Benchmarking results of SMIP project software components

NAILabs

September 15, 2003

1 Introduction

As packets are processed by high-speed security gateways and firewall devices, it is critical that system resources are optimized to allow for operation in a fast network. This paper outlines some benchmarking tests performed on the components of the SMIP project[1] to measure their efficiency in achieving this goal.

The machine used in calculating these benchmarks was a Dell Latitude CPx, with a 650 MHz Pentium III, 256 MB of memory, running RedHat Linux 7.2, and using kernel version 2.4.18-3. At the time of this writing, it is about 2 years old and is already significantly slower than more modern systems.

Unless otherwise stated, all networking benchmarks were performed over the loop-back interface to remove the real network specific dependencies from the results. All network bandwidth calculations were done using the netperf[2] benchmarking tool testing maximum transfer rates over a TCP connection.

The architectural components of the SMIP project are shown in Figure 1. An administrator uses the management console to describe the required network policy which is then stored in the management server's database. The policy management engine then utilizes the SNMP protocol to configure each of the network devices it is responsible for.

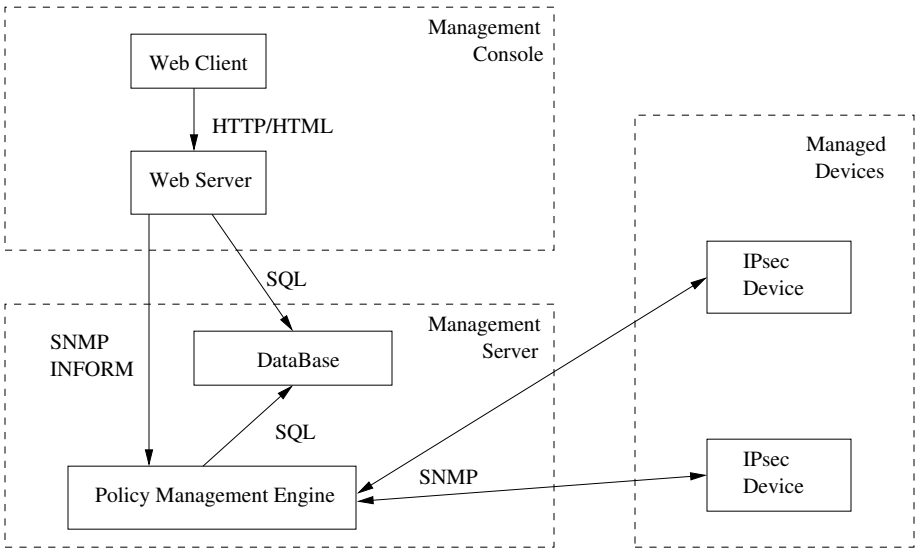

Figure 1: SMIP Project Architecture Components

2 Policy Enforcement Efficiency

At a Policy Enforcement Point (PEP), packets are processed and policy decisions are enforced to ensure that the configured network policy is being met. PEPs are managed by the Policy Management Engine and are a managed device specific to IPsec. Since PEPs are where network traffic is affected directly by network policies, they are the most critical point for optimization. Slow policy enforcement directly relates to the network bandwidth to and from a given PEP.

Within the SMIP project, IPsec and network policy definitions are constructed using a series of rules which are made up of filters and actions. These rules are then executed for each packet traversing a PEP in an assigned and deterministic order such that rules with a higher priority are always executed before rules with a lower priority. If all of the filters associated with a given rule are evaluated positively, then the action for that rule is executed and the processing of the network packet is finished. This decision making process is depicted in Figure 2. Since this process is required for every packet passing through a PEP, it is critical that this be optimized for performance. The critical elements of this process are: how fast the packets can be filtered and how fast the actions can execute. For IPsec security gateways, this means the IPsec and IKE specific actions must process IPsec and IKE actions as fast as possible.

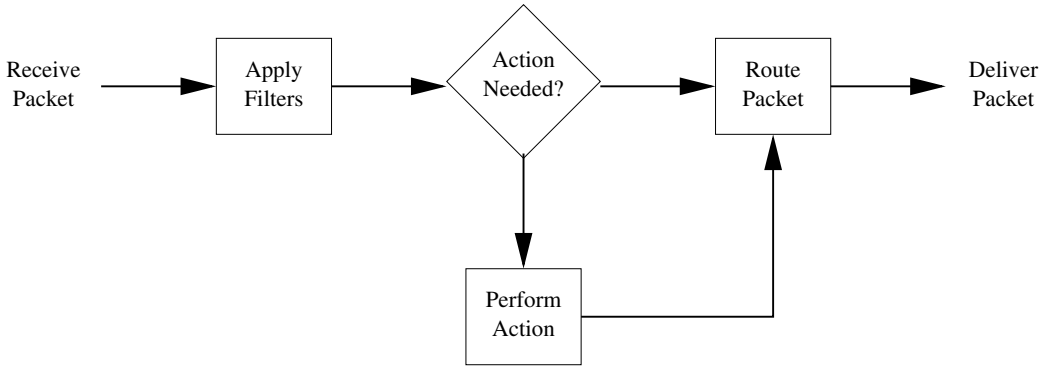

Figure 2: PEP Packet Decision Process

2.1 Filter Processing

As packets traverse a PEP they will be subject to various types of packet filters as the policy rules are evaluated. As the number of filters which must be processed per packet increase, the amount of processing time required for each packet must also increase. The graphs in Figures 3 through 6 show that as more packet filters are needed, the processing speed of both installing and applying those filters begins to consume more system resources.

For benchmarking purposes, all the filters in these tests were always evaluated to be false so that processing of the test packets did not stop before reaching the final filter. I.E., every filter was applied to every network packet for the test. This was done to test the speed at which a large number of filters could be evaluated.

It should be noted that in a real world scenario, it is highly unlikely that such a large number of filters would need to be processed for any given packet. The SMIP architecture, as well as most firewall filter processing engines including the one used within the Linux kernel, allows for the creation of decision trees which makes processing of all the filters in a large collection unnecessary. As an example, if you needed to compare a network packet against a list of 10,000

different IP addresses, you could optimize your performance by grouping your filters together by subnets so that only a subset of the entire collection of filters will be evaluated for any filter falling into the containing subnet.

These figures show which filters are more expensive than others. This helps in defining proper processing ordering of filters such that the faster filters are executed first. Additionally, they show that all of the filters are quite fast. The application of 100 of the slowest filters will still leave the machine able to process packets at a fast enough rate to handle a 1 Gb network interface (even on the two year old machine used for these tests).

The filters which are slightly slower within these graphs are ones that require external processing functions in order to evaluate the given packet. Filters such as the source address filter, for example, are handled directly and internally by the Linux packet filtering system. However, filters such as the IP-range filter are Linux iptables plugin-filters which are distributed with the SMIP project software. They require slightly more memory and some additional function calls and are thus slightly slower. The graphs show that even these externally defined filters, however, are extremely fast.

Some filters, not yet available for testing at the time of these benchmarks, are expected to be significantly slower. IKE credential checks, for example, will make use of public-key cryptography authentication tests. The speed of these filters will be evaluated in the last portion of the SMIP project.

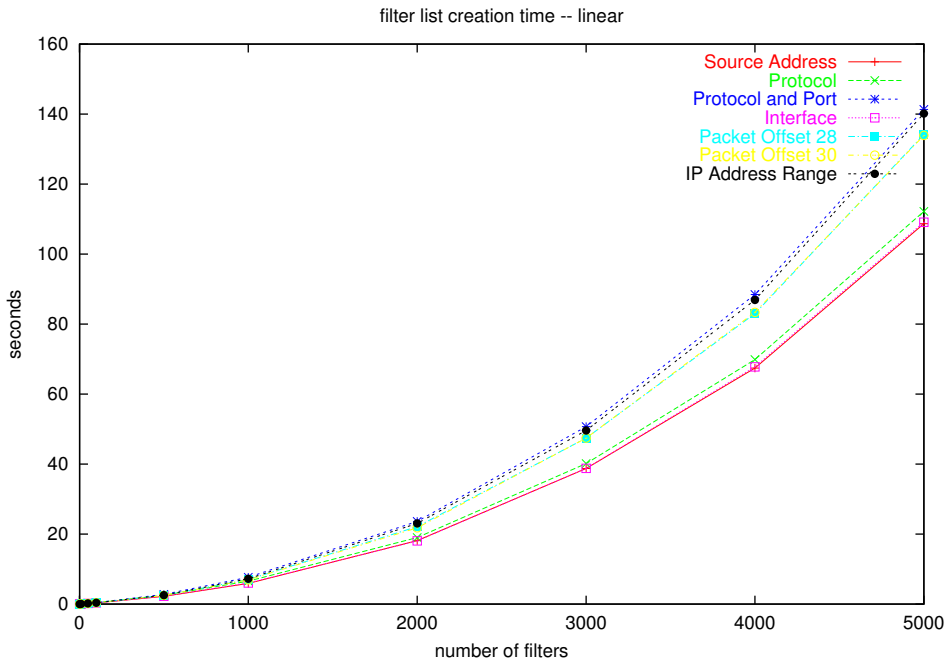

Figure 3: Filter insertion processing speed – linear scale

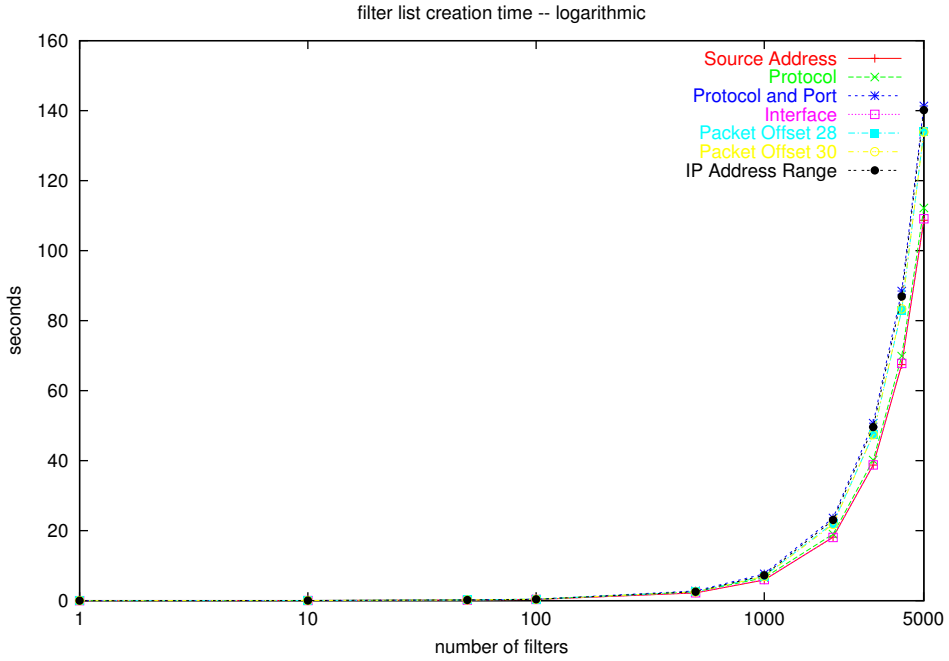

Figure 4: Filter insertion processing speed – logarithmic scale

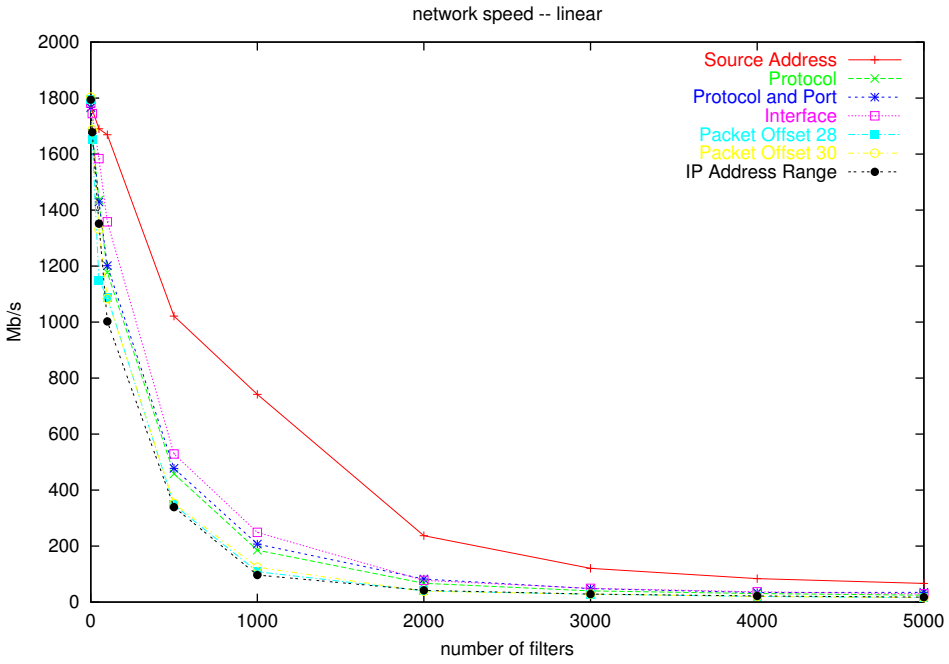

Figure 5: Filter processing and its effect on network speed – linear scale

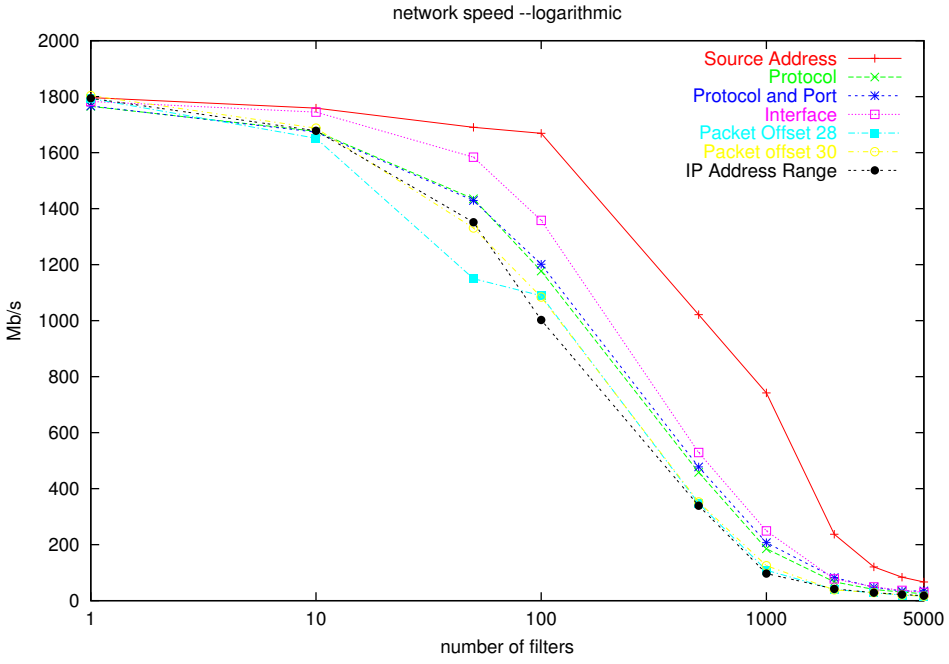

Figure 6: Filter processing and its effect on network speed – logarithmic scale

2.2 Action Processing

A few types of actions can be performed on packets traversing a PEP. Accepting and dropping actions are simple and require no benchmarking as they are effectively instantaneous. Packets can also trigger IPsec and IKE related actions. The IPsec actions fall into three categories: protecting traffic using existing IPsec Security Associations, bringing new statically configured Security Associations on-line and negotiating security association keying material and other parameters using the IKE negotiation service.

2.2.1 SADB Benchmarks

SADB lookups occur within a IPsec-enabled device to determine if any incoming or outgoing packet matches an existing IPsec Security Association and therefore must be processed by IPsec security transformations. The SADB is functionally a database of existing Security Association parameters within a device. It is critical that the SADB be as efficient as possible since every IPsec protected packet traversing a device will result in a SADB consultation.

The graphs shown in Figures 7 and 8 show the processing speed of SADB database insertions and the packet processing as affected by the number of entries stored in the SADB database.

These graphs are for the default programming methodologies of the Cerberus[3] package. It is likely that for systems requiring a huge number of connections, the code can be optimized so that the hash-table based lookups achieve even greater performance at the cost of memory utilization. Even with the default parameters in place, the graphs show that the SADB database implementation with Cerberus is extremely efficient.

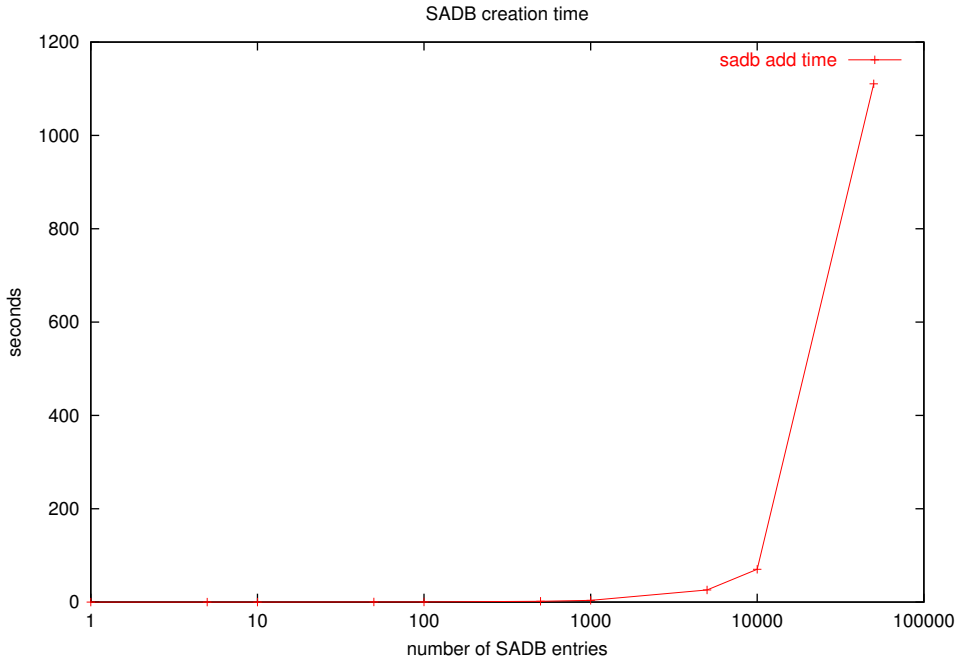

Figure 7: sadb insertion processing speed

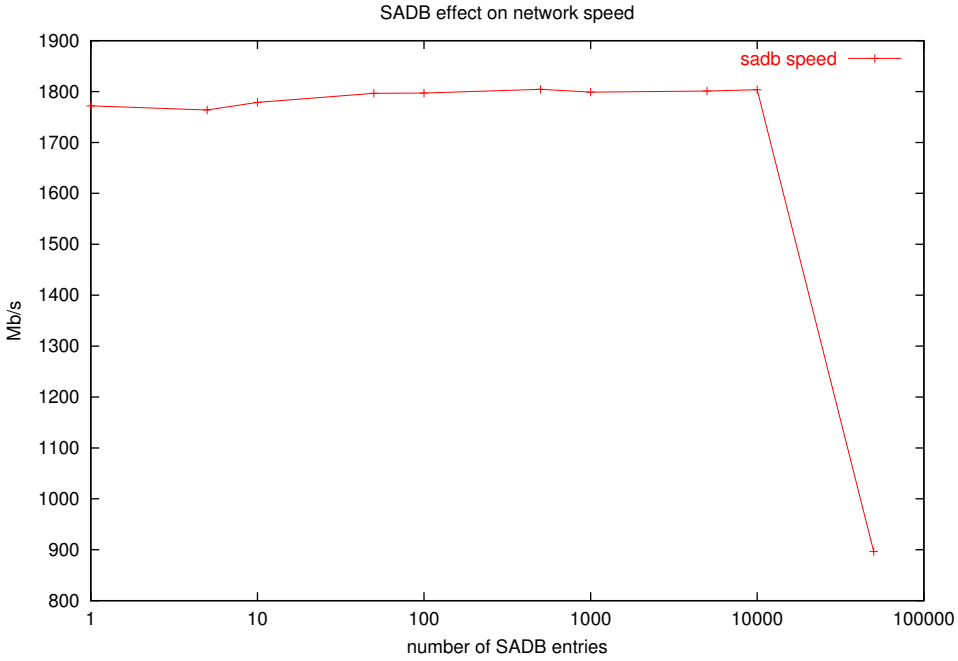

Figure 8: sadb processing and its effect on network speed

2.2.2 IPsec Encryption and Authentication Algorithm Efficiency

Table 1 shows the network performance of two machines conversing over an IPsec protected SA under various authentication and encryption algorithms. For these tests, the test laptop communicated with a much faster server over a 100Mb/s network link. Effective bandwidth was then measured between these two machines for every combination of IPsec authentication and encryption algorithms. The results are listed as a percentage of the baseline rate which had no intervening IPsec connection in use. Table 1 shows the results of this analysis.

	Authentication Algorithm		
Encryption Algorithm	None	HMAC-MD5-96	HMAC-SHA-96
None	100	76	65
DES/CBC	51	47	43
3DES/CBC	34	32	30
AES/128	47	44	38
IDEA/CBC	50	45	41
RC5/CBC	62	53	49
Blowfish/CBC	51	47	42

Table 1: Throughput percentage of SA Algorithm Efficiency

It is clear from the results in Table 1 that protecting traffic using authentication and/or encryption requires significant computational overhead. Authentication alone requires significant computation and reduced the effective throughput of the test laptop to 65-76 percent of it's original capacity depending on the algorithm used. The 3DES encryption algorithm is known to be one of the slowest and most computational encryption algorithms available to date. The data in this table shows this belief to be valid, as the network bandwidth dropped to 34 percent of the original speed even without authentication enabled. The worst performance occurs when using 3DES as an encryption algorithm and HMAC-SHA-96 as a authentication algorithm, which reduced the throughput to 30 percent of it's original potential.

It is clear from this data that protecting all traffic to and from heavily-utilized systems with both authentication and encryption will reduce the usefulness and effective bandwidth of that system. The SMIP project, with its rule-based policy architecture, solves this problem by allowing networks to be configured to only protect traffic when appropriate. Decisions can be made to use slower, but possibly more trusted algorithms, only when needed.

2.2.3 IKE Key Negotiation Efficiency

The IKE service, which negotiates dynamic keys for the IPsec protocol, also consumes resources. With statically configured IKE keys, IKE took 428 milliseconds to negotiate and configure an IPsec SA. With an RSA signature based identity in use, IKE required 1.8 seconds to configure an IPsec SA. This indicates that IKE negotiation also consumes significant resources on a system. IKE negotiations should only be used when needed. It should not be used to configure IPsec SAs which will not be used or for traffic which does not require protection.

On high-speed, heavily-utilized systems, IKE should be used only when IKE key negotiation is required. This is especially true when security polices require rapidly changing keys, as the overhead of frequent IKE transactions will quickly add up. If every device in a network had an always-up policy for IPsec connections and these connections were to be established using IKE, the network systems would spend far too much time performing keying negotiations which may

not be necessary or utilized. Because of the high cost of IKE-based key negotiation, a high volume gateway should not keep all its IKE derived IPsec connections open at all times. The SMIP project's policy-based decision architecture allows IKE negotiations to be used only when needed which will reduce the overhead resources consumed by a PEP.

3 Management Efficiency

In order for PEPs to make decisions on network packets that arrive at each network node, they must be first configured with the policies to enforce. The SMIP Policy Management Servers must be able to configure as many PEPs as possible in a given period of time. The management architecture of the SMIP project is depicted in Figure 9. Although the speed of policy distribution is not as critical as the rate at which PEPs can process individual packets when applying the policies, it is still important that policy distribution speed is optimized as much as possible. A change in network policy will not have full effect until it is distributed to all the PEPs that must enforce it.

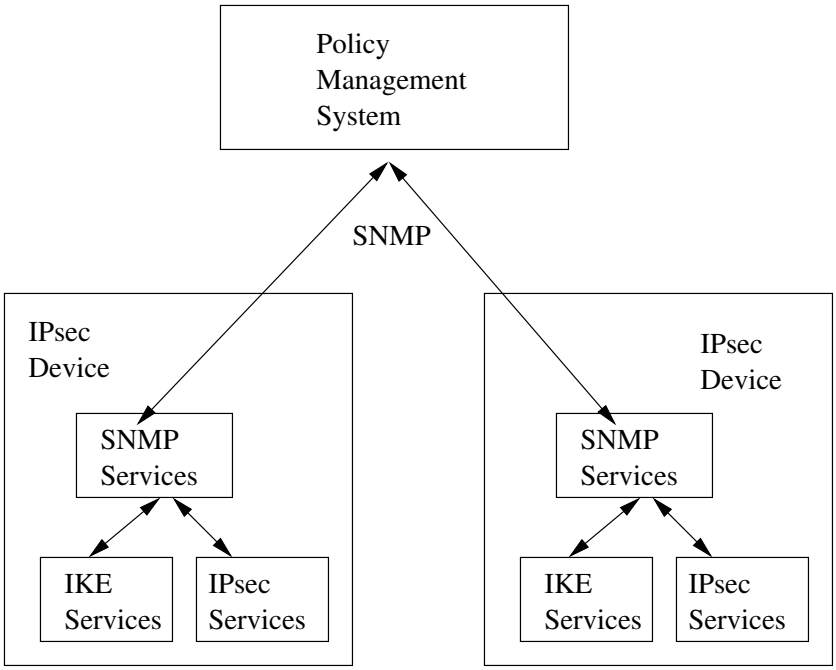

Figure 9: Management Architecture

There are two elements that effect how long it will take to configure a collection of PEPs across a network. The first element is the rate at which the PEPs themselves can accept management operations within the SNMP agent. The second is the rate at which the policy management servers can distribute policies to the PEPs.

3.1 SNMP Agent Efficiency

Figure 10 shows the time needed to configure a fixed number of filters and actions through the SNMP protocol. Figure 11 shows the time needed to retrieve the list of currently configured filters and actions through the SNMP protocol (using the least efficient SNMP method of data

retrieval). All of the SNMP transactions were performed with secure authentication enabled within the SNMPv3 protocol.

These graphs show that manipulating data within the PEPs via SNMP is fairly efficient. A large number of policy components can be inserted quickly into a running system. The speed discrepancy between the performance of filters and actions is the result of how much kernel interaction must be performed. The filters are implemented with the firewall handling rules in the Linux kernel, but the actions are implemented directly in the user-space level PlutoPlus agent.

It is important to note that the CVS head branch of the Net-SNMP package (targeting a Net-SNMP 5.1 release) contains a few speed improvements which will drastically improve these benchmarks. These benchmarks, however, were not performed against an agent constructed using the new technology not available yet in a public Net-SNMP release.

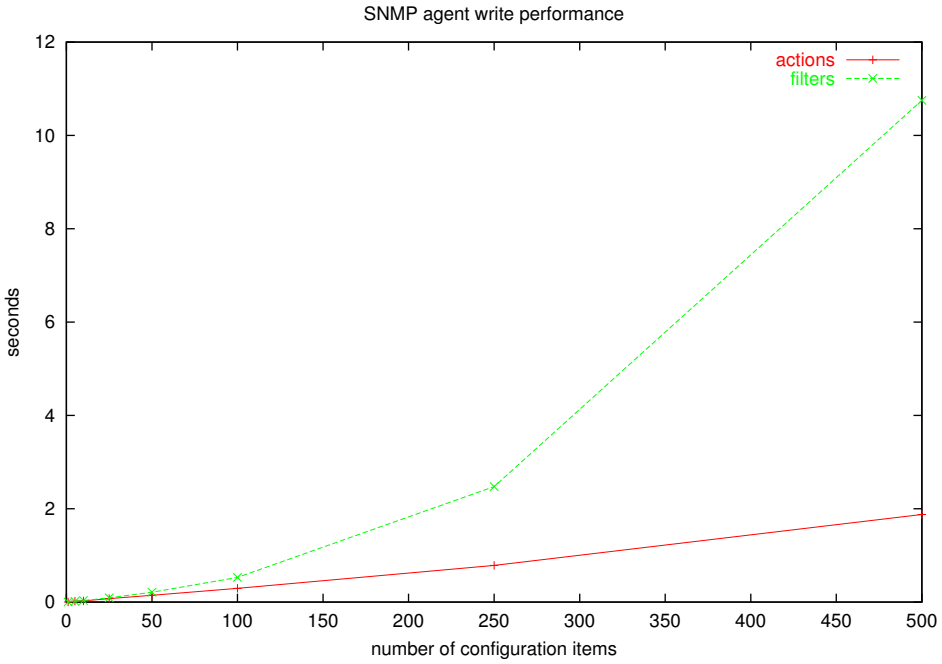

Figure 10: Time required to perform configurations via SNMP

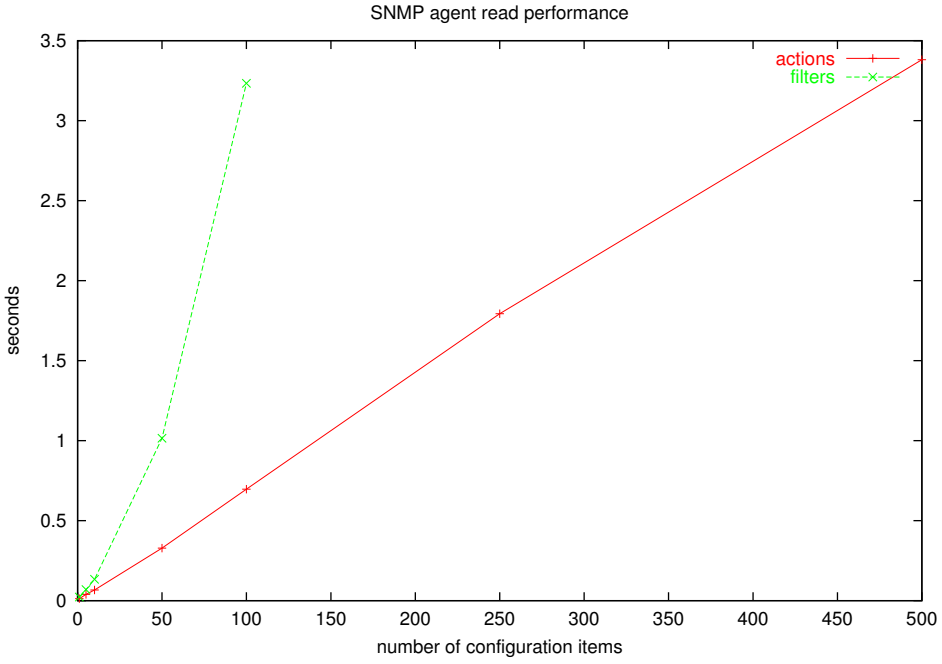

Figure 11: Time required to request current configuration via SNMP

3.2 Policy Management Server Efficiency

The SMIP architecture contains one or more policy management servers which are responsible for configuring a network with an administratively defined policy. At the time of this writing, the policy management engine is single-threaded and has not been optimized for performance.

Right now, the policy management server can configure a client on the same machine, over the loop-back interface, with the data needed to implement an IPsec SA with pre-configured static keys in approximately 345 milliseconds. The operations required for this configuration set amounted to inserting 11 rows of data into 11 SNMP tables. A large portion of this time (140 milliseconds) was spent performing a one time initialization. This was needed since the policy management server had never contacted the host in question before. On average, after initialization, a single SNMP table row can be distributed from the policy management server in under 10 milliseconds.

Because of the reuse features designed into the SMIP architecture, it should be possible to build a second SA to a different host in merely 2 additional SNMP table rows (requiring approximately only 12 milliseconds to distribute). This can be done, for example, if the same secret keys were to be used in a second policy thus eliminating the need for retransmission of the keying data. Nearly everything within the SMIP architecture is reusable in this way. Because the SMIP architecture allows for a lot of data re-use, the amount of data needed to instrument a network policy will vary depending on its re-usability as well as its complexity.

3.2.1 Policy Management Server Database Efficiency

The SMIP architecture makes heave use of a SQL database for storage of policy definitions. The database architecture has proven to be well designed for speed. Adding 10000 otherwise-unused rows of data into some of the database tables used by the policy management engine had no measurable effect on the management's performance. There are likely to be certain conditions where database access will actually slow down the policy management server's performance but none of these conditions have been found yet. Further study about these possible bottlenecks might be undertaken in the future.

4 Conclusions

Many of the SMIP architecture components are fairly well optimized already but some of the components do still need more optimizing. A few remaining components, in particular the policy management server, need more performance analysis and speed optimizations.

References

[1] Network Associates Laboratories. A Scalable IPsec Policy Configuration System. November 2001.

[2] Hewlett-Packard's NetPERF network benchmarking tool. http://www.cup.hp.com/netperf/NetperfPage.html.

[3] NIST Cerberus: An IPsec reference implementation for Linux. http://www.antd.nist.gov/itg/cerberus/.