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

Ricardo Nabinger Sanchez  
rnsanchez@cscience.org

Rodrigo Pereira  
rspereira@exatas.unisinos.br

Luciano Paschoal Gaspary  
paschoal@exatas.unisinos.br

Universidade do Vale do Rio dos Sinos (UNISINOS)  
Centro de Ciências Exatas e Tecnológicas, Brazil

http://prav.unisinos.br/~trace

Iguassu Falls, Brazil - September 4 - 6, 2003
Outline

Introduction
The Challenge of Network Monitoring
Brief RMON2 Review
RMON2 Agent’s Internals
  Agent’s Architecture
  What Does the Agent Collect?
  How Does the Agent Store Collected Data?
Performance Analysis of the Agent
Conclusions and Future Work
Introduction

Currently, there is a large number of high-layer network protocols.

For each high-layer protocol, there is a growing number of applications allowing its usage.

It also exists a growing number of users that use an also growing number of applications directed to an even larger number of protocols.

The ever growing network utilization implies in:

- constant alterations in the network infrastructure
- upgrading or buying equipment for newer (and faster) networks
Such modifications involve costs, which must be justified

The IETF (Internet Engineering Task Force) has been making efforts to standardize mechanisms that allow characterization and measurement of both protocols and networked applications behavior

Since the end of the 90’s, the rmonmib working group has been working on MIBs (Management Information Base) which provide accurate network information to the network manager, giving him means to justify investments

These standard MIBs also help in capacity planning, traffic characterization and network optimization
Our major goal is to provide the network community with a RMON2 compliant monitoring agent, designed to be used in GNU/Linux environments, which should be:

- **Efficient**, so it can be deployed on shared x86 workstations or dedicated low-end x86 stations
- **Fast**, so it can monitor links such as Fast Ethernet, which operates at 100Mbps, with low or even no packet loss at all
- **Open and free software**, as there is no such implementation available, on the hope of basing other network researches
The Challenge of Network Monitoring

Ethernet is the most widely used network type, whose speed may reach 10Gbps currently. On a 100Mbps Fast Ethernet link, we have:

- packet rates from 8,127 to 148,810 packets per second

Thus, we have:

- best case of 123.05µs to process a single packet at 8,127 packets per second
- worst case of 6.72µs to process a single packet at 148,810 packets per second
Brief RMON2 Review
What Does the Agent Collect?

The agent recognizes and analyses only IP over ethernet packets. If so, it looks for IP version 4, discarding the packet if not. Being an IPv4 over ethernet, the agent collects the relevant data: network source and destination addresses, application source and destination ports. Additional info is not on the packet itself, such as network interface and system uptime on packet’s processing.
How Does the Agent Store Collected Data?

Simple and small structures are implemented over **array of pointers**, as happens with the **control tables**. They are so simple that any other data structure imposes too much overhead.

Larger structures (simple or not) are implemented over **hash tables**, as happens with the **data tables**. These structures take advantage from the fast recovery times inherent to hash tables.

The agent uses a hash function known as “double hashing”, which deals better with collisions.

While simpler hash functions deal with collisions by probing another table position within some prefixed range (generally 1, $n$ or $n^2$), double-hashing probes key-dependant positions, which are almost random. This greatly reduces the chance of a second collision for the same key.
Hash Tables Revisited

First, we create the key to be used in the hash function. We need to pick data which is very intimate to the packet that is very unlikely to be present on other packets, such as the network address. To get better results, we mix some data, like transport and application protocols.

Then we apply the hash function over the key, with an initial offset of zero to get the first possible data index. This index is used to access the hash table, and the position must be checked to see if it’s the wanted one. In case of collision, the next offset is probed.

Example of hash table used in the alHost subtree
Performance Analysis of the Agent

Simple setup, connecting 2 hosts through a crossover UTP cable, at 100Mbps, full-duplex

Agent host is a 1.7GHz Intel Pentium 4, running GNU/Linux Slackware Linux 8.1, with a 2.4.21 Linux kernel (compiled with architecture-specific optimizations enabled)

Two test types:

- Stress test to measure agent’s limits, consisting in the transmission of the same UDP packet one million times, varying only packet size and without any delay between packets. Linux kernel packet generator was used.

- Realistic test, to verify how the agent would behave under real network traffic. A tcpdump trace file, containing one million packets (duration of about 30 seconds) was used and replayed at nominal rate.
Stress Test Results

Packet Loss Analysis (at 100Mbps)

Packet Loss Rate (%) vs Packet Size (Bytes)
Stress Test Results - Detail

Packet Loss Analysis (at 100Mbps)

- Y-axis: Packet Loss Rate (%)
- X-axis: Packet Size (Bytes)

The graph shows the packet loss rate (%) at different packet sizes (Bytes) when the network is under stress. The analysis is conducted at 100Mbps.
The test was run 20 times, and the average packet loss was of 23.9608%.

We found several factors which contributed to this significative packet loss rate:

- The `protocolDir` cache mechanism still imposes too much CPU usage.
- Under heavy network load, the Linux kernel uses a considerable amount of CPU. Measurements showed that this usage can get as high as 48%, due to network card interrupt handler. This was improved on Linux kernels 2.5, and will be available on 2.6.
- `libpcap` default installation presents a small buffer, insufficient to cope with high network load.
- The agent is still under optimization to be able to handle this standard MIB efficiently and with lower CPU usage.
Conclusions and Future Work

Passive network monitoring demands high CPU power, and thus, any further processing must be very efficient to avoid considerable packet loss rate.

As happens with other MIBs standardized by IETF, the large quantity of objects demands a high computational effort in order to keep these data real-time updated, which turns into a complex task.

Although not tested again, the RMON2 agent is expected to deal a lot better with realistic traffic, due to the protocolDir table conversion to hash table.

Modifying libpcap is planned, giving it larger buffers to support heavy network load while we design a better solution.
Questions?

Thank you!

Ricardo Nabinger Sanchez
rnsanchez@cscience.org
http://prav.unisinos.br/~trace

The RMON2 Agent is ready to be downloaded!
Please contact us so we can provide instructions