A flexible parallel architecture for high-performance network traffic processing

Yanchao Jing
School of Shanghai
Jiao Tong University
Email:13541333146@163.com

Guangtao Xue
School of Shanghai
Jiao Tong University
Email: homer@thesimpsons.com

ShiyouQian
School of Shanghai
Jiao Tong University
Email:

***Abstract*—Nowadays,the deep packet inspection(DPI) becomes a crucial technique to monitor the security of network which needs the completed information of network traffic packets.This technique claim an extremely strict requirement on the functionality and performance of a monitor subsystem.Therefore, a high-performance easy-use network traffic capture library is extremely required to help to build some attractive DPI applications.**

To feed this demand,this paper proposes a flexible parallel network library(noff) which has ability to process the network traffic in multiple Gbps speed.With several innovative software techniques,noff has remarkably optimized the packets delivery procedure and the streams reassembly procedure compared with some traditional methods.Supporting two kinds of parallel processing on multi-core architectures, minimizing memory allocation and dropping silent streams effectively are the main features of noff.Some well-designed experimental evaluations indicate that noff can perform better than other software libraries in streams reassembly and http parser on single core, and can process much higher traffic loads when using multi-core platform.

I. Introduction

Nowadays the demand for quality of service (Qos) and network security including policy-based routing,firewall and virtual private network(VPN) is ever-increasing [1],In order to satisfy the requirement,An in-depth understanding of the Internet traffic profile is needed.Accordingly, the deep packet inspection(DPI) becomes a crucial hot technology [2].In a nutshell,a DPI system Firstly has to collect packets from the network interface cards(NIC),then to create a data structure to represent the incoming packets as network flows and then to forward or store the received packets for further processing.So it is extremely necessary to capture all exchanged packets between end-systems on the monitored link in very high speed links (e.g. multiple Gbps). [2].Many DPI applications deeply rely on information from the network packets.Therefore a high-performance easy-use network traffic capture library is extremely required.

Recently, For high-performance,enormous relative theoretical methods and tools have been proposed such as zero-copy technology, multi-core capture,multi-thread method and FPGA tools [3] and so on. This paper mainly focuses on software optimization. The zero-copy and multi-core technology can help avoid sparse sampling [4].A renowned tool to accomplish zero-copy is PF_RING [5] [6].The PF_RING kernel module runs as a software interrupt handler that stores incoming packets to a memory-mapped buffer,shared with user-level stub.For multi-core technology,there are also various multi-thread proposed methods that can improve the entire performance. The key of multi-thread is to make the best of multi-core performance.

Although there are many kinds of theoretical technologies, there is hardly a flexible high-performance network traffic library. Under such conditions, developing a strong DPI application becomes a huge challenge.

To overcome this difficulty, in this paper we present a parallel network traffic library(noff),a high-performance passive network traffic monitoring framework.Noff provides the high-level functions needed by many monitoring applications and provide a quite flexible framework to adjust the system structure .Application developers can easily create magic high-level applications based on noff.

To acquire aggressive performance optimization, we use a flexible multi-thread structure which can make the best of the multi-core performance to achieve a high throughput.

To reduce the overhead of unnecessary memory copying.Noff uses the PF_RING as underlying kernel component. The subzero packet copy scheme of PF_RING can contribute to reduce the total overhead and noff also uses the TCMalloc [8] [10] framework to reduce the memory allocation frequency.Meanwhile,noff adopts some novel techniques to reduce the overhead of thread synchronization.

To accommodate heavy loads, noff introduces the notion of Least Recently Used (LRU) stream management algorithm.Under heavy load,traditional monitoring systems usually drop arriving packets in a random way, which would severely affect the following stream reassembly process.Noff always drop the dead stream which has no data exchange during a long time.This method can relatively reduce the impacts from dropping streams.

Noff also provides a flexible and expressive application programming interface that allows programmers to develop many kinds of interesting traffic applications.Our design introduces two novel features:(I) Noff uses the C++ bind and function features to design callback interfaces between two modules.Programmers can flexibly add new modules or change modules just through main configure file. And(II)it offers more control for tolerating packet loss under high load throughput. And noff also supports single queue NIC's multi-thread scaling and multiple queue based on RSS(receive-side scaling) [11] transparent parallelization of stream processing

simultaneously.

We have evaluated noff in a 10GbE environment using real campus traffic and showed that it outperforms existing alternatives like Libnids and Snort's [7] stream5 processor in a variety of scopes.For example,our results demonstrate that noff can capture and deliver all streams with lower CPU utilization. The main contributions of this paper are:

1.We propose noff, which is an universal network traffic library.It provides a flexible and aggressive software framework to offer programmers an easy-use platform to develop network plug-in compartments.

2.We introduce some applied software techniques to optimize traditional products and approaches.Through these techniques,noff can deliver transport-layer streams for higher traffic rates than previous approaches.

3.We introduce Least Recently Used (LRU), an innovate algorithm that enables a graceful adaptation to overload conditions by dropping packets of dead streams,and favoring streams that exchange data frequently.

II. SOFTWARE FEATURE

The design of noff focuses attention on two key objectives: structure flexibility and runtime performance.In this part, we introduce the main aspects of software feature of noff across these following six dimensions.

A. Flexble Module Framework and Interface

Noff adopts a novel callback module interface which is based on the function template feature of C++. Noff is composed of several modules. These modules can be structured into several parts according to functionality or logical structure. The logical structure of noff includes capturer,dispatcher,IP,TCP,UDP,application and so on,which is correspond to the TCP/IP network model.Each layer will offer one or more callback interface to many upper layer modules. Programmers can build their modules on lower layer modules without changing any existing code.Furthermore, noff can selectively assemble different modules into a same layer to adapt to different demands.

B. PF_RING Patch

PF_RING is a new type of network socket that can dramatically improve the packet capture speed.It uses User-space ZC (new generation DNA, Direct NIC Access) drivers for extreme packet capture transmission speed as the NIC NPU (Network Process Unit) is pushing getting packets to user land without any kernel intervention. Noff uses PF_RING as the kernel driver and can thus arrive more aggressive runtime speed.

C. LRU Scheme

Noff has proposed Least Recently Used(LRU):a stream algorithm which enables the system to control its resources effectively during overload. This algorithm can resolve the problem that sudden traffic bursts or overload conditions may force the packet capturing subsystem to fill up its buffers and randomly drop streams in a haphazard manner.In order to arrive this requirment,noff maintains a novel timer data structure:timing wheel [12]which store the streams into different buckets by the time gap.Timing wheel would drop the streams which have been silent for at least one minute.In this way,the new stream will be accommodated with a higher probability.Besides,Programmers can also force to maintain some priority streams by updating the timestamp of those streams.The detail implementation of timing wheel would be described in the implementation section.

D. Flexible Stream Reassembly

Libnids [13] has accomplished its TCP reassembly machine by simplifying the Linux network stack. However,the callback interface of libnids is a little poor and it has no ability to cope with the connection abnormity because of none timer. Noff adds a passive keep-alive timer to close the dead abnormal connection when a stream connection has no data interaction during a long time. Noff also abandons the TCP memory buffer in libnids to reduce memory copy overhead. Besides,noff reserves nearly all of the other functions of libnids.

E. Memory Allocations Optimization

On the one hand,Noff uses TCMalloc [9] [10] to optimize the memory copy performance. The TCMalloc implementation performs better than the traditional tools.TCMalloc assigns each thread a thread-local cache. Small allocations are satisfied from the thread-local cache. when a cache is insufficient,memories are moved from central data structures into the thread-local cache as needed, and periodic garbage collections are used to migrate memory back from a thread-local cache into the central data structures ,which clearly reduce the frequency of memory allocation.

On the other hand,Noff also uses smart pointer to manage objects to enhance the security of memory management.

F. Parallel Processing

Noff offers two kinds of multi-thread platforms which inherent support for multi-core systems.The two kinds are all achieved by transparently creating a number of worker threads for user-level stream processing the number of those threads is equal to the number the available cores.All processing of a particular stream is done on the same core, reducing,in this way, context switches, cache misses, and interthread synchronization operations.

To balance the network traffic load across multiple NIC queues and cores, Scap [14] uses kernel based approach such as Receive Side Scaling (RSS) [11]to process balance a lot number of packets.While,in our implementation,when the NIC supports multiple queue,noff could use RSS to assign one stream to a corresponding thread equably.Otherwise,noff could even use customized load balancer to assign each stream.

III. ARCHITECTURE

This section describes the architecture of the noff framework for capturing and processing.

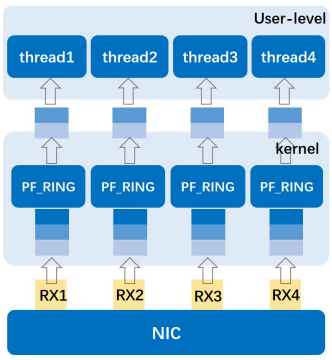

Fig. 1. Multiple Queues Capture

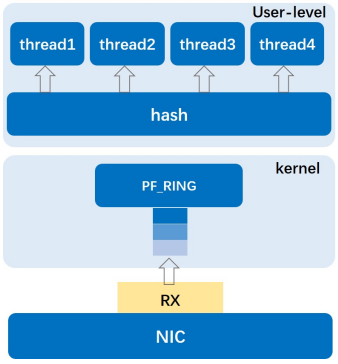

Fig. 2. Single Queue Capture

A. Kernel Capture

The Capture module of noff has two shapes:the first one mainly support Multiple queue NICs which use a loadable kernel module and a user-level API stub,as shown in Figure 1. In this case,Noff would initialize several threads which have same number towards NIC queues. Then PF_RING kernel module transfers stream packets between queue and thread.

The second one is mainly designed for NICs which only have one receive queue.This case is shown in Figure 2. Programmer can choose a suitable shapes according to practical situation which Reflects the versatility of our capture module.

B. Processing Procedure

Noff uses a multi-layer framework. Each layer is independent and could include a number of modules.A module would offer one or more callback interface for upper layer module to invoke. A stream packet's processing procedure is shown as Figure 3. Programmer can add a new module based on one existing layer by themselves.

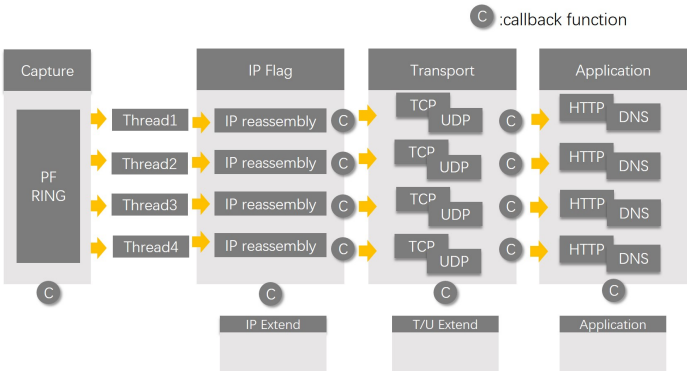

Fig. 3. Packet Processing Procedure

IV. Noff Modules

This section describes the module of noff and shows how to build and register a new module.

A. Module Structure

Noff's module is a C++ class. Each module should be responsible for a corresponding procedure.A module use a public function to invoke another module's interface in lower layer to acquire the necessary data.Meanwhile,this module also needs to offer a callback interface for upper layer modules through the function template feature.For instance, a module class is just simplify shown as Figure 4.

Figure 4 displays the nucleus functions of the capture module. vector<PacketCallback> and vector<IpFragmentCall> save all of the upper layer callback functions. Because these two vector are all private members. So programers should use **addPacketCallBack** and **addIpFragmentCallback** to operate these two private vectors.The **internalCallback** function is the key interface for lower layer modules to invoke.In conclusion,a valid module of noff should at least include a key interface for lower layer to invoke and at least one callback vector for upper layer modules to register callback function.

B. Module Configuration

After accomplishing one module,it is necessary to configure this module into the main file. The DNS module configuration is shown as following code block.As noff is a multi-thread platform,so we use the thread local variable to allocate module instance for each thread.Using this method,we need not to consider thread synchronization issues.In a module code block,we need firstly acquire the instance reference.Then we should register the upper callback interface to callback vector and register the key interface of this module to lower layer module.Finally,we should put our module function into **initInThread** function. In this way, this new module would work normally.

```
1  void setDnsCounterInThread()
2  {
3      auto& udp = threadInstance(Udp);
4      auto& dns = threadInstance(DnsParser);
5      udp.addUdpCallback(bind(
6      &DnsParser::processDns, &dns,
7      _1, _2, _3, _4));
8
9      dns.addRequstecallback(bind(
10     &UdpClient::onData<DnsRequest>,
11     dnsRequestOutput.get(), _1));
12
13     dns.addResponsecallback(bind(
14     &UdpClient::onData<DnsResponse>,
15     dnsResponseOutput.get(), _1));
16 }
17 void initInThread()
18 {
19  setDnsCounterInThread();
20 }
```

V. Implementation

We now give more details on the implementation of the noff monitoring framework.

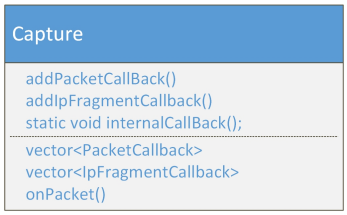

Fig. 4. An Example of The Module

A. Multi-Thread

Noff supports user-level multi-thread mechanism. Considering the compatibility of a network platform ,We offer two kinds of sharing scheme for programmers.The first one is built on NIC's RSS(receive-side scaling) scheme.

Multiple queue NICs provide multiple transmit and receive queues, allowing packets received by the NIC to be assigned to one of its receive queues. Each receive queue is assigned to a separate interrupt; by routing each of those interrupts to different cores. Processing of the interrupt requests triggered by the network traffic received by a single NIC can be distributed among multiple cores, bringing additional performance improvements in interrupt handling. Usually, a NIC distributes incoming traffic between the receive queues using a hash function.The first mechanism is accomplished through the PF_RING driver.

The second multi-thread mechanism is built on user-level hash algorithm because some NICs may not support Multiple queues. In this scheme,a sharding module should compute the hash index for each captured packet which is in the light of the four tuples of this packet and then assigns it to the corresponding thread.The mapping of threads to CPU cores is likewise practically one-to-one.

Through the multi-thread mechanism, noff has advanced the runtime performance effectively.

B. Powerful TCP Reassembly

TCP reassembly is a resource-consuming procedure. Thus it is central to optimize this procedure. noff develops a new stream reassemble module based on Libnids.We then describe this from three aspects:

1) Thread Safety: Libnids' stream reassembly module is accomplished by C language and it has many global variables which may cause thread unsafety.Whereas noff's stream reassembly module is accomplished by C++ and all of the variables are capsulated which ensures that each thread is assigned an independent copy.

2) Timing wheel: The connection flood is a common network attack.If a TCP connection only has the SYN packets without the FIN packets,this connection will not be erased until the stream buffer size exceeds the threshold.In this case,the new stream may have no chance to be processed. Actually,a number of situations can finally cause this case. Libnids' reassembly engine has no ability to conquer with this circumstance.So referring to the current problems,noff adopts an innovative Keep-alive timer solution.Once a connection is established,the connection's keep-alive timer should be also established simultaneously.

In order to manage the timer effectively, we adopt a wheel like data structure [12] which can flexibly insert a new stream timer and erase a dead timer which has expired.A simple schematic diagram is given in Figure 5. In this Figure there are totally seven time slots.Each slot represents one second and it is corresponding to a bucket which stores stream objects.When the system startups,the clock pointer should move to the next slot per second,Meanwhile the 0 slot would be dropped which means the stream objects in it also need to be dropped.And a new tail slot would be created.Through this way noff could knock out some dead streams.

noff need not only to knock out the dead streams but also to update the alive streams.In order to conveniently accomplish this function.we make use the reference count feature of the smart pointer.When we need to update a stream's timestamp,we just need to insert a shared_ptr object of this stream into the current time slot bucket which means the reference count of this stream has an increment.When we drop a time slot bucket and the objects in it,only if a stream's reference count become zero,this stream could be erased from stream hash table drastically.Through this way,we can easily find out that we can just update a stream by just inserting a shared pointer of this stream.

Furthermore, through this mechanism,programmers can also set priority for one stream.A simple method is that they can update one stream continuously whether the stream has in exchange packets or not.

Noff uses circle queue to accomplish the wheel timer and set the number of the time slot to be 60 which means once a stream has no data in exchange for 60 seconds,it would become a dead stream and would be then erased.

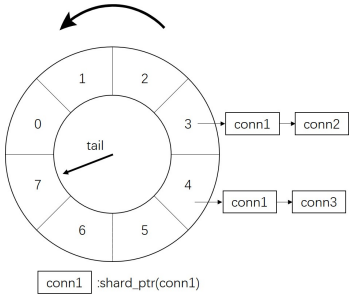

Fig. 5. Schematic Diagram of Timing Wheel

3) Memory Optimization: The TCP reassembly module needs to resort TCP fragments.So it needs to prestore packet data which is meant by a large number of dynamic memory allocation. Libnids not only prestore packets but also store the completed packets between client and server. This scheme has a palpable short slab that there are too many memory allocations and copies which could bring huge overhead.

Basically, noff has abandon the scheme that the reassembly module needs to store the completed packets. Because though libnids offer this ability. The upper applications also have to

store the TCP stream data by themselves again. After optimizing this procedure,the performance of stream reassembly module has been improved explicitly. And further more,noff uses TCMalloc to optimize the memory allocation speed.

C. Packets Filter and Statistic

Noff has afforded three kinds of filters.The first one is PF_RING filter,you should use **pfring_add_filter_rule** function to add a filter rule on an existing ring. The second one is BPF_filter. BPF supports filtering packets, allowing a userspace process to supply a filter program that specifies which packets it wants to receive. The third one is protocol filter which is embedded in Ip fragment module.It can filter the packets in the callback interface.Now it can filter packets according to registered module and port.For instance,TCP modules can use **registerRule** function to specify which packets they don't need.In this way,some packets may not be sent to the upper layer module again.So filters can increase the system performance to a certain extent.

Noff also provides much statistical information.Noff provides not only the basic kernel dropping information but also the number of processed packets in each filter and each thread.All of this information would be indicated through the system log. The following code block exhibits part of system log which lists the number of packets handled by each thread.

```
1 20170528 08:44:57.964400Z 128795 INFO
2 worker1: 7754 - Dispatcher.cc:48
3 20170528 08:44:57.964403Z 128795 INFO
4 worker2: 16902 - Dispatcher.cc:48
5 20170528 08:44:57.964405Z 128795 INFO
6 worker3: 20434 - Dispatcher.cc:48
```

Our implementation is based on an intel 520 XR2 NIC and a BCM5720 NIC.The XR2 is a modern 10GbE NIC supporting multiple queues and RSS while the BCM5720 is a 1000MbE NIC which only supports single receive queue.Noff can be effectively used with both these NICs. Actually according to our test,Noff can be executed on nearly all NICs platform.

D. Callback Interface

Noff use C++ bind/function feature to actualize callback interface.Bind scheme generates a forwarding call wrapper for a member function of a class.Calling this wrapper is equivalent to invoking a member function with some of its arguments bound to args. All module classes in noff should register their member function through this way.Once one function has been registered in a module.This module would store it into a vector.When a specific event happens,all call wrappers in the vector should be invoked one by one.This scheme is described vividly in Figure 6.

Actually this scheme is similar to Observer Pattern.

E. Packet Handle

An application may be interested in receiving both reassembled streams, as well as their individual packets to detect TCP-level attacks.Different from traditional methods or to extract meritorious information from network traffic data. Because

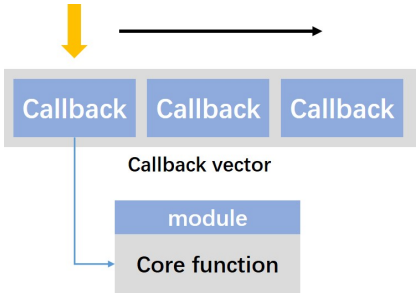

Fig. 6. Callback Interface Scheme

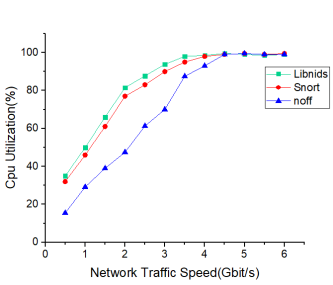

Fig. 7. Packet loss

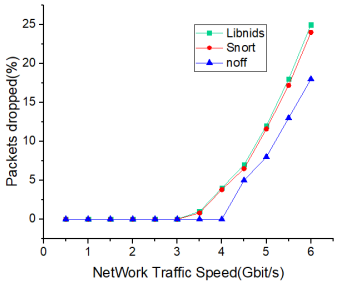

Fig. 8. CPU Utilization

of callback interface in each module,noff's interfaces only offer necessary data for each module rather than delivering the original packets from the network.If an application indicates that it needs original data,in noff it needs to register with the capture module's interface.

VI. Experimental Evaluation

We experimentally evaluate the performance and feature of noff, comparing it to other similar libraries, for common monitoring tasks, such as flow statistics export and http parser, while replaying a trace of real network traffic at different rates.

A. Experimental Environment

1) Hardware: In this paper,we use a testbed comprising two PCs interconnected directly through an Intel 520XR2 10GbE NIC and MT26448 10GbE NIC.Intel NIC is used as a stream capturer while MY26448 NIC is used for traffic generation. The two PCs are both equipped with one eight-core Intel Xeon 1.8GHz CPUs with 10MB L2 cache, 4GB RAM.Both PCs run 64-bit Ubuntu Linux(kernel version 4.4.50).

2) Traffic: To evaluate stream reassembly implementations with real traffic,we replay a traffic file which comes from campus network having totally two million TCP packets.

3) Parameters: We compare noff with the following libraries: (I)Libnids v1.25 [13] (II)and the Stream5 preprocessor of Snort v3.0.0. Libnids and Snort rely on Libpcap [15], which uses the PF_PACKET socket for packet capture on Linux while noff use PF_PACKET directly .In our experiments, the size of this buffer is set to 512MB, and the buffer size for reassembled streams is set to 500000 for all of these three library.The other parameters for Libnids and Snort are all default. We set the keep-alive timer of noff as 60 seconds.As

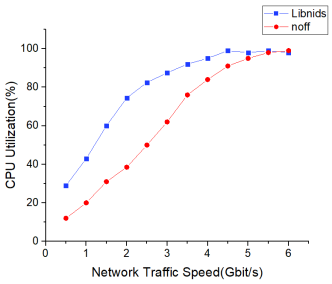

Fig. 9. CPU Utilization of HTTP Parser

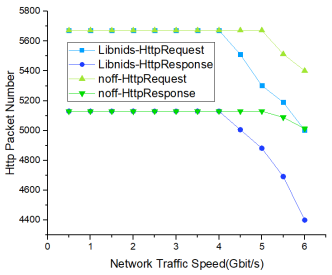

Fig. 10. Http Packet Number

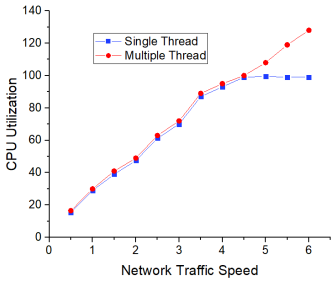

Fig. 11. CPU Utilization of Single and Multiple Thread

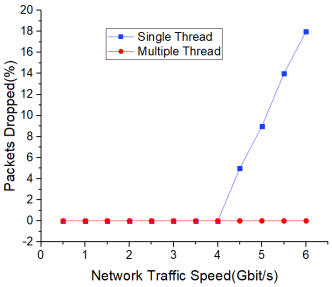

Fig. 12. Packet Loss of Single and Multiple Thread

we replay the traffic data circularly,an inactivity timeout of 60 seconds is a reasonable choice.

B. TCP Stream Processing Comparision

The first experiment we evaluate the TCP stream reassemble performance of noff comparing with Snort Stream5 processor and a Libnids-based simple sniffer program that just receives reassembled streams. In this experiment, noff only assembles TCP fragment module without advanced processing.Noff also drop UDP and ICMP packets directly.Similarly Libnids and Snort also drop all of them.When these libraries receive FIN or RST TCP packets,they just erase the entire TCP connection from hash table.For a fair comparison, we configure noff to only use a single worker thread, as Snort and Libnids are just single-threaded. And they all use PF_RING as NIC driver.

Figures 7,Figure 8 present the percentage of dropped packets,the average Cpu utilizing of the monitoring application on a single core. We use the tool tcpreplay to replay the static network traffic at the rate from 500Mbit/s to 6Gbit/s.The result Figures indicate that The Libnids starts losing packets when the traffic rate exceeds 3Gbit/s. It is becasue that when the traffic rate exceeds 3Gbit/s, the CPU utilization of the Libnids exceeds 90%. Also the Snort performs slightly better than Libnids,But it still start to losing packets when the traffic rate exceeds 3Gbit/s. At this time, the CPU utilization of the Snort also exceeds 90%.Because the Snort and Libnids have similar stream processing schemes.

Noff performs explicitly better than the Libnids and Snort.It starts to drop packets after the traffic rate exceeds 4Gbits/s.It is critical to figure that The Libnids,The Snort and the noff are all based on single-core PF_RING driver.Which means that noff has better user-level optimization. In my opinion,the key optimization is the memory allocation.

C. Http Information Processing Comparision

The second experiment is involved with the application of noff. The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. In this experiment,We have separately developed a HTTP parser application module for the three library(noff,Libnids,Snort).the modules collect HTTP data from TCP layer then extract critical information from the headers of HTTP packets and finally export the statistic information for each HTTP connection.

The procedure of HTTP parsing is shown as this:The module firstly collects the TCP streams which have been reassembled.Then the module filters the useless streams through port matching. And then the module should put the TCP data into a http parser tool to obtain the HTTP header's information.

In this experiment, we evaluate the performance of parsing HTTP packets of noff comparing with the Libnids which has assembled the same HTTP module as noff.In order to acquire relatively stable experimental results. We use a static TCP streams dataset from SJTU campus.All of TCP packets' port in this dataset are all 80 and there are totally 5672 HTTP request packets and 5130 HTTP response packets. Figure 9 and Figure 10 indicate the average CPU utilization of the monitoring application on a single core,and the HTTP packets number statistic information on a single core while varying the traffic rate from 500Mbit/s to 6Gbit/s.We see that Libnids starts losing HTTP packets when the traffic rate exceeds 4Gbit/s. The reason can be seen in Figure 9,where the total CPU utilization of Libnids exceeds 90% at 4Gbit/s.Meanwhile,noff starts to drop HTTP packets at 5Gbit/s.Comparing the experiment of TCP reassembly, we can find that the performance of noff application is also better than Libnids.

D. Multiple Thread and Multiple Core

In the Implementation section,we declare that the PF_RING driver can support multiple queue NIC packet capture. In this experiment,we should verify the value of multiple thread.

Our 10GBE NIC can support 8 receive queues. So we would compare the performance of noff's TCP reassemble on single thread with the performance on multiple thread. Actually, multiple thread does not bring too much additional overhead.Each thread should burden the same procedure. Through NIC's RSS,each thread would only process one-eighth streams compared with the single thread scheme.

The Figure 11 and Figure 12 indicate the experiment result which include the performance of noff on two cases.From the result, we can clearly see that the single thread case starts to drop packets when the traffic rate exceeds 4Gbit/s while the multiple thread case drops no packet even the the traffic rate exceeds 6Gbit/s.We can find the reason from 11.when the traffic rate exceeds 4Gbit/s,the single core utilization arrives

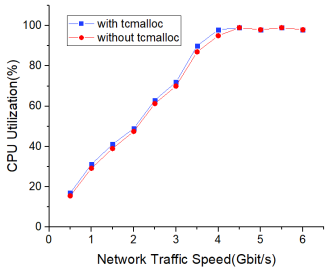

Fig. 13. CPU Utilization with TCMalloc or not

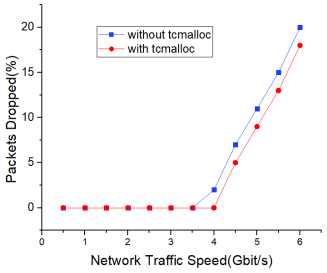

Fig. 14. Packet Loss with TCMalloc or not

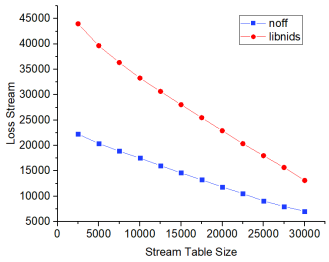

Fig. 15. Stream Loss with Different Hash Table Size

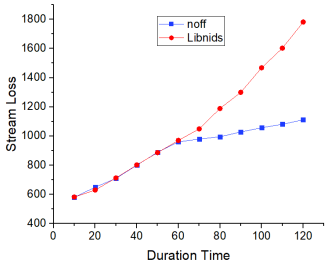

Fig. 16. Stream Loss with Different Duration

the threshold(100%) which means noff arrives its limitation on single core.But on multiple core platform,noff can use several cores to improve its processing ability.The utilization of it can exceed 100%.This experiment prove that by multiple thread scheme,noff can obtain better performance extremely.

In the implementation section, we also refer an user-level multiple thread scheme.Different from the RSS method.The user-level method has an extra memory copy and brings a synchronization Lock.So the user-level multiple thread can not always improve performance.According to our experiment,when the work per thread is too heavy,the user-level multiple thread may become a better choice.

E. TCMalloc Optimization

Memory copy always bring remarkable influence to performance.In this paper we adopt the TCMalloc tool to accelerate the memory copy process.In this experiment, we would compare the noff's performance in the two cases:with TCMalloc and without TCMalloc.In order to control the variables,we decide to reuse the test scenario in TCP stream reassembly experiment.

The Figure 13 and Figure 14 indicate the result of experiment.From the Figure 13,we can see that the CPU utilization is slightly higher when we don't use TCMalloc. So the consequence is that when we don't use TCMalloc noff starts to drop packets when the network traffic exceeds 3.5Gbit/s.This check experiment prove that TCMalloc can improve the memory copy performance to a certain extent.

The full name of TCMalloc is thread cache malloc. One of the key exploring spots is the thread local cache. In multiple thread environment,noff's module classes are all thread cache variables. So In this environment,the TCMalloc tool is a much better memory allocation optimization method.

F. Concurrent Traffic

In the previous section,we have involved the Least Recently Used algorithm implementation and timing wheel scheme to manage the stream hash table.In this experiment, we need to confirm the effectiveness of these two methods.In order to attain the objective, we hope to verify this system from two aspects: multiple concurrent traffic processing and the no FIN streams processing.

For the first aspects,we use a real network TCP traffic set from SJTU campus network as the test data.In this set,there are totally about 50000 streams.We would vary the size of stream hash table of noff and libnids from 500 to 30000.Then record the number of loss streams of libnids and noff. The figure 15 indicates the result of experiment.From the figure,we can explicitly see that libnids drops more streams than noff under the same circumstances. Actually these two library would eventually drop about 4000 streams when they have plenitudinous table size.

For the second aspects,we have imitated multiple streams. These streams are all no FIN packets.In this way, if the capture library has no scheme to erase the dead streams.The streams would take up one position of the table.Noff can use time wheel to kick away the dead streams.So it should have a better performance in this situation.The figure 16 indicates the result of this experiment.we imitate the no FIN stream at the speed of 10Mbit/s and compare the number of loss stream of noff and libnids from 10s to 120s.From the figure 16 we can see that noff and libnids have similar performance before 60s while noff obviously drop less packets after 60s for that noff's keep-alive timer works.The timer drops the dead stream from stream table.

Through these two experiments,we can reach the conclusion that noff has a better stream manager scheme.It can process multiple streams more flexibility.

VII. Related Work

In this section, we would introduce some previous related work of other researchers.

A. Kernel Capture Optimization

Scap [14] is a novel capture library whose most attractive distinguishing characteristic is that it offer a NIC driver to process packet data in multiple receive queue.Its modules are mainly deployed in kernel level which means it has an aggressive performance.And scap also provide some meticulous system optimization method.To tell the truth,scap is actually an excellent research work. By contrast, noff directly use PF_RING driver to accomplish the similar function.But noff also support an user-level multiple thread platform to enhancing the versatility of the library.After all,not all NIC support multiple receive queue.

B. Multicore Architecture

Multicore architecture creates a new epoch in research on parallel processing.Many researches have been done.For example Low-level CPU optimizations like those of L2-cache is one practical solution for fast parallelization [17] and Chen et al. [18] perform parallel processing of Big Data but optimize resource use.Marat Zhanikeev [17] proposed a lockfree method to speed up capture process.Scap [14] is another kernel-level based optimization method and Schultz [22] claims that baremetal can handle the 10Gbps traffic on a single core.Noff adopt a lockfree user-level multi-thread platform.

C. TCP Stream Reassembly

Libnids [13] and Stream5 [19] perform TCP reassembly based on the emulation of a network stack. Noff propose a more high-performance reassembly engine which optimize the drawbacks of libnids.

D. Structure Optimization

Luca Deri [20] proposed a method based on polling frame to optimize the packets processing procedure.Yeim-Kuan [21] make use of hierarchical binary prefix search scheme to process packets.Noff uses flexible callback interfaces to help programmers to build flexible structure easily.

VIII. Conclusion

In this paper,we have analyzed the significance in capturing and parsing the completed traffic network stream. So a high-performance capture traffic is badly demanded.Actually most programmers need a high-Performance capture as well as a flexible extendible library to add some aggressive applications.To cope with this conundrum. We have proposed our work:noff which is a flexible high-performance network traffic capture library.It has both kernel-level and user-level based multi-thread scheme to adapt to nearly all kinds of NICs.And it offers many flexible callback interfaces to help programmers to organise the system framework effectively.Compared with existing library libnids, noff has Imposed a variety of optimization such as memory allocation,TCP reassembly and so on.In this paper we evaluate our implementation through several experiments.

The results of our experimental evaluation demonstrate that noff can capture network traffic for speed up to 4Gbit/s without packets loss using a single core which is better than the other existing approaches.A HTTP parsing application based on noff's platform is also performed better than the the other existing approaches.Moreover,noff support multi-thread,using this scheme noff can cope with the network traffic whose speed is up to 6Gbit/s easily.Meanwhile,noff's stream management algorithm help it to drop nearly 50% fewer streams than the existing approaches when facing the high concurrence traffic.We can also build a new application based on noff faster.

We believe our implementation can bring some novel idea for developing an attractive network library.

Acknowledgment

We would like to thank the junior student Guangqian Peng.He has offered a huge contribution to this research work.This research was performed with the financial support from an industrial project. The experts of SJTU also have been involved in this project and offered some valuable feedback.

References

[1] Liu, Duo, et al. "High-performance packet classification algorithm for many-core and multithreaded network processor." International Conference on Compilers, Architecture and Synthesis for Embedded Systems ACM, 2006:334-344.

[2] Antonello, Rafael, et al. "Deep packet inspection tools and techniques in commodity platforms: Challenges and trends." Journal of Network & Computer Applications 35.6(2012):1863-1878.

[3] Qiao, Siyi, et al. "Network recorder and player: FPGA-based network traffic capture and replay." International Conference on Field-Programmable Technology IEEE, 2015.

[4] Ali, Sardar, et al. "On mitigating sampling-induced accuracy loss in traffic anomaly detection systems." ACM SIGCOMM Computer Communication Review 40.3(2010):4-16.

[5] PF_RING. http://www.ntop.org/products/packet-capture/pf_ring/

[6] Luca Deri, Modern Packet Capture and Analysis: Multi-Core, Multi-Gigabit, and Beyond, Internet Measurement (IM) Tutorial, 2009.

[7] Sourcefire vulnerability research team (vrt). http://www. snort.org/vrt/.

[8] J. Evans. Scalable memory allocation using jemalloc. 2011. http://www.facebook.com/notes/facebook-engineering/scalable-memory-allocation-using-jemalloc/480222803919.

[9] Intel Threading Building Blocks. https://software.intel.com/en-us/intel-tbb/

[10] Lee, Sangho, T. Johnson, and E. Raman. "Feedback directed optimization of TCMalloc." The Workshop on Memory Systems PERFORMANCE and Correctness ACM, 2014:3.

[11] Intel Server Adapters. Receive Side Scaling on Intel Network Adapters. http://www.intel.com/support/network/adapter/pro100/sb/cs-027574.htm.

[12] Varghese, G., and A. Lauck. "Hashed and hierarchical timing wheels: efficient data structures for implementing a timer facility." IEEE/ACM Transactions on Networking 5.6(2002):824-834.

[13] Libnids. http://libnids.sourceforge.net/.

[14] Papadogiannakis, Antonis, M. Polychronakis, and E. P. Markatos. "Scap: stream-oriented network traffic capture and analysis for high-speed networks." Conference on Internet Measurement Conference ACM, 2013:441-454.

[15] S. McCanne, C. Leres, and V. Jacobson. Libpcap. http://www.tcpdump.org/. Lawrence Berkeley Laboratory.

[16] Machdi, Imam, T. Amagasa, and H. Kitagawa. "Executing parallel TwigStack algorithm on a multi-core system." International Conference on Information Integration and Web-Based Applications & Services ACM, 2009:176-184.

[17] Zhanikeev, Marat. A lockfree shared memory design for high-throughput multicore packet traffic capture. Wiley-Interscience, 2014.

[18] Chen R, Chen H, Zang B. Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling. In 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2010;523534.

[19] J. Novak and S. Sturges. Target-Based TCP Stream Reassembly. http://assets.sourcefire.com/snort/developmentpapers/stream5-model-Aug032007.pdf, 2007.

[20] L Deri,NSPA Via,B Km,LL Figuretta.Improving passive packet capture: beyond device polling.Proceedings of Sane;2004.

[21] Chang, Yeim Kuan, and F. C. Kuo. "Towards optimized packet processing for multithreaded network processor." International Conference on High PERFORMANCE Switching and Routing IEEE, 2010:127-132.

[22] Schultz, Michael J., and P. Crowley. "Performance Analysis of Packet Capture Methods in a 10 Gbps Virtualized Environment." International Conference on Computer Communications and Networks IEEE, 2012:1-8.