Linuxflow: A High Speed Backbone Measurement Facility

LI Zhichun, ZHANG Hui, YOU Yue, HE Tao
lizc@serv.edu.cn hzhang@cernet.edu.cn
China Education and Research Network Center (CERNET)
Tsinghua University, Beijing, P.R. China

***Abstract* - Both traffic metering which can measure high speed network and data aggregating which can reduce collected data size for fine-grained are difficulties in network passive measurement area. For the needs of network management and IP accounting of CERNET [15], this paper designs and implements a highly scalable performance measurement facility: Linuxflow, which now has become the core of passive network measurement system deployed on CERNET backbone.**

***Index Terms* - network measurement, traffic meter, high speed networking, flow, Linux**

I. INTRODUCTION

CERNET is a non-profit national foundation that provides telecommunication and network services to universities and other research institutions in China. It is constructed and operated by Tsinghua and other leading universities. Up to now CERNET has connected with 1000+ universities in China, and has above 2.5 Gigabit PoPs.

To support the large number of users that run various types of applications, CERNET has been using self-developed network applications for flow-based network traffic analysis. Its applications include: usage-based accounting and billing for "transatlantic" traffic, network per-protocol user usage monitoring, traffic characteristics analysis and data mining, the detection of anomalies such as attacks, unwanted routing asymmetries and various types of network abuses. All of these applications need a highly scalable performance measurement system; this results in the design and implementation of network passive measurement facility. We call it Linuxflow.

Since 1997, the number of active IP addresses in CERNET has been multiplied 10 times, as CERNET backbone has been upgraded from 64K to OC3 to OC48.

Historically, CERNET has used several kinds of measurement system. Figure 1 presents the measurement methods by means of which CERNET can consume the increasingly large number of traffic packets at network gateway during last 5 years, as well as their capabilities respectively.

This research was supported by the Hi-Tech Research and Development Program of China (863) and the National Natural Science Foundation Of China under agreement numbers: 2001AA112041, 60103005.

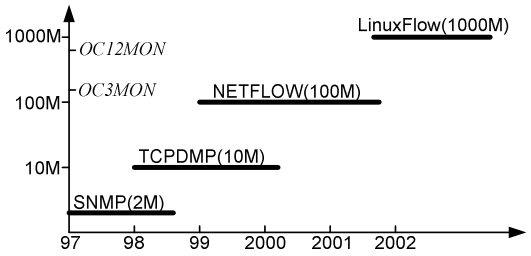


Figure 1. The using history of network passive measurement systems in CERNET

SNMP[5][6] is a standard network management protocol. Many network equipments support it and have its own extended MIB. But it only can get coarse-grained data. The extended MIB like CISCO IP Accounting table (1.3.6.1.4.9.2.4.7.1) can have more specific IP level information used to usage-billing system. But it can not fit high speed network.

LIBPCAP LIB is a common system-independent interface for user-level packet capture on UNIX system. TCPDUMP is the most famous program using the interface. TCPDUMP/LIBPCAP is very easy to use, but can not be used on high-speed network either [3].

Cisco NetFlow [8][9] is a flow-based network measurement mechanism. CERNET used it on our usage-based billing system for a long time. But if the network worms such as code-red break out, the NDE & NFC combination will suffer severe performance penalty. And Cisco GSR 12000 series can provide only sampling NetFlow which can't apply to billing system.

Some hardware network measurement devices such as OCXmon [10] and DAG [7] card can be used on very high speed network backbone link. But these systems are more expensive than software solution. We haven't tested this kind of devices in our network.

Since these techniques cannot satisfy our application's needs, especially network usage-based billing system. We design and implement our passive measurement system based on Linuxflow. This facility, as the name self-explanation, is based on Linux system. Based on the implementation of network protocol stack in Linux kernel [12][13], we have designed a special standalone network packet capture protocol stack, which dedicate to packet capture. This not only can decrease the times of context switch between kernel space and user space, but can insert easily implemented filter and let kernel only copy the information interesting us to user level, which provides much

more efficiency than LIBPCAP. Furthermore, we implement a flow-based data aggregating program packet-to-flow, the idea behind which is derived from IETF RTFM [1][2] and NetFlow [8], i.e. flow-based traffic data aggregation.

In section 2, we look at the traffic collection network environment of Linuxflow. Section 3 discusses the design and implementation details of Linuxflow. In section 4 we consider some experiments and practical results from the system, before concluding this paper with a final analysis in section 5.

II. REQUIRED ENVIRONMENT

A. *Methods of sniffing*

To collect traffic, we require a network interface upon which a copy of all relevant network traffic is available. This can be done using hub, port mirroring, or a tap mechanism.

Insert a hub in network link, all ports of the hub can get a copy of data. This method only can be used in Megabit PoP including 10M and 100M.

Port or interface mirroring is a technique by which the traffic from one or more interfaces on a network switch (the mirrored interfaces) can be copied to another port (the mirroring interface). In theory, this provides a mechanism to transparently observe traffic passing over the mirrored interfaces by observing traffic over the mirroring interface.

A tap mechanism is a piece of hardware that takes a single network input and duplicates it to transparently produce two identical outputs. This can be thought of as an optical splitter. This hardware works at the physical level (electrons or photons "on the wire") by splitting a physical signal and possibly enhancing it.

B. *Traffic collection network environment*

As shown in Figure 2, we can configure more than one Linuxflow servers to sniff different network links simultaneously and collect fine-gained flow data, and then send resulting data to a centralized flow collection and storage server via LFEP (LinuxFlow Export Protocol) protocol. This sever in CERNET has a 1.2TB RAID connected to it to store flow data. This mechanism can support different network analysis applications, which can get data from flow storage server for long term analysis, or can get flow data use LFEP protocol for real-time monitoring and anomaly detecting.

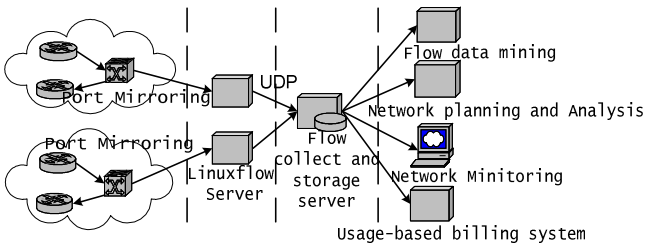


Figure 2. Linuxflow traffic collect and analysis environment

III. LINUXFLOW DESIGN

A. *Linuxflow structure*

Figure 3 presents our standalone network packet capture protocol stack based on Linux 2.4.x kernel, and user space multi-thread flow aggregation program. By using several kernel modules, the special standalone network packet capture protocol stack implements the protocol stack's hierarchy. User space daemon get data from kernel, aggregate packets to flows, assemble flows to LEFP packets and send to other collector machine.

B. *Packet capture protocol stack*

Our standalone packet capture protocol stack includes three modules in figure 2. In these modules we have used SMP and caching techniques, such as APIC interrupt control and multi-tasks, so they can easily and efficiently run on high-performance Intel PC server.

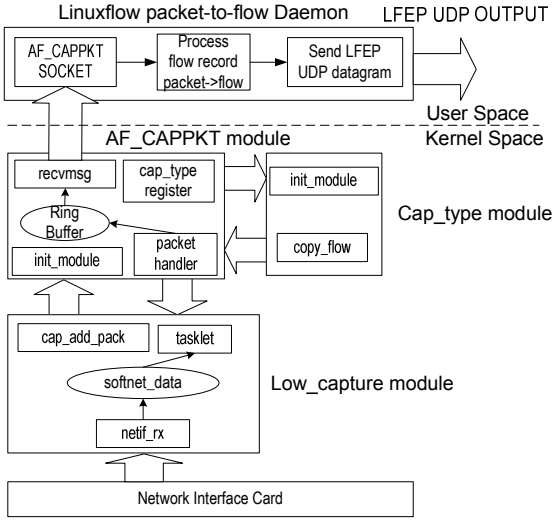


Figure 3. Linuxflow Network Measurement System Structure

1) *Low_capture module*

As we known, when a packet arrives at network interface card, the interface triggers a hardware interrupt. In the ISR of this interrupt, kernel calls kernel symbol *netif_rx* to make a packet *skbuff* and send it to network protocol stack. In this module we redefine the *netif_rx* kernel symbol. The packet interface card received will not be sent to Linux standard stack, but our special standalone packet capture stack.

Also, in this module we define our tasklet to handle the packet sent by *netif_rx*, and define the *cap_add_pack* hook function to bind a type of socket AF_CAPPKT.

2) *AF_CAPPKT module*

This module registers AF_CAPPKT protocol family to Linux kernel, and implements the AF_CAPPKT socket. It uses ring buffer like BPF and filters out the packet header and such information as time-stamp from packet *skbbuff* data structure to

fill in the ring buffer. By calling the system call *read*, *recvmsg* etc., user space program can get the contents of buffer as system call return once the buffer is full.

3) Cap_type module

This module provides us with the ability to implement different *cap_type* hook function to collect one or more packet header fields and other packet properties such as time-stamp. For example, we can define one *cap_type* to collect 5-tuple (source address, destination address, source port, destination port, and protocol type), and anther to collect all IP and TCP header fields. This can collect different information from a packet flexibly. The less information is collected from a packet, the more data record a buffer will return to user space. In this way, the program will proceed more quickly.

4) API in User Space

A program in user space that wants capture packet information only needs to call socket function like this.

sock = socket (AF_CAPPKT, CAP_COPY_FLOW, ntohs(ETH_P_IP)) ;

It can use *read* or *recvmsg* to read buffer which contain packet information structure defined in *cap_type*.

C. Multi-thread flow aggregation program

Multi-threading provides program with the ability to run on different architectures, ranging from microprocessor to mainframe to supercomputer, and the ability to buffer more data from kernel. This decreases the possibility of losing packet data from queueing theory's point of view [4].

The program has three threads: one for reading packet data buffer from kernel, another one for processing packet data to flow record, the last one for assembling flow record into LEFP UDP packet and sending it to other machines for further analysis.

The flow definitions in [14] [11] [8] [2] differs a little. In this paper we identify the flow using the classical 5-tuple (source address, destination address, source port, destination port, and protocol type) definition and flow is bi-directional packets stream. This definition is much more like the *streams* concept defined in [1], which is individual IP session (e.g. TCP or UDP) between ports on pairs of hosts.

A major issue is to determine when a flow has terminated. We defined two types of timeout for a flow, *active timeout* and *inactive timeout*. If a flow meets following conditions we think of it as being terminated.

- Flows which have been idle for a specified time (*inactive timeout*) are expired and removed from the flow table.
- Long lived flows are reset and exported from the flow table, when they have been active for a specified time (*active timeout*). The consecutive packets of a long lived flow which has been exported will make up a flow with a *cont* flag, this can notify collector "I am not a new one".
- TCP connections which have reached the end of byte stream (FIN) or which have been reset (RST)

In processing thread hashing algorithm is used to aggregate packets information to flows. When flows are expired, they will be exported to *send buffer*, where sending thread assembles flows to LEFP packet. We defined the LEFP, i.e. LinuxFlow Export Protocol to send the flow records to destination machine for analysis. The LEFP uses UDP protocol, and the packet format is shown as figure 4.

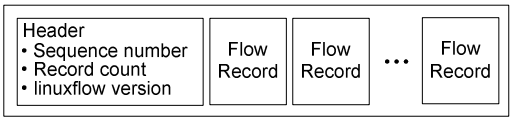


Figure 4 Linuxflow Export UDP Datagram Format

IV. PERFORMANCE EVALUATION

A. Linuxflow performance and accuracy test

1) Experimental environment

We use CERNET-CHINANET (china telecom) Gigabit interconnection backbone link as our test link. This link interconnected the biggest research network and biggest commercial network in China.

The test used an Intel server features as follows.

Processor	PIII XEON 700Mhz × 4
Memory	16GB DRAM
Accessory	64-bit/64MHz
Disk	35GB SCSI disk × 2
Network Card	Intel 1000BaseSX × 2

Table 1. Parts of the test machine

Linuxflow measurement system runs on RedHat 6.2 with self-configured kernel 2.4.17.

2) Experimental Dimensions

In this test, we have studied the relationship between the CPU load and the network packet rate/network bandwidth. From experiments we find the CPU load is mainly depended on the packet rate. Same packet rate with different network bandwidths (this means the packet length is different) leads to the almost same CPU load.

In accuracy experiment, we use sampling test method. With different network bandwidths, we transport same traffic between a point in CERNET and another point in CHINANET. By comparing what we send with what our measurement system gets, we can calculate the collecting ratio.

3) Experimental Results

The experimental result is shown in figure 5

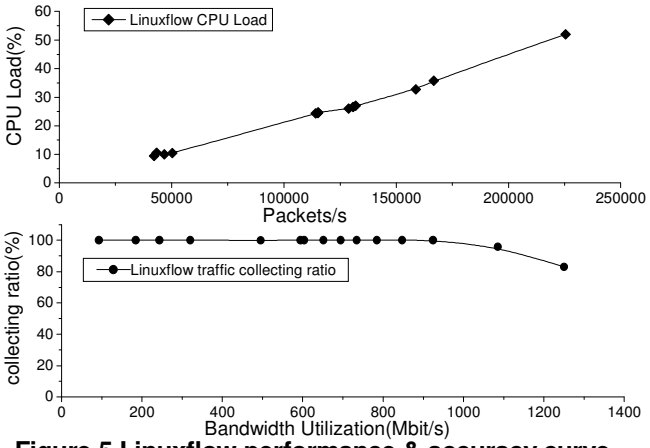


Figure 5 Linuxflow performance & accuracy curve

B. Compared with TCPDUMP and LIBPCAP API program.

TCPDUMP/LIBPCAP can't aggregate packet information to flow. So we only compare AF_CAPPKT to TCPDUMP/LIBPCAP.

Experiment is taken on CERNET-Internet OC3 international link. The test machine is a dual PIII 933MHz processors with 2GB DRAM, and a 3COM Gigabit Ethernet card to listen both direction of the link.

The packet loss ratio of TCPDUMP run on standard mode is more than 20%, so it can not pass the test. So we write a program based on LIBPCAP in our own. This test program is compared with another test program written based on AF_CAPPKT. Both test programs only do one thing: get each packet's IP header. Since LIBPCAP is a portable lib, it can run on different variants of UNIX. So we test the program on Linux and FreeBSD.

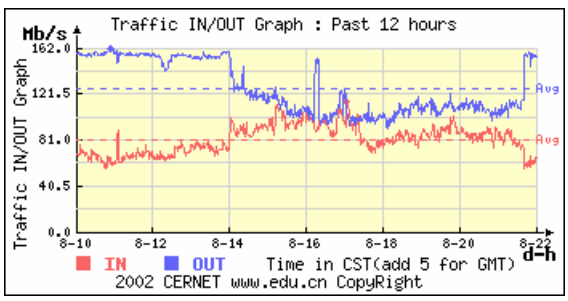
The experimental result is shown in table 2.

Program	**lpftest**	pcaptest	pcaptest
OS Platform	**Linux 2.4.17**	Linux 2.4.17	FreeBSD4.4
Accuracy	**99.8%**	99.6%	99.9%
CPU Load	**1%**	16%	15%

Table 2: testing results

C. Graphical presentation on CERNET

Graphs such as shown in figure 6 are generated hourly using our PHP graph interface.



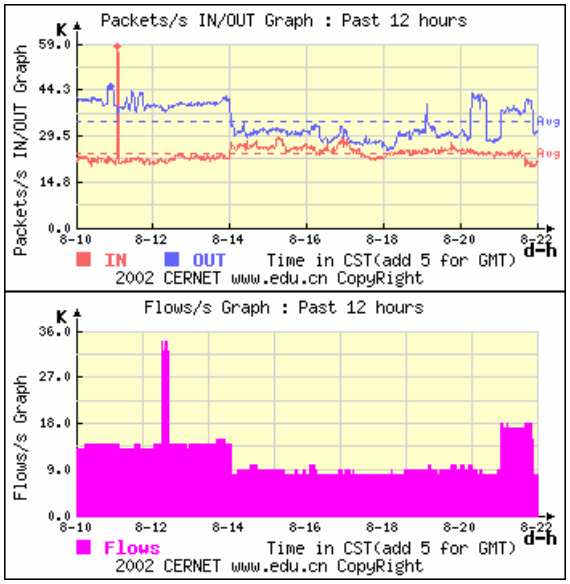


Figure 6. Web monitoring graph

V. CONCLUSION

We have designed and implemented a flow-based highly scalable performance measurement facility, Linuxflow. It includes a special standalone packet capture protocol stack and a flow-based multi-thread aggregation program.

We have proven its capability of handling gigabit network backbone not only by special tests, but also by the fact that it has been used on CERNET backbone.

Flow based network measurement system has provided significant benefit to CERNET. It enabled the usage-based billing scheme; it let us know what traffic transported on our network, and detect anomalies on our network.

In the future we may make Linuxflow more suitable to the current standard [14]. And we plan to use clustering techniques to make it more scalable and powerful.

Moreover, we are building our IDS systems and traffic data mining systems based on this measurement facility.

REFERENCES

[1] Nevil Brownlee, Margaret Murray. Streams, Flows and Torrents PAM2001

[2] Nevil Brownlee, Network Management and realtime Traffic Flow Measurement, pp 223-227, Journal of Network and Systems Management, Vol 6, No 2, 1998

[3] Eric Weigle, Wu-chun Feng. TICKETing High-Speed Traffic with Commodity Software and Hardware PAM2002

[4] L. Kleinrock, Queueing Systems, vol.1 & vol.2, John-Wiley, 1975/1976

[5] Case JD,Fedor M,Schoffstall M Letal. Simple Network managerment protocol(SNMP) RFC 1157 , 1990. http://www.ietf.org/rfc/rfc1157.txt

[6] J. Case,K. McCloghrie,M. Rose,S. Waldbusser Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2) 1993. http://www.ietf.org/rfc/rfc1450.txt

[7] The DAG project, http://dag.cs.waikato.ac.nz.

[8] White Paper NetFlow Services and Applications Cisco Corp. 2000 http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm
[9] NetFlow Services Solutions Guide Cisco Corp. 2001 http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm
[10] Joel Apisdorf,k claffy,Kevin Thompson. OC3MON: Flexible, Affordable, High-Performance Statistics Collection http://www.isoc.org/inet97
[11] K.C. Claffy, H.W. Braun, G.C. Polyzos, A parameterizable methodology for Internet traffic flow profiling, IEEE JSAC 1997
[12] Bovet, Daniel P., and Marco Cesati. Understanding the Linux Kernel. O'Reilly Press 2000
[13] Alessandro Rubini Linux Device Drivers, 2nd Edition O'Reilly Press 2001
[14] Internet-Drafts: Architecture Model for IP Flow Information Export
[15] CERNET www.edu.cn