NG-MON: Next Generation High-Speed IP Traffic Monitoring and Analysis System

Myung-Sup Kim, Se-Hee Han and James W. Hong
Dept. of Computer Science and Engineering, POSTECH
{mount, sehee, jwhong}@postech.ac.kr

Abstract

This paper presents the design and implementation of a next generation network traffic monitoring and analysis system, called NG-MON (Next Generation MONitoring), for multi-gigabit high-speed networks. Packet capturing and analysis of such high-speed networks is extremely difficult using traditional approaches. Using distributed, pipelining and parallel processing techniques, we have developed a flexible and scalable monitoring and analysis system, which can run on off-the-shelf, cost-effective computers. The monitoring and analysis tasks in NG-MON are divided into five phases: packet capture, flow generation, flow store, traffic analysis, and presentation. Each phase can be executed on separate computer systems and cooperates with adjacent phases using pipeline processing. Also, each phase can be composed of a cluster of computers for load distribution wherever the system load of the phase is higher than the performance of a single computer system. We have defined efficient communication methods and message formats between phases suitable for pipelined parallel processing. The implementation details and the results of performance testing are also provided.

Keywords

IP network monitoring and analysis, passive monitoring, distributed processing, pipeline processing, load sharing.

1. Introduction

Today, multi-gigabit networks are becoming common in Internet service provider (ISP) and enterprise networks. The bandwidth of ISP’s backbone networks is evolving from OC-48 (2.5Gbps) to OC-192 (10Gbps) to support rapidly increasing Internet traffic. Also, enterprise networks are evolving from 100-Mbps or 1-Gbps to multi-gigabit networks. Further, the types of traffic on these links are changing from simple text- and image-based traffic to more sophisticated and higher volume (such as streaming rich media or peer-to-peer). Monitoring and analyzing such high-speed, high-volume and complex network traffic is needed, but it lies beyond the boundaries of most traditional monitoring systems. When utilized fully, the amount of traffic in the 10Gpbs network link is 2.5Gbytes in one second [6]. This means that the number of packets to be handled in one second is about 6.7 million; this amount of traffic cannot be easily processed by a single computer system. To cope with this large amount of traffic we need a more powerful, scalable and flexible monitoring system.

Network traffic monitoring and analysis is a fundamental task for understanding the behavior of the network. To date, various standards, monitoring methods and systems have been introduced by
many research groups and vendors. Active monitoring methods have been used to measure the performance of networks (such as round-trip delay, jitter, packet loss) and passive monitoring methods have been used to measure throughput of networks [1].

Sampling is a popular method that most of the existing monitoring systems have adopted to overcome this problem [2]. However, the sampling method is neither accurate nor adequate for some applications (e.g., usage-based billing, intrusion detection, service-level agreement management, customer relationship management). Another approach is by the adoption of purpose-built hardware [3]. Unfortunately, the development cost of such hardware approach is very high, and the hardware can become outdated quickly. ISPs would be required to replace them to meet the requirement as network bandwidth increases. Therefore, we need a solution that is flexible, scalable, and cost-effective.

This paper suggests such a solution. In our earlier work, we developed a passive network traffic monitoring system, called WebTrafMon [3, 4]. It could monitor traffic lower than the 100-Mbps links and was able to capture packets without losses. When we used it to monitor traffic more than 100-Mbps links, we encountered several problems. The amounts of incoming packets were beyond the processing capacity of the probe. And the required storage space for flow data increased linearly as the link speed increased. Also, the single analyzer took a long time to complete the traffic analysis tasks.

Thus, we had to come up with a new approach to solve the problem. First, we subdivided the monitoring processes into multiple phases and distributed the processing load over them by allocating a system for each phase. If the distributed load in each phase was still beyond the capability of the system, it could be composed of a cluster of systems. With this approach, we have developed a flexible and scalable network traffic monitoring and analysis system, called NG-MON (Next Generation MONitoring). NG-MON uses the passive monitoring method [6]. This paper presents the design and implementation of NG-MON.

The organization of this paper is as follows. The requirements of NG-MON are enumerated in Section 2 and the design of NG-MON is presented in Section 3. The implementation details of NG-MON and the performance testing are described in Section 4. In Section 5, we compare our approach with other approaches proposed thus far. Finally, concluding remarks are given and possible future work is mentioned in Section 6.

2. Requirements

The following are the major requirements we have considered in designing NG-MON [6].

**Distributed, load-balancing architecture:** With a single general-purpose computer system, it is impossible to monitor and analyze entire packets on a multi-gigabit network. So it is necessary to divide monitoring the tasks into several functional units and distribute processing loads. With respect to the distribution method, we considered the pipelined and paralleled methods.

**Lossless packet capture:** We need to capture all packets on the link without any loss in order to provide the required information to various applications.

**Flow-based Analysis:** When analyzing, it is better to aggregate packet information into flows for efficient processing. By doing this, packets can be compressed without any loss of information.

**Consideration of limited storage:** The amount of captured packets in high-speed networks is more than hundreds of megabytes per minute even if we aggregate them into flows [3]. An efficient method is needed for storing these large amounts of flows and analyzed data in the limited storage.

**Support for various applications:** It should be flexible enough to provide data to various applications in diverse forms. When a new application needs to use the monitored data of the system, it should be able to easily support the application without changing the structure of the system.
3. Design of NG-MON

In the design of NG-MON, the key features we have employed are pipelined distribution and load balancing techniques. In Figure 1, traffic monitoring and analysis tasks are divided into five phases: packet capture, flow generation, flow store, traffic analysis, and presentation of analyzed data. These five phases are serially interconnected using a pipelined architecture. One or more processing systems may be used in each phase to distribute and balance the processing load. Each phase performs its defined role in the manner of a pipeline system. This architecture can improve the overall performance. And each phase is configured with cluster architecture for load balancing. This provides better scalability, which is achieved by adding more processing systems to the cluster in each phase whenever the processing load exceeds the processing capacity of the cluster. That is, as the system monitors higher bandwidth links and thus more traffic would need to be processed, we can add more processing power whenever it is needed.

![Figure 1. Architecture of NG-MON](image)

We have also defined efficient communication methods between the phases. Each phase can be replaced with more optimized modules as long as they provide and use the same defined interface. This provides flexibility. Rather than using expensive server-level computers, we use inexpensive off-the-shelf PC-level computers. Since our solution is all software-based, as more processing power is needed one can simply replace existing hardware or add more wherever is needed. We believe this to be a very cost-effective and scalable approach.

In this distributed and clustered architecture, the time synchronization among cluster components is important. For example, using multiple probes (packet capturers) in the packet capture phase raises the issues of time synchronization among the probes for accurate time stamping. Also for on-line and real-time analysis the time synchronization among each phase is necessary. However, in the deployment of NG-MON, the subsystems of NG-MON would tend to be interconnected to each other in the same LAN segment. In the case of throughput analysis, which is our current analysis target, the throughput information of captured traffic data is merged into a long-period data over time. Thus, the timestamp at long-run traffic data disappears. Therefore, the time synchronization protocol like NTP [7] can give enough precision to synchronize the system clocks of the components. We use one system to synchronize the system clock from a satellite, and other systems use NTP to synchronize the system clock form this system.

In the following sections, we describe each phase in detail.

3.1. Packet Capture

In the packet capture phase, one or more probes collect entire raw packets passing through the network link. By using the splitting function provided by an optical splitter [8], all packets on the link can be directed toward multiple probe systems as illustrated in Figure 2. We can also use the mirroring function provided in network devices such as switches and routers for distributing traffic to multiple probes. Splitting and mirroring function can divide a high-speed traffic into several low-speed traffics that a general-purpose computer can handle. Each probe processes incoming packets...
and keeps the minimum packet header information that we are interested in. In Figure 2, the output of each probe is a collection of the packet header information that is derived from raw packets.

![Figure 2. Packet Distribution and Packet Capture](image)

Each probe has as many export buffer-queues as the number of flow generators. Each export buffer-queue is for the corresponding flow generator. The probe fills the buffers with header information using the 5-tuple based hashing over export buffer-queues. When this buffer-queue is full, the probe constructs a packet header message and then sends it to the next phase, the flow generator. The destination of the constructed packet header message is assigned among the flow generators as to the buffer-queues. Therefore, temporarily scattered packets in the same flow would be placed together into the same flow generator. One packet header message is composed of up to 50-packet header information. The format of raw packet header data is given in Figure 3.

The size of packet header data kept is 24 bytes for each packet. All the fields except Timestamp and Capture ID are extracted from IP and TCP/UDP headers of each packet. The Timestamp indicates the time when a packet is captured by a probe. The Capture ID indicates the probe which captured that packet for later use.

![Figure 3. Packet Header Data Format](image)

3.2. Flow Generation

There are various definitions about the flow [9, 10, 11]. In this paper, we use the most widely-used one, which defines the flow as a sequence of packets with the same 5-tuple: source IP address, destination IP address, protocol number, source port, and destination port. Figure 4 shows that the packet header messages from the packet capture phase are distributed over flow generators by assigning their destinations to the corresponding flow generators. The packets in the same flow can be scattered among probes. However by 5-tuple bashed hashing in the packet capture phase, all the packet header data in the same flow which are scattered among probes are destined to go to the same flow generator. The hashing mechanism in the packet capture phase also provides scalable use of flow generator. For lossless communication between a packet capture and a flow generator, the TCP is used.

A flow generator stores the flow data in its memory area for processing efficiency. When a packet header message containing raw packet header data arrives, the flow generator searches the corresponding flow data from its flow table and then updates it, or creates a new flow if one does not exist. Packets in the same flow are aggregated into the same entry of the table by increasing the
packet count and adding the length to the total packet size. The flow generator exports the flow data to the flow store when one of the following conditions is satisfied: when the flow is finished (if TCP, when a FIN packet received), the time has expired, or the flow table is full.

The flow data format is given in Figure 5. Besides the 5-tuple information, the flow data has several other fields such as flow start time and flow end time. The flow start time indicates the time when the first packet of a flow is captured by a probe, and the flow end time means the capture time of the last packet of the flow.

3.3. Flow Store

In our earlier work [4], we recognized that one of the bottlenecks of the monitoring process is the storing of flow data. Therefore, when the flow data is stored to the flow store, the load balancing should be considered. In Figure 6, the destination of the exported flow messages from flow generators is assigned among the flow stores in turn by a round-robin manner. The assigning period is determined by the transfer rate of export flow data, capabilities of the flow stores, and the number of flow stores. In this way, the processing load to store the flow data is distributed over the flow stores.

From Figure 6, in a single time period $t_1$ all flow generators send the flow data that is composed into flow messages to flow store #1. And in the next time period $t_2$, all flow generators send flow messages to flow store #2. In the same manner, at a certain time period only one flow store receives flow data and stores this flow information into the storage. At the same time, the other flow stores process requests from analyzers.
In our system, we separate write (i.e., insert) operations from database query operations performed by the analyzers. Insertion does not occur at the same time as other operations in a single flow store. Thus, traffic analyzers query databases of flow stores when they are not receiving flow data. An example is illustrated in Figure 6. At time t1, the flow store #1 receives flow data from flow generators and the flow stores #2 and #3 process the query from traffic analyzers. That is, the flow store concentrates on operation requests of one side at a time. Flow stores discard the flow data table when they are finished with analysis by traffic analyzers. Only the most recent flow data is stored in the flow store, so the flow store only requires a small, fixed amount of disk space.

The flow messages from a flow generator are sent by TCP communication to a flow store for lossless delivery of flow data. The flow store provides two kinds of methods to provide the stored data to analyzers. One is DB to provide various forms of information from stored data. The other is binary file for fast data delivery from a flow store to an analyzer.

There can be various traffic analyzers for supporting various applications after the flow store phase. This means that the flow store should provide an analysis API to various analyzers.

3.4. Traffic Analysis

In this phase, the traffic analyzer queries the flow data stored in the flow store according to the various analysis scopes. It sends query messages to the flow stores and makes various statistics from the response. If all the scope of analysis were put into one table, the size of a single table would be too large to manage. This was one of the problems we encountered in our previous work [5]. Therefore, we placed several reduced set of tables corresponding to each analysis scope, such as host analysis, protocol analysis, and temporal analysis.
For example, NG-MON Analyzer illustrated in Figure 7 provides the details on network throughput with host, protocol, and temporal analysis. And in order to provide a temporal analysis, the analyzer has a set of tables according to every time unit of minute, hour, day, month, and year. The maintenance of each temporal table follows the same round-robin store mechanism that is used in RRDTOOL [12].

Furthermore, it is impractical to store all the flow data into the time-series tables because of voluminous data and limitation of storage space. To reduce the storage requirement, we preserved tables with only the most significant N entries. Thus, the total size of database will have some degree of boundary. The analyzer fills up the tables simultaneously in a pipelined manner. If the reception time period of flow stores is 1 minute, there can be 60 tables for storing every 1-hour’s analyzed flow data. After updating the 1-minute table, the corresponding hour table gets updated. There should be these kinds of time-series tables for each scope of analysis in the analyzer, as illustrated in Figure 7.

3.5. Presentation

The presentation phase can provide an analysis to users about the traffic in various forms using the Web interface. Before developing an analyzer, we first had to determine analysis items to be shown in this phase. Then the traffic analyzer can generate the corresponding DB tables based on these items. That is, a different analyzer is required to support different purpose presenters. Because tables contain analyzed data which is ready to be shown, the time needed to create reports and HTML pages is very short, typically less than a second.

This phase also provides analyzed data to corresponding applications. Because the header information of all packets has been stored to the flow store after being compressed into flows, it can provide any information to applications in a flexible and efficient way. NG-MON can provide necessary information to the billing applications on IP networks, IDS systems, and so on.

4. Implementation of NG-MON

Using the design of NG-MON architecture presented in the previous section, we have implemented an NG-MON prototype system. This implementation has been deployed to monitor the campus backbone and Internet network traffic at POSTECH. The POSTECH campus backbone network is composed of 1 Gbps Ethernet networks. The Internet connection is composed of two 100-Mbps Metro Ethernet links.

We used TCP messages as the data format exported from the probe and flow generator, which is illustrated in Figure 8. This data format was influenced by Cisco Netflow format [13], but we simplified the record contents of each TCP message. In our first implementation of NG-MON, the messages were transmitted using the UDP socket, but we changed it to the TCP socket because UDP socket cannot guarantee reliable data delivery between the phases. The size of packet header data format is 28 bytes. In a single TCP message, a maximum of 51 packet records can be included. In the same way, the size of the flow data format is 32 bytes. Our flow generator can send up to 46 flows in a single TCP message of approximately 1500 bytes. These aggregated TCP messages with records reduces the network overhead and the number of messages to be sent.

![Figure 8. TCP message format used between phases](image)

4.1. Packet Capturer and Flow Generator

Figure 9 illustrates the implementation architecture of the probe in NG-MON. The probe is
composed of three main components: a packet info extractor, a pkt exporter and a timer. When the traffic of a high-speed link is distributed to multiple probes by the use of the mirroring function of a network device (such as a router or switch) or splitting function, the packets belonging to the same flow can be scattered among different probes. But it is necessary that the scattered packet header information in the same flow should be sent to the same flow generator. That is why we are using the same number of packet buffers in the packet info extractor module, as in Figure 9.

![Figure 9. Probe Architecture](image)

When a raw packet is captured, the header information of the packet is extracted and added to one of the packet buffers, which is decided by hashing with 5-tuple information of the packet. When a packet buffer is full, the pkt exporter sends the packet header message to the corresponding flow generator. The timer is used to check if there are any packets which have not been exported for a long time, but this situation rarely happens in a high-speed link with lots of traffic.

Figure 10 illustrates the implementation architecture of the flow generator in NG-MON. The flow generator has a buffer manager module which generates flows using three tables: two flow tables and one fragmentation table. The fragmentation table is used to reassemble fragmented IP packets. In the testing on our 1-Gbps campus backbone network, the average number of fragmented IP packets was 4,000 out of 1,200,000 packets for 1 minute. The ratio was 0.3%, which is not a negligible number. The captured fragmented IP packets may be out of order, retransmitted, or duplicate packets. Thus, reassembling fragmented IP packets is a considerable challenge. In the current version of our flow generator, we solved this problem by using a timer to check for the completion of reassembly. This scheme is very simple and fast but incurs a slight delay in inserting them to the main flow table. The slight delay (e.g., several seconds), however, does not cause problems in flow generation.

![Figure 10. Flow Generator Architecture](image)
The buffer manager inserts a packet header data to only one flow table at a time. Concurrently, the other flow table is handled by the flow exporter module. The two flow tables are switched alternately when the specified time expires, this time is 1 minute as default but can be changed. The timer module sends a SIGALM signal every 1 minute, then the assembled fragmented packets are inserted into the current flow table and the current flow table is replaced by another flow table. The flow exporter module, which is running in another thread, takes over the switched flow table and exports the flow data to the corresponding flow store.

The flow export bandwidth is very important because there is a limitation in processing flow data at a flow store and flow data delivery should be performed without loss. During 1 minute, the maximum number of flow data is 429,000 in our testing environment, which generated 9327 TCP messages. By using TCP socket we accomplished this lossless delivery but it takes more than 10 seconds in the worst case to send all the TCP messages to a flow store. That is the main reason we run the flow exporter in a different thread from the buffer manager and use a switching mechanism with two flow tables.

The flow record pool module and pkt record pool module store empty unused records that are created at the starting time of the flow generator. We used this pool mechanism to reduce the overhead of memory allocation and release during the flow generation execution.

4.2. Flow Store

The default flow export period is 1 minute. That is, every 1 minute the flow exporter module in the flow generator exports flows to the flow stores. This exporting process and the receiving process in the flow store should be finished before the next export starts. So there can be more than one flow store to receive flow messages during a certain export time in case that a single flow store cannot handle the entire flow messages during 1 second.

The flow store mainly performs two functions. One function is to receive flow messages from flow generators and to store them into a DB or binary file. We are using the MySQL Database for storage. The other function is to process query requests from the analyzer, which is the responsibility of MySQL DB. To handle entire flow data in a single flow store may be impossible, because the total size of flow data for 1 minute for 10 Gbps network is 540Mbytes [6]. So there should be several flow stores for storing the flow data during a single flow export period. We can calculate the number of flow stores needed for NG-MON by the following equation.

\[
S_{\text{num}} = N \times \lceil \frac{M + 1}{M} \rceil,
\]

where  \( N \) : the number of flow stores needed to store all flow data in DB during 1 minute  
\( M \) : the time in minutes needed to analyze the 1-minute flow data by the analyzer

![Figure 11. Interactions and Operations of Flow Stores](image-url)
The flow store performs only one function at a time; either the flow data receive operation or the query processing operation. For example, Suppose \( N=2, M=1.4 \) then \( S_{num} = 2 \times 3 = 6 \). Figure 11 shows the detail of this situation. During the time period \( t_1 \), the flow generator group sends flow data to the flow store group 1, to the flow store group 2 at time period \( t_2 \), and to the flow store group 3 at time period \( t_3 \), etc. At \( t_2 \), the analyzer group starts to analyze the flow data stored at flow store group 1, which takes 1.4 minutes. At \( t_3 \), the analyzer group starts to handle the flow store group 2, and still analyzing the flow store group 1. Figure 11 is a snapshot of data flow at \( t_3 \). At \( t_4 \), the flow generator group will send flow data to the flow store group 1 again. And the analysis of the flow store group will finish before \( t_4 \) and at \( t_4 \) the flow store group 2 and 3 are being analyzed.

In our implementation of the flow store, the flow data is kept for only a certain amount of time to reduce the required disk space. Thus, before the flow data is deleted, all analysis processing should be finished. Suppose that only recent 10 minutes’ flow data is stored, the amount of required storage size is 330 Mbytes in our 1-Gbps campus backbone network. In the case of 10-Gbps we can presume the required storage space in the flow store will be 3.3 Gbytes.

There are four kinds of tables, which are managed by the flow store. Those are flow minute table, source subnet minute table, destination subnet minute table and minute summary table. The flow minute table stores the flow data directly without any change. The two subnet minute tables store the flow data after categorizing the IP address in the flow data into subnet. And the minute summary table stores the total packet count and the total packet size according to protocols for each minute.

4.3. Traffic Analyzer and Presenter

The traffic analysis is a very crucial part of NG-MON and the pipelined architecture of NG-MON can support various analysis purposes, such as security, billing, network planning and traffic engineering. The current implementation of the analyzer is focused on the throughput analysis in temporal and compositional viewpoint. In temporal analysis the analyzer provides from real-time analysis to long-term analysis: real-time (minute), hourly, daily, monthly and yearly analysis. In compositional analysis, the analyzer provides the throughput according to sent data, received data and exchanged data of hosts and subnets. Also, the analysis according to protocol such as network layer, transport layer and application layer is provided. There were several problems in the traffic analysis of our previous work. One is that it took a very long time to analyze the traffic data because of the massive traffic data size. The other was that the content of the analysis was very limited. So the DB schema of our previous work was not suitable for analysis of the high-speed traffic link.

In NG-MON, we designed a new DB schema to overcome the problems. The philosophy of our

![Figure 12. DB Schema for Data-Sent Analysis](image)
DB schema design is that, in the analysis phase sub-tables corresponding to the content of analysis should be made to minimize the query processing time. We organized the content of analysis to eight categories: data sent, data received, data exchanged, subnet sent, subnet received, network layer, transport layer and application layer. The user interfaces are composed based on these eight categories. Thus, we made different tables for each category. Figure 12 shows the DB schema for data sent category. For the temporal analysis of each category, the same schema tables corresponding to hour, day, month and year are created. Figure 12 shows 5 tables; a sent table at 10:11 2002-08-10, a sent table at 10:10 2002-08-10, a sent table at 09:00 2002-08-10, a sent table at 2002-08-09 and a sent table at 2002-07. To reduce the storage space we discard past tables. That is, we keep only recent 3-hour’s minute tables, recent 3-day’s hour tables and recent 3-month’s day tables.

Another scheme we applied to the analyzer is to keep only top-N significant data. From the test on our campus Internet link, we found that among the 55581 flows generated for 1 minute the number of a single packet flows was 10063 (about 18%). And the number of flows, which are composed of less than 10 packets, is 26339 (about 47%). The maximum number of packets in a flow was 2364. From this result we know the significant flows (i.e. the flows that contain large number of packets) is very small, so we do not need to keep all the flow data in a table. Discarding insignificant flow data results in saving considerable storage and processing time.

Figure 13 is a sample screen shot of the NG-MON user interface provided by the presenter module. We used PHP and Apache web server to develop the presenter module. In our NG-MON presenter, the target response time should be less than 1 second after an administrator sends a request. With the help of sub-table and top N list scheme used in the analyzer module, we were able to achieve this goal. Figure 13 shows top 10 lists of data sent host for certain 1 minute. The graph in the right upper corner indicates the variation of total bandwidth and packet size during the specific hour.

Figure 14 shows a detail sent-data view of a specified subnet. From this window, the administrator can view various statistics about the subnet; protocol distribution in pie-chart and temporal distribution of total bandwidth the specified subnet sent during the specified time period. Also, the details about traffic information sent from the specified subnet are displayed in the table form.
4.4. Implementation and Testing Environment

Table 1 shows the development tools and system configuration we used for our prototype implementation. For capturing packets, we used the pcap library [14] to capture all of the raw packets in the promiscuous mode. We have used the C language because of the execution performance in the phases of packet capture, flow generator, flow store and analyzer. The presenter was developed with PHP for web-based user interface and jpgraph php library [15] for various useful graphs. To store the flow data and analyzed data at flow store and analyzer we used the MySQL DB. There are some other tools to store flow data such as the ARTS format [16]. But we chose to use DB instead of ARTS, because our NG-MON needs to show the information with various forms from the collected traffic and requires a more flexible and extendable data storing method. The development of NG-MON was performed on Pentium-III PCs running on Linux.

We deployed our NG-MON prototype in our campus backbone network that is composed of multiple 1-Gbps Ethernet links. During 1-minute capture of packets, the 100,000 packets (1,400,000 maximum) were captured and 60,000 flows (420,000 maximum) were generated. We used three systems to monitor and analyze the traffic. One for probe module and flow generator module, and another for the flow store module. The third one was used for the presenter and analyzer. The CPU load of each system was less than 50%. The bandwidth of tested is not very close to our target bandwidth, which is a multi-gaga bit network. But from the testing in 1-Gbps network, we were able to verify that our NG-MON has a very flexible and scalable architecture.

Table 1. Development Tools for NG-MON
5. Related Work

Table 2 compares NG-MON with other passive network monitoring systems. Ntop [17] is a monitoring software system that provides a detailed protocol analysis and a graphical user interface. However, Ntop is not suitable for monitoring high-speed networks from our experience in deploying at an operational network. It cannot scale beyond monitoring a 100-Mbps link.

FlowScan [21] is a NetFlow analysis software system that uses cflowd [22], RRD Tool [12], and arts++[16]. It can only process NetFlow [13] data format and is not suitable for monitoring high-speed networks either. Ntop and FlowScan are appropriate for analyzing relatively low speed networks such as a WAN-LAN junction or a LAN segment. Our approach of distributing the processing load may be applied to Ntop and FlowScan to improve their processing capabilities.

CoralReef [19] is a package of libraries, device drivers, classes, and applications for passive data collection and analysis, developed by CAIDA [20]. Its merits are flexibility and the power of its libraries and components for building new applications. CoralReef can receive from files for offline analysis, or from libpcap for analysis of IP traffic carried over any physical medium or datalink protocol, or from device drivers of specialized collection hardware such as Apptel POINT cards and DAG cards [23, 24]. However currently it can monitor up to OC-12 network links. With respect to load distribution, only CoralReef suggests a separation of flow generation and traffic analysis, but without consideration of clustering of processing systems in each stage.

<table>
<thead>
<tr>
<th>Table 2. A Comparison of NG-MON with Related Work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>Ntop</td>
</tr>
<tr>
<td>FlowScan</td>
</tr>
<tr>
<td>CoralReef</td>
</tr>
<tr>
<td>Sprint IPMon</td>
</tr>
<tr>
<td>NG-MON</td>
</tr>
<tr>
<td>Raw Traffic, NetFlow</td>
</tr>
<tr>
<td>NetFlow</td>
</tr>
<tr>
<td>Raw Traffic</td>
</tr>
<tr>
<td>Raw Traffic</td>
</tr>
<tr>
<td>Raw Traffic</td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td>Packet Trace</td>
</tr>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
</tr>
<tr>
<td>&lt;&lt;100 Mbps</td>
</tr>
<tr>
<td>&lt;&lt;155 Mbps</td>
</tr>
<tr>
<td>&lt;&lt;622 Mbps</td>
</tr>
<tr>
<td>10 Gbps</td>
</tr>
<tr>
<td>10 Gbps</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Hardware+ Software</td>
</tr>
<tr>
<td>Hardware+ Software</td>
</tr>
<tr>
<td>Software</td>
</tr>
<tr>
<td>Sampling used</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Yes(device)</td>
</tr>
<tr>
<td>Configurable</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>No but Configurable</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
</tr>
<tr>
<td>On-line</td>
</tr>
<tr>
<td>On-line</td>
</tr>
<tr>
<td>On-line</td>
</tr>
<tr>
<td>Off-line</td>
</tr>
<tr>
<td>On-line</td>
</tr>
</tbody>
</table>

Sprint’s IPMon project [18] developed monitoring systems, called IPMON systems, for collecting traffic traces, which is used for off-line analysis after transferring to a laboratory. Their approach uses purpose-built hardware to assist the packet capturing and processing, known as the DAG card. The DAG card captures from ATM and PoS links and extracts the IP packets, then generates a timestamp for the packet using DUCK\(^{1}\) and transfers to main memory in the PC. They try to support OC-192 links with the current PCI technology by using a compression technique. But the IPMON system tends to highly depend on the performance of the DAG card.

Our NG-MON has been developed for monitoring high-speed IP networks. It takes raw traffic packets as input, and analyzes captured data online and then generates various throughput related data. NG-MON is a software-based solution (i.e., does not depend on any specific hardware), which can be easily installed and run on a variety of Unix and Linux platforms. NG-MON does not use sampling for capturing packets without any loss. However, the system configuration user interface allows the system to be configured to capture packets using sampling if needed.

\(^{1}\) Dag Universal Clock Kit
6. Conclusion and Future Work

In this paper, we have presented the design and implementation of NG-MON, a scalable and flexible monitoring and analysis system for high-speed IP networks. NG-MON adopted a pipelined and parallel architecture for achieving our goal. NG-MON, with its pipelined and parallel architecture, can process packets without any loss. Multi-gigabit networks generate a lot of traffic and thus the amount of data generated from packet capturing is incredibly large. Our packet and flow processing method requires a small, fixed amount of disk space on each flow store. We have implemented an NG-MON system and verified that our system has flexible and scalable architecture by deploying it in our campus backbone network. We plan to test our system on an ISP network in Korea, where multi-gigabit networks are widely deployed throughout the country.

NG-MON can play a major role in providing the necessary information to multitudes of applications. For example, it can be used as the basis for billing on IP-based applications (such as VoIP, Internet access). It can be also used as the basis for intrusion detection where capturing all packets is essential. Customer relationship management (CRM) is a hot topic for ISPs these days. NG-MON can provide the useful user usage information for such purpose.

References