# On the Development of IETF-based Network Monitoring Probes for High Speed Networks

Ricardo Sanchez, Rodrigo Pereira, Luciano Gaspary

Universidade do Vale do Rio dos Sinos
Centro de Ciências Exatas e Tecnológicas
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada (PIPCA)
Av. Unisinos 950 – CEP 93.022-000 – São Leopoldo, Brazil
{rnsanchez,rspereira,paschoal}@exatas.unisinos.br

**Abstract**

In the recent years network managers have increasingly relied on monitoring tools to characterize and measure high-layer protocol traffic in order to (a) justify investments on network equipment acquisition, (b) identify most network-consuming users, (c) detect bottlenecks, to mention just a few reasons. The Internet Engineering Task Force (IETF), aware of the mentioned demand, has been making efforts to standardize management mechanisms that allow the characterization and measurement of both protocols and networked applications behavior. However, the development of IETF compliant probes so that they sustain the traffic generated in high speed networks is a current challenge, since communication links operating at 100 Mbps or higher rates require efficient packet filtering and processing mechanisms so that probes do not discard packets. This paper reports the development, by our research group, of an RMON2 compliant SNMP agent. The paper focuses on the project decisions, including the architecture and data structures used (having in mind that the agent is supposed to be deployed in high speed network environments). The paper also presents a performance evaluation of the agent.

**Keywords:** High speed network, monitoring, RMON2, SNMP.

## 1 Introduction

The popularization of computer networks in the recent years brought about the appearance of a high number of distributed applications and high-layer protocols. Online games such as Quake, videoconferencing tools (e.g. NetMeeting and CUseeMe) and applications for message and file exchange, like ICQ and Kazaa, are some examples of the "new" menu of networked applications. These applications and many other have been increasingly incorporated by network users to their daily routine, which for the network manager means the need for constant alterations in the network infrastructure to support the increasing traffic imposed by them. Such modifications involve costs and, therefore, they need to be justified. Due to the diversity and complexity of these applications, not only *capacity planning* has become a more challenging task, but also *traffic characterization* and *monitoring* as well as *network optimization*.

The Internet Engineering Task Force (IETF), aware of the mentioned problem, has been making efforts to standardize management mechanisms that allow the characterization and measurement of

both protocols and networked applications behavior. Remote Network Monitoring MIB version 2 (RMON2) [1], Application Performance Measurement MIB (APM) [2] and Real-time Application QoS Monitoring MIB (RAQMON) [3] are examples of initiatives where the *rmonmib* working group has been working on since the end of the 90's.

The importance of these standards is unquestionable. By using standard MIBs to accomplish protocols and networked applications monitoring, it becomes possible to manage an heterogeneous infrastructure, with network devices and probes from different vendors, using both the same interface (e.g. RMON2, APM and RAQMON MIBs) and communication protocol (e.g. SNMP). Besides, in opposition to what happens to most of the proprietary monitoring tools, a management station can collect data from several probes. Therefore, it is possible to monitor several subnets from one central manager.

The development of IETF compliant probes so that they sustain the traffic generated in high speed networks is a current challenge. On the one hand, communication links operating at 100 Mbps or higher rates require efficient packet filtering and processing mechanisms so that probes do not discard packets. On the other, the size and complexity of the supported MIBs do not help to make the monitoring process efficient. IETF MIBs, specially RMON2, are composed of several groups that provide the network manager with different views about the monitored traffic. Hence, regardless of the architecture and data model used, each observed packet in the network traffic requires one or more updates in the internal structures maintained by the probe (delaying the accounting process).

This paper reports the development, by our research group, of an RMON2 compliant SNMP agent. The paper focuses on the project decisions, including the architecture and data structures used (having in mind that the agent is supposed to be deployed in high speed network environments). The paper also presents a performance evaluation of the agent. Beside discussing the difficulties of, at the same time, developing probes in conformance with IETF standards and make them able to operate in high speed networks, it is also a major contribution of this work the release of an open and free software-based RMON2 agent, which is an extension to NET-SNMP [4], and that can be used as an alternative to the expensive probes as there are not many requirements to run it on Intel x86 stations (e.g. PCs). The agent can be deployed in any institution at close to zero costs and be used as a base to other academic researches[1].

The paper is organized as follows: section 2 describes some related works. In section 3 we present the developed agent. The paper follows with a detailed presentation, in section 4, of the performance evaluation done with the agent. Section 5 concludes with some final considerations and prospects for future research.

# 2 Related Work

Research works related to real time network traffic monitoring aim, in general, at proposing software architectures that are able to handle a large number of packets with the lowest possible discard rates. An approach to accomplish this objective is the use of efficient filtering and packet matching mechanisms. Monitoring tools proposed by Malan and Jahanian [5] and Anagnostakis et al. [6] take this aspect into account. Windmill is a passive network protocol performance measurement tool. The tool provides the underlying filtering mechanism as well as the ability to reconstruct the high-level protocol streams. It utilizes dynamic code generation for fast packet matching and is designed to demultiplex packets to a set of receivers (one-to-many). Through the combination of dynamic compilation and a fast matching algorithm, Windmill's WPF can match an incoming packet with five components in less than 350ns on a 200MHz Pentium-Pro. FLAME is a programmable packet monitoring system that provides a mechanism for loading code in the system kernel. It guarantees safety

[1] It is important to highlight that there is no other open and free RMON2 agent implementation available.

by using a type-safe language and run-time checks. Developers claim that the system sustains itself even under Gigabit per second traffic rates.

Other complementary and equally important approach to develop efficient real time traffic monitoring probes focuses on using data strucures that provide fast store and update procedures, since they are invoked at least once for every analyzed packet. In the case of ntop [7], an open-source web-based network usage monitor that enables users to track relevant network activities including network utilization, established connections, network protocol usage and traffic classification, hosts information is stored in a large hash table whose key is the 48 bit hardware (MAC) address that guarantees its uniqueness. Each entry contains several counters that keep track of the data sent/received by the host, sorted according to the supported network protocols. For each packet, the hash entry corresponding to packet source and destination is retrieved or created if not yet present. IPTraf is a network monitoring utility for IP networks that uses a similar approach [8]. The main data structures used by the various facilities are doubly-linked lists. This makes it easier to scroll forward and backward, and the maximum number of entries is limited only by available memory. Search operations on most facilities, are performed linearly, and have a mild hit [8]. The IP Traffic Monitor (part of IPTraf) though uses a hash table for better search efficiency, due to its propensity to grow quite rapidly.

Triticom, Network Harmoni, Cisco and Enterasys sell RMON2 probes that support 10 to 100 Mbps traffic rates. However, information about packet filtering optimization and processing are not provided by these vendors. The RMON2 agent our research group has developed, as presented in the next sections, makes use of a user-level packet capture library and, therefore, tends to be less efficient than approaches that push this functionality into the kernel (e.g. FLAME). However, the station where our agent is installed can be used to run other applications concurrently. Regarding the data structures used to accomodate statistics provided by RMON2, we have mostly employed tables (implemented as vectors) indexed by *hash* functions. Besides, we have used additional mechanisms such as *caching* to improve agent efficiency (to support high-speed network traffic).

# 3 Internals of the RMON2 Agent

In this section we describe the RMON2-compliant SNMP agent developed by our research group. The section starts with a brief review of the RMON2 MIB (sub-section 3.1), followed by a detailed description of the agent's architecture (sub-section 3.2). Then we present (a) the mechanisms used by the agent to store collected information (sub-section 3.3) and (b) relevant optimizations implemented (sub-section 3.4).

## 3.1 A Brief Review of the RMON2 Management Information Base

The works to extend RMON MIB and include mechanisms to monitor higher-layer protocols began in 1994. This initiative, called RMON2, resulted in the creation of RFC 2021 in January 1997 [1]. When monitoring high-layer protocols such as network and application-layer protocols, it is possible to visualize the whole corporate network instead of individual segments. Briefly, the groups defined in RMON2 MIB (as illustrated in figure 1) are:

- *protocol directory* (`protocolDir`): repository that indicates all the protocol encapsulations that the probe is capable to interpret;
- *protocol distribution* (`protocolDist`): statistics about the amount of traffic generated by each protocol encapsulation observed by the probe;
- *address map* (`addressMap`): associates each network-layer address to the respective MAC address, storing it in a table;

- *network-layer host* (`nlHost`): collects statistics about the amount of input/output traffic of the hosts based on their network-layer addresses;
- *network-layer matrix* (`nlMatrix`): provides statistics about the amount of traffic between host pairs based on their network-layer addresses;
- *application-layer host* (`alHost`): collects statistics about the amount of input/output traffic of the hosts based on their application-layer addresses;
- *application-layer matrix* (`alMatrix`): provides statistics about the amount of traffic between host pairs based on their application-layer addresses;
- *user-history collection* (`usrHistory`): periodically samples objects specified by the user (manager) and stores the collected information in tables;
- *probe configuration*(`probeConfig`): controls the configuration of various operational parameters that are supported by the probe, software and hardware revision numbers of the probe, a trap destination table, and so on;
- *rmon conformance* (`rmonConformance`): describes the requirements for conformance to the RMON2 MIB.

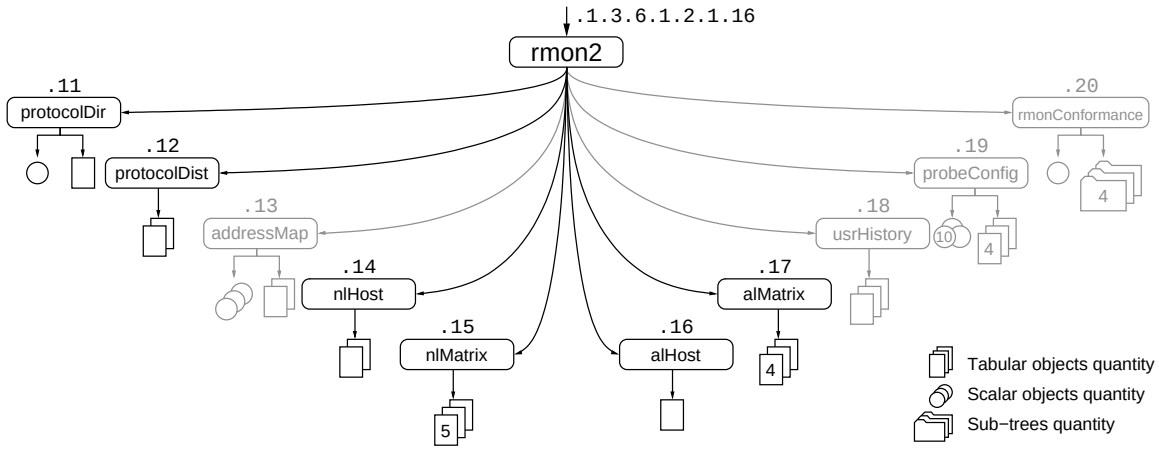


Figure 1: Structure of the RMON2 MIB

The agent we have developed comprises the groups that provide statistical information (highlighted in figure 1): *protocol directory*, *protocol distribution*, *network-layer host*, *network-layer matrix*, *application-layer host* e *application-layer matrix*. The *protocol directory* group is composed of a single table called `protocolDir`. The *protocol distribution* group, on its turn, consists of two tables: `protocolDistControl`, which controls the collection of basic statistics, and `protocolDistStats`, which stores the collected data. Similarly, the *network-layer host* group comprises a control table (`hlHostControl`) and a data table (`nlHost`). *Network-layer matrix* consists of five tables: `hlMatrixControl`, `nlMatrixSD`, `nlMatrixDS`, `nlMatrixTopNControl` e `nlMatrixTopN`. The *application-layer host* group is composed of a control table (`hlHostControl`), which is the same as the *network-layer host* group, and a data table (`alHost`). And last but not least, *Application-layer matrix* shares the `hlMatrixControl` table with *Network-layer matrix* and has four tables of its own: `alMatrixSD`, `alMatrixDS`, `alMatrixTopNControl` e `alMatrixTopN`. Gaspary et al. describe in [9] the purpose of each of these tables in detail.

## 3.2 Architecture of the Agent

The agent runs on GNU/Linux stations and was developed as an extension to the the Net-SNMP framework [4], using the C language, the POSIX thread library and the packet capture `libpcap` library [10]. Figure 2 shows the agent architecture. The `PM` (*Processing Module*) module is responsible for receiving and analyzing the captured packets (detailed in sub-sections 3.2.1 and 3.2.2). Essential information of the analyzed packets are identified and stored in an auxiliary data structure to be used later on by the `UM` (*Update Module*) module. The `UM` module, with the obtained information, updates the tables that store the statistics provided by the RMON2 MIB (sub-section 3.2.3). Finally, the `W` (Wrapper) module is an interface of the developed agent with the Net-SNMP daemon (sub-section 3.2.4).

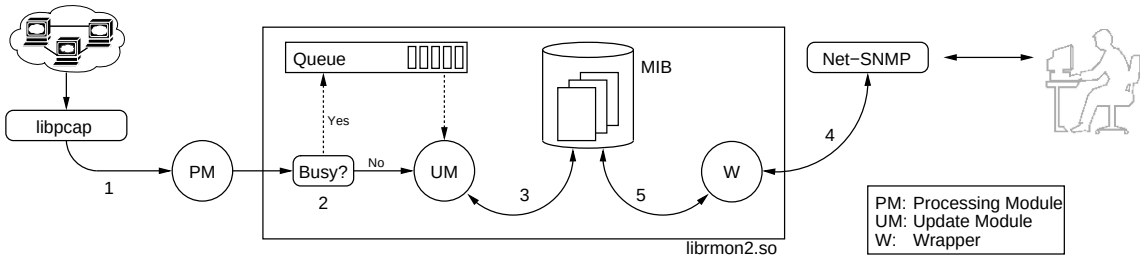


Figure 2: Internal organization of the agent

### 3.2.1 Packet capture

Packets are captured using the `libpcap` (Packet Capture library) library, which provides the developer with a high-level interface to capture all the packets flowing on the network segment. When the agent requests the library to start capturing packets in a certain interface, it registers a call-back function that is invoked whenever a packet is captured, delivering it to the `PM` module (see flow 1 in figure 2). The agent explores the filtering mechanisms available in the library. Hence, only packets whose encapsulation are registered at `protocolDir` table are captured, minimizing the number of packets that would be unnecessarily processed otherwise. When packets are discarded, the quantity of dropped packets is measured and `protocolDistControl`, `hlHostControl`, `hlMatrixControl`, `alHostControl` and `hlMatrixControl` tables are updated.

### 3.2.2 Packet processing

To speed up the process of updating the tables that compose the RMON2 MIB, each packet delivered by the `libpcap` to the `PM` module is analyzed and essential information are grouped in a PEDB (Packet Essential Information Block). If the `UM` module is not busy, the PEDB is directly dispatched to it. Otherwise it is inserted in a circular queue (as illustrated in flow 2 of figure 2). This queue leads to packet loss rate reduction (especially during network traffic peaks). Figure 3 shows the essential information that are extracted from each packet and grouped in a PEDB.

### 3.2.3 Update of the RMON2 tables

Concurrently to the process just mentioned, the `UM` module checks if there is any pending PEDB in the queue. If not, it is going to block until the `PM` module delivers new data. Whenever the `UM` module receives a new PEDB, it updates the RMON2 MIB tables executing the procedure described below. To illustrate it, consider an `http/tcp/ipv4/ether2` packet being processed by an agent configured to identify the encapsulations listed in figure 4.

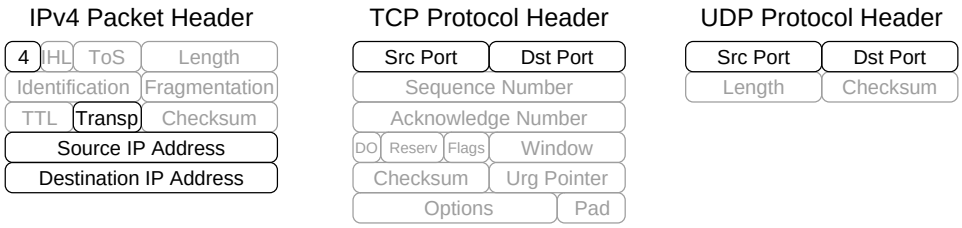


Figure 3: Essential information extracted from the captured packets

- First, the `UM` module checks if there is an encapsulation registered at the `protocolDir` table that matches the encapsulation of the PEDB being processed, taking only **link** and **network** layers into account. If so, as occurs in figure 4 (a), then tables `protocolDistStats`, `nlHost` and `nlMatrix` must be updated.
- Second, the `UM` module checks if there is an encapsulation registered at the `protocolDir` table that matches the encapsulation of the PEDB being processed, taking only **link**, **network** and **transport** layers into account. If so, as occurs in figure 4 (b), then tables `protocolDistStats`, `nlHost`, `nlMatrix`, `alHost` and `alMatrix` must be updated.
- Third, the `UM` module checks if there is an encapsulation registered at the `protocolDir` table that matches the encapsulation of the PEDB being processed, taking all **link**, **network**, **transport** and **application** layers into account. If so, as occurs in figure 4 (c), then tables `protocolDistStats`, `nlHost`, `nlMatrix`, `alHost` and `alMatrix` must be updated.

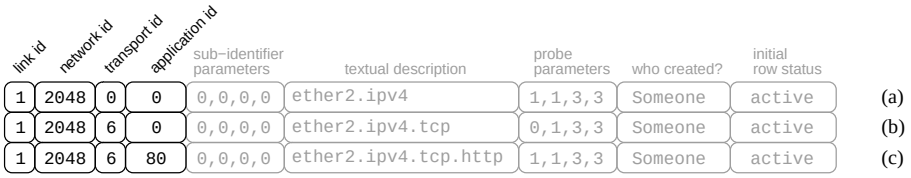


Figure 4: Examples of encapsulations registered at the `protocolDir` table

### 3.2.4 Integration of the agent to the Net-SNMP framework

The integration between the RMON2 agent and Net-SNMP is done through a module (Wrapper) that registers several call-back functions. These functions are invoked whenever the Net-SNMP daemon receives a request (`get`, `getnext`, `walk` or `set`) referring to RMON2 MIB objects (see flows 4 and 5 in figure 2). The Wrapper module had its basic structure created by a tool called `mib2c` (included in Net-SNMP distribution). The skeleton automatically generated has been populated with functions that (a) access the data structures developed (detailed in next sub-section) and (b) retrieve/set the information being requested/informed.

## 3.3 Storing of Collected Information

This section describes how the statistics that comprise the RMON2 MIB are internally stored by the agent. Basically, three types of data structures have been used: with direct access, hash function-based access and cache-based access (detailed in the following sub-sections).

### 3.3.1 Data structures with direct access

The `protocolDir` and control tables (`protocolDistControl`, `hlHostControl`, `hlMatrixControl`, `hlHostControl` and `alMatrixControl`) have been implemented as vectors with direct access. These tables consist of vectors of pointers, which are used to allocate specific records of each table. Due to the static nature of these data structures, table sizes must be configured before compiling the agent.

### 3.3.2 Data structures with hash function-based access

Data tables (`protocolDistStats`, `nlHost`, `nlMatrix`, `alHost` and `alMatrix`) have been implemented as vectors of pointers whose indexing is done through a hash function applied to a key. This approach provides fast data retrieval (in the first attempt in general). For each data table there is a particular way of generating the key that will be used to access or insert an entry in its respective storage slot. As an example, figure 5 shows both the composition of the key used to access or insert an entry in `alHost` and the process of solving table conflicts.

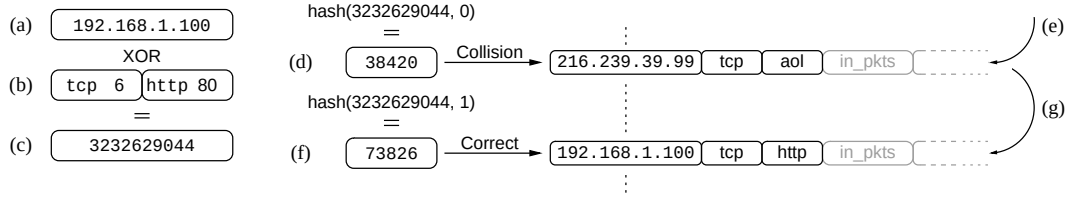


Figure 5: Key generation, access to the `alHost` table and conflict resolution

To generate the key, a `XOR` logical operation is performed on the source network-layer address, illustrated in figure 5(a), and the transport protocol identifier plus the destination application port (b), producing the key (c). By applying the hash function to this key, one obtain the first vector access index (d), enabling the access to the table, illustrated in (e). In the example, one can observe that the entry in position (d) is not the one to be updated (network-layer addresses and application ports do not match), resulting an access collision. When it happens, it is necessary to calculate the second vector access index using the same key (c). By passing an offset of 1 as argument to the hash function, a skip (spacing between the keys) of 35406 slots has been added to the initial index (d), generating the second index (f). As one can see in the figure, this index will correctly point to the entry to be updated (g). It is worth mentioning that each key has a different spacing between keys reducing the probability of collisions.

Had other collision occurred, the previous step would be repeated using an offset of 2, and so on, until the corresponding entry had been found. After using all possible offsets, it means a new entry is supposed to be created and inserted in the table. Besides, the value for the maximum number of collisions must be updated, since the last entry inserted is located after the search limit the agent does.

### 3.3.3 Data structures with cash-based access

The `protocolDir` table, beside being directly accessed, has an auxiliary structure that provides the functionality of a cache. As some encapsulations are identified more than others in the network traffic, they are promoted and are moved to the header of a double linked list. Hence, encapsulations that occur more frequently are located in the `protocolDir` table in a shorter time (and with lower computational cost). To speed up the search for encapsulations even more, the table uses three caches, one for each encapsulation level (network, transport and application).

## 3.4 Optimizations

During the development of the agent we have put efforts into increasing the processor cache hit rates (in an attempt to maximize the number of operations that are retrieved from the processor cache memory). The data structures have been designed aiming at obtaining a better usage of the new features and resources provided by current processors. For example, current `i686` processors have multiple parallel execution units, allowing anticipated instruction decoding. Hence, while an instruction is being executed, the next is decoded, reducing the time the processor has to wait for instruction decoding. There are also special parallel execution channels that allow operations on integer and float point numbers to be executed in parallel.

# 4 Performance Analysis of the Agent

In this section we describe the performance analysis we have carried out to determine the sustained capacity of the agent. The experimental setup used to run the experiments are presented in sub-section 4.1. Then we present in sub-section 4.2 the measurements performed (using both homogeneous and realistic network traffic) to figure out the performance of the agent.

## 4.1 Experimental Setup

To access the performance of the agent we have used the setup illustrated in figure 6. It consists of two PCs connected through a cat 5e UTP crossover cable at 100Mbps. The source host has a 333MHz Intel Celeron (Mendocino) CPU and 64MB RAM. Its Ethernet controller is the 100Mbps 3COM 3c905B Cyclone. The operating system used in the source host is the GNU/Linux (Slackware Linux 8.0, kernel 2.4.20). The destination host (where the RMON2 agent is installed) has a 1.7GHz Intel Pentium 4 CPU, 512MB RAM, an Intel i845 (Brookdale) chipset host bridge, a 100 Mbps 3COM 3c905C-TX/TX-M Tornado network interface card and runs GNU/Linux (Slackware Linux 8.0, kernel 2.4.20).

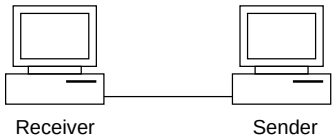


Figure 6: Setup used to run the experiments

## 4.2 Measurements

The first experiment consisted of transmitting 1,000,000 UDP packets (with the same protocol header) from source to destination, at 100Mbps, using the packet generator module provided by the Linux kernel. This transmission has been repeated 43 times varying the size of the packets generated, aiming at achieving different packet per second (pps) rates. The purpose of this experiment was to access the maximum number of packets the agent is able to process and to identify the interference of the operating system on the agent monitoring capacity.

Packet sizes from 64 to 1472 bytes have been used. From 64 to 224 bytes, the packet sizes have been increased of eight bytes in each run so that we could identify the maximum agent capacity. From 256 to 1472 bytes the packet sizes have been enlarged 64 bytes in each run. In figure 7 one can observe that the agent capacity is highly affected by the heavy system load. From 208-byte long packets on, the system load allowed the agent to process 58,900pps with a 8.68% loss rate. It is worth

noticing that, although when using 216-byte long or larger packets the system has already some free CPU time, the agent still loose some packets. It occurs because there is a bottleneck in the deliver of packets captured by the libpcap to the RMON2 agent, evidenced by the fact that with 960-byte long packets there was a 0% loss rate (12,700pps), while with 1472-byte long packets (8,355pps) the loss rate reached 0.0015%.

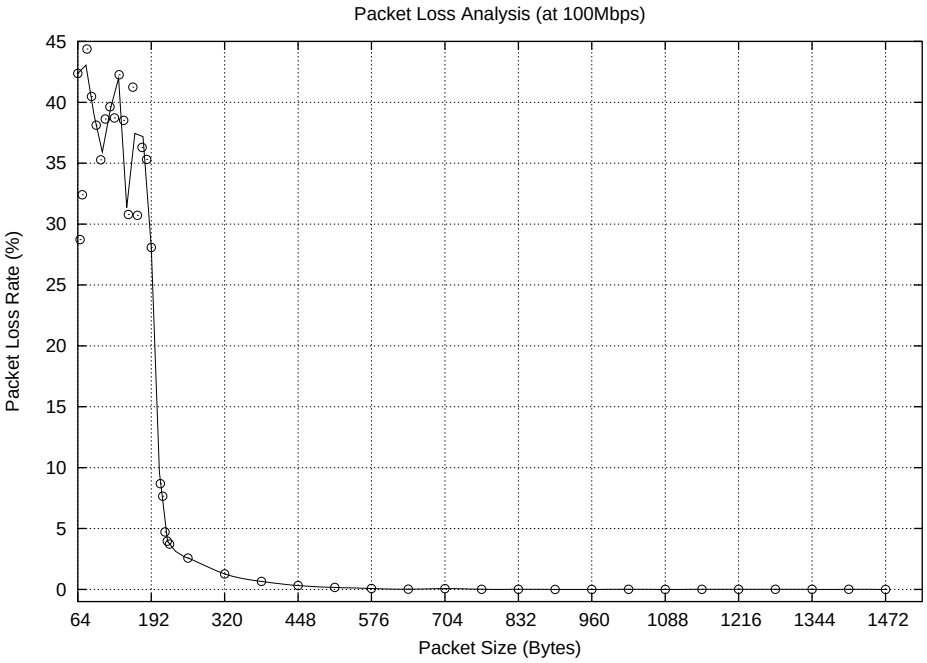


Figure 7: Packet loss rate at 100Mbps

Para realizar o teste com tráfego realístico, foi necessário usar uma máquina mais potente para substituir o transmissor. A máquina adotada como novo transmissor tem exatamente a mesma configuração do receptor, permitindo que o tráfego fosse retransmitido na mesma velocidade que fora capturado. O tráfego utilizado está caracterizado na figura 8, contendo 97.6482% de pacotes IPv4 sendo 96.4051% de pacotes TCP (sobre o total de 1,000,000 de pacotes) e 0.8486% de pacotes UDP, o restante dos pacotes atualmente não é reconhecido pelo agente RMON2.

A perda média de pacotes nos experimentos realizados foi de 23.9608%. Durante os experimentos, verificou-se que os picos de utilização de CPU estão diretamente relacionados com o descarte de pacotes. Um levantamento mais apurado indicou que o mecanismo de cache da tabela `protocolDir` apresenta deficiências quando submetido à trafego muito diversificado, como ocorre no experimento realizado, forçando o agente a descartar pacotes. Uma possível solução para o problema mencionado já está em fase de implementação, onde o mecanismo de cache será substituído por tabelas hash.

# 5 Conclusions and Future Work

A MIB RMON2, a exemplo do que ocorre com outras MIBs padronizadas pelo IETF, apresenta uma quantidade grande de objetos, oferecendo diferentes visões do tráfego sendo monitorado. Para manter estas visões atualizadas em tempo real, para cada pacote coletado é preciso atualizar um conjunto numeroso de tabelas. Por exemplo, ao capturar um pacote `http/tcp/ipv4/ethernet`, este deve gerar atualizações nas estatíticas associadas aos protocolos de rede, transporte e aplicação. Estas atualizações demandam um custo computacional elevado, que influencia diretamente o desempenho da ferramenta (como ilustrado na figura 8).

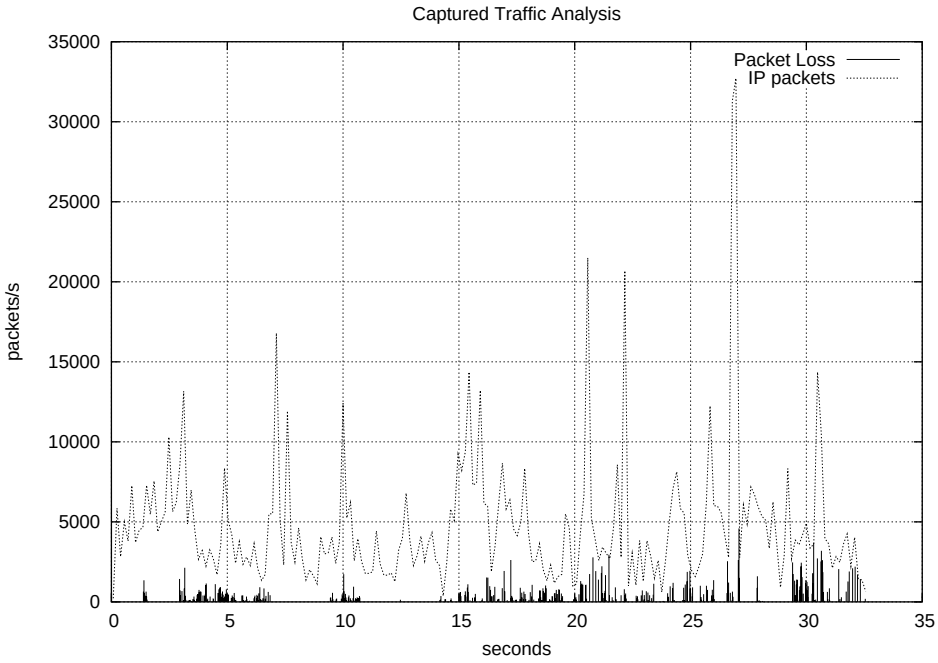


Figure 8: Characterization of the realistic traffic used to access the agents performance

Os mecanismos usados para suportar a monitoração da rede e atualizações dos objetos presentes na MIB são cruciais para atingir a escalabilidade necessária para lidar (cope with) com redes de alta velocidade. No caso da monitoração passiva, intensamente utilizada pelo nosso agente, a utilização de uma biblioteca de captura localizada em user space torna-se inadequada para atender à captura de pacotes a taxas próximas ou acima de 100Mbps. Nosso grupo de pesquisa acredita que uma estratégia a adotar seria o desenvolvimento de um módulo para o kernel Linux com funcionalidade semelhante à libpcap, voltado a atender às necessidades específicas do agente. Outros pontos de otimização já identificados estão relacionados a um melhor aproveitamento de CPU, como sugerido em [11].

Nossos experimentos apontam ainda que a estrutura cache, usada para melhorar o desempenho de acesso à tabela `protocolDir`, mostrou-se pouco adequada em determinadas condições (tráfego com protocolos muito diversificados), afetando o desempenho do agente. Essas observações nos levaram a reprojetar sua estrutura interna, substituindo a atual cache por uma tabela hash (já em desenvolvimento).

# References

[1] S. Waldbusser. “Remote Network Monitoring Management Information Base Version 2 using SMIv2”. RFC 2021, INS, Jan. 1997.

[2] S. Waldbusser. “Application Performance Measurement MIB”. Internet Draft, Mar. 2003.

[3] A. Siddiqui, D. Romascanu, E. Golovinsky and R. Smith. “Real-time Application Quality of Service Monitoring (RAQMON) MIB”. Internet Draft, Oct. 2002.

[4] Net-SNMP Project Homepage. http://net-snmp.sourceforge.net/.

[5] G. Malan and F. Jahanian. “An Extensible Probe Architecture for Network Protocol Performance Measurement”. In *Proc. of ACM SIGCOMM*, Vancouver, 1998, pp. 215–227.

[6] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M. Greenwald and J. M. Smith. “Efficient Packet Monitoring for Network Management”. In *Proc. of the 8th IEEE/IFIP Network Operations and Management Symposium*, Florence, 2002, pp. 423–436.

[7] L. Deri and S. Suin. “Ntop: Beyond Ping and Traceroute”. In *Proc. of the 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Zurich, 1999, pp. 271–283.

[8] IPTraf Homepage. http://cebu.mozcom.com/riker/iptraf/.

[9] L. P. Gaspary and L. R. Tarouco. “Characterization and Measurements of Enterprise Network Traffic with RMON2”. In *Proc. of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, p. 229–242.

[10] tcpdump Homepage. http://www.tcpdump.org/.

[11] AMD Athlon Processor x86 Code Optimization Guide. http://www.amd.com/.