The CoralReef Software Suite as a Tool for System and Network Administrators

David Moore, Ken Keys, Ryan Koga, Edouard Lagache, and kc claffy – CAIDA

ABSTRACT

Until now, system administrators have lacked a flexible real-time network traffic flow monitoring package. Such a package must provide a wide range of services but remain flexible enough for rapid in-house customization. Existing passive data collection tools are typically narrow in scope, designed for specific tasks from packet capture (tcpdump [9]) to accounting (NeTraMet [4]). In response, CAIDA has created the CoralReef suite designed to provide network administrators and researchers with a consistent interface for a wide range of network analysis applications, from raw capture to flows analysis to real-time report generation. CoralReef provides a convenient set of passive data tools for a diverse audience.

CoralReef is a package of device drivers, libraries, classes, and applications. We briefly outline the architecture and provide relevant case studies and examples of CoralReef's use as applied to real-world networking situations. We will show how CoralReef is a powerful, extensible, and convenient package for network monitoring and reporting.

Introduction

With the growth in traffic volume and increasing diversity of applications on the Internet, understanding and managing networks has become increasingly difficult and important. To this end we have created the CoralReef passive traffic monitoring suite, which allows network users, administrators, and researchers to measure and analyze network traffic. The CoralReef software suite is a comprehensive collection of tools developed by CAIDA to collect, store, and analyze traffic data. CoralReef can be deployed on a dedicated monitor host using data capture cards that tap a fiber optic link, or on virtually any UNIX system without special hardware using libpcap interfaces. CoralReef software handles everything from the low level details of cell and packet capture to the production of high level HTML reports in near real-time. Network and system administrators can use the CoralReef suite to monitor and interpret a wide range of observed network behavior.

CoralReef evolved from OCXmon monitors, developed jointly by MCI and NLANR [1, 2]. The OCXmon monitors ran on MS-DOS, could only monitor ATM links, and provided only basic cell capture (in device-dependent format) and limited flow summary capability. CoralReef runs on UNIX, and supports device independent access to network data from OCXmon hardware, native OS network interfaces, and trace files; programming APIs; a variety of bundled analysis applications; and greater flexibility in remote access and administration. CoralReef is developed and tested under FreeBSD, Linux, and Solaris, although specialized hardware drivers are not available for all operating systems. CoralReef has two releases, a "public" non-commercial use version and a version available only to CAIDA members. Both versions implement the same set of libraries and APIs, but the members-only version incorporates performance and operational enhancements geared toward CAIDA members. What makes CoralReef unique is that it supports a large number of features at many layers, and provides APIs and hooks at every layer, making it easier for anyone to apply it in unanticipated ways and develop new applications with minimum duplicated effort.

Because commercial software tools lack sufficient flexibility, network administrators often develop their own network analysis tools, typically based on tcpdump [9]. A part of tcpdump is the library libpcap [13] which provides a standard way to access IP data and BPF (Berkeley Packet Filter) devices. The tcpdump tool also has a packet data file format (pcap) which has become a *de facto* industry standard. Several network analyzer tools are built on top of libpcap, such as the Ethereal [7] protocol analyzer and NeTraMet (RFC 2722 [5] and RFC 2724 [8]), which are geared toward long term collection for metering and billing. Other network analysis tools include the MEHARI [12] ATM/IP analysis system; Narus [16] for long-term workload and billing; the DAG ATM/POS capture cards and software [19] by the WAND group at the University of Waikato, New Zealand; Clevertool's netboy [6]; Network Associates Sniffer Pro [3]; and NIKSUN's NetVCR [17].

An Overview of the CoralReef Software Suite

CoralReef is a package of libraries, device drivers, classes, and applications written in, and for use with, several programming languages. The overall architecture and programming interfaces are described in a separate paper [11] and are not covered here. Figure 1 shows an overview of the relationships between

CoralReef applications. Detailed descriptions of the software tools can be found at the CoralReef web site (http://www.caida.org/tools/measurement/coralreef/).

Most CoralReef applications fall into one of two categories: those with names beginning with "crl_", which operate on raw packet data; and those with names beginning with "t2_", which operate on aggregated flow data. We will refer to these groups of applications as crl_* and t2_*, respectively. Sources of raw data include custom Coral drivers for special collection cards, the libpcap library for commodity network interfaces, and trace files generated by crl_trace, tcpdump, or other software.

Raw Traffic Applications

All of the crl_* applications take a common set of command line and configuration options. These options include stopping after a specified number of packets or ATM cells or after a specified time duration; link specific parameters; filtering by ATM virtual channels; number of bytes to capture from each packet; and debugging level. Additionally, applications that operate on packets can filter their input with BPF (tcpdump) filter expressions. Applications that operate at regular time intervals have a common syntax for specifying the interval size.

Pure utilities:

- crl_trace: captures network traffic to a .crl trace file
- crl_info: reports hardware and link configuration details of a trace file
- crl_time: outputs timestamps and inter-arrival time information for packets or ATM cells
- crl_encode: encodes the IP addresses in a .crl file to protect privacy
- crl_to_*: captures network traffic or converts trace files to other file formats

Simple tools:

- crl_print: prints headers and payloads of ATM cells
- crl_print_pkt: prints multiple layers of protocol headers and payloads of packets
- crl_rate_layer2: at regular time intervals, outputs cell count and bit rate for each ATM channel
- crl_rate: at regular time intervals, outputs IPv4 and IPv6 packet and byte counts, and counts of non-IP packets

Static Reports:

- crl_hist: reports packet and byte counts by IP length and protocol, port summary matrices for TCP and UDP, fragment counts by protocol, packet length histograms for the entire trace and for a list of applications, and the top 10 source and destination port numbers seen for TCP and UDP traffic
- crl_bycountry: reports the amount of traffic flowing to and from networks, and between networks, ASes, and countries

Specialized Utilities:

- crl_portmap: captures all packets from any hosts that connect to another host's portmap port
- crl_flow: at regular time intervals, aggregates packet data into flows by source and destination IP addresses, protocol, and source and destination ports

Traffic Flow Applications

The t2_* applications operate on tables generated by crl_flow or other t2_* applications, with the same time intervals.

- t2_report: generates HTML summary reports
- t2_ASmatrix: with a routing table, source and destination ports
- t2_top: sorts a table by packets, bytes, or flows, and displays the top N entries

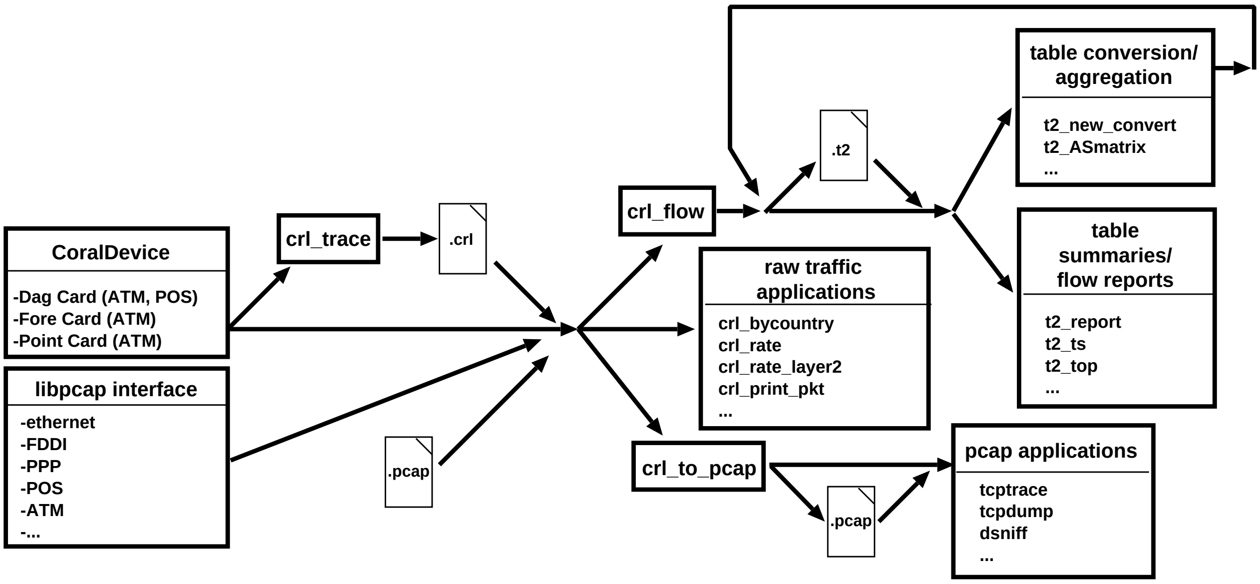


Figure 1: Overview of CoralReef applications.

- t2_ts: outputs counts of IP packets, bytes, and flows
- t2_convert: aggregates tables by specified keys

Other Applications

- crl_to_pcap: converts Coral traces or live data to pcap format for use with existing libpcap tools
- parse_bgp_dump: converts Cisco router ''sho ip bgp'' output to the routing table format used by t2_ASmatrix, t2_report, and crl_bycountry
- parse_bgp_mrtd: converts MRTd [15] output to the routing table format used by t2_ASmatrix, t2_report, and crl_bycountry

Libraries

- libcoral: reads trace files and live network interfaces, and provides common functionality for all crl_* applications
- Coral.pm: perl interface to libcoral
- ASFinder: maps IP addresses to AS numbers and network prefixes
- AppPorts: maps protocols and port numbers to application names
- NetGeoClient: maps IP addresses and AS numbers to geographic locations
- Tables: manipulating and processing the tables used by the t2_* applications

Using CoralReef in an Operational Setting

CoralReef can only monitor traffic that is visible to a network interface. If the network you want to monitor is a shared medium such as non-switched Ethernet or FDDI, any interface on that network is sufficient. Monitoring a link between routers or on a switched network requires directing traffic into additional dedicated interfaces, which may be either standard interfaces read via libpcap, or special hardware accessed through Coral drivers. A link can be tapped either with a physical splitter (Figure 2a) or by configuring a span or mirror port on the appropriate switch or router (Figure 2b). Note that tapping both directions of a link with splitters requires a dedicated interface for each direction.

The hardware needed depends on the utilization of the links being monitored and the amount of aggregation desired. For straightforward packet traces, the main constraint is usually disk performance and capacity; we recommend ultra-wide SCSI rather than an IDE drive. For flow collection and analysis, memory and CPU speed are more important. Individual applications in a CoralReef pipeline can run on separate machines to distribute the load. A common example of this is to run crl_flow on the monitor machine and t2_report on a different machine.

Examples

In this section, we briefly present examples of using CoralReef in an operational setting. A more complete outline of uses can be found in the CoralReef documentation or the user community mailing list.

Several -C options are common to all crl_* applications. In the following examples, we use -Ci=*time* to specify the repeated interval at which the application processes data and outputs results, and -C'filter *expression*' to specify a BPF filter expression that selects the packets to be measured. To stop the applications after a specific duration, you would use the -Cd=*time* option. The crl_* applications can read traffic from a variety of sources; in these examples, the data source is ''if:fxp0'', a native Ethernet interface (fxp0) read via libpcap.

IP addresses in the sample outputs have been encoded for privacy. Some output has been edited to better fit the page and for illustrative value.

Timestamps in application output are printed in UNIX timestamp format.

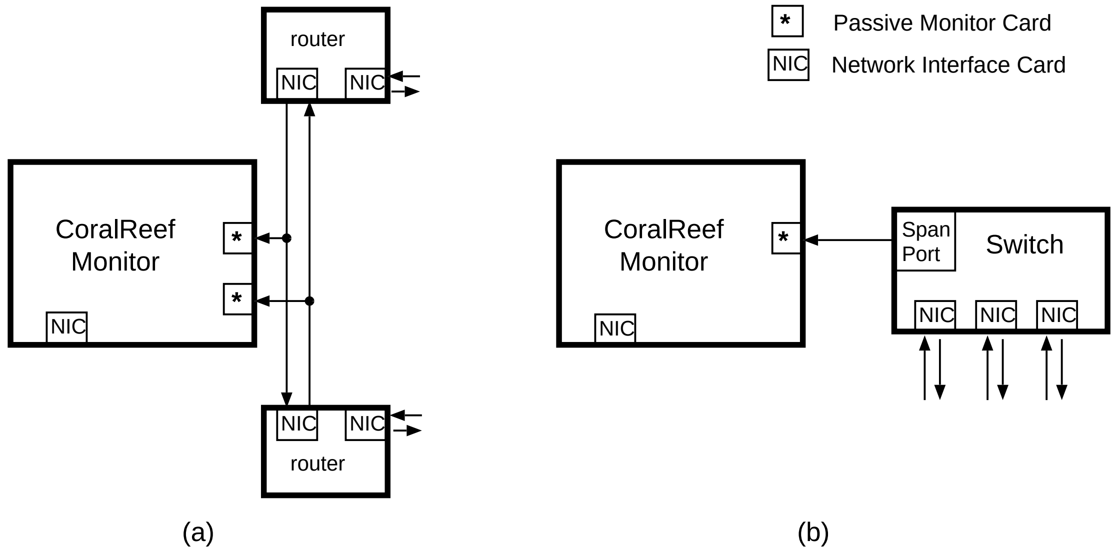


Figure 2: Examples of tapping a link.

Using crl_rate to Check Utilization of Subinterfaces

The crl_rate application counts packets and bytes on interfaces and subinterfaces at regular intervals. On Ethernet interfaces, subinterfaces are IEEE 802.1Q VLANs. On ATM interfaces, virtual channels are reported as subinterfaces. Other types of interfaces do not have subinterfaces.

Goal: Continuously Measure Traffic on a Link, in Packets and Bytes

Command line: crl_rate -Ci=10 if:fxp0
See Listing 1 for the sample output.
Explanation:

This output shows that there was traffic on seven VLANs on Ethernet interface fxp0 in a 10 second period. A similar table would be printed every 10 seconds. The "non_ip" column counts packets of protocols like ARP, AppleTalk, and IPX. The total IPv4 traffic on this link in 10 seconds was about 99.5 Megabytes, or 79.6 Megabits per second. Note that the bytes counted are those in layer three and above; bytes in lower layer encapsulations like Ethernet and ATM are not counted.

This simple example is a good way to test your CoralReef monitor and software setup to verify that the output matches your expectations.

There was only one interface in this example, labeled 0 in the output, but it is possible to monitor multiple interfaces simultaneously.

Goal: Find Out How Much KaZaA Traffic is on Your Link

Command line:
crl_rate -s -Ci=300 -C'filter port 1214' if:fxp0
See Listing 2 for sample output.
Explanation:

In this example, we were not interested in subinterfaces (VLANs), so we used the -s option to omit them. Because of high variability in these kinds of measurements, larger intervals are usually more useful. In this example, we used a five minute interval. To limit the measured traffic to KaZaA [10], we used a BPF filter to match only traffic to or from KaZaA's well-known port (1214). In the first of the two intervals shown, there were about 1.33 Gigabytes of KaZaA traffic, or 35.5 Megabits per second.

Using crl_flow to Collect Flow Data

The crl_flow application summarizes data by IP flows. In this context, a flow is identified by the 5-tuple of source address (src), destination address (dst), protocol (proto), source port (sport), and destination port (dport). A flow is unidirectional, so there will be one flow for each of the two directions of a network connection, with sources and destinations swapped.

The definition of flow termination can be chosen by a command line option. The -I option specifies that flows terminate at the end of each interval, which is the most useful definition for this kind of continuous monitoring. Other definitions are typically more useful in offline analysis of historic data, as is often needed in research situations.

A table of these 5-tuples, along with counts of packets, bytes, and flows for each, is called a Tuple_Table. The "ok" column contains a 1 if sport and dport are meaningful for the protocol and were not truncated by capturing too few header bytes. Ports are meaningful for TCP, UDP, and ICMP (for ICMP, the sport and dport columns actually contain ICMP type and code, respectively).

Goal: Continuously Collect Data on Link Use, Summarized by Hosts, Protocols, and Ports.

Explanation:

The -h option tells crl_flow to print in human-readable format. With no formatting option, crl_flow prints a tab-separated format more suitable for input to other scripts. Additionally, crl_flow -b outputs a binary

```
# time 1001975450.054545 (10.000000), packets lost: 0
# if[subif]    ipv4pkts      ipv4bytes   ipv6pkts      ipv6bytes     non_ip
0[135]                1             40          0              0          0
0[110]             2269        2536140          0              0          0
0[169]             9397        3761410          0              0          0
0[170]            40097       20640233          0              0          0
0[130]             5659        1921566          0              0          0
0[131]           118429       70553909          0              0          0
0[108]             1774          92307          0              0          0
0   TOTAL        177626       99505605          0              0          0
. . .
```

Listing 1: crl_rate output: traffic on a link.

```
# time 1001977053.013038 (300.000000), packets lost: 0
# if[subif]    ipv4pkts      ipv4bytes   ipv6pkts      ipv6bytes     non_ip
0   TOTAL       1999660     1330861215          0              0          0

# time 1001977353.013038 (300.000000), packets lost: 0
# if[subif]    ipv4pkts      ipv4bytes   ipv6pkts      ipv6bytes     non_ip
0   TOTAL       1956821     1215396030          0              0          0
. . .
```

Listing 2: crl_rate output: KaZaA traffic.

format that is readable by the t2_* applications, more efficiently than either of the text formats.
Command line: crl_flow -I -h -Ci=10 if:fxp0
See Listing 3 for sample output.

The output shown here has been edited to fit the page. Real output would have a Tuple_Table for each interface and subinterface, repeated every 10 seconds; an interface or subinterface summary preceding each table; and two additional columns in each table showing the first and last packet timestamp observed within each flow. Remember that IP addresses have been encoded for privacy.

The sample output shows one UDP DNS flow (protocol 17, port 53), one HTTP flow from a web server to a client, several HTTP flows from clients to web servers, and one large flow on TCP port 1214 (KaZaA).

Using t2_* to Monitor Utilization and Flows

Although crl_flow does some aggregation, its output is still typically too voluminous to be directly useful. The t2_* applications further aggregate or filter the output of crl_flow for more specific needs. In particular, t2_ts outputs a single line per interval summarizing the packets, bytes, and flows observed. t2_top sorts table entries by packets, bytes, or flows, and prints only the top *N*.

Most t2_* applications accept different table types as input, so the user must specify the type on the command line. crl_flow outputs a Tuple_Table; we will introduce other table types in later examples.

Goal: Continuously Measure Traffic on a Link, in Packets, Bytes, and Flows

Command line:
crl_flow -I -b -Ci=10 if:fxp0 | t2_ts Tuple_Table
See Listing 4 for sample output.
Explanation:

In this example, t2_ts prints a line for every 10 second interval, the beginning of which is indicated in the first column (time). The next three columns show the total number of packets, bytes, and flows observed during the interval. The ''entries'' column shows the number of table entries, which, in the case of a Tuple_Table, is equal to the number of flows. The last three columns show the average number of packets, bytes, and flows per second during the interval.

The -b option to crl_flow tells it to output in efficient binary format readable by t2_* applications. The use of this option can drastically improve performance, and is recommended when the intermediate output does not need to be read by a human.

Goal: Continuously Find Flows Consuming the Most Bandwidth on a Link

Command line:
crl_flow -I -b -Ci=10 if:fxp0 |
t2_top -b -n5 Tuple_Table

```
# crl_flow output version: 1.0 (pretty format)

# begin trace interval: 1001981488.441461
# trace interval duration: 10.000000 s
# Layer 2 PDUs dropped: 0
# IP: 101.8403 Mbit/s
# Non-IP: 0.0000 pkts/s
# Table IDs: 0[131], 0[108], 0[130], 0[110], 0[170], 0[169]

...
# begin Tuple Table ID: 0[131]
# expired flows
#src           dst           proto ok sport dport     pkts     bytes  flows
0.1.0.8        1.82.0.1         17  1    53    53        2       497      1
0.1.0.14       0.44.0.1          6  1    80  2223        4       646      1
0.3.0.148      1.95.0.1          6  1  1214 62772      125    187008      1
0.1.1.93       0.71.0.6          6  1 49200    80        3       565      1
0.1.1.93       0.71.0.6          6  1 49199    80        5       647      1
0.1.1.93       0.71.0.6          6  1 49198    80        5       647      1
0.1.1.93       0.71.0.6          6  1 49196    80        6       708      1
0.1.2.59       11.88.0.1         6  1 51643    80        6       817      1
...
# end of text table
...
# end trace interval
...
```

Listing 3: crl_flow output: Continuous data collection.

```
time                 pkts    bytes    flows  entries   pkts/s      bytes/s    flows/s
1001982746.151594  143315  97458948    6624     6624  14331.500  9745894.800  662.400
1001982756.151594  143985  98792491    6465     6465  14398.500  9879249.100  646.500
...
```

Listing 4: ts_ts output: Utilization and flows.

See Listing 5 for sample output.
Explanation:

The "KEYS" columns are the same as the keys in the input table, which in this example is a Tuple_Table from crl_flow. The -p, -b, or -f option tells t2_top to sort by packets, bytes, or flows, and the -n option specifies how many entries to print.

Using t2_* to Find Hosts Generating the Most Traffic

Often we want to aggregate the flows of a Tuple_Table by a subset of its keys. For example, we may want to count the bytes sent between pairs of hosts, regardless of their protocols and ports; or, all the packets sent from a particular TCP port, no matter what host sent or received them.

Table Type	Keys
Tuple_Table	source IP, destination IP, IP protocol, ports ok, source port, destination port
IP_Table	IP
IP_Matrix	source IP, destination IP
Proto_Ports_Table	IP protocol, ports ok, source port, destination port
Port_Table	port
Port_Matrix	source port, destination port
Proto_Table	IP protocol
AS_Table	AS
AS_Matrix	source AS, destination AS
Country_Table	country
Country_Matrix	source country, destination country
App_Table	application
VPVC_Table	vp/vc pair
Prefix_Table	prefix/masklength
Prefix_Matrix	source prefix/masklength, destination prefix/masklength
Length_Table	length

Table 1: CoralReef table types.

In addition to the Tuple_Table, CoralReef has other table types defined by different sets of keys. For example, the keys of an IP_Matrix are source and destination IP addresses, and the key of an IP_Table is a single IP address. Table 1 shows all tables and their keys.

The t2_convert application converts one table type to another by aggregating entries with common keys. A conversion operator determines which subset of input table keys to use as the keys of the output table. For example, applying the src_IP_Table operator to a Tuple_Table generates an IP_Table whose keys are the source addresses of the input table. The pkts, bytes, and flows counts of each entry in the new table are the sums of the corresponding counts of the Tuple_Table entries with the same source IP address. Figure 3 shows all the operators that can be applied to the various table types.

Goal: Find the Top Five Hosts by Bytes of Traffic Generated

Command line:
```
crl_flow -I -b -Ci=10 if:fxp0 |
  t2_convert Tuple_Table src_IP_Table |
  t2_top -b -n5 IP_Table
```
See Listing 6 for sample output.
Explanation:

The output of crl_flow is a Tuple_Table, with keys src, dst, proto, ok, sport, and dport. To aggregate those flows by source IP address, we apply the src_IP_Table operator with t2_convert. Since the flows column in a Tuple_Table is always 1, the flows column in the resulting IP_Table is the number of flows with that source IP address. Sorting this IP_Table by bytes and taking the top five entries shows the hosts sending the most bytes.

Goal: Find the Top Five Web Servers by HTTP Flows

Command line:
```
crl_flow -I -b -Ci=10 -C'filter tcp src port 80' if:fxp0 |
  t2_convert Tuple_Table src_IP_Table |
  t2_top -f -n5 IP_Table
```
See Listing 7 for sample output.
Explanation:

This is similar to the previous example, except we limit the traffic to web servers by using a filter option to crl_flow and sort by flows (-f) instead of bytes. Aggregating this Tuple_Table by source IP address and then sorting the resulting IP_Table by flows shows the

```
src         dst          proto  ok    sport    dport
#KEYS                                                   pkts    bytes     flows
0.4.0.27    0.98.0.1       6     1    46978    64671    16035   22534236      1
0.4.0.30    0.19.0.2       6     1    22       64156    2965    4230700       1
0.4.0.44    0.158.0.1      6     1    22       33222    3647    3919348       1
0.4.0.4     0.17.0.1       6     1    80       58013    1831    2702668       1
0.4.0.3     0.15.0.1       6     1    45925    20       2244    2390668       1
# end of text table
#KEYS                                                   pkts    bytes     flows
0.4.0.27    0.98.0.1       6     1    46995    64683    9311    13084084      1
0.4.0.27    0.98.0.1       6     1    46978    64671    9095    12780460      1
0.4.0.30    0.19.0.2       6     1    22       64156    3097    4417620       1
0.4.0.44    0.158.0.1      6     1    22       33222    3185    3350880       1
0.4.0.21    0.73.0.2       6     1    60971    119      1362    1915352       1
# end of text table
```
Listing 5: t2_top output: High bandwidth consumers.

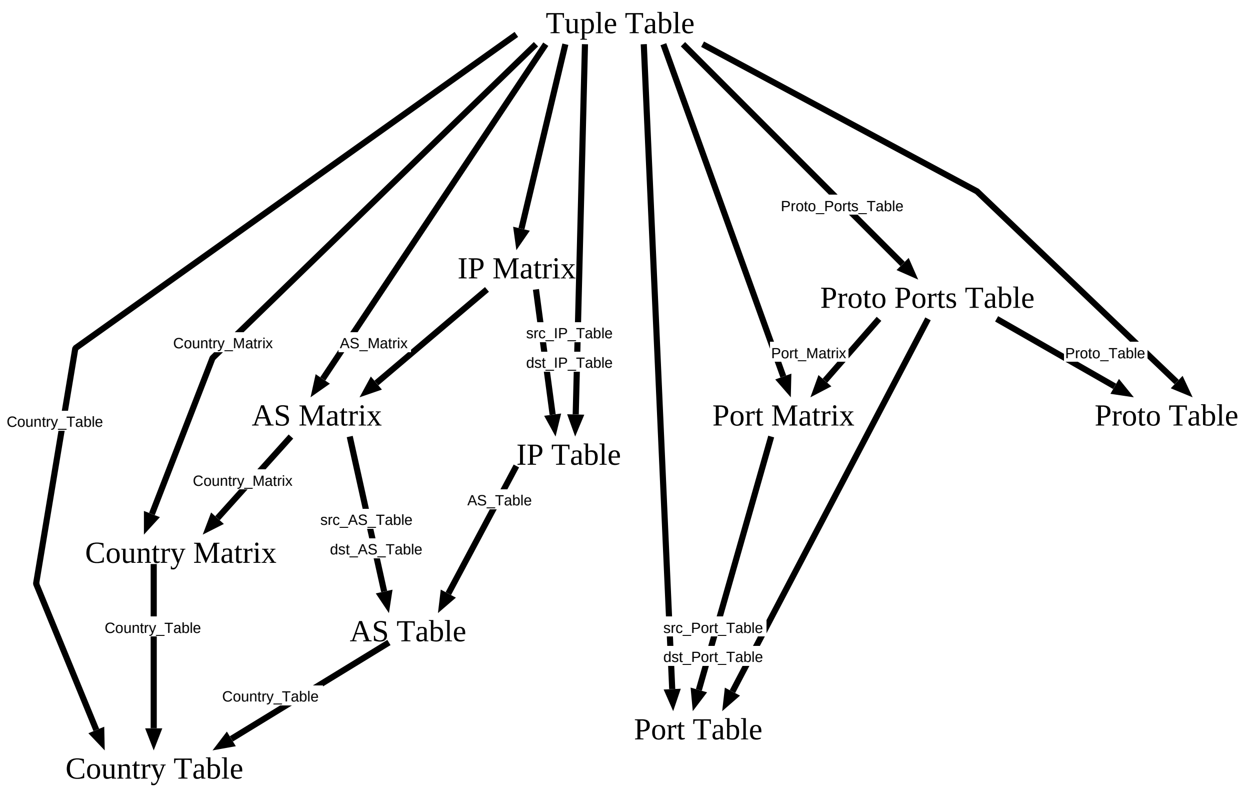


Figure 3: Table conversion operations.

```
#KEYS           pkts    bytes   flows   (top 5 sorted by bytes)
0.4.0.27        16035   22534236        1
0.4.0.21        12202   13663537        46
0.4.0.30        2965    4230700         1
0.4.0.44        3647    3919348         1
0.4.0.4         1831    2702668         1
# end of text table
#KEYS           pkts    bytes   flows   (top 5 sorted by bytes)
0.4.0.27        18409   25864829        3
0.4.0.21        13900   15515873        46
0.4.0.30        3097    4417620         1
0.4.0.44        3185    3350880         1
0.4.0.64        1347    1948443         7
# end of text table
...
```

Listing 6: t2_top output: Top five source hosts.

```
#KEYS           pkts    bytes   flows   (top 5 sorted by flows)
0.3.0.77        126     101845  23
0.1.0.108       102     52397   17
0.1.0.52        180     78166   15
0.1.0.42        8       320     8
0.1.0.182       24      1713    5
# end of text table
#KEYS           pkts    bytes   flows   (top 5 sorted by flows)
0.1.0.108       192     80733   36
0.3.0.77        131     119220  22
0.1.2.145       5       205     5
0.1.2.135       66      72221   5
0.1.0.119       17      25500   4
# end of text table
...
```

Listing 7: t2_top output: Top five web servers.

web servers with the most HTTP connections during each 10 second interval.

Using t2_* to Find Hosts Talking to the Most Hosts

Normally, the flows column in each entry of the output table of t2_convert is the sum of the flows column of the input table entries with the same output keys. But with the -F option of t2_convert, the flows column in each output entry will be the *number* of input entries with the same output keys. For example, given this IP_Matrix table:

```
src       dst       pkts bytes flows
0.0.0.1   0.0.0.2     3   120     3
0.0.0.1   0.0.0.3     1    40     1
0.0.0.1   0.0.0.4    10   400     5
```

The command "t2_convert IP_Matrix src_IP_Table" yields an IP_Table in which the flows column shows the number of 5-tuple flows with the given source address:

```
src                 pkts bytes flows
0.0.0.1               14   560     9
```

but "t2_convert -F IP_Matrix src_IP_Table" yields an IP_Table in which the flows column shows the number of IP pairs with the given source address:

```
src                 pkts bytes flows
0.0.0.1               14   560     3
```

Goal: Find the Number of Unique Destination Hosts for Each Source Host

Command line:

```
crl_flow -I -b -Ci=10 if:fxp0 |
  t2_convert Tuple_Table IP_Matrix |
  t2_convert -F IP_Matrix src_IP_Table |
  t2_top -f -n5 IP_Table
```

Sample output:

```
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
0.1.0.120      1059    135597  159
0.1.0.5        1224    149885  143
0.4.1.16       188     13280   110
0.4.0.7        1239    140900  87
0.1.0.1        799     67787   78
# end of text table
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
0.1.0.120      1159    93601   167
0.1.0.5        1553    119823  157
0.4.0.7        1042    109109  79
0.1.0.1        884     75684   71
0.1.0.65       941     106510  65
# end of text table
```

Explanation:

Remember, since the -F option was used on the second t2_convert, the flows column is actually the number of corresponding entries in the IP_Matrix input table, i.e., the number of IP pairs with the given source address. So, in the first 10 second interval, host 0.1.0.120 sent traffic to 159 different destination hosts, totaling 135597 bytes.

Goal: Find the Top Five Web Servers by Number of Clients

Command line:

```
crl_flow -I -b -Ci=10 -C'filter tcp src port 80' if:fxp0 |
  t2_convert Tuple_Table IP_Matrix |
  t2_convert -F IP_Matrix src_IP_Table |
  t2_top -f -n5 IP_Table
```

Sample output:

```
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
0.1.0.52       180     78166   11
0.1.0.108      102     52397   6
0.1.0.62       33      33203   4
0.1.0.182      24      1713    4
0.1.0.231      40      33100   3
# end of text table
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
0.1.0.108      192     80733   11
0.1.2.135      66      72221   5
0.1.0.182      20      7135    4
0.3.0.77       131     119220  2
0.1.1.7        145     39893   2
# end of text table
```

Explanation:

This example is similar to the previous one, except that we first filter the traffic to measure only packets sent by HTTP servers. So, in the first 10 second interval, host 0.1.0.52 sent HTTP traffic to 11 different destination hosts, totaling 78166 bytes.

Goal: Find Hosts on Your Internal Network Trying to Spread the CodeRed Worm

Command line:

```
crl_flow -I -b -Ci=60 -C'filter tcp dst port 80 and \
    src net 10.0.0.0/8' if:fxp0 |
  t2_convert Tuple_Table IP_Matrix |
  t2_convert -F IP_Matrix src_IP_Table |
  t2_top -f -n5 IP_Table
```

Sample output:

```
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
10.0.39.61     7680    460800  7680
10.0.198.103   8960    358400  6144
10.0.39.60     6144    368640  6144
10.0.0.190     7168    299008  4864
10.0.19.1      12544   602112  4608
# end of text table
#KEYS          pkts    bytes   flows
# (top 5 sorted by flows)
10.0.39.61     8448    506880  8448
10.0.39.60     7424    445440  7424
10.0.198.103   7680    307200  6656
10.0.0.141     6144    256000  4352
10.0.213.103   4352    188416  3584
# end of text table
```

Explanation:

Hosts infected with the CodeRed worm try to infect large numbers of other hosts by attempting to open an HTTP connection to random IP addresses (which may or may not actually exist or be running a web server) [20]. With the exception of web caches, most hosts do not open HTTP connections to more than a few different servers per second, so we should be suspicious of any host that tries to connect to significantly more servers. In particular, hosts infected with CodeRed attempt to open HTTP connections to many tens or hundreds of hosts per second. By using a filter that selects only packets from the internal network (10.0.0.0/8 in this example) to HTTP servers, and by seeing which of the hosts sending those packets are attempting to communicate with the most servers, we produce a list of internal hosts that are behaving suspiciously.

Report Generator

The CoralReef report generator provides a web interface to continuously updated link usage reports. The report generator (t2_report) is a Perl application, using C backends for speed, which receives (via a pipe) either live data or traces taken from crl_flow. t2_report collects and displays timeseries information by using RRDtool [18].

The report generator utilizes many of the features of the CoralReef suite and thus illustrates some of the capabilities of the suite. At configurable intervals (e.g., every five minutes), t2_report produces pie charts and tables of traffic data from the most recent sample interval, and timeseries graphs of data over the last hour, day, week, month, and year. All three report forms present data as bytes, packets, and flows. The pie charts and tables show protocol breakdown, applications, flows, source/destination hosts, unknown TCP and UDP, and source/destination ASes and countries. The timeseries graphs show absolute counts and percentages for protocol breakdown and applications. There are two sets of application timeseries graphs.

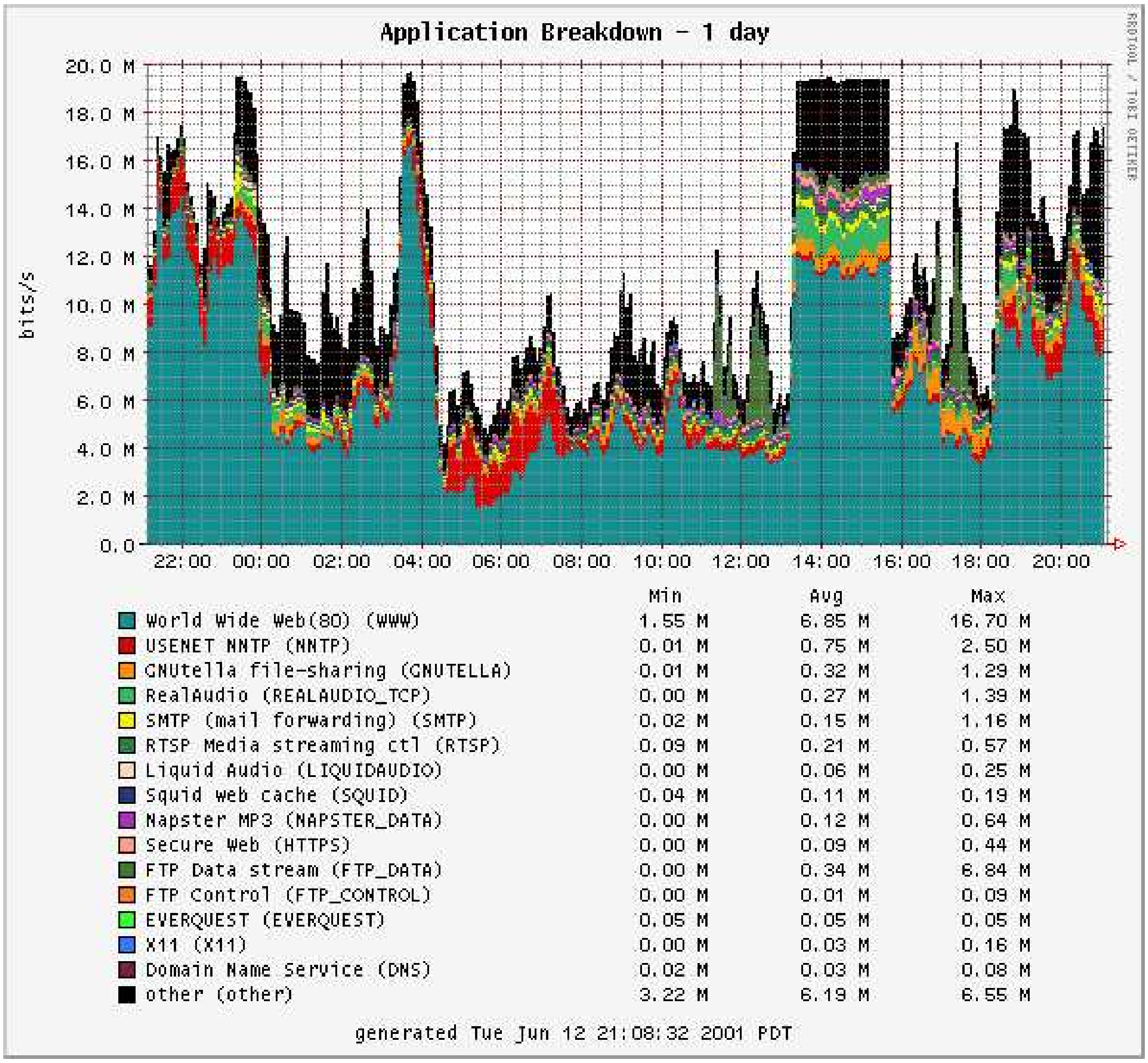


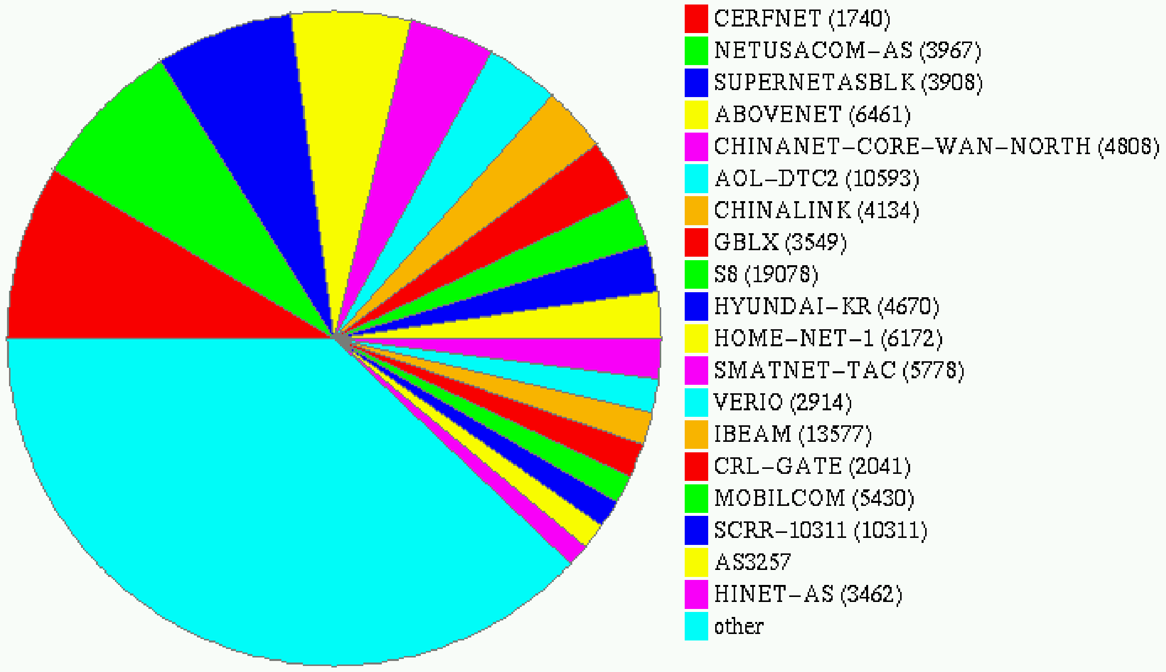
Figure 4: Example of a timeseries plot of application breakdown by bytes.

One shows only the applications specified in the t2_report configuration file, and the other shows the applications with the most traffic in each interval.

To report traffic by AS number, countries, and application names, t2_report must use external data not present in the packets themselves. t2_report uses a library called ASFinder and a routing table, as output by parse_bgp_*, to map IP addresses to AS numbers. To get countries and AS names from AS numbers, t2_report uses NetGeo [14]. Application names are found by the AppPorts library, which uses a prioritized ruleset to map protocol and port numbers to applications. Users can add or modify application rules by editing a simple text file.

Figure 4 shows an example of a timeseries plot of application breakdown by bytes. Figure 5 shows an example of the top source ASes by bytes in a five minute period. The CoralReef web site has a live demonstration of the report generator monitoring the commodity traffic link for the U. C. San Diego.

Vpvc data 0[1:137]: CERFnet to UCSD/SDSC inbound traffic
Wed Jun 13 04:08:00 2001 UTC
300.00s



Overall Performance:

Byte rate: 15.5966 Mbits/s
Packet rate: 4.1023 Kpackets/s
Tuple rate: 163.2733 tuples/s
Total unique Source AS entries: 1789 (top 100 by bytes shown)

Source AS	Mbits/s	% bytes	Kpkts/s	% packets	tuples/s	% tuples
CERFNET (1740)	1.3413	8.60	0.2237	5.45	0.7033	0.43
NETUSACOM-AS (3967)	1.1639	7.46	0.1853	4.52	11.8267	7.24
SUPERNETASBLK (3908)	1.0676	6.85	0.0890	2.17	0.0233	0.01
ABOVENET (6461)	0.9234	5.92	0.1097	2.67	2.3067	1.41
CHINANET-CORE-WAN-NORTH (4808)	0.6644	4.26	0.0940	2.29	3.0333	1.86
AOL-DTC2 (10593)	0.5656	3.63	0.0865	2.11	4.2400	2.60
CHINALINK (4134)	0.5024	3.22	0.0760	1.85	1.6933	1.04
GBLX (3549)	0.4740	3.04	0.0596	1.45	2.8133	1.72

Figure 5: Example of top source ASes by bytes in a five minute period.

Conclusion

CoralReef provides a suite of tools to aid network administrators in monitoring and diagnosing changes in network behavior. CoralReef provides a unified platform to a wide range of capture devices and a collection of tools that can be applied at multiple levels of the network. Its components provide measures on a wide range of real-world network traffic flow applications, including validation and monitoring of hardware performance for saturation and diagnosis of network flow constraints. CoralReef can be used to produce standalone results or produce data for analysis by other programs. CoralReef reporting applications can output in text formats that can be easily manipulated with common UNIX data-reduction utilities (e.g., grep), providing enormous flexibility for customization in an operational setting.

CoralReef provides a balanced collection of features for network administrators seeking to monitor their network and diagnose trouble spots. It serves as a useful bridge between higher level monitoring tools which only work at a coarse level of aggregation and ''dump'' utilities which may overwhelm the administrator with detail. By covering the range from raw packet capture to real-time HTML report generation, CoralReef provides a viable toolkit for wide range of network administration needs.

Acknowledgments

Support for CoralReef is provided by NSF Grant NCR-9711092, DARPA NGI Contract N66001-98-2-8922, DARPA NMS Grant N66001-01-1-8909, and by CAIDA members. We would like to thank Mike Tesch (formerly CAIDA) and Jambi Ganbar of MCI (formerly CAIDA) for early prototypes and testing; Sue Moon of Sprint Advanced Technology Laboratories and Chris Rapier of Pittsburgh Supercomputing Center for their feedback on CoralReef; Nevil Brownlee, Young Hyun, Colleen Shannon, Daniel J. Plummer, and everyone else at CAIDA for their input and support.

Availability

The CoralReef software package is available for non-commercial use from http://www.caida.org/tools/measurement/coralreef/ . Questions about CoralReef can be e-mailed to coral-info@caida.org .

Author Information

David Moore is the Co-Director and a PI of CAIDA (the Cooperative Association for Internet Data Analysis). David's research interests are high speed network monitoring, denial-of-service attacks and infrastructure security, and Internet traffic characterization. His current research includes using the backscatter analysis technique to track and quantify global DoS attacks and Internet worms.

Ken Keys is lead developer of CAIDA's CoralReef project. He has been involved with network research for three years and programming UNIX networking code for over a decade. Ken is known to many for his years of work on TinyFugue, a popular MUD client. Reach him electronically at kkeys@caida.org .

Ryan Koga is CAIDA's resident expert at integrating C and C++ with Perl. He spends most of his time developing libraries for CoralReef and writing assorted programs for other CAIDA projects.

Edouard Lagache is a Researcher and Perl developer with CAIDA. He received his Ph.D. from the University of California, Berkeley in 1995. He has done research on human/computer interaction and social aspects of learning.

kc claffy is Co-Director and a PI of CAIDA, and a resident research scientist based at the University of California's San Diego Supercomputer Center. kc's research interests include Internet workload/performance data collection, analysis and visualization, particularly with respect to commercial ISP collaboration/cooperation and sharing of analysis resources. kc received her Ph.D. in Computer Science from UCSD in 1994.

References

[1] J. Apisdorf, k claffy, K. Thompson, and R. Wilder, ''OC3MON: Flexible, Affordable, High-Performance Statistics Collection,'' *INET'97 Proceedings*, http://www.isoc.org/isoc/whatis/conferences/inet/97/proceedings/F1/F1_2.HTM, June 1997.

[2] Apisdorf, J., k claffy, Kevin Thompson, and Rick Wilder, ''OC3MON: Flexible, Affordable, High Performance Statistics Collection,'' *Proceedings of the 1996 LISA X Conference*, 1996.

[3] Network Associates, ''Sniffer home,'' http://www.sniffer.com .

[4] Brownlee. N., *RFC 2123: Traffic Flow Measurement: Experiences with NeTraMet*, March, 1997.

[5] Brownlee, N., C. Mills, and G. Ruth, *RFC 2722: Traffic Flow Measurement: Architecture*, October, 1999.

[6] Clevertools, ''Analyzer – Packet-Sniffer – Network Tools,'' http://www.clevertools.com/ .

[7] Combs, Gerald, et al., ''Ethereal – A Network Protocol Analyzer,'' http://www.ethereal.com/ .

[8] Handelman, S., S. Stibler, N. Brownlee, and G. Ruth, *RFC 2724: RTFM: Net Attributes for Traffic Flow Measurement*, October, 1999.

[9] Jacobson, V., C. Leres, and S. McCanne, ''tcpdump,'' Lawrence Berkeley Laboratory, Berkeley, CA, ftp.ee.lbl.gov, June 1989.

[10] KaZaA, ''KaZaA media sharing,'' http://www.kazaa.com/ .

[11] Keys, Ken, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and k claffy, ''The

Architecture of CoralReef: An Internet Traffic Monitoring Software Suite,'' *PAM2001 – A Workshop on Passive and Active Measurements*, CAIDA, RIPE NCC, http://www.caida.org/outreach/papers/pam2001/coralreef.xml, April, 2001.
[12] Lizcano, P. J., A. Azcorra, J. Solé-Pareta, J. Domingo-Pascual, and M. Alvarez Campana, ''MEHARI: A system for Analysing the Use of Internet Services,'' *Computer Networks*, Vol. 81, pp. 2293-2307, 1999.
[13] McCanne, S., C. Leres, and V. Jacobson, ''libpcap,'' Lawrence Berkeley Laboratory, ftp.ee.lbl.gov, Berkeley, CA.
[14] Moore, David, Ram Periakaruppan, Jim Donohoe, and kc claffy, ''Where in the World is netgeo.caida.org?'' *INET 2000 Proceedings*, June, 2000.
[15] MRTd, ''MRT – multi-threaded routing toolkit,'' http://www.mrtd.net/ .
[16] Narus, ''Narus IBI Platform,'' http://www.narus.com/ibi/ .
[17] Niksun, ''NetVCR,'' http://www.niksun.com/products/netvcr.html .
[18] Oetiker, T., *RRDtool – Round Robin Database.*
[19] Waikato Applied Network Dynamics Group, ''The DAG Project,'' http://dag.cs.waikato.ac. nz/ .
[20] eEye Digital Security, ''.ida 'Code Red' Worm,'' http://www.eeye.com/html/Research/Advisories/AL20010717.html .