

# Rate-based Internet Accounting System Using Application-aware Traffic Measurement

TS Choi, CH Kim, SH Yoon, JS Park, HS Chung, BJ Lee, HH Kim,  
TS Jeong

Electronics Telecommunications Research Institute  
E-mail: {[choits](mailto:choits@etri.re.kr), [kimch](mailto:kimch@etri.re.kr), [shpyoon](mailto:shpyoon@etri.re.kr), [chunghs](mailto:chunghs@etri.re.kr), [jspark](mailto:jspark@etri.re.kr), [bjlee](mailto:bjlee@etri.re.kr), [hhkim](mailto:hkim@etri.re.kr),  
[tsjeong](mailto:tsjeong@etri.re.kr)}@etri.re.kr

APNOMS2003

## Abstract

As the Internet is quickly evolving from best-effort networks to business quality premium networks, billing based on the precise traffic measurement becomes an important issue for Internet Service Providers (ISPs). Billing settlement is necessary not only between ISPs and customers but also between ISPs. Currently, most ISPs use a flat rate charging policy. Besides the degree of difficulty in deriving appropriate charging policies agreeable by a concerned party, there are substantial technical challenges to come up with a good usage-based accounting system. Usage-based accounting depending on IP packet header information only is no longer sufficient due to highly dynamic nature of the development and the use of the Internet applications such as peer-to-peer, streaming and network games whose traffic takes substantial amount. They are using port numbers dynamically and even several applications can use the same port number. Packet fragmentation and asymmetric routing contribute to this difficulty as well. In this paper, we propose a high performance, adaptable, configurable, and scalable application-aware passive traffic measurement and analysis system which can achieve very accurate usage-based accounting. This system consists of dedicated traffic capturing card(s) to improve its performance, an agent which generates flow and packet records based on application signatures, and a server which classifies applications precisely with our novel classification algorithm and correlates sub-flows of incoming and outgoing directions if necessary. It then analyzes their traffic usages in a detailed sub-flow level (e.g., HTTP request/reply, HTTP reply/request, FTP control request/reply, FTP data request/reply, KAZZA down sender/receiver flow etc.) within an application.

## Introduction

- As the Internet is evolving from the best-effort to business quality premium network
  - ◆ a strong demand to measure precise usage of network resources is emerging
  - ◆ It can be used for traffic profiling, usage-based accounting, and/or traffic engineering, QoS monitoring, usage-based pricing, and intrusion/security anomaly detection
- However, the related research community is facing significant challenges to provide technically viable solutions due to
  - ◆ highly dynamic nature of the development and the use of the current Internet applications
  - ◆ Traffic asymmetry
  - ◆ packet fragmentation
- There have been many research and development efforts in the field of traffic measurement and analysis for the past decade.
  - ◆ CAIDA's OCxmon, Tcpdump, Ethereal, Cisco's Netflow, CAIDA's CoralReef, Flowscan, NetraMet, and SPRINT's IPMon are some examples
  - ◆ However, accurate traffic usage accounting in the Internet requires a cleverly combined mechanism of per-packet payload inspection, flow-based analysis, correlation of associated sub-transaction flows, and wire-speed packet capturing performance
- In this paper, we propose a high performance, adaptable, configurable, and scalable application-aware traffic measurement and analysis system

Historically, the Internet has provided a best-effort service which means that it treated all packets equally in terms of both service quality and pricing. As the Internet is evolving from the best-effort to business quality premium network, a strong demand to measure precise usage of network resources is emerging. Although there is still prevalent support of flat rate charging for the usage of network resources, the emerging demand is getting a momentum these days. Rationale for such a movement is maximization of network performance, service quality and profitability. However, the related research community is facing significant challenges to provide technically viable solutions due to the nature of the Internet.

The main difficulties come from highly dynamic nature of the development and the use of the current Internet applications. Traditionally, Internet traffic was dominated mostly by the client-server type of applications such as WWW, ftp, telnet, etc. However, this characteristic has been changed significantly when new applications such as peer-to-peer, multimedia and network game applications were introduced. These applications use a range of port numbers or dynamically allocated ones for their sub-transactions. (e.g., EDONKEY uses 4661, 4662, 4665, 6667 and RTSP streaming application allocates a port number dynamically for a stream data transfer) This means that distinguishing flows based on a port number and other header properties is not safe and accurate enough. Thus, application header information and application signature matching in a packet payload are needed for the precise measurement.

Also since the Internet is asymmetric in nature, an application transaction consists of multiple sub-transactions, a series of Requests and Replies, which may follow different routing paths. Accurate usage accounting for such a case requires distributed monitoring and correlation of sub-transactions which appeared in different paths. Another problem can be caused by packet fragmentation. It is reported that not a small portion of the Internet backbone traffic is transported in a fragmented state [1], and this tendency may deepen because more and more encapsulated services, such as IPv6 over IPv4, IPsec, etc., is getting popular. Therefore, per-packet and/or packet payload inspection based usage accounting only do not solve the problem.

There have been many research and development efforts in the field of traffic measurement and analysis for the past decade. CAIDA's OCxmon [2], Tcpdump [3], and Ethereal [4] can be used to capture full packets and analyze them off-line for the purpose of various traffic profiling. They are per-packet analysis system. Cisco's Netflow [5], CAIDA's CoralReef, Flowscan, and NetraMet [6], and SPRINT's IPMon [7] are flow-based traffic analysis system. They all claim that flow-based usage accounting functionality is supported. However, accurate traffic usage accounting in the Internet requires a cleverly combined mechanism of per-packet payload inspection, flow-based analysis, correlation of associated sub-transaction flows, and wire-speed packet capturing performance. Technically speaking, it is a truly challenging task with the state-of-the-art technologies.

In this paper, we propose a high performance, adaptable, configurable, and scalable application-aware traffic measurement and analysis system which can achieve very accurate usage-based accounting.

## Internet Application Classification

- Type S: **Simple Application Type**
  - ◆ for an application which uses a well-known port number or which uses a registered port number but are popularly used
- Type P: **Payload Application Type**
  - ◆ for an application which uses a registered or ephemeral port number but requires payload inspections for precise classification
- Type R: **Reverse Application Type**
  - ◆ for an application which uses a registered or ephemeral port number but requires comparison with a correlated reverse flow for the precise classification
- Type C: **Complex Application Type**
  - ◆ for an application which uses a dynamic port number assignment
- Type U: **Unknown Application Type**
  - ◆ for applications which do not use registered port numbers and do not belong to any of the four types mentioned above

Our architecture is unique in that we classified Internet applications into five categories based on their behavior. An application is classified into not only per application level but also per sub-transaction level within an application. Each application comprises multiple sub-transactions such as request, reply, and data transfer. The objective to classify applications into such fine granular levels is to account precise traffic usage and to reduce the unknown traffic volume as much as possible.

### • Type S: Simple Application Type

This type is for an application which uses a well-known port number or which uses a registered port number but are popularly used. This classification is similar to the port-based traditional method such as Cisco's Netflow. Per packet application recognition is possible and payload inspection is not required. Also traffic forwarding direction doesn't influence application identification. Some examples are Telnet, HTTP using port number 80, FTP (non-passive), and SMTP, etc. There exists, however, a rather higher probability of misrecognition.

### • Type P: Payload Application Type

This type is for an application which uses a registered or ephemeral port number but requires payload inspections for precise classification. Since an application signature can appear in a random position within an outstanding session, the entire application flow has to be assembled before the payload can be examined. The flow consists of a flow record and a set of packet records which belong to the flow. The direction doesn't matter as the case in the simple type. Some example applications are HTTP with port 8080, EDONKEY file transfer transaction, and MS messenger file transfer transaction. HTTP with port 8080 should contain "HTTP" or "GET" in the first few packets of the flow in order to be the WWW application.

### • Type R: Reverse Application Type

This type is for an application which uses a registered or ephemeral port number but requires comparison with a correlated reverse flow for the precise classification. For example, a TCP request flow is typically associated with a response flow. And most of the response flows consist of 40-byte SYN ACK packets. By simply looking at the port number of them, it is hard to identify the application to which it belongs. To make precise measurement of such TCP based applications, the response flows have to be identified in relation with request flows. These reverse direction response flows are classified as type R. The only correct way to identify such response flows is actually comparing two flows' source & destination addresses, port numbers and most importantly accurate timestamps. Without this process, we cannot guarantee a particular flow is the right one by just looking at its 5-tuple IP header information.

### • Type C: Complex Application Type

This type is for an application which uses a dynamic port number assignment. To recognize such an application, we need to find an inducing control flow of that flow. For example, many streaming applications and passive FTP behave like this way. During a control session, a server sends dynamically assigned port number to a client. Then either the server or the client uses it to transfer user data. In this case, the flow used to transfer user data is classified as a type C.

# Application Recognition Configuration Language (ARCL)

```

application WWW {
  port_rep_name HTTP port 80 protocol TCP{
    decision_group HTTP_REQ_REP_ACK {
      src_port >= 1024
      dst_port == 80
    }
    decision_group HTTP_REQ_REP_ACK {
      src_port == 80
      dst_port >= 1024
    }
  }
  port_rep_name HTTP_ALT port 8080 protocol TCP{
    src_disc_pattern=="HTTP" in pkt 0-2 at byte 0 - 4
    ( dst_disc_pattern=="GET" in pkt 0-3 at byte 0 - 10 ||
      dst_disc_pattern=="POST" in pkt 0-3 at byte 0 - 10 )
    decision_group HTTP_ALT_REQ_REP_ACK {
      src_port >= 1024
      dst_port == 8080
    }
    decision_group HTTP_ALT_REP_REQ_ACK {
      src_port == 8080
      dst_port >= 1024
    }
  }
}

application EDONKEY {
  port_rep_name EDONKEY_DOWN port 4662 protocol TCP{
    dst_disc_pattern=="0xe33d000000" in pkt 2-3 at byte 0 - 4
    decision_group EDONKEY_DOWN_REQ_REP_ACK {
      src_port >= 1024
      dst_port == 4662 ~ 4666 || 4242 || 4224 || 4660 || 5555
    }
    decision_group EDONKEY_DOWN_REP_REQ_ACK {
      src_port == 4662 ~ 4666 || 4242 || 4224 || 4660 || 5555
      dst_port >= 1024
    }
  }
}

application FTP {
  port_rep_name FTP port 21 protocol TCP{
    src_ref_pattern=="r/227 Entering Passive Mode
\\(\\d{1,3},\\d{1,3},\\d{1,3},\\d{1,3},(\\d{1,4}),\\d{1,4})\\)/$src_port =
atoi($1)*1024 + atoi($2)" in pkt any at byte 0-35 induce
FTP_DOWN_P
    decision_group FTP_REQ_REP_ACK {
      src_port >= 1024
      dst_port == 21
    }
    decision_group FTP_REP_REQ_ACK {
      src_port == 21
      dst_port >= 1024
    }
  }
}

```

APNOMS2003

4

In the previous slide, we described how to categorize the Internet applications into a set of distinctive types. Another important issue is then how to actually capture and analyze the contents of packets based on the above classification criteria. This involves the development of a high performance packet capturing device, a software or hardwired filter which inspects the contents of packets and aggregates them into meaningful flow information, and server software which classifies them into the distinctive application types and analyzes their traffic usage. Considering highly dynamic nature of the development and the use of Internet applications, an adaptive and extensible content filter is essential component.

For this purpose, we propose ARCL which succinctly describes the way to capture and analyze the contents of packets. The language is simple and effective in that the system can be swiftly reconfigured to detect and recognize unprecedented or modified applications without the developer's writing and distributing extension modules for processing the newer applications, which usually results in fairly long period of shrunk recognition coverage even after detecting and analyzing the new applications.

The above example script illustrates the usage of this language. Due to the space limit, we omit a syntax represented in BNR format of this language. It shows three applications: WWW, FTP passive mode, and EDONKEY. First application, WWW, represents two application flows, HTTP with 80 and HTTP with 8080. As mentioned before, the former belongs to a simple type, thus it doesn't have to specify source or destination content signatures. The latter is P type which requires content signature which is specified as src\_disc\_pattern and dsc\_disc\_pattern. P2P application EDONKEY is P type, thus both port number and payload filter is specified together. Passive FTP is C type and a dynamic port number is specified in src\_disc\_pattern. Also we also divided each application into a number of sub-transaction flows to profile each traffic volume accurately.

## Flow Definition Extension for Application aware Traffic Measurement

0	8	16	24	32
Protocol	Flag	Record Number		
Start time (sec)				
Start time(MicroS)				
End time (sec)				
End time(MicroS)				
Source IP address				
Destination IP address				
Source Port (ICMP : Type, 0 in case of IP)		Destination Port (ICMP : code, 0 in case of IP)		
Number of Packets				
Number of Bytes				
Time Stamp (resolution : MicroS)				
Total Length (Packet)		Packet record length		
TOS	TTL	TCP Flags(padding if not TCP)	Flag	
Payload (variable length)				

APNOMS2003

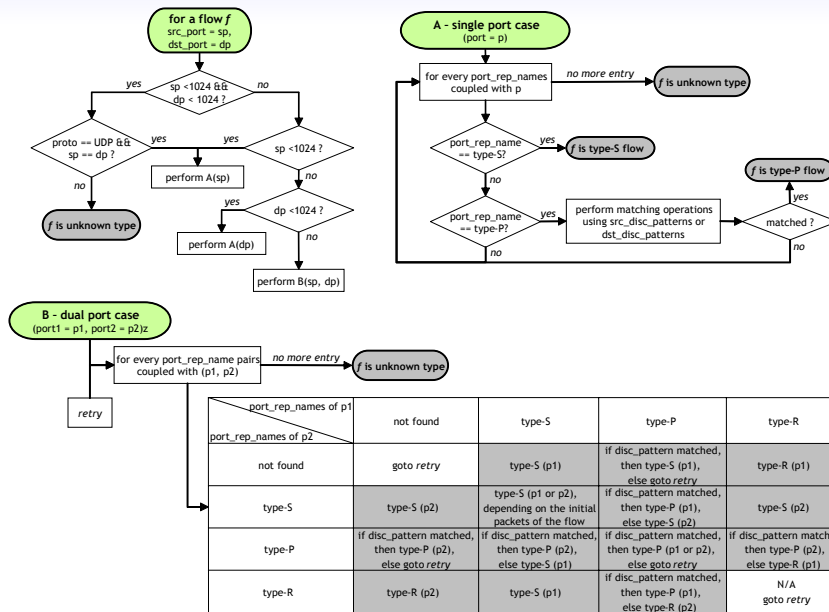
5

Once each packet is examined and classified into the right type, captured information has to be aggregated into a flow and packet records for an analysis process. Typically, flow-based monitoring system generates flow records only. Our system adds packet records and associated payloads which include signature information. Note that we are not capturing all packet contents but the ones with application signatures only. Analysis server uses this information for the detailed analysis of traffic usage accounting per sub-transaction flow and eventually an aggregated application flow. We decided such an extension because none of the existing flow definitions serve our purpose. These records play a crucial role for the accurate application recognition.

Above figures show our flow and packet record formats. Besides 5-tuple IP packet header information, we add additional fields: flag, record number, number of packets and number of bytes. Flag identifies a long-term flow when flows are generated by an active timeout timer. Currently two bits are used: Last Flow Record (LFR) and Previous Record Existence (PRE). LFR indicates the last flow and PRE indicates intermediate flows of the long-term flow. Our capturing device is capable of time stamping in microsecond level, which is provided by a GPS module. Source and destination port fields are used to identify TCP/UDP port numbers and ICMP type/code if the protocol is ICMP. Number of packets and bytes fields represents the number of packets and bytes within the flow.

In packet record, the timestamp indicates that the elapsed time since the start of this flow. Total length is the length of IP packet and packet record length is the length of this packet record. TOS, TTL, and TCP flags are imported from the IP and TCP headers. Two flags bits are used. Last Record (LR) means the last packet of the flow. Payload bit tells whether payload is attached in this record or not. Actual payload is attached whenever it contains matched application signature(s).

# Classification Algorithms



APNOMS2003

6

In this slide, we describe an algorithm to classify the captured packets into a set of application types and to perform a detailed traffic accounting process. It can be divided into two main parts: one performed by an agent and another by a server. Algorithm in an agent is in charge of fragmented packet assembly, payload pattern matching with given application signatures, and flow/packet records generation. On the other hand, the algorithm in a server accumulates traffic statistics per application flows and maps them into autonomous system (AS) numbers and country codes.

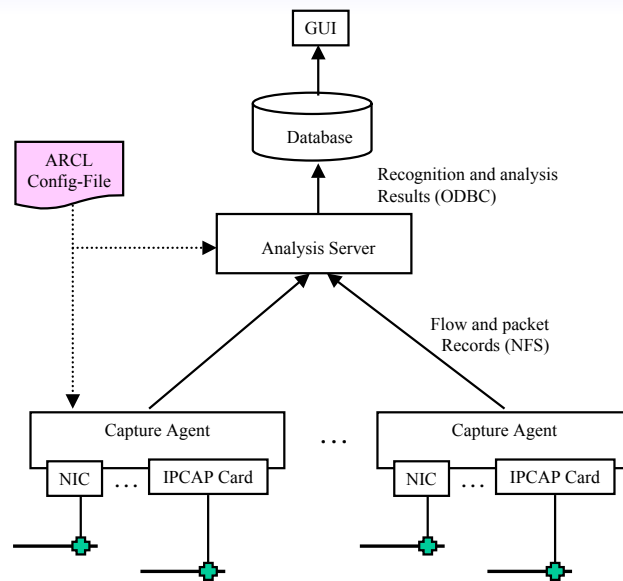
For the fragmented packet processing, each packet is checked whether it's a fragmented packet. If so, it checks the protocol number. If the packet isn't TCP, UDP, or ICMP, it is considered as unknown packet and passed onto flow generation process. If the packet is either TCP, UDP, or ICMP, it is checked whether it is the first fragmented packet. If true, the packet header information is stored for the further processing and stored in the FIFO queue. Also any fragmented packets which have arrived earlier is checked and stored in a right order. If it is not the first packet then fragmented packet header table is consulted. If the needed information is not present, timer is set until the necessary packet arrives. If the timer times out then the packet is treated as ill-formed packet and discarded. If the needed information is present, the appropriate next steps are performed.

The payload pattern matching processing is performed during the flow/packet record generation. A configuration file prepared with ARCL is used for this matching. For the flow/packet record generation, the algorithm looks at each packet and checks if it belongs to the existing flow. If so, perform protocol check and pattern matching and updates timestamp and flow statistics. It also generates a packet record and attaches a payload if necessary. If it is a new flow packet then a new flow record is generated and similar to the continuous flow record generation steps are repeated.

The server algorithm consists of two parts: pre- and post-processing. Pre-processing algorithm visits each flow and generates complete S- and P-type flow tables and R- and C-reference tables. During this process, it consults ARCL configuration file for the pattern matching and behaves differently depending on the status of the flow, that is, a continuous flow or not. Post-processing is mainly contributed to identifying R- and C-type flows, generates various usage-based flow accounting tables: AS accounting, Country accounting and Link accounting tables.

More details are described when agent and server components are explained. The above figure is a flow chart elaborating the key idea of the two steps to map a flow to a port\_rep\_name. The essence of the first resolution is that a well known port possesses a higher priority than a registered port in terms of the power of influence. This is intuitive because when a port of a flow belongs to the well known range and the other to the registered range, the well known one is the effective service port and the other is ephemeral. The key point of the second resolution is that the order of the priorities among the recognition types is type-P, type-S, and type-R in descending order; Type-C flows do not compete with the others.

# System Architecture Overview



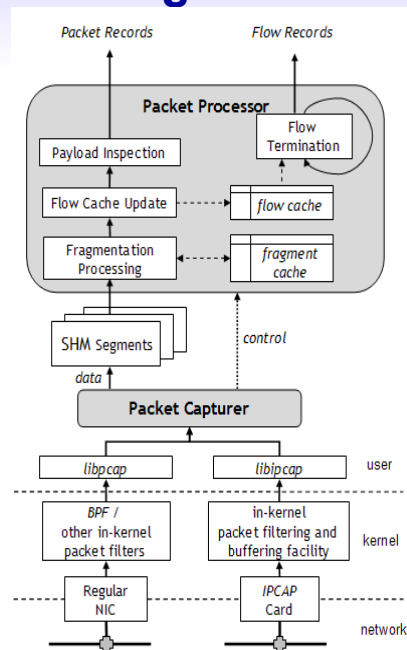
APNOMS2003

7

Figure above shows a bird's-eye view of the proposed architecture. Capture Agent (the "Agent") captures packets which are tapped and transferred to the Agent by an electronic or optical signal splitter. If service disruption is not allowable, port mirroring at a packet switching node can act as a substitute. Upon receiving raw packets the Agent at first filters out non-IP packets. Then, by processing the remaining IP packets, the Agent composes flows and generates two types of records that compactly describe the flow and packet information respectively. TCP, UDP, and ICMP packets are properly handled during the flow generation process, but for the sake of obtaining statistical information, the other types of packets are classified to a single special flow whose port numbers are zero. For ICMP packets, port numbers are substituted for ICMP types and codes. Although the Agent can monitor multiple physical links, planning operational capacity should be based on the Agent platform's performance and traffic volume on the target links. Analysis Server (the "Server"), at first, classifies the flow and packet records into one of the four types on the basis of source and destination ports. The specific processing routines that result from instantiating the recognition methods classify candidate flows of each type to corresponding applications. Finally, by means of chain-based rule matching, each packet in a flow is further classified to subgroups of applications. The Server processes records transmitted from multiple Agents in a comprehensive way. A type-C flow can take place on several independent links that are different from the link on which the inducing flow was monitored. A type-R flow, likewise, can occur on a link other than the coupled reverse link of the original type-PI flow because of the asymmetry of Internet routing. Considering these possibilities, the Server should keep the records from multiple Agents together. Recognition results are maintained in a database, and can be easily retrieved and represented by a GUI. In the meantime, an ARCL file configures the way the Agents and the Server operate in all of the above processes.

The Agent is designed to be able to operate on a real-time basis because reducing the capture-off period as much as possible is essential in most operational cases. The operation of the Server, however, may not be constrained by time. The recognition operations can be conducted on a batch basis. Nevertheless, if Agents are not too highly aggregated onto a Server and the Server possesses enough computing resources, the Server can also perform recognition tasks on a real-time basis. For very high-speed links, however, the Agents and Server may be severely overloaded, and accordingly, the way the Agents and the Server operate must not be tightly coupled.

# Agent

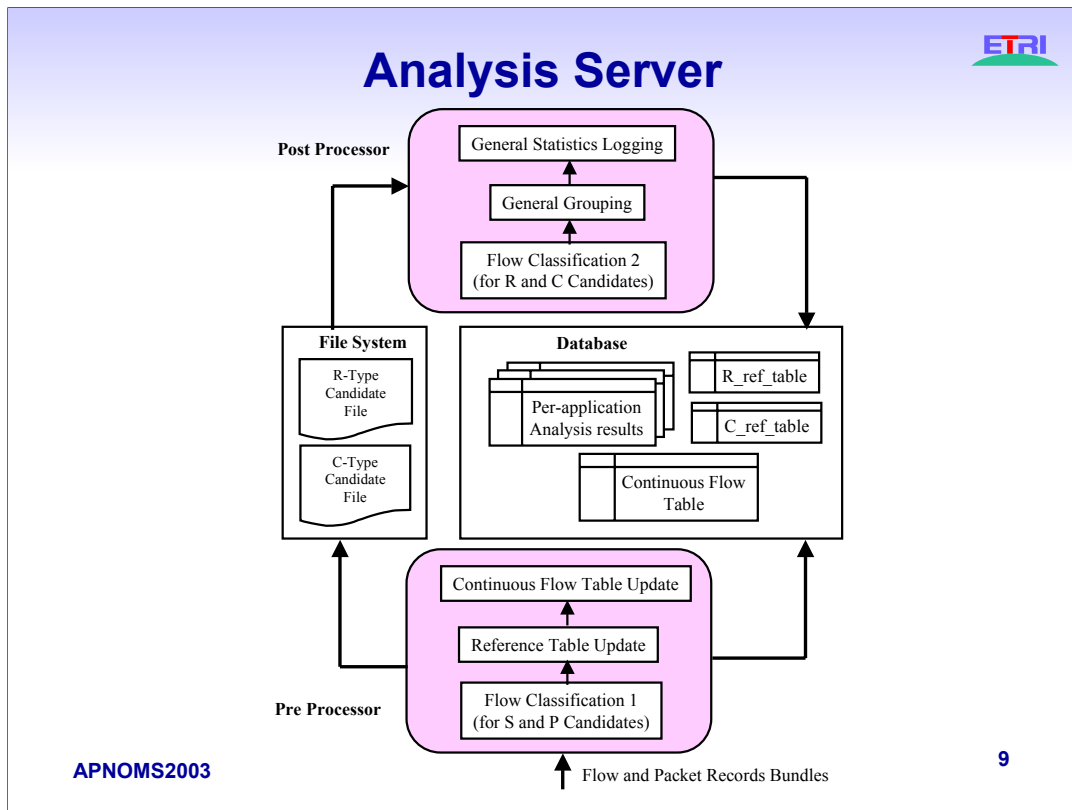


APNOMS2003

8

The detailed internal structure and procedure of the Agent are shown in the above figure. The Agent is composed of two main modules: Packet Capturer (PC) and Packet Processor (PP). A PC and PP pair handles a logical IP link, and a single Agent machine can carry on multiple instances of the PC and PP pair. The PC's primary goal is copying raw IP packets from the kernel to the user area with as little loss as possible. Packets transmitted to the PP are, at first, processed by the fragmentation handling routine that restores parts of each fragmented packets. The flow cache update routine performs two functions. For the first packet of a new flow, it creates a new entry in the flow cache, and for the other packets, it updates the existing entry's values. The flow termination routine is invoked every unit time and terminates flows on the basis of active, inactive, and TCP-FIN timeout. The last routine operating on a per-packet basis is the payload inspection routine. This routine inspects each packet's payload to search for patterns of type-P and type-C methods. In order to reduce the amount of records transmitted to the Server, only a pattern-matched packet's payload is appended to the packet record.

Although we also designed and implemented an intelligent packet capture (IPCAP) card suited for our specific architecture and needs, a detailed introduction of the hardware and its supporting kernel facilities is out of the scope of this paper. Nevertheless, because the capture agent is also designed to be able to operate on *bpf* and *libpcap* [8], common PC boxes or workstations with a UNIX operating system can be easily utilized.



The Server's internal structure and procedures for supporting the classification methods are shown in the slide above. The recognition and analysis is performed through a pre-processing and a post-processing routine. When there are several record files gathered from multiple links, the pre-processing routine is invoked for each of the file, while the post-processing routine is invoked once every server-activation interval (record file transmission interval) after pre-processing all of the record files. Type-S or type-P flows can be assertively recognized by the first flow classification routine because such flows do not require any preceding flow's information. Such flows are immediately mapped to a port\_rep\_name and their information is permanently updated on the corresponding application's table. The packets contained in such flows are also classified into corresponding decision\_groups on the basis of the matching rules specified in an ARCL file. Meanwhile, the pre-processing routine also identifies flows which may induce type-C and type-R flows and leave the inducing flows' referential notes in C\_ref\_table and R\_ref\_table respectively. The referential tables are common to all links. On the other hand, because candidate type-R flows can also be identified by the first classification subroutine with some uncertainty, such a flow is specially preserved in the R\_candidate file. All of the residual flows are finally preserved in the C\_candidate file, although some of them may be ended up being recognized as unknown applications. When a flow is the first sub-flow of a long-lived flow, the classification result of the flow is maintained in a continuous flow table, so that the successive flows might not cause repetitive operations later.

When all of these pre-processing subroutines are carried on for all of the record files falling under the given interval, the remaining settlement is required to the post-processing subroutines. The first task of the post-processing is classifying the flows preserved in the R\_candidate and the C\_candidate file. When a flow in the R\_candidate file is matched for a referential note in the R\_ref\_table, the flow is assertively classified to a type-R flow and, accordingly, mapped to a port\_rep\_name. A similar operation for the flows in the C\_candidate file is performed. An assertively recognized flow's information is permanently updated on the corresponding application's table, and the subsidiary packets are also assertively classified. By this routine, flow and packet recognition operations are completed. The remaining subroutines are for various statistical processing, such as country and AS-based grouping, a link or link set-based grouping.

# Implementation



APNOMS2003

10

We have implemented our prototype system based on the system architecture and features described in the motivation section. The agent and server software were implemented over Linux 8.0 platform using C and C++. We are using MYSQL database system to store node and link information and various classifications and accounting tables. GUI is web-based and PHP scripts are used for interaction with DB system and graphical representations of accounting information. The above figure shows a snapshot of our GUI.

Agent and server software is running in the separate systems. The agent system is a Pentium IV 2-CPU PC server which has 2 GB main memory and 80 GB hard disk space. It can hold upto two 64bit PCI II type capture cards. The server system is the same type of PC-server with 800 GB hard disk space and DB system is running in the same box.

For the packet capture and system performance improvement, we have designed and implemented several features. The capture card has two buffers (DMA and session buffers) and a filter. When packets are captured, it is stored in the DMA buffer and copied into session buffer periodically. Filtering is applied in between. Currently, we support MAC-address based filtering only to differentiate packet forwarding directions and is planning to enhance to support more generic pattern-based filtering. Session buffer resides in the kernel space and can be as big as 2 GB. Agent software periodically requests a bundle of raw packet data and processes it in the main memory to generate flow/packet records. The generated records are stored in the hash table until it holds up to one-hour amount of data and then it flushes into a flow/packet record bundle.

We have been deploying the prototype system at our company's internet (ETRINET) access link toward two major ISPs in Korea. There are two T-3 links and we are currently monitoring one link of both incoming and outgoing directions. The number of Internet users in ETRINET reaches around 2,500. Outgoing and incoming traffic is simultaneously fed into an Agent and to a Server from a mirror-enabled Ethernet switch. The average profile of the consolidated traffic is 46.52 Mbps, 5.325 Kpps, and 174.1 fps, and the peak profile is 148.23 Mbps, 18,242 Kpps, and 1,359 Kfps. Two logical links (outgoing and incoming) are monitored by a single Agent; in the Agent, two instances of a PC and PP pair operated for the incoming and the outgoing link respectively. Both of the two PC instances were operated in *pcap* mode; the capture operations were fully software-based. We are conducting various tests at the moment: packet loss monitoring, flow generation accuracy and statistics measurement, application recognition success ratio testing, etc.

## System Performance Evaluation

- Packet loss
  - ◆ ratio of 0.01% and 0.03% on the incoming and the outgoing link respectively
  - ◆ all of the packet losses were occurred on the layers below the PC (Packet Capturer)
- Agent can cover up to around 810 Mbps and 92.65 Kpps
- The system can be reactively or proactively configured to avoid a highly overloaded situation by means of adjusting the proportion of type-P or type-C flows in the configuration file

Although we observed average packet loss ratio of 0.01% and 0.03% on the incoming and the outgoing link respectively, all of the packet losses were occurred on the layers below the PC. In a Linux system, the default size of a kernel buffer allocated for a packet capture socket can contain only about tens of packets. Most packet losses, thus, were taken place by the kernel buffer shortage during a quite busy period. It is notable, however, that there was no loss at all between a PC and a PP. This happens because the time required for a PP to process a packet bundle is shorter than the actual time spent to capture and accumulate packets up to the bundle size. Even in case of the opposite situation, operators can prepare comparatively huge buffer space (8 \* 32 MB by default), because a PC and a PP communicate through a configurable number of shared memory segments in a configurable size. In the ETRINet's case, the average time required for a PP to process a packet bundle was 1.782 sec, and the average time spent for a PC to capture and accumulate packets up to the bundle size was 31.006 sec. Although it may not have any strong implications, a simple calculation shows a rough performance upper-bound of the PP under current circumstances; the existing Agent platform may be able to cover 92.65 ( $5.325 * 31.006 / 1.782$ ) Kpps and 809.43 ( $46.52 * 31.006 / 1.782$ ) Mbps on average. Regardless of these estimated figures, the system can be reactively or proactively configured to avoid a highly overloaded situation. By means of adjusting the proportion of type-P or type-C flows in the configuration file, the load of the Agent and the Server can be controlled. Thus, for a very high speed link, we can configure all of the flows to type-Ss resigning a high degree of correctness, and vice versa.

Besides ETRINet, we are experimenting our system in a large university and the largest Internet exchange point in Korea. The evaluation results are not completely compiled yet and will be reported in the camera-ready version of the paper.

## Conclusion and Future Work

- We proposed a high performance, adaptable, configurable, and scalable application-aware traffic measurement and analysis system
- We are satisfied with the initial set of testing results from the deployment at the relatively low speed Internet link in our campus
- Our next step is to test it at a backbone internet exchange link which is one of the busiest exchange points in the government run public Internet in Korea
- Although we have focused on the usage-based accounting functionality of our system in this paper, it can be utilized in many other areas such as traffic profiling and security anomaly detection. These additional capabilities will be explored as our future enhancement

Due to highly dynamic nature of the development and the use of the current Internet applications, accurate traffic usage accounting in the Internet requires a cleverly combined mechanism of per-packet payload inspection, flow-based analysis, correlation of associated sub-transaction flows, and wire-speed packet capturing performance. In this paper, we proposed a high performance, adaptable, configurable, and scalable application-aware traffic measurement and analysis system to tackle such challenges. We are satisfied with the initial set of testing results from the deployment at the relatively low speed Internet link in our campus. Our next step is to test it at a backbone internet exchange link which is one of the busiest exchange points in the government run public Internet in Korea. It has a dozen Gb speed links and a number of flows it handles per day is much greater than our campus network. The testing result can be included in our next version of the paper. Although we have focused on the usage-based accounting functionality of our system in this paper, it can be utilized in many other areas such as traffic profiling and security anomaly detection. These additional capabilities will be explored as our future enhancement.

### References

- [1] Colleen Shannon, David Moore, and k claffy, "Characteristics of Fragmented IP Traffic on Internet Links", Proc. of ACM SIGCOMM Internet Measurement Workshop, San Francisco, USA, Nov. 2001.
- [1] <http://www.caida.org/tools/>.
- [3] <http://sourceforge.net/projects/tcpdump/>.
- [4] <http://www.ethereal.com/>.
- [5] <http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm>.
- [6] <http://www.caida.org/tools/>.
- [7] Sprint ATL, "IP Monitoring Project," <http://www.sprintlabs.com/Department/IP-Interworking/Monitor/>.
- [8] Steven McCanne, Van Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture, In Proc. of the Winter 1993 USENIX Conference, 259--269. USENIX Association, Jan 1993.