,
    notSupported);
    menu_IPv4_bits.addItem(menu_IPv4_bits_rcvdByTime);
    menu_IPv4_bits_sentByTime = new MenuItem(notsup + "Sent per
Unit of Time", true,
    notSupported);
    menu_IPv4_bits.addItem(menu_IPv4_bits_sentByTime);
    menu_IPv4.addItem("Bits" + submenu, true, menu_IPv4_bits);
    menu_IPv4_errors_chksum = new MenuItem(notsup + "Checksum",
true, notSupported);
    menu_IPv4_errors.addItem(menu_IPv4_errors_chksum);
    menu_IPv4_errors_hdr = new MenuItem(notsup + "Header", true,
notSupported);
    menu_IPv4_errors.addItem(menu_IPv4_errors_hdr);
    menu_IPv4_errors_chksumByTime = new MenuItem(notsup +
"Checksum per Unit of Time", true,
    notSupported);
    menu_IPv4_errors.addItem(menu_IPv4_errors_chksumByTime);

```

```

    menu_IPv4_errors_hdrByTime = new MenuItem(notsup + "Header
per Unit of Time", true,
        notSupported);
    menu_IPv4_errors.addItem(menu_IPv4_errors_hdrByTime);
    menu_IPv4.addItem("Errors" + submenu, true, menu_IPv4_errors);
    menu_IPv4_buffers_send_status = new MenuItem(notsup +
"Status", true, notSupported);
    menu_IPv4_buffers_send.addItem(menu_IPv4_buffers_send_status);
    menu_IPv4_buffers_send_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
    menu_IPv4_buffers_send.addItem(menu_IPv4_buffers_send_overflo
ws);
    menu_IPv4_buffers.addItem("Send" + submenu, true,
menu_IPv4_buffers_send);
    menu_IPv4_buffers_receive_status = new MenuItem(notsup +
"Status", true, notSupported);
    menu_IPv4_buffers_receive.addItem(menu_IPv4_buffers_receive_s
tatus);
    menu_IPv4_buffers_receive_overflows = new MenuItem(notsup +
"Overflows", true, notSupported);
    menu_IPv4_buffers_receive.addItem(menu_IPv4_buffers_receive_o
verflows);
    menu_IPv4_buffers.addItem("Receive" + submenu, true,
        menu_IPv4_buffers_receive);
    menu_IPv4.addItem("Buffers" + submenu, true,
menu_IPv4_buffers);
    menu_IPv4_responseTime_max = new MenuItem(notsup + "Maximum",
true,
        notSupported);
    menu_IPv4_responseTime.addItem(menu_IPv4_responseTime_max);
    menu_IPv4_responseTime_min = new MenuItem(notsup + "Minimum",
true,
        notSupported);
    menu_IPv4_responseTime.addItem(menu_IPv4_responseTime_min);
    menu_IPv4_responseTime_avg = new MenuItem(notsup + "Average",
true,
        notSupported);
    menu_IPv4_responseTime.addItem(menu_IPv4_responseTime_avg);
    menu_IPv4.addItem("Response Time" + submenu, true,
        menu_IPv4_responseTime);

/* Create the "IPv6" menu */
menu_IPv6.setAutoOpen(true);
menu_IPv6.addItem("(PLACEHOLDER)", true, notSupported);

/* Create the "Raw IP" menu */
menu_rawIP.setAutoOpen(true);
menu_rawIP.addItem("(PLACEHOLDER)", true, notSupported);

```



```

/* Create the "SCTP" menu */
menu_SCTP.setAutoOpen(true);
menu_SCTP.addItem("(PLACEHOLDER)", true, notSupported);

/* Create the "Wireless" menu */
menu_wireless.setAutoOpen(true);
wirelessOnOff = new MenuItem(off + "Enable/Disable", true,
wirelessCmd);
menu_wireless.addItem(wirelessOnOff);

/* Create the "LEGEND" menu */
menu_legend.setAutoOpen(true);
menu_legend.addItem(notsup + "Feature Not Yet Supported",
true, legendInfo);
menu_legend.addItem(on + "Feature Turned On", true,
legendInfo);
menu_legend.addItem(off + "Feature Turned Off", true,
legendInfo);

/* Create the "TEST" menu */
menu_test.setAutoOpen(true);
testItem = new MenuItem(off + "Show All Graphics", true,
testCmd);
menu_test.addItem(testItem);
//System.out.println("HTML: " + testItem.getHTML());

/* Add menu items to the top horizontal menu */
menu.addItem(new MenuItem("Router", menu_router));
menu.addItem(new MenuItem("TCP", menu_TCP));
menu.addItem(new MenuItem("UDP", menu_UDP));
menu.addItem(new MenuItem("IPv4", menu_IPv4));
menu.addItem(new MenuItem("IPv6", menu_IPv6));
menu.addItem(new MenuItem("Raw IP", menu_rawIP));
menu.addItem(new MenuItem("SCTP", menu_SCTP));
menu.addItem(new MenuItem("Wireless", menu_wireless));
menu.addItem(new MenuItem("LEGEND", menu_legend));
menu.addItem(new MenuItem("TEST", menu_test));

menu.setWidth("100%");
initWidget(menu);
}

public static String changeMenuStatus(String origHTML) {
    System.out.println("Original HTML: " + origHTML);

    String[] tokens = origHTML.split("<DGRP-SPLIT>");

```

```

        if (tokens[0].equals("&nbsp;<DGRP-ON>")) {
            System.out.println("Returning HTML: " + off +
tokens[2]);
            return (off + tokens[2]);
        }
        else if (tokens[0].equals("&nbsp;<DGRP-OFF>")) {
            System.out.println("Returning HTML: " + on + tokens[2]);
            return (on + tokens[2]);
        }
        else {
            System.out.println("Returning original HTML.");
            return (origHTML);
        }
    }
}

```

BandwidthInfo.java

```

package org.dgrp.client;

import com.google.gwt.user.client.rpc.IsSerializable;

/**
 *
 * @author chrisne
 */
public class BandwidthInfo implements IsSerializable {

    private final double LOW_BW_THRESHOLD = 0.33;
    private final double MED_BW_THRESHOLD = 0.67;
    private final double HIGH_BW_THRESHOLD = 0.9;

    private String ipAddress;
    private String bandwidthInUse;

    private int pktsSentTo, pktsRcvdFrom, dataBytesSentTo,
dataBytesRcvdFrom,
        pktsSentAndRcvd, dataBytesSentAndRcvd;

    public BandwidthInfo(String ipAddress) {
        this.ipAddress = ipAddress;
        bandwidthInUse = null;
    }
    public BandwidthInfo() { //no-argument constructor required
for GWT serialization
        this.ipAddress = "0.0.0.0";
        bandwidthInUse = null;
    }
}

```

```

    }

    public String getIPAddress() {
        return ipAddress;
    }

    public void setBandwidthInUse(int current, int max) {
        double percentage = current / max;

        if (percentage < LOW_BW_THRESHOLD) {
            bandwidthInUse = "low";
        }
        else if (percentage < MED_BW_THRESHOLD) {
            bandwidthInUse = "medium";
        }
        else if (percentage < HIGH_BW_THRESHOLD) {
            bandwidthInUse = "high";
        }
        else { //bandwidth usage nearing maximum
            bandwidthInUse = "blocked";
        }
    }

    public String getBandwidthInUse() {
        return bandwidthInUse;
    }

    public void setTotalPacketsSentTo(int totalPackets) {
        pktsSentTo = totalPackets;
    }
    public void setTotalPacketsReceivedFrom(int totalPackets) {
        pktsRcvdFrom = totalPackets;
    }
    public void setTotalDataBytesSentTo(int totalDataBytes) {
        dataBytesSentTo = totalDataBytes;
    }
    public void setTotalDataBytesReceivedFrom(int totalDataBytes)
    {
        dataBytesRcvdFrom = totalDataBytes;
    }
    public void setTotalPacketsSentToAndReceivedFrom(int
totalPackets) {
        pktsSentAndRcvd = totalPackets;
    }
    public void setTotalDataBytesSentToAndReceivedFrom(int
totalDataBytes) {
        dataBytesSentAndRcvd = totalDataBytes;
    }

    public int getTotalPacketsSentTo() {
        return pktsSentTo;
    }

```

```

    }
    public int getTotalPacketsReceivedFrom() {
        return pktsRcvdFrom;
    }
    public int getTotalDataBytesSentTo() {
        return dataBytesSentTo;
    }
    public int getTotalDataBytesReceivedFrom() {
        return dataBytesRcvdFrom;
    }
    public int getTotalPacketsSentToAndReceivedFrom() {
        return pktsSentAndRcvd;
    }
    public int getTotalDataBytesSentToAndReceivedFrom() {
        return dataBytesSentAndRcvd;
    }
}

```

BandwidthMonitor.java

```

package org.dgrp.client;
import com.google.gwt.user.client.rpc.RemoteService;

/**
 *
 * @author chrisne
 */
public interface BandwidthMonitor extends RemoteService{
    public void startServiceCountPackets(String s);
    public void startServiceCountDataBytes(String s);
    public void stopServiceCountPackets();
    public void stopServiceCountDataBytes();
    public BandwidthInfo getBandwidthInUse(String s);
    public BandwidthInfo getRandomBandwidthInUse(String s);
    public String[] getRefreshedIPs();
}

```

BandwidthMonitorAsync.java

```

package org.dgrp.client;
import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 *

```

```

    * @author chrisne
    */
public interface BandwidthMonitorAsync {
    public void startServiceCountPackets(String s, AsyncCallback
asyncCallback);
    public void startServiceCountDataBytes(String s,
AsyncCallback asyncCallback);
    public void stopServiceCountPackets(AsyncCallback
asyncCallback);
    public void stopServiceCountDataBytes(AsyncCallback
asyncCallback);
    public void getBandwidthInUse(String s, AsyncCallback
callback);
    public void getRandomBandwidthInUse(String s, AsyncCallback
callback);
    public void getRefreshedIPs(AsyncCallback callback);
}

```

DGRPEntryPoint.java

```

package org.dgrp.client;

import org.dgrp.client.SidebarItem.SidebarItemInfo;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.History;
import com.google.gwt.user.client.HistoryListener;
import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.HasAlignment;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.VerticalPanel;

/**
 *
 * @author Christopher Nelson
 */
public class DGRPEntryPoint implements EntryPoint,
        HistoryListener {

    /** Creates a new instance of DGRPEntryPoint */
    public DGRPEntryPoint() {
    }
    protected Sidebar list = new Sidebar();
    private SidebarItemInfo curInfo;
    private SidebarItem curItem;
    private HTML description = new HTML();
    private DockPanel panel = new DockPanel();
    private DockPanel mainPanel;

```

```

        public void onHistoryChanged(String token) {
            // Find the SidebarItemInfo associated with the history
            context. If one
            // is found, show it (It may not be found, for example,
            when the user
            // mis-types a URL, or on startup, when the first context
            will be "").
            SidebarItemInfo info = list.find(token);
            if (info == null) {
                showInfo();
                return;
            }
            show(info, false);
        }

        public void onModuleLoad() {
            // Load all the sidebar items.
            loadSidebarItems();

            // Put the sidebar on the left, and add the outer dock
            panel to the
            // root.
            mainPanel = new DockPanel();
            mainPanel.setStyleName("dgrp-MainPanel");

            VerticalPanel vp = new VerticalPanel();
            vp.setWidth("100%");
            vp.add(description);
            vp.add(mainPanel);

            description.setStyleName("dgrp-Heading");

            panel.add(list, DockPanel.WEST);
            panel.add(vp, DockPanel.CENTER);

            panel.setCellVerticalAlignment(list,
            HasAlignment.ALIGN_TOP);
            panel.setCellWidth(vp, "100%");
            panel.setCellHeight(vp, "100%");

            History.addHistoryListener(this);
            RootPanel.get().add(panel);

            // Show the initial screen.
            String initToken = History.getToken();
            if (initToken.length() > 0) {
                onHistoryChanged(initToken);
            } else {
                showInfo();
            }
        }

```

```

    }

    public void show(SidebarItemInfo info, boolean affectHistory)
    {
        // Don't bother re-displaying the existing item. This can
        // be an issue
        // in practice, because when the history context is set,
        // our
        // onHistoryChanged() handler will attempt to show the
        // currently-visible
        // item.
        if (info == curInfo) {
            return;
        }
        curInfo = info;

        // Remove the old item from the display area.
        if (curItem != null) {
            curItem.onHide();
            mainPanel.remove(curItem);
        }

        // Get the new item instance, and display its description
        // in the
        // item list.
        curItem = info.getInstance();
        list.setItemSelection(info.getName());
        description.setHTML(info.getDescription());

        // If affectHistory is set, create a new item on the
        // history stack. This
        // will ultimately result in onHistoryChanged() being
        // called. It will
        // call show() again, but nothing will happen because it
        // will request
        // the exact same item we're already showing.
        if (affectHistory) {
            History.newItem(info.getName());
        }

        // Display the new item.
        mainPanel.add(curItem, DockPanel.CENTER);
        mainPanel.setCellWidth(curItem, "100%");
        mainPanel.setCellHeight(curItem, "100%");
        mainPanel.setCellVerticalAlignment(curItem,
        DockPanel.ALIGN_TOP);
        curItem.onShow();
    }

    /**
     * Adds all items to the list. Note that this does not create

```

```

actual instances
    * of all items yet (they are created on-demand). This can
make a significant
    * difference in startup time.
    */
protected void loadSidebarItems() {
    list.addItem(Welcome.init());
    list.addItem(Analysis.init());
    list.addItem(Settings.init());
    list.addItem(Version.init());
    list.addItem(About.init());
}

private void showInfo() {
    show(list.find("Welcome"), false);
}
}

```

GetVersionInfo.java

```

package org.dgrp.client;
import com.google.gwt.user.client.rpc.RemoteService;

/**
 *
 * @author Christopher Nelson
 */
public interface GetVersionInfo extends RemoteService{
    public VersionContents getVersionInfo();
}

```

GetVersionInfoAsync.java

```

package org.dgrp.client;
import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 *
 * @author Christopher Nelson
 */
public interface GetVersionInfoAsync {
    public void getVersionInfo(AsyncCallback callback);
}

```


ImagePanel.java

```
package org.dgrp.client;

import com.google.gwt.user.client.ui.DockPanel;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.Image;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.Grid;

/**
 *
 * @author chrisne
 */
public class ImagePanel extends Composite {

    private final String NUM_PACKETS_LABEL = "# Packets";
    public final int NUM_PACKETS = 0;
    private final String NUM_DATA_BYTES_LABEL = "# Data Bytes";
    public final int NUM_DATA_BYTES = 1;

    private boolean isIPv4PacketsSentAndRcvdVisible = false;
    private boolean isIPv4BytesSentAndRcvdVisible = false;

    DockPanel dock, routerStatsDock, wirelessDock, laptop0Dock,
    laptop1Dock,
        laptop2Dock, laptop3Dock;
    HorizontalPanel ispPanel, routerPanel, pipePanel, laptopPanel;
    Image laptop0, laptop1, laptop2, laptop3, router, wireless,
    routerStats,
        laptop0Pipe, laptop1Pipe, laptop2Pipe, laptop3Pipe;
    Label laptop0IPAddrLabel, laptop1IPAddrLabel,
    laptop2IPAddrLabel,
        laptop3IPAddrLabel;
    Grid laptop0grid, laptop1grid, laptop2grid, laptop3grid;

    public ImagePanel() {

        laptop0 = new Image("images/placeholder.png");
        laptop0.setWidth("131px");
        laptop0.setHeight("104px");
        laptop0.setStyleName("dgrp-Images-Image");

        laptop1 = new Image("images/placeholder.png");
        laptop1.setWidth("131px");
        laptop1.setHeight("104px");
        laptop1.setStyleName("dgrp-Images-Image");
```

```

laptop2 = new Image("images/placeholder.png");
laptop2.setWidth("131px");
laptop2.setHeight("104px");
laptop2.setStyleName("dgrp-Images-Image");

laptop3 = new Image("images/placeholder.png");
laptop3.setWidth("131px");
laptop3.setHeight("104px");
laptop3.setStyleName("dgrp-Images-Image");

router = new Image("images/router.png");
router.setWidth("188px");
router.setHeight("166px");
router.setStyleName("dgrp-Images-Image");

wireless = new Image("images/placeholder.png");
wireless.setWidth("384px");
wireless.setHeight("231px");
wireless.setStyleName("dgrp-Images-Wireless");

routerStats = new Image("images/placeholder.png");
routerStats.setWidth("255px");
routerStats.setHeight("275px");
routerStats.setStyleName("dgrp-Images-RouterStats");

laptop0Pipe = new Image("images/placeholder.png");
laptop0Pipe.setWidth("371px");
laptop0Pipe.setHeight("54px");
laptop0Pipe.setStyleName("dgrp-Images-Laptop0Pipe");
laptop1Pipe = new Image("images/placeholder.png");
laptop1Pipe.setWidth("102px");
laptop1Pipe.setHeight("54px");
laptop1Pipe.setStyleName("dgrp-Images-Laptop1Pipe");
laptop2Pipe = new Image("images/placeholder.png");
laptop2Pipe.setStyleName("dgrp-Images-Laptop2Pipe");
laptop2Pipe.setWidth("102px");
laptop2Pipe.setHeight("54px");
laptop3Pipe = new Image("images/placeholder.png");
laptop3Pipe.setWidth("371px");
laptop3Pipe.setHeight("54px");
laptop3Pipe.setStyleName("dgrp-Images-Laptop3Pipe");

dock = new DockPanel();
wirelessDock = new DockPanel();
routerStatsDock = new DockPanel();
pipePanel = new HorizontalPanel();
laptop0Dock = new DockPanel();
laptop0IPAddrLabel = new Label();
laptop0IPAddrLabel.setHorizontalAlignment(Label.ALIGN_CEN
TER);

```

```

laptop0grid = new Grid(2, 2);
laptop0grid.setVisible(false);
//laptop0grid.setBorderWidth(1);
laptop0grid.setCellPadding(2);
laptop0grid.setStyleName("dgrp-Stat-Table");
//laptop0grid.setHTML(0, 0, "# Packets");
//laptop0grid.setHTML(1, 0, "# Data Bytes");
//laptop0grid.setHTML(0, 1, "0");
//laptop0grid.setHTML(1, 1, "0");
laptop1Dock = new DockPanel();
laptop1IPAddrLabel = new Label();
laptop1IPAddrLabel.setHorizontalAlignment(Label.ALIGN_CEN
TER);
laptop1grid = new Grid(2, 2);
laptop1grid.setVisible(false);
//laptop1grid.setBorderWidth(1);
laptop1grid.setCellPadding(2);
laptop1grid.setStyleName("dgrp-Stat-Table");
//laptop1grid.setHTML(0, 0, "# Packets");
//laptop1grid.setHTML(1, 0, "# Data Bytes");
//laptop1grid.setHTML(0, 1, "0");
//laptop1grid.setHTML(1, 1, "0");
laptop2Dock = new DockPanel();
laptop2IPAddrLabel = new Label();
laptop2IPAddrLabel.setHorizontalAlignment(Label.ALIGN_CEN
TER);
laptop2grid = new Grid(2, 2);
laptop2grid.setVisible(false);
//laptop2grid.setBorderWidth(1);
laptop2grid.setCellPadding(2);
laptop2grid.setStyleName("dgrp-Stat-Table");
//laptop2grid.setHTML(0, 0, "# Packets");
//laptop2grid.setHTML(1, 0, "# Data Bytes");
//laptop2grid.setHTML(0, 1, "0");
//laptop2grid.setHTML(1, 1, "0");
laptop3Dock = new DockPanel();
laptop3IPAddrLabel = new Label();
laptop3IPAddrLabel.setHorizontalAlignment(Label.ALIGN_CEN
TER);
laptop3grid = new Grid(2, 2);
laptop3grid.setVisible(false);
//laptop3grid.setBorderWidth(1);
laptop3grid.setCellPadding(2);
laptop3grid.setStyleName("dgrp-Stat-Table");
//laptop3grid.setHTML(0, 0, "# Packets");
//laptop3grid.setHTML(1, 0, "# Data Bytes");
//laptop3grid.setHTML(0, 1, "0");
//laptop3grid.setHTML(1, 1, "0");
ispPanel = new HorizontalPanel();
routerPanel = new HorizontalPanel();
laptopPanel = new HorizontalPanel();

```

```

laptopPanel.setHeight("150px");

ispPanel.setHeight("100px");
wirelessDock.setWidth("500px");
routerStatsDock.setWidth("500px");
laptop0Dock.setWidth("300px");
laptop0Dock.setHeight("100%");
laptop1Dock.setWidth("300px");
laptop1Dock.setHeight("100%");
laptop2Dock.setWidth("300px");
laptop2Dock.setHeight("100%");
laptop3Dock.setWidth("300px");
laptop3Dock.setHeight("100%");

dock.setWidth("100%");
dock.setHeight("100%");
dock.setHorizontalAlignment(DockPanel.ALIGN_CENTER);

routerStatsDock.add(routerStats, DockPanel.CENTER);
wirelessDock.add(wireless, DockPanel.CENTER);
routerPanel.add(routerStatsDock);
routerPanel.add(router);
routerPanel.setCellVerticalAlignment(router,
HorizontalPanel.ALIGN_BOTTOM);
routerPanel.add(wirelessDock);

pipePanel.add(laptop0Pipe);
pipePanel.setCellWidth(laptop0Pipe, "475px");
pipePanel.setCellHorizontalAlignment(laptop0Pipe,
HorizontalPanel.ALIGN_CENTER);
pipePanel.add(laptop1Pipe);
pipePanel.setCellWidth(laptop1Pipe, "125px");
pipePanel.setCellHorizontalAlignment(laptop1Pipe,
HorizontalPanel.ALIGN_CENTER);
pipePanel.add(laptop2Pipe);
pipePanel.setCellWidth(laptop2Pipe, "125px");
pipePanel.setCellHorizontalAlignment(laptop2Pipe,
HorizontalPanel.ALIGN_CENTER);
pipePanel.add(laptop3Pipe);
pipePanel.setCellWidth(laptop3Pipe, "475px");
pipePanel.setCellHorizontalAlignment(laptop3Pipe,
HorizontalPanel.ALIGN_CENTER);

laptop0Dock.add(laptop0IPAddrLabel, DockPanel.SOUTH);
laptop0Dock.add(laptop0grid, DockPanel.WEST);
laptop0Dock.add(laptop0, DockPanel.CENTER);
laptop0Dock.setCellHorizontalAlignment(laptop0,
DockPanel.ALIGN_CENTER);
laptop1Dock.add(laptop1IPAddrLabel, DockPanel.SOUTH);
laptop1Dock.add(laptop1grid, DockPanel.WEST);
laptop1Dock.add(laptop1, DockPanel.CENTER);

```

```

        laptop1Dock.setCellHorizontalAlignment(laptop1,
DockPanel.ALIGN_CENTER);
        laptop2Dock.add(laptop2IPAddrLabel, DockPanel.SOUTH);
        laptop2Dock.add(laptop2grid, DockPanel.WEST);
        laptop2Dock.add(laptop2, DockPanel.CENTER);
        laptop2Dock.setCellHorizontalAlignment(laptop2,
DockPanel.ALIGN_CENTER);
        laptop3Dock.add(laptop3IPAddrLabel, DockPanel.SOUTH);
        laptop3Dock.add(laptop3grid, DockPanel.WEST);
        laptop3Dock.add(laptop3, DockPanel.CENTER);
        laptop3Dock.setCellHorizontalAlignment(laptop3,
DockPanel.ALIGN_CENTER);

        laptopPanel.add(laptop0Dock);
        laptopPanel.add(laptop1Dock);
        laptopPanel.add(laptop2Dock);
        laptopPanel.add(laptop3Dock);

        dock.add(ispPanel, DockPanel.NORTH);
        dock.add(routerPanel, DockPanel.NORTH);
        dock.add(pipePanel, DockPanel.NORTH);
        dock.add(laptopPanel, DockPanel.NORTH);
        dock.setCellWidth(ispPanel, "100%");
        dock.setCellWidth(routerPanel, "100%");
        dock.setCellWidth(pipePanel, "100%");
        dock.setCellWidth(laptopPanel, "100%");

        initWidget(dock);
    }

    public void hideWireless() {
        wireless.setUrl("images/placeholder.png");
        wireless.setWidth("384px");
        wireless.setHeight("231px");
        wireless.setStyleName("dgrp-Images-Wireless");
    }

    public void showWireless() {
        wireless.setUrl("images/wireless_cloud.png");
        wireless.setWidth("384px");
        wireless.setHeight("231px");
        wireless.setStyleName("dgrp-Images-Wireless");
    }

    public boolean isWirelessVisible() {
        if (wireless.getUrl().endsWith("placeholder.png")) {
            return false;
        }
        else {
            return true;
        }
    }

```

```

    }

    public void hideRouterStats() {
        routerStats.setUrl("images/placeholder.png");
        routerStats.setWidth("255px");
        routerStats.setHeight("275px");
        routerStats.setStyleName("dgrp-Images-RouterStats");
    }

    public void showRouterStats() {
        routerStats.setUrl("images/placeholder_for_router_stats.p
ng");
        routerStats.setWidth("255px");
        routerStats.setHeight("275px");
        routerStats.setStyleName("dgrp-Images-RouterStats");
    }

    public boolean isRouterStatsVisible() {
        if (routerStats.getUrl().endsWith("placeholder.png")) {
            return false;
        }
        else {
            return true;
        }
    }

    public void hideIPv4PacketsSentAndRcvd() {
        isIPv4PacketsSentAndRcvdVisible = false;
    }
    public void showIPv4PacketsSentAndRcvd() {
        isIPv4PacketsSentAndRcvdVisible = true;
    }
    public boolean isIPv4PacketsSentAndRcvdVisible() {
        return isIPv4PacketsSentAndRcvdVisible;
    }

    public void hideIPv4BytesSentAndRcvd() {
        isIPv4BytesSentAndRcvdVisible = false;
    }
    public void showIPv4BytesSentAndRcvd() {
        isIPv4BytesSentAndRcvdVisible = true;
    }
    public boolean isIPv4BytesSentAndRcvdVisible() {
        return isIPv4BytesSentAndRcvdVisible;
    }

    public void hideLaptop(int position) {
        switch (position) {
            case 0:
                laptop0.setUrl("images/placeholder.png");
                laptop0.setWidth("131px");

```

```

        laptop0.setHeight("104px");
        laptop0.setStyleName("dgrp-Images-Image");
        laptop0grid.setVisible(false);
        break;
    case 1:
        laptop1.setUrl("images/placeholder.png");
        laptop1.setWidth("131px");
        laptop1.setHeight("104px");
        laptop1.setStyleName("dgrp-Images-Image");
        laptop1grid.setVisible(false);
        break;
    case 2:
        laptop2.setUrl("images/placeholder.png");
        laptop2.setWidth("131px");
        laptop2.setHeight("104px");
        laptop2.setStyleName("dgrp-Images-Image");
        laptop2grid.setVisible(false);
        break;
    case 3:
        laptop3.setUrl("images/placeholder.png");
        laptop3.setWidth("131px");
        laptop3.setHeight("104px");
        laptop3.setStyleName("dgrp-Images-Image");
        laptop3grid.setVisible(false);
        break;
    default:
        break; //ignore others for now
}
}

public void showLaptop(int position) {
    switch (position) {
        case 0:
            laptop0.setUrl("images/laptop.png");
            laptop0.setWidth("131px");
            laptop0.setHeight("104px");
            laptop0.setStyleName("dgrp-Images-Image");
            laptop0grid.setVisible(true);
            laptop0grid.setHTML(0, 1, "");
            laptop0grid.setHTML(1, 1, "");
            break;
        case 1:
            laptop1.setUrl("images/laptop.png");
            laptop1.setWidth("131px");
            laptop1.setHeight("104px");
            laptop1.setStyleName("dgrp-Images-Image");
            laptop1grid.setVisible(true);
            laptop1grid.setHTML(0, 1, "");
            laptop1grid.setHTML(1, 1, "");
            break;
        case 2:

```

```

        laptop2.setUrl("images/laptop.png");
        laptop2.setWidth("131px");
        laptop2.setHeight("104px");
        laptop2.setStyleName("dgrp-Images-Image");
        laptop2grid.setVisible(true);
        laptop2grid.setHTML(0, 1, "");
        laptop2grid.setHTML(1, 1, "");
        break;
    case 3:
        laptop3.setUrl("images/laptop.png");
        laptop3.setWidth("131px");
        laptop3.setHeight("104px");
        laptop3.setStyleName("dgrp-Images-Image");
        laptop3grid.setVisible(true);
        laptop3grid.setHTML(0, 1, "");
        laptop3grid.setHTML(1, 1, "");
        break;
    default:
        break; //ignore others for now
}
}

public boolean isLaptopVisible(int position) {
    switch (position) {
        case 0:
            if (laptop0.getUrl().endsWith("placeholder.png"))
            {
                return false;
            }
            break;
        case 1:
            if (laptop1.getUrl().endsWith("placeholder.png"))
            {
                return false;
            }
            break;
        case 2:
            if (laptop2.getUrl().endsWith("placeholder.png"))
            {
                return false;
            }
            break;
        case 3:
            if (laptop3.getUrl().endsWith("placeholder.png"))
            {
                return false;
            }
            break;
        default:
            break; //ignore others for now
    }
}

```



```

        return true;
    }

    public void showLaptopPipe(int position, String bwUsage) {
        switch (position) {
            case 0:
                if (bwUsage.equals("low")) {
                    laptop0Pipe.setUrl("images/laptop0_pipe_small
.png");
                }
                else if (bwUsage.equals("medium")) {
                    laptop0Pipe.setUrl("images/laptop0_pipe_mediu
m.png");
                }
                else if (bwUsage.equals("high")) {
                    laptop0Pipe.setUrl("images/laptop0_pipe_large
.png");
                }
                else if (bwUsage.equals("blocked")) {
                    laptop0Pipe.setUrl("images/laptop0_pipe_red.p
ng");
                }
                laptop0Pipe.setWidth("371px");
                laptop0Pipe.setHeight("54px");
                laptop0Pipe.setStyleName("dgrp-Images-
Laptop0Pipe");
                break;
            case 1:
                if (bwUsage.equals("low")) {
                    laptop1Pipe.setUrl("images/laptop1_pipe_small
.png");
                }
                else if (bwUsage.equals("medium")) {
                    laptop1Pipe.setUrl("images/laptop1_pipe_mediu
m.png");
                }
                else if (bwUsage.equals("high")) {
                    laptop1Pipe.setUrl("images/laptop1_pipe_large
.png");
                }
                else if (bwUsage.equals("blocked")) {
                    laptop1Pipe.setUrl("images/laptop1_pipe_red.p
ng");
                }
                laptop1Pipe.setWidth("102px");
                laptop1Pipe.setHeight("54px");
                laptop1Pipe.setStyleName("dgrp-Images-
Laptop1Pipe");
                break;
            case 2:

```

```

        if (bwUsage.equals("low")) {
            laptop2Pipe.setUrl("images/laptop2_pipe_small
.png");
        }
        else if (bwUsage.equals("medium")) {
            laptop2Pipe.setUrl("images/laptop2_pipe_mediu
m.png");
        }
        else if (bwUsage.equals("high")) {
            laptop2Pipe.setUrl("images/laptop2_pipe_large
.png");
        }
        else if (bwUsage.equals("blocked")) {
            laptop2Pipe.setUrl("images/laptop2_pipe_red.p
ng");
        }
        laptop2Pipe.setWidth("102px");
        laptop2Pipe.setHeight("54px");
        laptop2Pipe.setStyleName("dgrp-Images-
Laptop2Pipe");
        break;
    case 3:
        if (bwUsage.equals("low")) {
            laptop3Pipe.setUrl("images/laptop3_pipe_small
.png");
        }
        else if (bwUsage.equals("medium")) {
            laptop3Pipe.setUrl("images/laptop3_pipe_mediu
m.png");
        }
        else if (bwUsage.equals("high")) {
            laptop3Pipe.setUrl("images/laptop3_pipe_large
.png");
        }
        else if (bwUsage.equals("blocked")) {
            laptop3Pipe.setUrl("images/laptop3_pipe_red.p
ng");
        }
        laptop3Pipe.setWidth("371px");
        laptop3Pipe.setHeight("54px");
        laptop3Pipe.setStyleName("dgrp-Images-
Laptop3Pipe");
        break;
    default:
        break;
    }
}

public void hideLaptopPipe(int position) {
    switch (position) {
        case 0:

```

```

        laptop0.setUrl("images/placeholder.png");
        laptop0.setWidth("131px");
        laptop0.setHeight("104px");
        laptop0.setStyleName("dgrp-Images-Image");
        break;
    case 1:
        laptop1.setUrl("images/placeholder.png");
        laptop1.setWidth("131px");
        laptop1.setHeight("104px");
        laptop1.setStyleName("dgrp-Images-Image");
        break;
    case 2:
        laptop2.setUrl("images/placeholder.png");
        laptop2.setWidth("131px");
        laptop2.setHeight("104px");
        laptop2.setStyleName("dgrp-Images-Image");
        break;
    case 3:
        laptop3.setUrl("images/placeholder.png");
        laptop3.setWidth("131px");
        laptop3.setHeight("104px");
        laptop3.setStyleName("dgrp-Images-Image");
        break;
    default:
        break; //ignore for now
    }
}

public void setLaptopIPAddrLabel(int position, String text) {
    switch (position) {
        case 0:
            laptop0IPAddrLabel.setText(text);
            break;
        case 1:
            laptop1IPAddrLabel.setText(text);
            break;
        case 2:
            laptop2IPAddrLabel.setText(text);
            break;
        case 3:
            laptop3IPAddrLabel.setText(text);
            break;
        default:
            break;
    }
}

public void setLaptopStatValue(int position, int stat, String
value) {
    String label = "", valueUsed = "";

```

```

switch (stat) {
  case NUM_PACKETS:
    if (isIPv4PacketsSentAndRcvdVisible) {
      label = NUM_PACKETS_LABEL;
      valueUsed = value;
    }
    break;

  case NUM_DATA_BYTES:
    if (isIPv4BytesSentAndRcvdVisible) {
      label = NUM_DATA_BYTES_LABEL;
      valueUsed = value;
    }
    break;

  default: //ignore; invalid statistic
    break;
}

switch (position) {
  case 0:
    if (stat == NUM_PACKETS) {
      laptop0grid.setHTML(0, 0, label);
      laptop0grid.setHTML(0, 1, valueUsed);
    } else if (stat == NUM_DATA_BYTES) {
      laptop0grid.setHTML(1, 0, label);
      laptop0grid.setHTML(1, 1, valueUsed);
    } else {
      //ignore for now; invalid statistic
    }
    break;
  case 1:
    if (stat == NUM_PACKETS) {
      laptop1grid.setHTML(0, 0, label);
      laptop1grid.setHTML(0, 1, valueUsed);
    } else if (stat == NUM_DATA_BYTES) {
      laptop1grid.setHTML(1, 0, label);
      laptop1grid.setHTML(1, 1, valueUsed);
    } else {
      //ignore for now; invalid statistic
    }
    break;
  case 2:
    if (stat == NUM_PACKETS) {
      laptop2grid.setHTML(0, 0, label);
      laptop2grid.setHTML(0, 1, valueUsed);
    } else if (stat == NUM_DATA_BYTES) {
      laptop2grid.setHTML(1, 0, label);
      laptop2grid.setHTML(1, 1, valueUsed);
    } else {

```

```

        //ignore for now; invalid statistic
    }
    break;
case 3:
    if (stat == NUM_PACKETS) {
        laptop3grid.setHTML(0, 0, label);
        laptop3grid.setHTML(0, 1, valueUsed);
    } else if (stat == NUM_DATA_BYTES) {
        laptop3grid.setHTML(1, 0, label);
        laptop3grid.setHTML(1, 1, valueUsed);
    } else {
        //ignore for now; invalid statistic
    }
    break;

    default: //ignore for now; invalid position
        break;
}
}

void showAllGraphics() {
    showWireless();
    showLaptop(0);
    showLaptop(1);
    showLaptop(2);
    showLaptop(3);
    showLaptopPipe(0, "small");
    showLaptopPipe(1, "small");
    showLaptopPipe(2, "small");
    showLaptopPipe(3, "small");
    setLaptopIPAddrLabel(0, "IP Address Placeholder");
    setLaptopIPAddrLabel(1, "IP Address Placeholder");
    setLaptopIPAddrLabel(2, "IP Address Placeholder");
    setLaptopIPAddrLabel(3, "IP Address Placeholder");
}
}

```

Settings.java

```

package org.dgrp.client;

import com.google.gwt.user.client.ui.HTML;

/**
 * Settings page.
 */
public class Settings extends SidebarItem {

```

```

public static SidebarItemInfo init() {
    return new SidebarItemInfo("Settings", "System Settings...") {
        public SidebarItem createInstance() {
            return new Settings();
        }
    };
}

public Settings() {
    initWidget(new HTML(
        "<div class='dgrp-About-Prose'" +
        "This is a placeholder for the future implementation of
system settings." +
        "<br><br>Examples of settings that may be included:" +
        "<ul>" +
        "<li>Interface IP Addresses</li>" +
        "<li>Interface Netmasks</li>" +
        "<li>Interfaces Enabled or Disabled</li>" +
        "<li>Default Gateways (Routers)</li>" +
        "<li>DNS settings</li>" +
        "<li>Quagga routing protocol</li>" +
        "<li>Quagga packet filtering</li>" +
        "<li>Tunable DGRP Settings</li>" +
        "</ul>" +
        "</div>",
        true));
}

public void onShow() {
}
}

```

Sidebar.java

```

package org.dgrp.client;

import org.dgrp.client.SidebarItem.SidebarItemInfo;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.Hyperlink;
import com.google.gwt.user.client.ui.VerticalPanel;

import java.util.ArrayList;

/**
 * The left panel that contains all of the sidebar items along
 * with a short
 * description of each.
 */

```

```

public class Sidebar extends Composite {

    private VerticalPanel list = new VerticalPanel();
    private ArrayList items = new ArrayList();
    private int selectedItem = -1;

    public Sidebar() {
        initWidget(list);
        setStyleName("dgrp-Sidebar-List");
    }

    public void addItem(final SidebarItemInfo info) {
        String name = info.getName();
        Hyperlink link = new Hyperlink(name, name);
        link.setStyleName("dgrp-Sidebar-Item");

        list.add(link);
        items.add(info);
    }

    public SidebarItemInfo find(String sidebarItemName) {
        for (int i = 0; i < items.size(); ++i) {
            SidebarItemInfo info = (SidebarItemInfo) items.get(i);
            if (info.getName().equals(sidebarItemName)) {
                return info;
            }
        }

        return null;
    }

    public void setItemSelection(String name) {
        if (selectedItem != -1) {
            list.getWidget(selectedItem).removeStyleName("dgrp-Sidebar-Item-Selected");
        }

        for (int i = 0; i < items.size(); ++i) {
            SidebarItemInfo info = (SidebarItemInfo) items.get(i);
            if (info.getName().equals(name)) {
                selectedItem = i;
                list.getWidget(selectedItem).addStyleName("dgrp-Sidebar-Item-Selected");
            }
        }
    }
}

```

SidebarItem.java

```
package org.dgrp.client;

import com.google.gwt.user.client.ui.Composite;

/**
 * A 'SidebarItem' is a single panel of the application. They are
 * meant to be
 * lazily instantiated so that the application doesn't pay for
 * all of them
 * on startup.
 */
public abstract class SidebarItem extends Composite {

    /**
     * Encapsulated information about an item. Each item is
     * expected to have
     * a static init() method that will be called at startup.
     */
    public abstract static class SidebarItemInfo {
        private SidebarItem instance;
        private String name, description;

        public SidebarItemInfo(String name, String desc) {
            this.name = name;
            description = desc;
        }

        public abstract SidebarItem createInstance();

        public String getDescription() {
            return description;
        }

        public final SidebarItem getInstance() {
            if (instance != null) {
                return instance;
            }
            return (instance = createInstance());
        }

        public String getName() {
            return name;
        }
    }

    /**
     * Called just before this item is hidden.
     */
}
```



```

    */
    public void onHide() {
    }

    /**
     * Called just after this item is shown.
     */
    public void onShow() {
    }
}

```

TopologyInfo.java

```

package org.dgrp.client;

/**
 *
 * @author chrisne
 */
public class TopologyInfo {

    private int maxNodes;
    String[] nodeAddresses;

    public TopologyInfo(int maxNodes) {
        this.maxNodes = maxNodes;
        nodeAddresses = new String[maxNodes];
    }

    public int getMaxNodes() {
        return maxNodes;
    }

    public String getAddress(int position) {
        return nodeAddresses[position];
    }

    public void setAddress(int position, String address) throws
Exception {
        try { //except exception if address is new to the list
            int tempPosition = findPosition(address);
            if (position == tempPosition) { //this is OK
                nodeAddresses[position] = address;
            }
            else {
                throw new Exception("ERROR: Cannot add the same
address again.");
            }
        }
    }
}

```

```

        catch (Exception e) {
            if (e.getMessage().equals("ERROR: Address not
found.)) { //expected
                nodeAddresses[position] = address;
            }
            else { //do not catch exceptions we did not expect
                throw e;
            }
        }
    }
    public int findPosition(String address) throws Exception {
        for (int i=0; i<nodeAddresses.length; i++) {
            if (nodeAddresses[i].equals(address)) {
                return i;
            }
        }
        throw new Exception("ERROR: Address not found.");
    }
}
}

```

Version.java

```

package org.dgrp.client;

import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.rpc.ServiceDefTarget;

/**
 * Version page.
 */
public class Version extends SidebarItem {

    private HTML verInfo = new HTML(
        "<div class='dgrp-About-Prose'>" +
        "Retrieving version information from the server..." +
        "</div>",
        true);

    public static SidebarItemInfo init() {
        return new SidebarItemInfo("Version Info",
            "Version Information for the DTrace Graphical Router
Project...") {
            public SidebarItem createInstance() {

```

```

        return new Version();
    }
};
}

public Version() {

    initWidget(verInfo);

    // Create an asynchronous callback to handle the result.
    final AsyncCallback callback = new AsyncCallback() {
        public void onSuccess(Object result) {

            VersionContents verResults = (VersionContents) result;

            verInfo.setHTML(
                "<div class='dgrp-About-Prose'>" +
                "<table>" +

                //This software info
                "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
                "color=\"white\">DTrace Graphical Router
Project:" +
                "</font></th></tr>" +
                "<tr><td width=\"10\"></td><td>Version:</td>"
+
                "<td width=\"10\"></td>" +
                "<td>" + verResults.dgrpVersion +
                "</td></tr>" +
                "<tr><td width=\"10\"></td><td>Build
Number:</td>" +
                "<td width=\"10\"></td>" +
                "<td>" + verResults.dgrpBuildNumber +
                "</td></tr>" +
                "<tr><td width=\"10\"></td><td>Build
Date:</td>" +
                "<td width=\"10\"></td>" +
                verResults.dgrpBuildDate + "</td></tr>" +
                "<tr><td width=\"10\"></td><td>Author:</td>" +
                "<td width=\"10\"></td>" +
                "<td>" + verResults.dgrpAuthor + "</td></tr>"
+
                "<tr height=\"10\"><td
colspan=\"3\"></td></tr>" +

                //Solaris Info
                "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
                "color=\"white\">OpenSolaris:</font></th></tr>"

```

```

>" +
        "<tr><td width=\"10\"></td><td>Version:</td>"
+
        "<td width=\"10\"></td>" +
        "<td>" + verResults.solarisRelease + "</td></
tr>" +
        "<tr><td
width=\"10\"></td><td>Architecture:</td>" +
        "<td width=\"10\"></td>" +
        "<td>" + verResults.solarisArch +
"</td></tr>" +
        "<tr><td width=\"10\"></td><td>Install
Date:</td>" +
        "<td width=\"10\"></td>" +
        "<td>" + verResults.solarisInstallDate +
"</td></tr>" +
        "<tr><td width=\"10\"></td><td>Current
Uptime:</td>" +
        "<td width=\"10\"></td>" +
        "<td>" + verResults.solarisUptime +
"</td></tr>" +
        "<tr height=\"10\"><td
colspan=\"3\"></td></tr>" +
        //Quagga info
        "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
        "color=\"white\">Quagga:</font></th></tr>" +
        "<tr><td width=\"10\"></td><td>Version:</td>"
+
        "<td width=\"10\"></td>" +
        "<td>" + verResults.quaggaVersion +
"</td></tr>" +
        "<tr><td width=\"10\"></td><td>Install
Date:</td>" +
        "<td width=\"10\"></td>" +
        "<td>" + verResults.quaggaInsDate +
"</td></tr>" +
        "<tr height=\"10\"><td
colspan=\"3\"></td></tr>" +
        //Java info
        "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
        "color=\"white\">Java:</font></th></tr>" +
        "<tr><td width=\"10\"></td><td>Version:</td>"
+
        "<td width=\"10\"></td>" +
        "<td>" + verResults.javaVersion +

```

```

"</td></tr>" +
    "<tr><td width=\"10\"></td><td>Vendor:</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.javaVendor + "</td></tr>"
+
    "<tr><td width=\"10\"></td><td>Virtual
Machine (VM):</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.javaVMName + "</td></tr>"
+
    "<tr><td width=\"10\"></td><td>VM
Version:</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.javaVMVersion +
"</td></tr>" +
    "<tr><td width=\"10\"></td><td>VM
Vendor:</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.javaVMVendor +
"</td></tr>" +

    "<tr height=\"10\"><td
colspan=\"3\"></td></tr>" +

    //Web-Server info
    "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
    "color=\"white\">Apache/Tomcat Web
Server:</font></th></tr>" +
    "<tr><td width=\"10\"></td><td>Apache:</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.apacheVersion +
"</td></tr>" +
    "<tr><td width=\"10\"></td><td>Tomcat:</td>" +
    "<td width=\"10\"></td>" +
    "<td>" + verResults.tomcatVersion +
"</td></tr>" +

    "<tr height=\"10\"><td
colspan=\"3\"></td></tr>" +

    //Browser info
    "<tr><th colspan=\"4\"
bgcolor=\"black\"><font " +
    "color=\"white\">Browser:</font></th></tr>" +
    "<tr><td width=\"10\"></td><td>Version:</td>"
+
    "<td width=\"10\"></td>" +
    "<td>" + getBrowserInfo() + "</td></tr>" +

    "</table>" +

```

```

        "</div>"
    );
}

public void onFailure(Throwable caught) {
    verInfo.setHTML(
        "<div class='dgrp-About-Prose'>" +
        "Failed to retrieve version information from
the server." +
        "</div>"
    );
}

};

// Make remote call. Control flow will continue immediately
and later
// 'callback' will be invoked when the RPC completes.
getService().getVersionInfo(callback);
}

public static GetVersionInfoAsync getService(){
    // Create the client proxy. Note that although you are
creating the
    // service interface proper, you cast the result to the
asynchronous
    // version of
    // the interface. The cast is always safe because the
generated proxy
    // implements the asynchronous interface automatically.
    GetVersionInfoAsync service = (GetVersionInfoAsync)
        GWT.create(GetVersionInfo.class);
    // Specify the URL at which our service implementation is
running.
    // Note that the target URL must reside on the same
domain and port from
    // which the host page was served.
    //
    ServiceDefTarget endpoint = (ServiceDefTarget) service;
    String moduleRelativeURL = GWT.getModuleBaseURL() +
"getversioninfo";
    endpoint.setServiceEntryPoint(moduleRelativeURL);
    return service;
}

public static native String getBrowserInfo() /*-{
    return $wnd.navigator.userAgent;
} */;

public void onShow() {
}
}

```

VersionContents.java

```
package org.dgrp.client;

import java.io.Serializable;

/**
 *
 * @author Christopher Nelson
 */
public class VersionContents implements Serializable {

    //Solaris info
    public String solarisRelease;           //first line of /etc/
release
    public String solarisInstallDate;      //from SUNWcsr
    public String solarisArch;             //os.arch
    public String solarisUptime;           //uptime

    //Quagga info
    public String quaggaVersion;           //from SUNWquaggar
    public String quaggaInsDate;           //from SUNWquaggar

    //This software info
    public String dgrpBuildNumber;         //from
appinfo.properties
    public String dgrpAuthor;              //from
appinfo.properties
    public String dgrpBuildDate;           //from
appinfo.properties
    public String dgrpVersion;             //from
appinfo.properties
    public String dgrpDescription;         //from
appinfo.properties

    //Java info
    public String javaVersion;             //java.version
    public String javaVendor;              //java.vendor
    public String javaVMName;              //java.vm.name
    public String javaVMVersion;           //java.vm.version
    public String javaVMVendor;            //java.vm.vendor

    //Browser info
    public String browserInfo;             //determined on client

    //Web-Server info
    public String tomcatVersion;           //from SUNWtcatr
```

```
    public String apacheVersion;           //from SUNWapchr
}
```

Welcome.java

```
package org.dgrp.client;

import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.HTMLPanel;

public class Welcome extends SidebarItem {

    public static SidebarItemInfo init() {
        return new SidebarItemInfo("Welcome",
            "Welcome to the DTrace Graphical Router Project...") {
            public SidebarItem createInstance() {
                return new Welcome();
            }
        };
    }

    public Welcome() {
        HTMLPanel welcomeHTML = new HTMLPanel(
            "<h3>Welcome to the DTrace Graphical Router
Project</h3>" +
            "<p>Click a link to the left to continue...</p>"
        );

        VerticalPanel welcomePanel = new VerticalPanel();
        welcomePanel.setSpacing(8);
        welcomePanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTE
R);
        welcomePanel.setWidth("100%");

        welcomePanel.add(welcomeHTML);
        welcomePanel.setCellWidth(welcomeHTML, "100%");

        initWidget(welcomePanel);
    }

    public void onShow() {
    }
}
```


APPENDIX G. SOURCE CODE – PACKAGE org.dgrp.server

The complete source code for the reference implementation of this framework is provided in this and other appendices to this document for the reader's easy reference. For the simplest viewing experience or to use the code without copying and pasting it into a new source file, the reader is encouraged to review the soft-copy files available on the CD-ROM included with this document.

BandwidthMonitorImpl.java

```
package org.dgrp.server;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.util.Random;
import org.dgrp.client.BandwidthInfo;
import org.dgrp.client.BandwidthMonitor;
import org.dgrp.server.dtraceservices.*;

/**
 *
 * @author chrisne
 */
public class BandwidthMonitorImpl extends RemoteServiceServlet
implements
    BandwidthMonitor {

    private DTraceCountDataBytesService countDataBytesService;
    private DTraceCountPacketsService countPacketsService;

    public void startServiceCountPackets(String subnet) {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.startServiceCountPackets()...\n");
        countPacketsService = new DTraceCountPacketsService();
        countPacketsService.startService(subnet);
    }

    public void stopServiceCountPackets() {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.stopServiceCountPackets()...\n");
```

```

        countPacketsService.stopService();
    }

    public void startServiceCountDataBytes(String subnet) {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.startServiceCountDataBytes()...\n");
        countDataBytesService = new DTraceCountDataBytesService();
        countDataBytesService.startService(subnet);
    }

    public void stopServiceCountDataBytes() {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.stopServiceCountDataBytes()...\n");
        countDataBytesService.stopService();
    }

    public BandwidthInfo getBandwidthInUse(String ipAddr) {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.getBandwidthInUse()...\n");
        BandwidthInfo bwInfo = new BandwidthInfo(ipAddr);
        DGRPLogger.log("Created bwInfo...done\n");
        DGRPLogger.log("Setting Total Packets Received From...");
        bwInfo.setTotalPacketsReceivedFrom(countPacketsService.
            getNumberPacketsByIP(ipAddr,
countPacketsService.DIRECTION_RCVD));
        DGRPLogger.log("done.\nSetting Total Packets Sent To...");
        bwInfo.setTotalPacketsSentTo(countPacketsService.
            getNumberPacketsByIP(ipAddr,
countPacketsService.DIRECTION_SENT));
        DGRPLogger.log("done.\nSetting Total Packets Received
From and Sent To...");
        bwInfo.setTotalPacketsSentToAndReceivedFrom(countPacketsS
ervice.
            getNumberPacketsByIP(ipAddr,
countPacketsService.DIRECTION_TOTAL));
        DGRPLogger.log("done.\nSetting Total Data Bytes Received
From...");
        bwInfo.setTotalDataBytesReceivedFrom(countDataBytesServic
e.
            getNumberDataBytesByIP(ipAddr,
countDataBytesService.DIRECTION_RCVD));
        DGRPLogger.log("done.\nSetting Total Data Bytes Sent
To...");
        bwInfo.setTotalDataBytesSentTo(countDataBytesService.
            getNumberDataBytesByIP(ipAddr,
countDataBytesService.DIRECTION_SENT));
        DGRPLogger.log("done.\nSetting Total Data Bytes Received
From and Sent To...");
        bwInfo.setTotalDataBytesSentToAndReceivedFrom(countDataBy
tesService.
            getNumberDataBytesByIP(ipAddr,

```

```

countDataBytesService.DIRECTION_TOTAL));
    DGRPLogger.log("done.\n");

    //This is a fake for now...
    bwInfo.setBandwidthInUse(1, 4);

    DGRPLogger.log("Returning from
BandwidthMonitorImpl.getBandwidthInUse()\n");
    return bwInfo;
}

    public String[] getRefreshedIPs() {
        DGRPLogger.log("Entering
BandwidthMonitorImpl.getRefreshedIPs()...\n");
        return countDataBytesService.getBusiestIPsByDataBytes();
    }

    /*****
    *****
    * The following methods exist for the purpose of
demonstration and
    * testing and are not useful for the retrieval or display of
real data.
    *****/
    /*****/

    public BandwidthInfo getRandomBandwidthInUse(String ipAddr) {
        // Return a random bandwidth for testing/demo
        int maxBandwidth = 4;
        Random r = new Random();
        int bandwidth = r.nextInt(maxBandwidth) + 1;

        BandwidthInfo bwInfo = new BandwidthInfo(ipAddr);
        bwInfo.setBandwidthInUse(bandwidth, maxBandwidth);

        return bwInfo;
    }
}

```

DGRPLogger.java

```

package org.dgrp.server;

import java.io.*;

/**
 *
 * @author chrisne

```

```

*/
public class DGRPLogger {

    private static String logfile = "/var/tmp/dgrplog.txt";

    public static void log(String string) {
        try {
            BufferedWriter out = new BufferedWriter(new
FileWriter(logfile, true));
            out.write(string);
            out.close();
        } catch (IOException e) {
        }
    }
}

```

GetVersionInfoImpl.java

```

package org.dgrp.server;

import java.io.*;
import java.util.*;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import org.dgrp.client.GetVersionInfo;
import org.dgrp.client.VersionContents;
import java.net.URL;

/**
 *
 * @author Christopher Nelson
 */
public class GetVersionInfoImpl extends RemoteServiceServlet
implements
    GetVersionInfo {

    public VersionContents getVersionInfo() {

        DGRPLogger.log("Entering getVersionInfo()...\n");

        VersionContents ver = new VersionContents();

        InputStream in = null;
        Properties props = new Properties();

        try {
            in =
getClass().getResourceAsStream("/appinfo.properties");
            props.load(in);

```

```

        //Solaris info
        ver.solarisRelease = getSolarisRelease();
        ver.solarisInstallDate = getSolarisInstallDate();
        ver.solarisArch = System.getProperty("os.arch");
        ver.solarisUptime = getSolarisUptime();

        //Quagga info
        ver.quaggaVersion = getQuaggaVersion();
        ver.quaggaInsDate = getQuaggaInstallDate();

        //This software info
        ver.dgrpAuthor = props.getProperty("program.AUTHOR");
        ver.dgrpBuildDate =
        props.getProperty("program.BUILDDATE");
        ver.dgrpBuildNumber =
        props.getProperty("program.BUILDNUM");
        ver.dgrpDescription =
        props.getProperty("program.DESCRPTION");
        ver.dgrpVersion =
        props.getProperty("program.VERSION");

        //Java info
        ver.javaVMName = System.getProperty("java.vm.name");
        ver.javaVMVendor =
        System.getProperty("java.vm.vendor");
        ver.javaVMVersion =
        System.getProperty("java.vm.version");
        ver.javaVendor = System.getProperty("java.vendor");
        ver.javaVersion = System.getProperty("java.version");

        //Browser info
        ver.browserInfo = null; //determined client-side

        //Web-Server info
        ver.tomcatVersion = getTomcatVersion();
        ver.apacheVersion = getApacheVersion();

        in.close();
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    }

    DGRPLogger.log("Returning from getVersionInfo().\n");
    return ver;
}

private String getSolarisRelease() {
    try {
        BufferedReader rel = new BufferedReader(new

```

```

FileReader(
    "/etc/release"));
    StringTokenizer st = new
StringTokenizer(rel.readLine());
    String solRel = "";
    while (st.hasMoreTokens()) {
        solRel = solRel + st.nextToken() + " ";
    }
    return solRel;
}
catch (FileNotFoundException e) {
    return "<font color=\"red\"><i>Retrieval of this
property is " +
        "only supported when running this software on
" +
        "Solaris</i></font>";
}
catch (IOException e) {
    return "<font color=\"red\"><i>Failed to retrieve " +
        "property</i></font>";
}
}

private String getSolarisInstallDate() {
    try {
        String cmd = "pkginfo -l SUNWcsr";
        Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
            p.getInputStream()));

        String curLine = stdInput.readLine();

        while (curLine != null) {
            StringTokenizer st = new StringTokenizer(curLine);
            if (st.nextToken().equals("INSTDATE:")) {
                return(st.nextToken() + " " + st.nextToken()
+ " " +
                    st.nextToken() + " " +
st.nextToken());
            }
            curLine = stdInput.readLine();
        }

        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
    catch (IOException e) {
        return "<font color=\"red\"><i>Retrieval of this
property is " +
            "only supported when running this software on

```

```

" +
        "Solaris</i></font>";
    }
    catch (NoSuchElementException e) {
        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
}

private String getSolarisUptime() {
    try {
        String cmd = "uptime";
        Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
            p.getInputStream()));

        StringTokenizer st = new
StringTokenizer(stdInput.readLine());
        st.nextToken(); //skip the first
        st.nextToken(); //...and the second
        String days = st.nextToken();
        st.nextToken(); //skip the fourth
        String hours = st.nextToken(",");

        //assumes more than one day
        return(days + " day(s) and " + hours + " hour(s)");
    }
    catch (IOException e) {
        return "<font color=\"red\"><i>Retrieval of this
property is " +
            "only supported when running this software on
" +
            "Solaris</i></font>";
    }
    catch (NoSuchElementException e) {
        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
}

private String getQuaggaVersion() {
    try {
        String cmd = "pkginfo -l SUNWquaggar";
        Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
            p.getInputStream()));

        String curLine = stdInput.readLine();

```

```

        while (curLine != null) {
            StringTokenizer st = new StringTokenizer(curLine);
            if (st.nextToken().equals("DESC:")) {
                return(st.nextToken() + " " + st.nextToken()
+ " " +
                    st.nextToken() + " " +
st.nextToken());
            }
            curLine = stdInput.readLine();
        }

        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
    catch (IOException e) {
        return "<font color=\"red\"><i>Retrieval of this
property is " +
            "only supported when running this software on
" +
            "Solaris</i></font>";
    }
    catch (NoSuchElementException e) {
        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
}

private String getQuaggaInstallDate() {
    try {
        String cmd = "pkginfo -l SUNWquaggar";
        Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
            p.getInputStream()));

        String curLine = stdInput.readLine();

        while (curLine != null) {
            StringTokenizer st = new StringTokenizer(curLine);
            if (st.nextToken().equals("INSTDATE:")) {
                return(st.nextToken() + " " + st.nextToken()
+ " " +
                    st.nextToken() + " " +
st.nextToken());
            }
            curLine = stdInput.readLine();
        }

        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
}

```



```

        catch (IOException e) {
            return "<font color=\"red\"><i>Retrieval of this
property is " +
                "only supported when running this software on
" +
                "Solaris</i></font>";
        }
        catch (NoSuchElementException e) {
            return "<font color=\"red\"><i>Failed to retrieve " +
                "property</i></font>";
        }
    }

private String getTomcatVersion() {
    try {
        String cmd = "pkginfo -l SUNWtcatr";
        Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
                p.getInputStream()));

        String curLine = stdInput.readLine();

        while (curLine != null) {
            StringTokenizer st = new StringTokenizer(curLine);
            if (st.nextToken().equals("DESC:")) {
                return(st.nextToken() + " " + st.nextToken()
+ " " +
                    st.nextToken() + " " +
st.nextToken());
            }
            curLine = stdInput.readLine();
        }

        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
    catch (IOException e) {
        return "<font color=\"red\"><i>Retrieval of this
property is " +
            "only supported when running this software on
" +
            "Solaris</i></font>";
    }
    catch (NoSuchElementException e) {
        return "<font color=\"red\"><i>Failed to retrieve " +
            "property</i></font>";
    }
}

private String getApacheVersion() {

```

```

try {
    String cmd = "pkginfo -l SUNWapchr";
    Process p = Runtime.getRuntime().exec(cmd);
    BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
        p.getInputStream()));

    String curLine = stdInput.readLine();

    while (curLine != null) {
        StringTokenizer st = new StringTokenizer(curLine);
        if (st.nextToken().equals("DESC:")) {
            return(st.nextToken() + " " + st.nextToken()
+ " " +
                st.nextToken() + " " + st.nextToken()
+ " " +
                st.nextToken() + " " + st.nextToken()
+ " " +
                st.nextToken() + " " +
st.nextToken());
        }
        curLine = stdInput.readLine();
    }

    return "<font color=\"red\"><i>Failed to retrieve " +
        "property</i></font>";
}
catch (IOException e) {
    return "<font color=\"red\"><i>Retrieval of this
property is " +
        "only supported when running this software on
" +
        "Solaris</i></font>";
}
catch (NoSuchElementException e) {
    return "<font color=\"red\"><i>Failed to retrieve " +
        "property</i></font>";
}
}
}

```

APPENDIX H. SOURCE CODE – PACKAGE org.dgrp.server.dtraceservices

The complete source code for the reference implementation of this framework is provided in this and other appendices to this document for the reader's easy reference. For the simplest viewing experience or to use the code without copying and pasting it into a new source file, the reader is encouraged to review the soft-copy files available on the CD-ROM included with this document.

DTraceCountDataBytesService.java

```
package org.dgrp.server.dtraceservices;

import java.io.File;
import java.net.URL;
import java.util.*;
import org.dgrp.server.DGRPLogger;
import org.opensolaris.os.dtrace.*;

/**
 *
 * @author chrisne
 */
public class DTraceCountDataBytesService {

    public final int DIRECTION_SENT = 0;
    public final int DIRECTION_RCVD = 1;
    public final int DIRECTION_TOTAL = 2;

    private URL url = DTraceCountDataBytesService.class.
        getResource("/org/dgrp/server/dtraceservices/count_da
ta_bytes.d");

    private Consumer consumer;
    private boolean isRunning = false;

    public void startService(String subnet) {
        try {
            DGRPLogger.log("Entering
```

```

DTraceCountDataBytesService.startService()...\n");
    File scriptFile = new File(url.toURI());
    String macroArgs = new String(subnet);
    DGRPLogger.log("Creating DTrace consumer.\n");
    consumer = new LocalConsumer();
    DGRPLogger.log("Opening DTrace consumer.\n");
    consumer.open();
    DGRPLogger.log("Compiling DTrace script.\n");
    consumer.compile(scriptFile, macroArgs);
    DGRPLogger.log("Enabling DTrace consumer.\n");
    consumer.enable();
    DGRPLogger.log("Starting DTrace consumer.\n");
    consumer.go();
    isRunning = true;
    DGRPLogger.log("Leaving
DTraceCountDataBytesService.startService()...\n");
}
catch (Exception e) {
    e.printStackTrace();
}
}

public boolean isRunning() {
    return isRunning;
}

public void stopService() {
    consumer.stop();
    consumer.close();
    isRunning = false;
}

public String[] getBusiestIPsByDataBytes() {
    DGRPLogger.log("Entering
getBusiestIPsByDataBytes()...\n");

    if (!isRunning()) { //consumer not running, data not
available
        DGRPLogger.log("Consumer not running; returning null
from getBusiestIPsByDataBytes()...\n");
        return null;
    }

    final String totAgg = "tot";
    List ipAddrs = new ArrayList();
    Set<String> aggSet = new HashSet();
    aggSet.add(totAgg);
    Aggregation aggregation;
    try {
        DGRPLogger.log("Getting aggregation from consumer...\n");

```

```

        aggregation =
consumer.getAggregate(aggSet).getAggregation(totAgg);
    } catch (Exception e) {
        //consumer is probably not running, return null
        return null;
    }

    if (aggregation.equals(null)) {
        return null;
    }
    else { //aggregation exists
        DGRPLogger.log("Aggregation existed...\n");
        List list = aggregation.getRecords();
        Collections.sort(list, new AggRecordComparator());
        Iterator iterator = list.iterator();
        while (iterator.hasNext()) {
            AggregationRecord aggRec = (AggregationRecord)
iterator.next();
                String ip = (String)
aggRec.getTuple().iterator().next().getValue();
                ipAdrrs.add(ip);
                DGRPLogger.log("Adding IP: " + ip);
                long val = (long)
aggRec.getValue().getValue().longValue();
                DGRPLogger.log(" (value is " + val + ")\n");
            }
        }

        String[] ipAdrrsStrings = (String[]) ipAdrrs.toArray(new
String[0]);
        DGRPLogger.log("Returning from
getBusiestIPsByDataBytes().\n");
        return ipAdrrsStrings;
    } //end of method

    public int getNumberDataBytesByIP(String ipAddr, int
direction) {
        DGRPLogger.log("Entering getNumberDataBytesByIP(" +
ipAddr + ", " +
            direction + ")\n");

        if (!isRunning()) { //consumer not running, data not
available
            DGRPLogger.log("Consumer not running; returning zero
from getNumberDataBytesByIP().\n");
            return 0;
        }

        final String sndAgg = "snd", rcvAgg = "rcv", totAgg =
"tot";
        Set<String> aggSet = new HashSet();

```

```

aggSet.add(sndAgg);
aggSet.add(rcvAgg);
aggSet.add(totAgg);
Aggregation aggregation;

DGRPLogger.log("Getting aggregation from consumer...\n");
try {
    if (direction == DIRECTION_SENT) {
        aggregation =
consumer.getAggregate(aggSet).getAggregation(sndAgg);
    }
    else if (direction == DIRECTION_RCVD) {
        aggregation =
consumer.getAggregate(aggSet).getAggregation(rcvAgg);
    }
    else if (direction == DIRECTION_TOTAL) {
        aggregation =
consumer.getAggregate(aggSet).getAggregation(totAgg);
    }
    else {
        DGRPLogger.log("Invalid direction, returning
zero.\n");
        DGRPLogger.log("Returning from
getNumberDataBytesByIP()...\n");
        return 0;
    }
}
catch (Exception e) {
    //consumer is probably not running, return 0
    return 0;
}

if (aggregation.equals(null)) {
    return 0;
}
else { //aggregation exists
    DGRPLogger.log("Aggregation existed...\n");
    List list = aggregation.getRecords();
    Iterator iterator = list.iterator();
    while (iterator.hasNext()) {
        AggregationRecord aggRec = (AggregationRecord)
iterator.next();
        String tupleIP = (String)
aggRec.getTuple().iterator().next().getValue();
        if (ipAddr.equals(tupleIP)) {
            int val = (int)
aggRec.getValue().getValue().intValue();
            DGRPLogger.log("Matched IP, value is " + val
+ "\n");
            DGRPLogger.log("Returning from
getNumberDataBytesByIP()...\n");

```

```

        return val;
    }
}

DGRPLogger.log("IP not matched, returning zero.\n");
DGRPLogger.log("Returning from getNumberDataBytesByIP().\n");
return 0;
} //end of method

} //end of class

class AggRecordComparator implements Comparator {
    public int compare(Object obj1, Object obj2) {
        DGRPLogger.log("Using AggRecordComparator.compare.\n");
        AggregationRecord aggRec1 = (AggregationRecord) obj1;
        AggregationRecord aggRec2 = (AggregationRecord) obj2;
        long val1 = aggRec1.getValue().getValue().longValue();
        long val2 = aggRec2.getValue().getValue().longValue();
        if (val1 < val2)
            return 1;
        else if (val1 == val2)
            return 0;
        else
            return -1;
    }
}

```

DTraceCountPacketsService.java

```

package org.dgrp.server.dtraceservices;

import java.io.File;
import java.net.URL;
import org.opensolaris.os.dtrace.*;
import java.util.*;
import org.dgrp.server.DGRPLogger;

/**
 *
 * @author chrisne
 */
public class DTraceCountPacketsService {

    public final int DIRECTION_SENT = 0;
    public final int DIRECTION_RCVD = 1;
    public final int DIRECTION_TOTAL = 2;
}

```

```

        private URL url = DTraceCountPacketsService.class.
            getResource("/org/dgrp/server/dtraceservices/count_packets.d");

        private Consumer consumer;
        private boolean isRunning = false;

        public void startService(String subnet) {
            try {
                DGRPLogger.log("Entering
DTraceCountPacketsService.startService()...\n");
                File scriptFile = new File(url.toURI());
                String macroArgs = new String(subnet);
                DGRPLogger.log("Creating DTrace consumer.\n");
                consumer = new LocalConsumer();
                DGRPLogger.log("Opening DTrace consumer.\n");
                consumer.open();
                DGRPLogger.log("Compiling DTrace script.\n");
                consumer.compile(scriptFile, macroArgs);
                DGRPLogger.log("Enabling DTrace consumer.\n");
                consumer.enable();
                DGRPLogger.log("Starting DTrace consumer.\n");
                consumer.go();
                isRunning = true;
                DGRPLogger.log("Leaving
DTraceCountPacketsService.startService()...\n");
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }

        public boolean isRunning() {
            return isRunning;
        }

        public void stopService() {
            consumer.stop();
            consumer.close();
            isRunning = false;
        }

        public int getNumberPacketsByIP(String ipAddr, int direction)
        {
            DGRPLogger.log("Entering getNumberPacketsByIP(" + ipAddr
+ ", " +
                direction + ")\n");

            if (!isRunning()) { //consumer not running, data not
available

```



```

        DGRPLogger.log("Consumer not running; returning zero
from getNumberPacketsByIP().\n");
        return 0;
    }

    final String sndAgg = "snd", rcvAgg = "rcv", totAgg =
"tot";
    Set<String> aggSet = new HashSet();
    aggSet.add(sndAgg);
    aggSet.add(rcvAgg);
    aggSet.add(totAgg);
    Aggregation aggregation;

    DGRPLogger.log("Getting aggregation from consumer...\n");
    try {
        if (direction == DIRECTION_SENT) {
            aggregation =
consumer.getAggregate(aggSet).getAggregation(sndAgg);
        }
        else if (direction == DIRECTION_RCVD) {
            aggregation =
consumer.getAggregate(aggSet).getAggregation(rcvAgg);
        }
        else if (direction == DIRECTION_TOTAL) {
            aggregation =
consumer.getAggregate(aggSet).getAggregation(totAgg);
        }
        else {
            DGRPLogger.log("Invalid direction, returning
zero.\n");
            DGRPLogger.log("Returning from
getNumberPacketsByIP().\n");
            return 0;
        }
    }
    catch (Exception e) {
        //consumer is probably not running, return 0
        return 0;
    }

    if (aggregation.equals(null)) {
        return 0;
    }
    else { //aggregation exists
        DGRPLogger.log("Aggregation existed...\n");
        List list = aggregation.getRecords();
        Iterator iterator = list.iterator();
        while (iterator.hasNext()) {
            AggregationRecord aggRec = (AggregationRecord)
iterator.next();
            String tupleIP = (String)

```

```

aggRec.getTuple().iterator().next().getValue();
        if (ipAddr.equals(tupleIP)) {
            int val = (int)
aggRec.getValue().getValue().intValue();
            DGRPLogger.log("Matched IP, value is " + val
+ "\n");
            DGRPLogger.log("Returning from
getNumberPacketsByIP().\n");
            return val;
        }
    }
}

DGRPLogger.log("IP not matched, returning zero.\n");
DGRPLogger.log("Returning from
getNumberPacketsByIP().\n");
return 0;
} //end of method

} //end of class

```

count_data_bytes.d

```

#!/usr/sbin/dtrace -s

#pragma D option defaultargs

BEGIN /* Special probe upon script startup */
{
    givenSubnet = $$1; /* subnet either given or set as
empty string */

    printf("\n\n-----
-\n");
    printf("Counting data bytes sent and received by IP
address...\n");

    printf("-----\n"
);
}

ip:::send /* Probe for sent packets (by destination address) */
{
    @snd[args[2]->ip_daddr] = sum(args[2]->ip_length);
    @tot[args[2]->ip_daddr] = sum(args[2]->ip_length);
}

```

```

ip:::receive      /* Probe for received packets (by source
address) */
{
    @rcv[args[2]->ip_saddr] = sum(args[2]->ip_plength);
    @tot[args[2]->ip_saddr] = sum(args[2]->ip_plength);
}

END /* Special probe upon script termination */
{

printf("\n\n-----\n\n");
    printf("Printing results...\n");

printf("-----\n\n");

    printf("\nData bytes sent to:\n");
    printa("  %15s %8u\n", @snd);

    printf("\nData bytes received from:\n");
    printa("  %15s %8u\n", @rcv);

    printf("\nTotal data bytes received from and sent to:\n");
    printa("  %15s %8u\n", @tot);
}

```

count_packets.d

```

#!/usr/sbin/dtrace -s

#pragma D option defaultargs

BEGIN /* Special probe upon script startup */
{
    givenSubnet = $$1; /* subnet either given or set as
empty string */

printf("\n\n-----\n\n");
    printf("Counting packets sent and received by IP address...\n\n");
    printf("-----\n\n");
}

```

```

ip:::send /* Probe for sent packets (by destination address) */
{
    @snd[args[2]->ip_daddr] = count();
    @tot[args[2]->ip_daddr] = count();
}

ip:::receive /* Probe for received packets (by source address) */
{
    @rcv[args[2]->ip_saddr] = count();
    @tot[args[2]->ip_saddr] = count();
}

END /* Special probe upon script termination */
{
printf("\n\n-----\n\n");
printf("Printing results...\n");
printf("-----\n\n");

printf("\nPackets sent to:\n");
printa(" %15s %08u\n", @snd);

printf("\nPackets received from:\n");
printa(" %15s %08u\n", @rcv);

printf("\nTotal packets received from and sent to:\n");
printa(" %15s %08u\n", @tot);
}

```