NetFlow-Based Network Traffic Monitoring

ZHANG Weiwei GONG Jian GU Wenjie CAI Shaomin
College of Computer Science and Enginneering
Southeast University
Nanjing, China
{wwzhang, jgong, wjgu, smcai}@njnet.edu.cn

Abstract—**In order to achieve real-time traffic monitoring on high-speed backbone link, this paper proposed a series of effective solutions aiming at data collection, processing and statistical analysis with minimal packet loss rate (even none packet loss). First of all, with NetFlow sampled records as data source, the proposed method effectively improves data collection efficiency by using buffer structures and multi-threads concurrent mechanism. Secondly, on the purpose of avoiding redundant operations in latter analysis to improve efficiency of the whole system, the method introduces a common operational procedure to make a unified process on raw data. Finally, through geographical region partition of the network and appropriate time granularity, the method analyzes network traffic performance between managed network (or a single IP) and outer networks. Using an ordinary PC, the scheme could be able to collect, process and analyze data from *20Gbps* backbone network with a high performance (packet loss rate less than 10^{-6}, memory overhead of *38MB*).**

Key Words—traffic monitoring; backbone network; NetFlow; parallelization programming

I. INTRODUCTION

With the rapid expansion of the network scale and the increasing diversity of applications, the bandwidth of the backbone is growing to thousands of megabytes. It brings a new challenge to the traffic monitoring, analysis and management timely and efficiently in the condition of low packet loss rate. The main reason is data acquisition and processing cannot catch up with traffic growth.

The traditional technologies for traffic monitoring and analysis are SNMP and packet sniffer, but they fail to meet the needs of both data accuracy and performance requirements: SNMP cannot provide fine-grained traffic information based on end to end connection; Packet sniffer cannot be used in situation of backbone network.

In recent years, development of flow[1] technology makes traffic monitoring and analysis of backbone network possible. Flow technology provides rich traffic information based on the end to end connection, and it only needs to handle less than 5% data of the whole packets, thus reducing the pressure of the network device. There are many tools for data collection and analysis, and Flow-tools[2] and Nfdump[3] are representatives among them. However, most of them are base on the idea of serial design, and packet loss rate is quite high in the case of burst traffic, thus making it difficult to meet the performance requirements for real-time traffic monitoring in backbone network.

To solve the performance problem above, the paper proposed a series of improved schemes. First, the paper uses NetFlow[4] technology to collect data, decomposes data based on the concept of parallel design, and utilizes multi-buffers and multi-threads parallel mechanism to process data. Then, the paper also introduces a common operational procedure to reduce redundant operations and improve efficiency. Finally, through geographical region partition of the network and appropriate time granularity, the paper analyzes the network traffic and performance between the managed network and outer networks timely.

II. IMPROVED SCHEMES

The NetFlow-based network traffic monitoring framework concludes four steps: data export, data collection, data processing and statistical analysis. The last three steps will be introduced in detail.

A. Parallel Data Collection

With the increasing amount of data, packet loss will occur if the speed of data acquisition cannot catch up with the arrival rate of packets. In order to achieve real-time traffic monitoring on backbone link, we need to design an efficient scheme to make real-time data collection on backbone network with minimal packet loss rate. Based on the basic idea of parallel design, this paper increases the concurrency of the task from both data decomposition and functional decomposition, and then effectively improves data collection efficiency by using buffer structures and multi-threads concurrent mechanism.

1) Data Decomposition

Data decomposition divides one data set into several sub-data sets, and then uses parallelization method on the process of them to improve the system efficiency.

NetFlow UDP packets are continuously sent to the collector, so the data can be regarded as continuous in time space. As real-time traffic monitoring is concerned on the traffic behavior over the current time slice, so we can divide the whole time space into continuous time slices (eg. the length of each time slice is 5 minutes), the data on different time slices form independent sub-data sets. Only when all data within a time slice has been collected can we start to process and analyze it. Therefore, data collection and processing work on a different data set from the statistical analysis (the first two steps work on the data set within the current time slice, while the latter one works on the data set within the previous time slice). Through the vertical division of data set in time space,

statistical analysis is separated from data collection and processing, thus improving task concurrency.

Data set can also be executed horizontal division in time space. A data set within a specific time slice can be further divided into N disjoint sub-data sets to enhance efficiency by calling the appropriate number of threads.

2) Function Decomposition

Function decomposition divides the whole function into multiple sub-functions, and then makes parallel execution of them to improve the system efficiency.

NetFlow-based network traffic monitoring concludes data collection module, data processing module and statistical analysis module. These three modules work independently with each other and exchange data through shared buffers. On the one side, the statistical analysis module works on a data set within a different time slice; On the other side, though data collection module and data processing module work on the same data set within a time slice, they make parallel execution independently with each other through shared buffers. As above analysis, data collection module directly affects the packet loss rate, and it seems as the bottleneck of improving efficiency.

Data collection module specifically includes UDP packet capture, packet decoding and flow records storage. Existing open source tools often make the three steps a serial execution. As the packet decoding operation is very complex, putting it in the middle of the execution sequence will lead to a large amount of packet loss when meeting burst traffic. In order to minimize the loss rate, it needs to separate UDP packet capture operation from the other two operations and make them working parallelly. Because UDP packet capture is the simplest operation which has a minimum of time overhead, executing it dependently could effectively reduce the packet loss rate.

3) Parallel Design

Through the above data decomposition and function decomposition, we effectively increase the concurrent of the task. Next we will introduce how to improve data collection efficiency by using buffer structures and multi-threads concurrent mechanism.

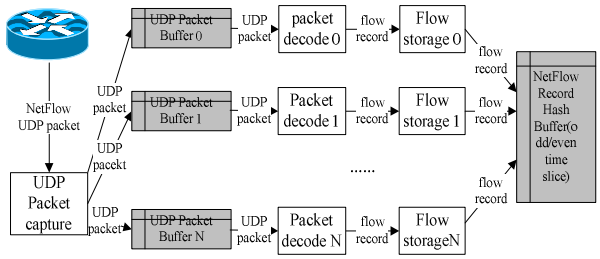

Figure 1: Parallel Design

Figure 1 shows a detailed design of parallel data collection module. Packet decoding and flow records storage operations conduct serially, while UDP packet capture operation executes in parallel with them and exchanges data through shared buffers. UDP packet capture is responsible for monitoring the corresponding UDP port on the specified IP address, collecting NetFlow UDP packets sent from the router, and then writing them into UDP Packet Buffer. Packet decoding operation reads the un-decoded UDP packet from UDP Packet Buffer, and then decodes flow record out of the packet. Flow records storage operation calculates hash value according to the quintuple of the flow (source/destination address, source/destination port, protocol number), and then writes the processed records into NetFlow Record Hash Buffer.

Data collection module involves two shared buffers: UDP Packet Buffer and NetFlow Record Hash Buffer. UDP Packet Buffer is used to store the received UDP packets, which is the key point to resolve the unexpected packet loss in the case of burst traffic. As shown in Figure 1, in order to further improve the efficiency of the packet decoding and record storage, we allocate N UDP Packet Buffers, and write the collected packets into them in turn. For each buffer, we call one decoding and storage thread. NetFlow Record Hash Buffer is used for data communications between data collection module and data processing module. For the convenience of data searching, we choose hash structure. In addition, as the data set are divided into sub-data sets, we also introduce two NetFlow Record Hash Buffers to store the sub-data sets in turn.

B. Data Processing

Considering that different statistical applications all need to execute some common processing steps. In order to avoid the overhead caused by redundant operations and improve the overall system efficiency maximally, this paper introduces a common operational procedure.

1) Basic Steps

From the detailed analysis of the data processing steps of a variety of statistical applications (such as traffic cost and QoS), we can extract the following basic steps.

Filtering data: Real-time traffic monitoring makes statistical analysis on all records of the current time slice, so we need to filter the records that do not belong to this time slice and abnormal records that with incorrect timestamp. The packet decoding operation discussed above needs to check timestamp of each record to determine which buffer (odd or even time slice buffer) it should be written into. Meanwhile, data filtering can be handled to reduce the system burden as early as possible.

Merging short-flows: NetFlow records with the same quintuple could be merged into one record. While merging records, bytes and number of packets should be accumulated for each record. This operation could reduce *1/3-1/2* quantities of records within a time slice, as well as reducing the memory cost of the buffer. The packet decoding operation also needs to calculate hash value according to the quintuple of the record. Meanwhile, short-flows merging operation can be conducted together.

Searching bidirectional flows: We first give the definition of bidirectional flows. Suppose, F_1 and F_2 are any two flow records with quintuples $Q_1=\{sip_1, dip_1, sport_1, dport_1, proto_1\}$ and $Q_2=\{sip_2, dip_2, sport_2, dport_2, proto_2\}$. If $sip_1=dip_2$, $dip_1=sip_2$, $sport_1=dport_2$, $dport_1=sport_2$, and $proto_1=proto_2$, then we call F_1 and F_2 are bidirectional flows. In order to further distinguish them, we define the direction of the flow. If $sip_1<dip_1$, then F_1 is forward-direction flow, or vice versa. For any flow F_1, the bidirectional flows searching operation is to find the corresponding reverse-direction flows in data set.

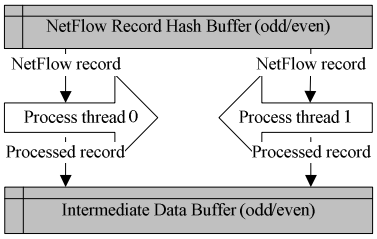

Figure 2: Parallel data processing

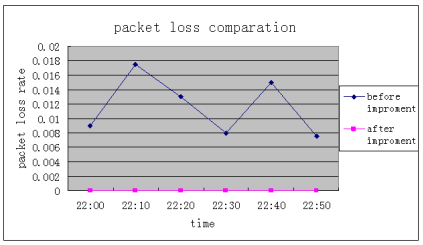

Figure 3: Packet loss comparation

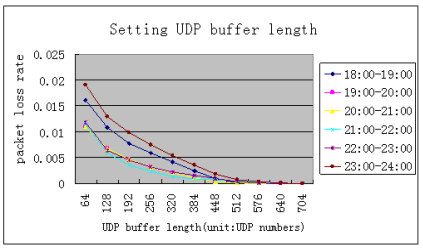

Figure 4: UDP buffer length setting

Locating geo-location and ISP of the IP addresses: By querying the IP information table, we can find ISP and geographic information of any IP address. The IP table is obtained by making the integration of Whois, Maxmind, IP2Location, QQ and some other databases.

Identifying application type of the flows: Recognize the applications type of flows according to protocol, port and some other information of the flow records.

2) Parallel Processing

Data processing module reads records from NetFlow Record Hash Buffer, searches bidirectional flows, locates geo-location and ISP of the IP addresses, identifies application type of the flows, and finally stores the processed records into Intermediate Data Buffer. If we begin to process data only after all records with a time slice have been collected, the data processing module would take too much time that should be reserved for the latter statistical analysis. Therefore, data processing module should check NetFlow Record Hash Buffer every a short time(less than the length of a time slice), and process the new records that have been written into the buffer just now. In this way, when finishing acquiring all records within a time slice, there would be a few non processed records left in the buffer that would cost just a little time. As shown in Figure 2, to improve efficiency, we can trigger two threads to process records from both ends of NetFlow Record Hash Buffer and write processed records into Intermediate Data Buffer from both ends.

Data processing module involves two shared buffers: NetFlow Record Hash Buffer and Intermediate Data Buffer. NetFlow record hash cache has already been introduced above. Intermediate Data Buffer is used to store processed records, and it is also used for data communications between data processing module and statistical analysis module. The length of Intermediate Data Buffer is associated with the number of unique quintuples. In order to further improve the concurrency of data processing, we also introduce two Intermediate Data Buffers to store data in turn.

C. Statistical Analyses

This section makes a fine-grained geographical region partition on Internet, and then analyzes network traffic performance between managed network (or a single IP) and outer networks.

Firstly, the whole Internet could be divided into managed network and un-managed network based on whether IP addresses are within control. To conduct fine-grained traffic monitoring, we then further divide networks into smaller subnets according to the ISP and geographical information of IP addresses. To make the subnet division convenient, we mark each IP address with a 32bit integer which reflects the corresponding ISP and geographic information.

Each flow record contains a source IP address and a destination IP address. According to their integer mark, the two IP addresses could be divided into two subnets. Therefore, the record could be mapped into two subnets, and then we could further calculate bandwidth, latency and packet loss rate between the two subnets. As many essays have detailed introduced the methods to assess traffic performance based on bandwidth, latency and packet loss rate, this paper does not declare repeatedly.

III. Performance Analyses

Real-time data collection, processing and statistical analysis on backbone link with minimum packet loss is a performance problem that the paper should study and improve. This section makes a detailed analysis on the proposed solutions from three aspects such as packet loss rate, time complexity and memory overhead. The schemes proposed in this paper have been actually used to collect and analyze traffic from a duplex 10Gbps backbone link of JSERNET (Chinese Education and Research Network that belongs to Jiangsu province).

A. Packet Loss Rate

1) Calculation Formula

$$lost\_rate = \frac{lost\_pkts}{received\_pkts + lost\_pkts} \quad (1)$$

lost_pkts is the number of UDP packets lost in the time slice while *reveived_pkts* shows the number of UDP packets captured in the time slice.

2) Testing

After improvement, packet collection, decoding and records storage operations were executed parallelly, meanwhile some complex operations called multiple threads to execute concurrently. As shown in Figure 3，the improvement effectively improves the efficiency of data collection and decreases packet loss rate to zero.

B. Time Complexity

To ensure the performance of traffic monitoring, the proposed scheme should reduce time overhead of data collection, processing and analysis to a certain limitation.

t_{slice}: length of a time slice; M: number of UDP packets in a time slice; N, N': number of records in a time slice before/after merging short-flows; t_{r_udp}, t_{w_udp}, t_{decode}: time spent to read/write/decode a UDP packet in memory; t_r, t_w, t_m: time spent to read/write/modify a record in memory; t_{ip2loc}: time spent to search a IP

address in IP information table; t_{app_identy}: time spent to identify the application type of a record; N_{cpu}: CPU numbers of a PC; H: bucket numbers of hash structure; $t_{control}$: time spent to control multi-threads; t_{stat}: time spent on statistical analysis; *MAX_REVERSE_TIME*: suppose T_i and T_j are two end timestamps of any two records which reached in order. If there exists a minimum number which always satisfy "$MAX\_REVERSE\_TIME > T_i - T_j$", we call it maximum wrong sequence time.

1) Data Collection

Time spent on collecting data concludes two parts: one part is to capture packets and the other part is to decode packets and store records into buffer. Packet capture operation receives UDP packets and then writes them into UDP buffer.

$$t_{packet\_capture} = M(t_{r\_udp} + t_{w\_udp}) \tag{2}$$

Packet decoding and record storage operation reads UDP packets from UDP buffer, decodes records from them and meanwhile checks timestamp to filter records, then writes records into NetFlow Record Hash Buffer and merges short-flows concurrently. It is considered to call N_{cpu} decoding and storage threads.

$$t_{packet\_decode} = \frac{M(t_{r\_udp} + t_{decode}) + N(\frac{N'}{H}t_r + t_w)}{N_{cpu}} \tag{3}$$

2) Data Processing

Time spent on processing data includes: time spent on reading records, identifying bidirectional flows, searching IP table, identifying application type of records, and writing records into Intermediate Data Buffer. It is considered to call two threads to read records from both ends of NetFlow Record Hash Buffer and write records into Intermediate Data Buffer from both end.

$$t_{process} = \frac{1}{2}N'(t_r + \frac{N'}{H}t_r + t_{ip2loc} + t_{app\_identy} + t_w) + t_{control} \tag{4}$$

3) Statistical Analysis

Each statistical application reads data from the same Intermediate Data Buffer, but they execute independently with each other. To ensure performance, time spent on each operation should satisfy the following inequality:

$$t_{packet\_capture} < t_{slice};\ t_{packet\_decode} < t_{slice};$$
$$t_{process} + t_{stat} < t_{slice} - MAX\_REVERSE\_TIME \tag{5}$$

Now, an ordinary PC has two CPUs, read-write speed in memory is *1GB/s*. On the *20Gbps* backbone link, the number of UDP packets could not exceed *400000* in *5* minutes. One packet has *1500B* and one record has *68B*. $t_{packet\text{-}capture} \approx 1.2s; t_{packet\text{-}decode} \approx 1.02s; t_{stice} = 300s; t_{process} \approx 0.26s;$ If $t_{stat} < 269.74s$, then formula 5 established. As analysis above, with enough UDP buffers, an ordinary PC could achieve real-time traffic monitoring on backbone link.

C. Memory Overhead

The proposed scheme involves three kinds of buffers: N_{cpu} UDP buffers, two NetFlow Record Hash Buffers and two Intermediate Data Buffers (one for odd time slice and the other for even time slice).

UDP buffer is used to buffer burst traffic, the length of the UDP buffer affects packet loss rate directly. On the *20Gbps* backbone link, observe packet loss rate each hour from *18:00* to *24:00* (traffic peak of a day). As shown in figure 4，with the length of the UDP buffer increasing，packet loss rate decreases along logarithmic trend down. And when buffer length comes *704*, packet loss rate decreases to zero. Here we set UDP buffer length to *768(512+256)* UDP packets.

NetFlow Record Hash Buffer is used for data communications between packet capture operation and packet decoding operation in the data collection module. And Intermediate Data Buffer is used to store processed records for the latter statistical analysis module. On the one hand, to increase the concurrency of the task, the system doubles the memory overhead (use two buffers alternately for odd and even time slices); On the other hand, data filtering and short-flows merging effectively reduce the number of records (reducing *1/3-1/2* quantities of records), as well as the buffer overhead. The length set for these two kinds of buffers should be corresponding with the maximum quantity of records in a time slice after merging short-flows. On the one side, a small setting of length would lose data when meeting a large number of records in some time slice and the buffer has been written full. On the other side, a large setting of length would waste memory because of the less use of the buffer in many cases. To improve the efficiency of the buffer use, it should be able to allocate memory dynamically when the buffer is full, and release the minimum free memory on a particular time every day.

Through memory overhead test, the scheme reduced the total memory cost from the previous *56MB* to *19MB×2,* and could efficiently coupe with burst traffic.

IV. CONCLUSION

This paper proposed a series of effective solutions aiming at data collection, processing and statistical analysis to achieve real-time traffic monitoring on backbone link: use NetFlow sampled records as data source; increase the concurrency of the task from both data decomposition and functional decomposition, then effectively improve data collection efficiency by using buffer structures and multi-threads concurrent mechanism; extract common steps to make a unified process on purpose of avoiding redundant operations. Through the performance test, an ordinary PC which makes use of the scheme could be able to collect, process and analyze data from the *20Gbps* backbone link with a high performance (packet loss rate less than 10^{-6}, memory overhead of *38MB*) in the real time.

REFERENCES

[1] http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.html

[2] Ryu, B., Cheney, D., Braun, HW. Internet flow characterization: adaptive timeout strategy and statistical modeling. In Workshop on Passive and Active Measurement (PAM), 2001: 94-105

[3] http://www.splintered.net/sw/flow-tools/

[4] http://nfdump.sourceforge.net/