Traffic Data Repository at the WIDE Project

Kenjiro Cho
Sony CSL
kjc@csl.sony.co.jp

Koushirou Mitsuya
Keio University
mitsuya@sfc.wide.ad.jp

Akira Kato
The University of Tokyo
kato@wide.ad.jp

Abstract

It becomes increasingly important for both network researchers and operators to know the trend of network traffic and to find anomaly in their network traffic. This paper describes an on-going effort within the WIDE project to collect a set of free tools to build a traffic data repository containing detailed information of our backbone traffic. Traffic traces are collected by *tcpdump* and, after removing privacy information, the traces are made open to the public. We review the issues on user privacy, and then, the tools used to build the WIDE traffic repository. We will report the current status and findings in the early stage of our IPv6 deployment.

1 Introduction

In this paper, we introduce an on-going effort within the WIDE project to collect a set of free tools to build a traffic data repository containing detailed information of our backbone traffic. The WIDE project makes the resulting data sets publicly accessible so that this project is not only on freely-redistributable software but also on freely-redistributable traffic data sets.

The WIDE project is a research consortium in Japan established in 1987. The members of the project include network researchers, engineers and students of universities, industries and government. The focus of the project is empirical study on a live large-scale internet. Thus, WIDE runs its own internet testbed carrying both commodity traffic and research experiments. WIDE is also responsible for various Internet operations including the M-root name server, NSPIXP(Network Service Provider Internet eXchange Point), AI3(Asian Internet Interconnection Initiatives), and 6Bone in Japan.

The goals of our traffic repository are to promote traffic analysis research as well as to promote development of tools. Traffic characteristics in a backbone network are considerably different from those in a local area network but few people have access to traffic traces from backbone networks. Obtaining details of backbone traffic is getting harder as more backbone networks are shifting to commercial ISPs, which motivate us to build a traffic repository [KMKA99]. Traffic traces from the 6Bone is also made available to promote development of IPv6-ready tools.

Traffic traces are collected at several points within the WIDE backbone. Traces are in the *tcpdump* raw format so that all header information is available and can be used for detailed analysis.

We use commodity hardware and the existing freely-available tools for building our traffic repository so that it has nothing technically fancy. Our focus is rather continuity in making the latest traces available. At this writing, daily traces at one sample point have added up to the record of more than a year.

2 Related Work

Packet Monitoring

Packet capturing was brought with the advent of Ethernet. The first personal computer, Xerox Alto, already had programs to monitor Ethernet. As Ethernet came into wide use, dedicated network monitors became indispensable to developers and operators. The CMU/Stanford enet packet filter is the first UNIX based packet filter developed in 1980 [MRA87]. It eventually evolved into the Ultrix Packet Filter at DEC, NIT under SunOS, and BPF.

Userland programs that prints the headers of packets appeared with UNIX workstation. Sun implemented NIT (Network Interface Tap) to capture packets and *etherfind* to print packet headers. The advantage of UNIX-based monitoring tools is that users can use other software tools available on UNIX for manipulating and analyzing packet traces.

tcpdump [JLM89] is probably the most popular packet capturing tool in the UNIX community. *tcpdump* first appeared in 1989 and merged into BSD Net Release2 in 1991. *tcpdump* is based on a powerful filtering mechanism, the BSD packet filter (BPF) [MJ93]. The packet capturing and filtering facilities of *tcpdump* are implemented in a separate library, *pcap* [JLM94]. The *pcap* library became independent from *tcpdump* in 1994, and there are a wide range of network monitoring or analysis tools which integrate the *pcap* library. In 1999, tcpdump.org [tcp99] was organized by volunteers to maintain the *tcpdump* code.

High-performance monitoring systems are explored by OC3MON [ACTW96] and its successors that are based on a PC hardware but exclusively for ATM . CoralReef [CAI99] is a package developed at CAIDA to analyze the output of OCxMON.

Packet monitoring techniques have been used to gather long-term statistics. A pioneering work is *statspy* [Bra88] in the NNStat package developed at ISI. As SNMP becomes widely available, network statistics tools are geared toward SNMP. MRTG [Oet96] and its successor RRDtool [Oet99] are popular tools to collect traffic counters from routers through SNMP. More recently, cflowd [McR99] is developed at CAIDA to make use of Cisco's NetFlow [Cis98] that exports statistics of flow cache entries.

Traffic Archive

The Internet Traffic Archive (ITA) [DMPS95] was created in 1995 by Danzig *et al.* to promote research on network analysis. ITA has several traces studied in published papers as well as unstudied traces. ITA is an important step towards open traffic data sets because research based on open data sets can be confirmed or further analyzed by someone else, which leads to deeper studies.

There are several different formats in the ITA archives but the majority of the available traces are in the *tcpdump* ascii output format. A set of shell scripts, called *sanitize*, are written by Paxson and used to scramble addresses in the *tcpdump* ascii output format to provide anonymity to network users.

Our traffic repository was motivated in part by the effort at ITA. We employ automatic traffic sampling at regular intervals since the archives at ITA are not updated much. We also thought that the *tcpdump* raw format is preferable to the ascii format because the raw format has more information, and powerful tools are available to manipulate the raw format. Also, it would be useful for developers of tools which handle traces in the *tcpdump* raw format.

3 Motivation

WIDE installed several traffic sampling points within the backbone since traffic data has been essential to both network research and operation. However, traffic information tend to be confined to a small set of members, and it is difficult to share detailed information without a framework to support sharing. This leads to the idea of a traffic trace repository in which detailed traffic traces are archived and easily accessible to everyone.

In order to build a traffic trace repository and make good use of traces, we had to solve two problems. One is to create a safety measure for handling traces that include privacy information. The other is automation of the trace acquisition process.

Traffic traces include private information of the network users. Special care is needed to handle traces, and thus, only limited members are allowed to handle raw traces. Still, there is always a risk of accidents when we handle raw traces. Hence, even if traces are available only for limited members, it is important to make traces safe enough to prevent possible accidents. On the other hand, if traces are made free from user privacy, we can make the traces open to the public since WIDE does not need to worry about its impact to stock prices.

Automation of the maintenance process is the other important factor. Collecting traffic traces in a long term needs perseverance, and cannot be achieved unless most of the work are automated. Not only automation of acquisition but also automation of summarization and visualization are essential to maintaining the repository because, if no feedback is given, people tend to run out of energy.

There are strong concerns about security and privacy with regard to making traces publicly available. After a long discussion, we have reached a conclusion that the benefits outweigh the risks. Or, at least, it is worth a challenge.

4 Privacy Issues

Traffic traces contain privacy information including network addresses and application payload so that it is important to understand issues involved in user privacy.

There are two major issues involving user privacy.

Removing user data: User private data must be removed from traces. Traffic traces should have only protocol headers, Protocol payload which contains user data should be removed.

Providing anonymity: IP address is unique and can be used to identify a user, and thus, addresses should be scrambled to provide anonymity to users.

There are a wide variety of research purposes that have different requirements for traces. No single method will satisfy all the requirements and still keep user privacy. We are trying to provide traces which can be used for a wide range of research. For research which has specific requirements, our traces will provide a starting point, and can be used to narrow down its requirements. Once specific requirements become clear, it is easy to find a specific method to meet the requirements.

4.1 Removal of Payload

As a general rule, we should remove the payload of TCP or UDP that contains users' private information. If another protocol header exists on top of a TCP or UDP header and the inner header does not contain user private information, the inner header may be maintained. If it is difficult to judge whether a header contains user private information or not, the header should be removed as a precaution.

Once protocol payload is removed, the risk of jeopardizing user privacy is considerably reduced. It would be safe enough for use within a closed group. However, in order to make traces open to the public, we need a further level of security. That is, we need to provide anonymity to network users.

4.2 Address Scrambling

We should provide anonymity to individuals and organizations by scrambling source and destination addresses in IP headers. IP addresses, however, have hierarchical structures and special addresses such as broadcast addresses, multicast addresses and private addresses. It is not easy to provide anonymity but still keeping the structures and special meanings. We should chose an appropriate method according to the importance of anonymity in traces and the purpose of the data set.

Address Scrambling Methods

Address scrambling maps one IP address to another IP address. There are a number of methods to scramble addresses.

1. the sequential numbering method maps each IP address occurrence to a sequential number. Although this method is easy to understand, it is difficult to preserve other meanings of addresses.

2. the hash method maps an IP address to another IP address using a hash function in order to provide random mapping. It is also possible to preserve the common address prefix between 2 addresses by maintaining an ordered tree of addresses similar to a routing table. In this method, if 2 IP addresses have a common address prefix, they are mapped to addresses with a common address prefix of the same length. Note that, although it preserves routing information, this method has a risk of being reverse-engineered. For example, one can use a well-known server's address as a clue to de-scramble the address prefix [Ylo96]. The impact of this threat, however, depends on the importance of hiding the network topology.

There are several choices regarding address consistency between two or more data sets.

1. all occurrences of an address are to be mapped to a single address within a data set.

2. all occurrences of an address are to be mapped to a single address across different data sets.

Longer consistency is convenient for analysis but it also makes reverse-engineering easier.

Address Issues

non-unique addresses Addresses not containing user identifiers may be left without scrambling. Those addresses include broadcast addresses, multicast addresses, and private addresses. In the case of IPv6, link-local addresses and site-local addresses could contain unique interface identifier (e.g., MAC address). A solicited-node multicast address contain lower bits of the global address. Therefore, these IPv6 addresses should be scrambled as well.

addresses in upper layers IP addresses could be contained in an upper protocol message. For instance, ICMP and DNS contain IP addresses in their protocol payload. These addresses must be scrambled in the same manner, or removed.

MAC addresses Link-layer headers (e.g., Ethernet headers) contain MAC addresses. A MAC address contains vendor and model information which could be part of user privacy or lead to a security hole. However, traces from backbone networks do not contain MAC addresses of user nodes since MAC addresses recorded in the trace are only from local nodes on the same segment.

IP/TCP options IP options can contain IP addresses. Addresses in IP options should be scrambled in the same manner. Otherwise, IP options should be replaced by NOP options, or removed.

On the other hand, TCP options do not contain privacy information. TCP options carry useful information to analyze TCP behaviors so that TCP options may be preserved.

5 Methods

We use several tools to automatically maintain the traffic repository. The details of these tools are described later in this section. New trace data is collected from sampling

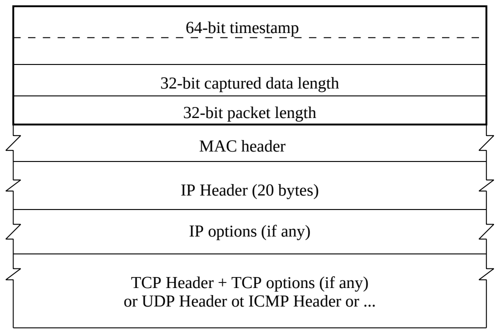
64-bit timestamp
32-bit captured data length
32-bit packet length
MAC header
IP Header (20 bytes)
IP options (if any)
TCP Header + TCP options (if any)
or UDP Header ot ICMP Header or ...

Figure 1: Pcap header format

points to the repository during the night. A web page for the new trace is automatically created.

At a sampling node, a script is invoked from *cron* to run *tcpdump* and compress the trace. The obtained raw trace file is placed under a certain directory.

At the repository node, another script is invoked from *cron* to fetch the raw trace and process it. The script copies the compressed raw trace from the sampling point over a secure session using *scp*. Then, the script uncompresses the trace and invokes *tcpdpriv* to remove privacy information from the trace. The trace is fed into *tcpdstat* to get a summary output. The script creates a web page for the trace, and updates the index page to include the newly created page. Finally, the script compresses the trace data again, and place it for the ftp service.

5.1 tcpdump

We use *tcpdump* to obtain traffic traces because *tcpdump* is widely used, and installed as part of the default tools on many systems. In addition, there are many tools that integrates the *pcap* library and be able to read *tcpdump* output files. Those tools include *tcptrace*, *tcpslice*, *tcpdstat* and *ttt*.

tcpdump, by default, puts the network interface into *promiscuous mode* to capture every packet going across the wire. In the BSD-derived kernel, BPF is implemented as a packet capture mechanism. When BPF is enabled, the network driver in the kernel passes both sending and receiving data-link level frames to BPF. BPF performs packet filtering if necessary, adds timestamp, and copies the fixed length from the head of the frame into the store buffer. *tcpdump* in the user space can read multiple frames in a single read from the store buffer in the kernel in an efficient manner. *tcpdump*, by default, prints the header information of each packet in a text format. With *-w* option, *tcpdump* writes out the packet frames into a specified file. With *-r* option, *tcpdump* reads from a saved file instead of a network interface to replay a saved file. The *pcap* library is used to read or write data in the raw format. Thus, it is easy to write a program to read or write packets in the *tcpdump* format.

Figure 1 shows the format of raw *tcpdump* output. In the BSD systems, the kernel uses *microtime()* for timestamp; the precision of the timestamp is 1 usec on the PC architecture. Timestamp is taken when a packet is passed to BPF from the network driver so that it is the time that the driver sees that packet.

5.2 tcpdpriv

We use *tcpdpriv* to remove user data and scramble addresses. *tcpdpriv* was developed by Greg Minshall at Ipsilon Networks in 1996. *tcpdpriv* removes privacy information in a raw *tcpdump* output. *tcpdpriv* uses the *pcap* library to read and write *tcpdump* output files. *tcpdpriv* removes the payload of TCP and UDP, and the entire IP payload for other protocols. *tcpdpriv* implements several address scrambling methods; the sequential numbering method and its variants, and a hash method with preserving address prefix.

However, the original *tcpdpriv* lacks several features we need:

- it does not support IPv6.
- it does not preserve TCP options that are essential to analyzing TCP behaviors.
- we also want to preserve other protocols such as ICMP, ARP and DNS.

Thus, we have modified the original *tcpdpriv* to support these features. The default settings are also changed to meet our requirements since the options seem to be too complex and a mistake of option selection could be fatal to user privacy.

5.3 tcpdstat

We developed *tcpdstat* to get summary information of a *tcpdump* file. *tcpdstat* reads a *tcpdump* file using the *pcap* library and prints the statistics of a trace. The output includes the number of packets, the average rate and its standard deviation, the number of unique source and destination address pairs, and the breakdown of protocols.

tcpdstat is intended to provide a rough idea of the trace content. The output can be easily converted to a HTTP format. It also provides helpful information to find anomaly in a trace. For example, if the traffic volume of ICMP is unusually large, or if the traffic volume of a specific address pair is unusually large, it could be a sign of some form of a DoS attack.

5.4 Other Tools

There are other tools that are not used to create the traffic repository but can read *tcpdump* files and useful for analyzing traces afterwards.

tcpslice by Vern Paxon extracts portions of a trace. *tcptrace* by Shawn Ostermann produces detailed information about each TCP connection in a trace. *tracelook* by Greg Minshall provides *xgraph* plots of TCP connections in a trace. *flstats* also by Minshall prints flow statistics. *ethereal* by Gerald Combs is a traffic analyzer with a graphical user interface. *ethereal* uses the *pcap* library, and thus, can replay a *tcpdump* file. Our *ttt* (Tele Traffic Tapper) tool displays composition graphs of protocols and host addresses in real time. *ttt* can replay a trace file at a given speed so that it is possible to replay a 1-hour trace in 1 minute.

6 Current Status

Currently, we are collecting daily-traces from the following sampling points.

trans-pacific is a 1.5Mbps T1 line, one of the several international links of WIDE. The sampling point is on an Ethernet segment one hop before the T1 line. The incoming traffic (from U.S. to Japan) of this link is fairly congested.

6Bone is located on a FastEthernet segment connected to NXPIXP-6 (An IPv6 internet exchange point in Tokyo) [WID99]. The segment located at an AS boundary, and the traffic includes only native IPv6 and does not include IPv4 except tunneled IPv4 over IPv6. Because NXPIXP-6 is built on a FastEthernet switch, only the traffic crossing a single port of the switch can be captured.

Traces are sampled at a fixed time of day. This is obviously not desirable and we need to find a better sampling method.

We started daily data collection at the *trans-pacific* point in February 1999. Since WIDE has a number of connections to the Internet exchange points, the return path of a session does not necessarily go through the same link.

The maximum size of each trace file is limited to about 100M bytes (about 40MB when compressed). We believe 100MB is an appropriate size for handling on a commodity PC as well as for fetching over the network, still it has enough information for statistical analysis.

Figure 2 is a sample output of *tcpdstat* from the *trans-pacific* point on February 12, 2000. This 1-hour-long trace contains about 2 million packets, the number of unique address pairs is about 56K. HTTP is dominant in the trace, 70% of the total packets and 62% of the total bytes.

Among the collected traces, some data sets contain traces of DoS attacks such as *portscan* and *smurf*. These traces could be useful for developing tools to detect such attacks.

The *6bone* point has been added in January 2000. The traffic volume of the *6bone* point is still low; the average rate is around 100Kbps and the majority of traffic is BGP and ICMPv6. However, we expect IPv6 traffic will increase in a few years as major router and OS vendors have started shipping IPv6 support in their base systems. Our intention is to record the evolution of IPv6 traffic in a long term.

Figure 3 is the output of *tcpdstat* from the *6Bone* point on the same day. This 3.5-hour long trace contains about 200K packets, the number of unique address pairs is about 270.

We expect that IPv6 traces will be useful for developing tools to support IPv6 since IPv6 traffic traces, especially on a backbone link, are not widely available.

Although we started collecting traces and made them available, we have not studied the traces thoroughly. Rather, one of the purposes of our open traffic repository is to leave analysis to those who are interested in doing it.

If other organizations start building similar but possibly closed traffic repositories, it would be possible to share experiences and development of tools. Especially, there are demands to develop a counter measure against the ever-growing threat of DoS attacks.

7 Future Work

Our focus at this moment is long-term data collection. So far, we have set sampling points only on relatively slow connections. Data collection from faster links is an obvious direction, but we have limited storage capacity and network capacity.

As for high-performance packet capturing, we can benefit from advanced research such as OC3MON [ACTW96]. OC3MON uses a DOS-based capturing tool to monopolize CPU, and takes advantage of the processor on the ATM card for offloading.

However, today's commodity PC is already quite powerful: Gigabit Ethernet is about 125MB/sec. The bus bandwidth of 32bit PCI at 33MHz is 132MB/sec, and 64bit PCI at 66MHz is 528MB/sec. The disk interface is getting faster as well. Ultra160 SCSI provides 160MB/sec. A single high-end disk has sustained rate of about 30MB/sec but disks can be used in parallel so that 4 disks provide about 120MB/sec. CPU power itself seems to be catching up.

There are also issues to run *tcpdump* on non-realtime UNIX; preemption could affect reliable data collection, resource contention and kernel-user data copy could affect performance, network cards and drivers are not designed to obtain precise timestamp. Still, if the system is correctly tuned, a commodity PC seems to be capable of capturing packets even at a gigabit network.

8 Conclusion

We have presented the WIDE traffic repository, an ongoing effort to create archives of *tcpdump* files collected at several points within the WIDE backbone. Our attempt is a challenge to the legitimacy of concerns about revealing detailed traces for privacy and security reasons. We hope our repository will be useful for traffic analysis and for development of tools.

9 Availability

The WIDE traffic repository is at http://tracer.csl.sony.co.jp/mawi/. The traffic traces along with the tools we use can be downloaded from there. The traffic traces can be used only for research purposes. Actions that trespass upon users' privacy are prohibited.

References

[ACTW96] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder. Oc3mon: Flexible, affordable, high performance statistics collection. In *Proceedings of LISA X*, pages 97–112, Chicago, IL, September 1996.

[Bra88] R. T. Braden. A pseudo-machine for packet monitoring and statistics. In *Proceedings of SIGCOMM '88 Symposium*, pages 200–209, Stanford, California, August 1988.

[CAI99] Coralreef. http://www.caida.org/Tools/CoralReef/, 1999.

[Cis98] Cisco NetFlow. http://www.cisco.com/warp/public/732/netflow/, 1998.

[DMPS95] Peter Danzig, Jeff Mogul, Vern Paxson, and Mike Schwartz. The Internet Traffic Archive. http://ita.ee.lbl.gov/, 1995.

[JLM89] Van Jacobson, Craig Leres, and Steve McCanne. tcpdump. ftp://ftp.ee.lbl.gov/, 1989.

[JLM94] Van Jacobson, Craig Leres, and Steve McCanne. libpcap. ftp://ftp.ee.lbl.gov/, 1994.

[KMKA99] Akira KATO, Jun MURAI, Satoshi KATSUNO, and Tohru ASAMI. An internet traffic data repository: The architecture and the design policy. In *Proceedings of SOSP*, San Jose, CA, June 1999.

[McR99] Daniel McRobb. cflowd. http://www.caida.org/Tools/Cflowd/, 1999.

[MJ93] Steven McCanne and Van Jacobson. A BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of USENIX Winter Conference*, pages 259–269, San Diego, California, January 1993. Usenix.

[MRA87] J. C. Mogul, R. F. Rashid, and M. J. Accetta. The packet filter: An efficient mechanism for user-level network code. In *Proceedings of SOSP*, pages 39–51, Austin, TX, November 1987.

[Oet96] Tobias Oetiker. MRTG: Multi Router Traffic Grapher. http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html, 1996.

[Oet99] Tobias Oetiker. RRDtool. http://www.caida.org/tools/utilities/rrdtool/, 1999.

[tcp99] tcpdump.org. http://www.tcpdump.org/, 1999.

[WID99] WIDE IPv6 Internet Exchange Point in Tokyo. http://www.wide.ad.jp/nspixp6/, 1999.

[Ylo96] Tatu Ylonen. Thoughts on how to mount an attack on tcpdpriv's "-a50" option... included with the tcpdpriv source distribution, 1996.

```
DumpFile: 200002121359.dump
FileSize: 140.35MB
Id: 200002121359
StartTime: Sat Feb 12 13:59:00 2000
EndTime: Sat Feb 12 15:06:29 2000
TotalTime: 4048.47 seconds
TotalCapSize: 108.37MB CapLen: 76 bytes
# of packets: 2095754 (449.89MB)
AvgRate: 932.21Kbps stddev:312.68K

Packet Size Histogram (including MAC headers)
 [   32-   63]:    1315693
 [   64-  127]:     258761
 [  128-  255]:     121532
 [  256-  511]:     113037
 [  512- 1023]:     137300
 [ 1024- 2047]:     149431

IP flow (unique src/dst pair) Information
# of flows: 56157 (avg. 37.32 pkts/flow)
Top 10 big flow size (bytes/total in %):
8.0% 6.0% 4.8% 2.8% 2.6% 2.0% 1.4% 1.2% 0.8% 0.7%

Protocol Breakdown
     protocol           packets                bytes           bytes/pkt
------------------------------------------------------------------------
 total         2095754 (100.00%)       471744043 (100.00%)    225.10
 ip            2095736 (100.00%)       471743089 (100.00%)    225.10
  tcp          1768533 ( 84.39%)       400258906 ( 84.85%)    226.32
   http        1474686 ( 70.37%)       292981631 ( 62.11%)    198.67
   squid         42778 (  2.04%)        36118457 (  7.66%)    844.32
   smtp          74280 (  3.54%)        29130167 (  6.17%)    392.17
   nntp           1270 (  0.06%)          101659 (  0.02%)     80.05
   ftp           23779 (  1.13%)         7180413 (  1.52%)    301.96
   pop3           5601 (  0.27%)         2537763 (  0.54%)    453.09
   telnet          995 (  0.05%)           88678 (  0.02%)     89.12
   ssh            1950 (  0.09%)          230243 (  0.05%)    118.07
   dns            1169 (  0.06%)           94179 (  0.02%)     80.56
   bgp            6090 (  0.29%)          419164 (  0.09%)     68.83
   other        135935 (  6.49%)        31376552 (  6.65%)    230.82
  udp           264967 ( 12.64%)        62915121 ( 13.34%)    237.45
   dns          187377 (  8.94%)        29884111 (  6.33%)    159.49
   rip             135 (  0.01%)            8910 (  0.00%)     66.00
   other         77455 (  3.70%)        33022100 (  7.00%)    426.34
  icmp           51228 (  2.44%)         5609707 (  1.19%)    109.50
  igmp             801 (  0.04%)           48060 (  0.01%)     60.00
  ospf            8909 (  0.43%)         2283318 (  0.48%)    256.29
  ipip               3 (  0.00%)             246 (  0.00%)     82.00
  ip6             1295 (  0.06%)          627731 (  0.13%)    484.73
  frag             112 (  0.01%)          157111 (  0.03%)   1402.78

tcpdump file: 200002121359.dump.gz (45.94 MB)
```

Figure 2: sample output of tcpdstat at trans-pacific

```
DumpFile: 200002120900.dump
FileSize: 17.64MB
Id: 200002120900
StartTime: Sat Feb 12 09:00:00 2000
EndTime: Sat Feb 12 12:33:45 2000
TotalTime: 12825.20 seconds
TotalCapSize: 14.69MB CapLen: 94 bytes
# of packets: 193424 (48.84MB)
AvgRate: 40.31Kbps stddev:63.46K

Packet Size Histogram (including MAC headers)
 [   64-  127]:     100654
 [  128-  255]:      66924
 [  256-  511]:       1179
 [  512- 1023]:       2322
 [ 1024- 2047]:      22345

IP flow (unique src/dst pair) Information
# of flows: 270 (avg. 716.39 pkts/flow)
Top 10 big flow size (bytes/total in %):
53.9% 4.0% 3.8% 3.7% 3.6% 3.6% 3.1% 2.9% 2.9% 2.9%

Protocol Breakdown
     protocol           packets                bytes           bytes/pkt
------------------------------------------------------------------------
 total          193424 (100.00%)        51210692 (100.00%)    264.76
 ip6            193424 (100.00%)        51210692 (100.00%)    264.76
  tcp6          184430 ( 95.35%)        49453242 ( 96.57%)    268.14
   smtp            402 (  0.21%)           54893 (  0.11%)    136.55
   ftp           51229 ( 26.49%)        29569720 ( 57.74%)    577.21
   ssh              53 (  0.03%)            6820 (  0.01%)    128.68
   bgp          132476 ( 68.49%)        19798783 ( 38.66%)    149.45
   other           270 (  0.14%)           23026 (  0.04%)     85.28
  udp6             469 (  0.24%)           36610 (  0.07%)     78.06
   other           469 (  0.24%)           36610 (  0.07%)     78.06
  icmp6           7346 (  3.80%)         1489628 (  2.91%)    202.78
  ip4             1179 (  0.61%)          231212 (  0.45%)    196.11

tcpdump file: 200002120900.dump.gz (2.99 MB)
```

Figure 3: sample output of tcpdstat at 6bone