# NETWORK STACK OPTIMIZATION FOR IMPROVED IPSEC PERFORMANCE ON LINUX

Michael G. Iatrou
*Department of Electrical and Computer Engineering, University of Patras, GR26504, Patras Greece*

Artemios G. Voyiatzis
*Department of Electrical and Computer Engineering, University of Patras, GR26504, Patras Greece*
*Industrial Systems Institute, Building of the Patras Science Park, Stadiou Str, Platani, 26504 Patras, Greece*

Dimitrios N. Serpanos
*Industrial Systems Institute, Building of the Patras Science Park, Stadiou Str, Platani, 26504 Patras, Greece*
*Department of Electrical and Computer Engineering, University of Patras, GR26504, Patras Greece*

Keywords: IPsec, Performance, Petworking, Security, Linux.

Abstract: Virtual Private Network (VPN) connectivity is a necessity in the public Internet, for accessing in a secure fashion private resources from anywhere. Internet Protocol Security (IPsec) is a standardized VPN technology for serving multiple connectivity scenarios. Implementation of cryptography is widely considered as a performance bottleneck and a target for optimization.
We present a set of system configuration optimizations for the Linux 2.6 kernel network stack implementation, supported by extensive measurements. These optimizations achieve significant throughput gains. Our work demonstrates that comparable performance between plain IP and IPsec connections is possible without altering the implementation of the cryptographic algorithms.

## 1 INTRODUCTION

Internet Protocol (IP) is the standard protocol for moving packets between a source and a destination host in the Internet. There are no inherent security mechanisms defined in IP. Thus, it is easy to manipulate IP packets e.g., alter their contents, change source or destination addresses, and inject fake ones (Bellovin, 2004). Internet Protocol Security (IPsec) is a set of open protocols defined by IETF in RFC 4301–4309. IPsec aims to offer security in the IP layer, for both IPv4 and IPv6.

IPsec defines three fundamental security protocols: Authentication Header (AH), Encapsulating Security Payload (ESP), and Internet Key Exchange (IKE). AH provides sender authentication, data integrity, and anti-replay protection. ESP additionally provides data confidentiality. IKE is the key management protocol and provides the mechanisms to initiate and to periodically refresh cryptographic parameters of the AH and ESP.

IPsec operates in either *transport* or *tunnel* mode. In transport mode IPsec protects only the IP payload i.e., the upper layer information contained in an IP datagram. In tunnel mode IPsec protects the whole IP datagram i.e., the IP header and the IP payload.

IPsec in tunnel mode is commonly used to realize a Virtual Private Network (VPN) over the public Internet. VPN connectivity is attracting significant interest lately due to the increasing need of accessing private resources in a secure manner from any point of the public Internet. The IPsec endpoints can be routers (site-to-site VPN), end-systems (host-to-host VPN), or mixed (host-to-network VPN).

We focus on a site-to-site VPN, like between the head and branch offices of a company. In this case, IPsec operates in tunnel mode and both endpoints hold the same pre-shared keys. Thus, the gateways need not engage in key management operations through the IKE protocol. As a control scenario, we also considered transport mode. Our focus is on attainable throughput for saturating IPsec-protected links of 10, 100, and 1000 Mbps .

In this paper we provide insight on the performance of IPsec as implemented in Linux kernel version 2.6. We present experimental data and propose a set of system configuration optimizations. These optimizations lead to significant throughput gains without intrusive modifications of the implementation. Our work demonstrates that careful system engineering should be applied first, before trying to optimize (an already optimized) implementation of cryptography.

The paper is organized as follows. Section 2 presents IPsec with emphasis on implementation on Linux kernel and performance issues.

Section 3 describes the testbed environment used for performance analysis and Section 4 the measurements and discusses our findings. Section 5 presents the set of system optimizations we propose and discusses their benefits. Finally, Section 6 concludes our findings and proposes future work in the area.

## 2 IPSEC BACKGROUND

### 2.1 Protocols

IPsec is a suite of protocols for securing IP communications by authenticating and encrypting each IP packet of a data stream.

A set of extra headers for the IP datagram that actually provide the VPN services is defined by two new protocols: ESP and AH. The AH protocol offers data source authentication, data integrity, anti-replay protection, but does not offer confidentiality. The ESP protocol provides all the features offered by AH protocol and in addition data privacy. The cryptographic transformations that implement the security services require a set of keys that are available between the send peers. These can be either preshared keys or can be negotiated using IKE protocol.

AH is used to provide connectionless integrity and data origin authentication for IP datagrams using an Integrity Check Value (ICV), as well as protection against replays utilizing a monotonically increasing number for each packet, which can be optionally verified by the receiver. AH provides authentication for as many fields of IP header as possible, as well as for next level protocol data. However, some IP header fields may change in transit and values of such fields cannot be protected by AH. Thus, the protection provided to the IP header by AH is piecemeal. The size of the AH header is 12 bytes plus the variable length ICV. For point-to-point communication, suitable integrity algorithms for the ICV computation include keyed Message Authentication Codes (MACs) based on symmetric encryption algorithms (e.g., AES) or on one-way hash functions (e.g., MD5, SHA1, etc.) Usually the output of the algorithm is truncated to 96 bit for its use in ICV.

ESP can provide the same services as AH and furthermore data confidentiality using cryptographic transformations. Whenever ESP only is used, both confidentiality and integrity services are recommended, in order to avoid some attacks (Bellovin, 1996), (Degabriele and Paterson, 2007). The primary difference in the integrity provided by ESP and AH is the extent of the coverage. Specifically, ESP does not protect any IP header fields unless those fields are encapsulated by ESP e.g., via use of tunnel mode.

IKE is a component of IPsec used for performing mutual authentication and establishing and maintaining security associations (SAs). Among other things, these define the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms. Although establishing this shared state in a manual fashion is possible, it does not scale well, and thus the use of IKE is required.

### 2.2 Transport and Tunnel Mode

VPN tunnels are usually implemented through packet encapsulation. The full packet is wrapped in a new header at the layer VPN operates, in order to provide transparent peer-to-peer connectivity. IPsec supports two modes of operation instead: *transport* mode and *tunnel* mode.

In transport mode, encryption and authentication services are provided only for the payload of the original IP datagram. Transport mode is used for host-to-host communication and it is not compatible with gateway services.

In tunnel mode the original IP datagram is fully encapsulated, providing security services for both the IP header and the payload. Since a new IP header is added, tunnel mode is appropriate for gateway-to-gateway service setups. ESP and AH protocols are available in both modes.

### 2.3 Cryptography

Implementation experience with IPsec in both manual keying mode and in IKE protocol mode has shown that there are so many choices for system administrators to make, that it is difficult to achieve interoperability without careful pre-agreement (Eastlake 3rd, 2005). Thus, the IPsec Working Group agreed that there should be a small number of *named* suites that cover typical security policies (Hoffman, 2005).

These suites are optional and do not prevent implementers from allowing individual selection of each security algorithms. The proposed cryptographic algorithms are 3DES and AES128 in CBC mode for encryption operations and SHA1 and AES-XCBC for integrity operations.

### 2.4 Linux Implementation

FreeS/WAN project provided the first implementation of IPsec for Linux. The implementation consisted of a kernel IPsec stack (KLIPS) and user-space key management daemon named `pluto`. The communication between the two parts is realized over the IPsec-standardized `PF_KEY` interface (McDonald et al., 1998).

Starting from Linux kernel 2.6 series, a "native" IPsec implementation was opted. Being standards-compliant, the kernel component implements the `PF_KEY` interface and can be used in conjunction with any standards-compatible user-space key management component. Examples of such components include `pluto` (now part of the OpenSwan/StrongSwan project), `isakmpd` from the OpenBSD project, `racoon` from the KAME project, and manual keying (no IKE component).

The in-kernel IPsec component interacts with the network processing stack through the standardized `XFRM` in-kernel framework. The Linux kernel provides as generic, in-kernel modules heavily-optimized implementations of various cryptographic algorithms. IPsec reuses through XFRM these modules for its cryptographic needs.

### 2.5 Performance Issues

IPsec introduces extra processing overhead to the network stack, in terms of performing mutual authentication and establishing and maintaining security associations (SAs), as well as cryptographic data transformations. Cryptographic operations for encryption, decryption and hashing introduce overhead unrelated to characteristics of the traffic imposed by protocols beyond the network layer. Instead, SAs manipulation has an impact to the performance that is highly correlated to session-like properties of the traffic (Shue et al., 2005).

In the past, various performance compensating methodologies have been proposed, such as crypto offloading to specialized hardware (Bellows et al., 2003), key caching (Shue et al., 2007), and usage of specific cryptographic algorithms (Elkeelany et al., 2002).

To the best of our knowledge, the impact of IPsec on generic bulk data transfers remains unaddressed, as well as the opportunity to utilize related protocol characteristics and implementation specifics to further compensate the performance overhead of IPsec, for both network throughput and CPU usage. We seek to address these issues in the following.

## 3 METHODOLOGY

For the performance analysis of IPsec we built a testbed that provides both an easy to deploy and reproduce environment and versatility in a variety of measurements scenarios. Our primal focus is on the performance impacts of IPsec for bulk data transfers.

### 3.1 Testbed

The testbed environment consists of personal computers running Slackware 10.1 Linux distribution, with custom, vendor independent kernel builds and optimized-for-throughput TCP/IP stack. In preliminary tests, various kernel versions were used, namely 2.6.0, 2.6.4, 2.6.7, and 2.6.11. All reported results in this paper are taken with kernel version 2.6.11.7.

To ensure the reproducibility of the tests and the accuracy of CPU usage instrumentation, we configure the systems without any unneeded services running, except OpenSSH, which is used to control them and doesn't essentially affect measurements. Furthermore, network interfaces were connected directly (back-to-back) with shielded twisted-paired cables, certified for speeds up to 1 Gbps.

To measure network throughput, we use the `netperf` tool (Jones, 2009). Netperf is capable of measuring network throughput, generating multiple packet sizes, and utilizing different socket sizes, using a single stream. We measure the amount of data sent using a packet stream in a predefined time period. In our tests, we use packet sizes of 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32768 bytes. Each experiment runs for a constant time of ten seconds. This time is sufficient for TCP throughput stabilization. Netperf is configured for up to 12 iterations (minimum 4) to a reach confidence level of 95 with a 3% width of confidence interval. We use the stock kernel socket sizes. We will show later that careful socket size changes contribute to achieving higher throughput.

The built-in features of netperf can be used for CPU usage estimation. In order to to achieve more fine-grained estimations, we opted to collect the appropriate statistics directly from the kernel, using the

Table 1: Testbed components.

| ID | CPU core | Clock/FSB (MHz) | RAM (MB) | NIC | (Mbit/s) |
|---|---|---|---|---|---|
| P2 | Intel Pentium 2 | 541 / 75 | 384 | RTL8139 | 10/100 |
| A | AMD Athlon XP | 1667 / 266 | 512 | RTL8139 | 10/100 |
| P41, P42 | Intel Pentium 4 | 1800 / 100 | 256 | Intel 82546 | 10/100/1000 |
| P4M | Intel Pentium 4M | 2200 / 266 | 512 | BCM 4401 | 10/100 |
| X1, X2 | Intel Xeon | 2800 / 533 | 512 | Intel 82546EB | 10/100/1000 |

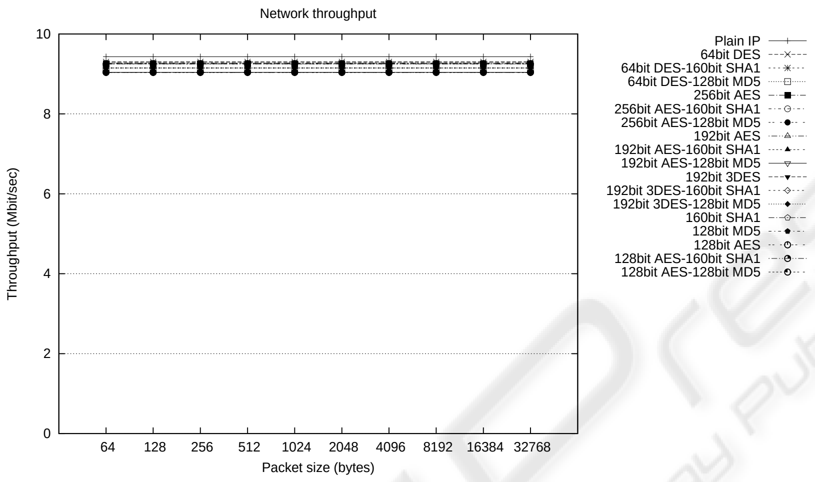

Figure 1: IP and IPsec comparison - 10 Mbps link.

`/proc` interface and `/proc/stat` in particular. A custom application controls remotely the configuration for each system and collects the results from netperf and the samples from `/proc/stat`.

Finally, for an in-depth analysis of performance bottlenecks, we use `oprofile`, a transparent, low-overhead, system-wide profiler (Levon, 2008). In total, we use nine personal computers with varying capabilities, as shown in Table 1.

### 3.2 Experiments

We set up IPsec in both transport and tunnel mode with preshared keys, and measure the performance of plain IP and IPsec with different algorithms for authentication (namely MD5 and SHA1) and for encryption (namely DES, 3DES, and AES), for a single unidirectional TCP stream traffic. We collect measurements for link speeds of 10, 100, and 1000 Mbps.

We use two pairs of systems for testing the 10 Mbps link: (P2,P4M) and (X1,X2). Systems P2 and X1 act as senders and system P4M and X2 as the receivers of the TCP stream. In all experiments, the MTU was set to the maximum allowed of 1500 bytes.

We use three pairs of systems, covering the full range of available hardware capabilities, for testing the 100 Mbps link: (P2,P4M), (A,P4M), and (X1,X2). This variety allows us to compare the scalability of the IPsec implementation on different hardware.

We use the high-end systems for testing the 1 Gbps link: (P41,P42) and (X1,X2). For these systems, we further experiment with customized TCP/IP options, interrupt coalescence capabilities, MTU size, and different network cards. We use the same IPsec setup for all experiments.

## 4 RESULTS

We group the results of the experiments according to the link speed, since it is the definitive constraint in terms of network throughput. Furthermore, it provides a metric for the performance characterization of each hardware platform.

### 4.1 Link of 10 Mbps

IPsec has negligible impact for both network throughput and CPU utilization overhead, even for the low-end systems, as shown in Figure 1. Compared to plain IP, encryption and authentication modes of IPsec can saturate the link with small increase in CPU utilization. There is only a small difference in maximum throughput achieved. This difference is the result of the increased packet sizes due to IPsec encapsulation.

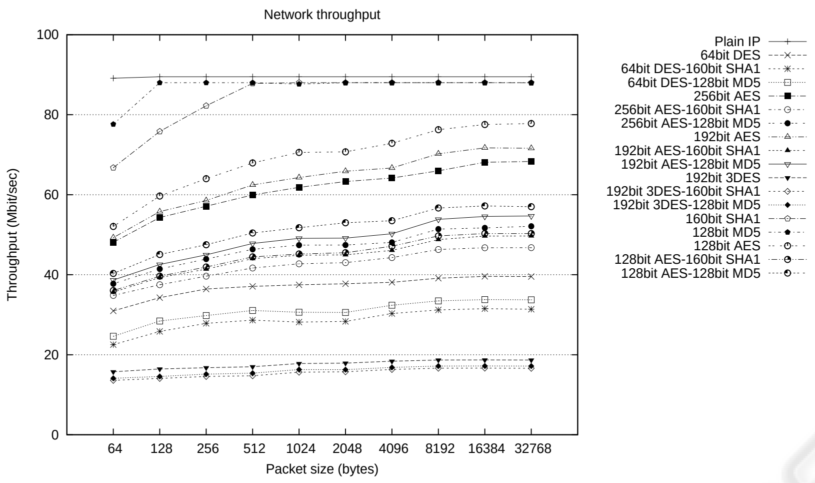

Figure 2: (P2,P4M) IP and IPsec comparison - 100 Mbps link.

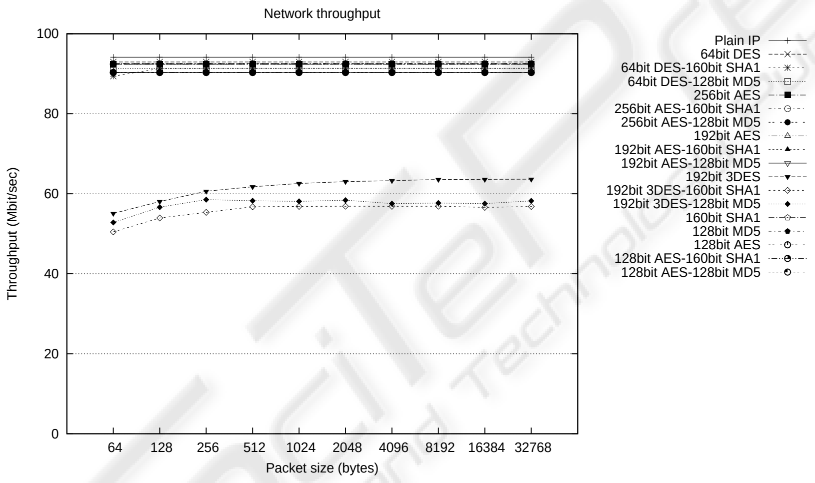

Figure 3: (X1,X2) IP and IPsec comparison - 100 Mbps link.

Tunnel mode exhibits a 3% throughput loss compared to transport mode. The primal reason for this is that tunnel mode encapsulates the full IP datagram and thus, the packet size in the wire is increased.

## 4.2 Link of 100 Mbps

We distinguish three system setups here: the low-end (P2,P4M), the medium (A,P4M), and the high-end (X1,X2). In the low-end (P2,P4M), the impact of cryptographic operations is significant and proportional to their computational complexity, as Figure 2 depicts. Also, the number of packets to process per time unit strongly affects the overall throughput. The throughput grows in a nearly logarithmic rate with the packet size in all but four cases: two of low computational complexity (MD5 and SHA1) and two of high one (3DES+MD5, 3DES+SHA1).

The pair (A,P4M) has enough processing power to handle all algorithm and key size combinations, with virtually no throughput loss for the whole spectrum of packet sizes. The only exceptions are the setup of DES+SHA1 and the ones of 3DES. The setup of 3DES suffers a 40-50 Mbps penalty on throughput and exhibits a 5 Mbps average variation with the packet size.

The high-end setup (X1,X2) doesn't bridge the performance gap of 3DES identified in (A,P4M). However, the gap is now more narrow: 35-40 Mbps, as Figure 3 depicts.

In all setups, the observed difference between transport and tunnel mode is bound to 3-5%.

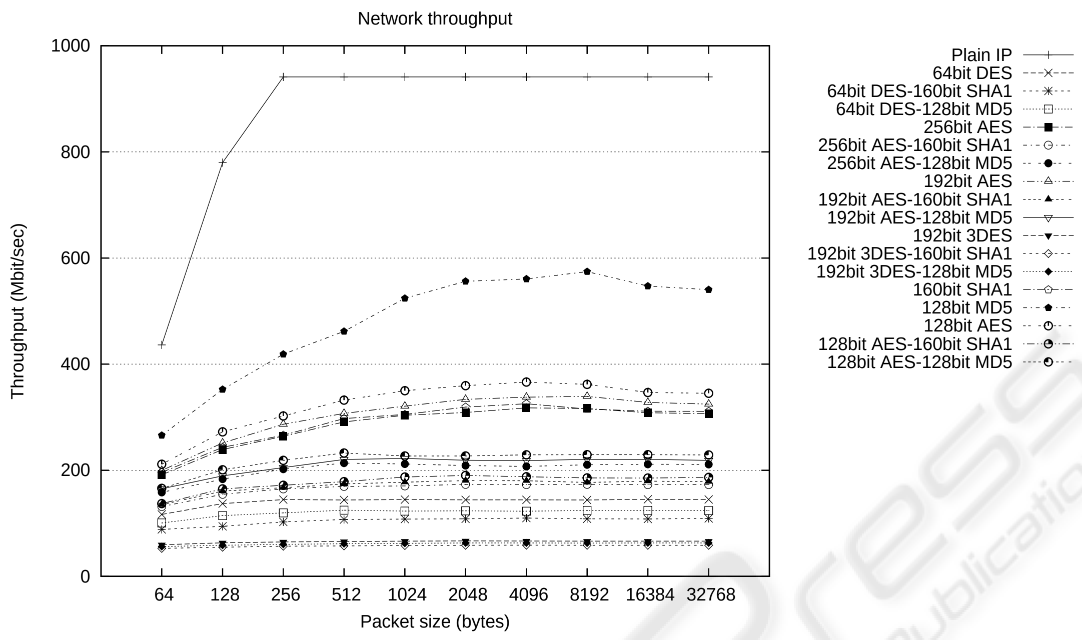

Figure 4: (X1,X2) IP and IPsec comparison - 1 Gbps link.

### 4.3 Link of 1 Gbps

The setup (P41,P42) does not provide enough processing power to saturate the link, even for plain IP. Furthermore, IPsec suffers from a throughput penalty of more than 50%. The total throughput results in respect to packet size and cryptographic algorithm complexity are similar to those of (P2,P4M) for the 100 Mbps link.

The setup (X1,X2) of Xeon processors saturates the link in the case of plain IP for packet sizes larger than 256 bytes, as Figure 4 depicts. However, it is able to provide confidentiality and integrity protection only for up to 300 Mbps, as show in Figure 4. In all setups, the difference between transport and tunnel mode is negligible. It is interesting to note the case of AES128, combined or not with some integrity algorithm. In all cases, the achieved throughput is almost doubled moving from packet size of 64 bytes to 8192 bytes.

Further examination of the collected traces reveals that the CPU usage is 100%, as Figure 5 depict. The vast majority of the time is spent in the *softirq* state. The cryptographic processing of each packet takes place in this state. In the case of (X1,X2) the second more-time consuming state is *IRQ*. In this state the processor handles the interrupt received from the network card. These interrupts occur whenever a packet event takes place, such as on sending and receiving a packet.

## 5 OPTIMIZATIONS

The results of Section 4 indicate that commodity systems of medium capabilities can be utilized to implement Linux IPsec gateways for links up to 100 Mbps. However, there is some area for improvement for 1 Gbps links. The results from the 1 Gbps link on high-end systems provide an interesting insight: serving the interrupts caused by the packet events seems to have a considerable impact on cryptographic algorithm execution of protocol processing.

The implementation of IPsec in a system can be considered as a latency component: each and every packet must pass through the IPsec implementation for one or more of the following operations: encryption, decryption, hash generation, hash verification. Since the function calls required to implement these accumulate and form a longer execution path, it is preferable to process as many bytes as possible on each path traversal. In the following, we explore possible optimizations in all layers of TCP/IP that can affect packet processing time.

TCP/IP is not a static and monolithic set of protocols. Protocol parameters can be configured for specific end-to-end link characteristics. We saw that interrupt processing has a critical role on performance; interrupt coalescence is an excellent candidate for our purpose. Also, MTU size can have an immediate impact, since it can affect maximum allowed packet size. This can reduce the number of packets needed

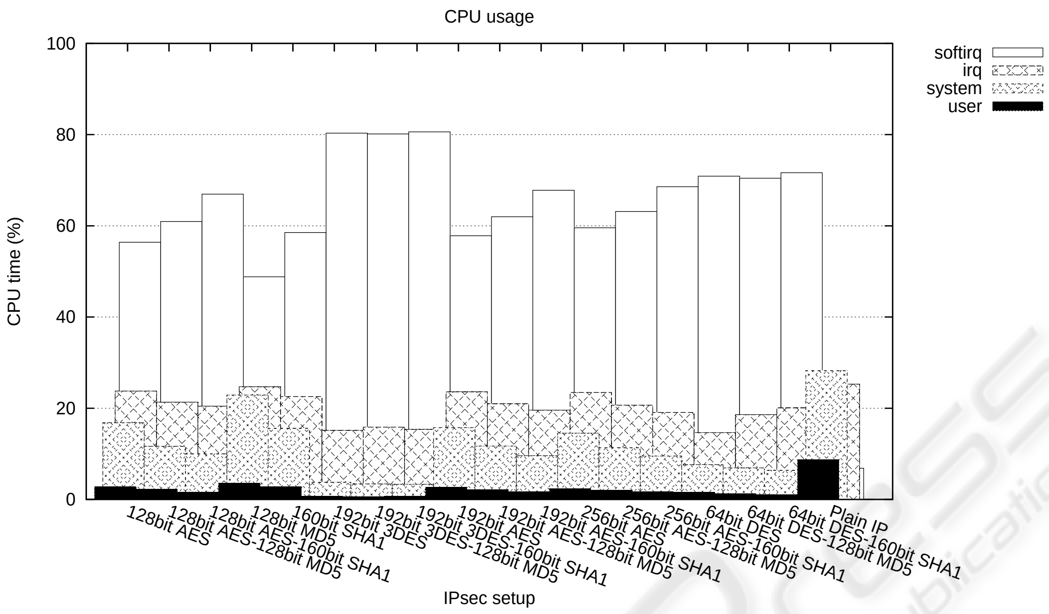

Figure 5: (X1,X2) CPU utilization - 1 Gbps link.

to transmit a specific volume of information and thus, reduce overall packet processing time and total number of interrupts.

The IEEE 802.3 standard dictates a backwards-compatible MTU size of 1500 bytes. For a saturated Gigabit link, the kernel must be able to cope with more than 80,000 packets per second. The so-called “Jumbo frames” have been proposed as a means to reduce packet rate for a given transmission rate. The size of jumbo frames is not standardized but there are currently available products that support MTU sizes of 4096, 8192, 9000, and up to 16110 bytes.

There is no formal agreement on the maximum MTU size for devices supporting jumbo frames. Furthermore, usage of the “path MTU discovery” protocol is not widely adopted (Mogul and Deering, 1990), (Mathis and Heffner, 2007). These facts can lead to connection problems when jumbo frames are enabled along a network path with network devices of different vendors. For controlled environments, where an a priori agreement between interested parties can be achieved, jumbo frames are a desirable feature, since it leads to higher performance, in the means of less CPU and bus utilization which can possibly induce higher throughput.

We explore in the following the throughput improvement gained by each of the above parameters.

## 5.1 TCP/IP Stack Optimizations

Performance improvement of TCP over large bandwidth-delay products paths is accomplished with the addition of standardized extensions (Jacobson et al., 1992), (Mathis et al., 1996). The Linux kernel supports some of these extensions for high performance networking in a customizable, online configurable way. From the available arsenal, we choose to enable the options for timestamps, window scaling and SACK. Furthermore, we experiment with the TCP window size and the default values for TCP send and receive buffers. Specifically, we got the optimum results by setting the min, default and max values for both sending and receiving socket buffers to 87380, 4194304 and 4194304 bytes accordingly. The kernel selects the appropriate value depending on the available memory.

These customizations lead to a gain of 20 Mbps for plain IP and IPsec in AH mode.The larger the size of the packets, the bigger the gain. Notably, the TCP/IP stack optimizations are more beneficial to IPsec in ESP mode (AES, DES, and 3DES). They contribute up to 150 Mbps more throughput. In general, TCP optimizations not only provided a throughput boost, but also exhibited more stable behavior.

## 5.2 Interrupt Coalescence

Whenever a packet is received by the network interface card (NIC), it raises an interrupt. This interrupt

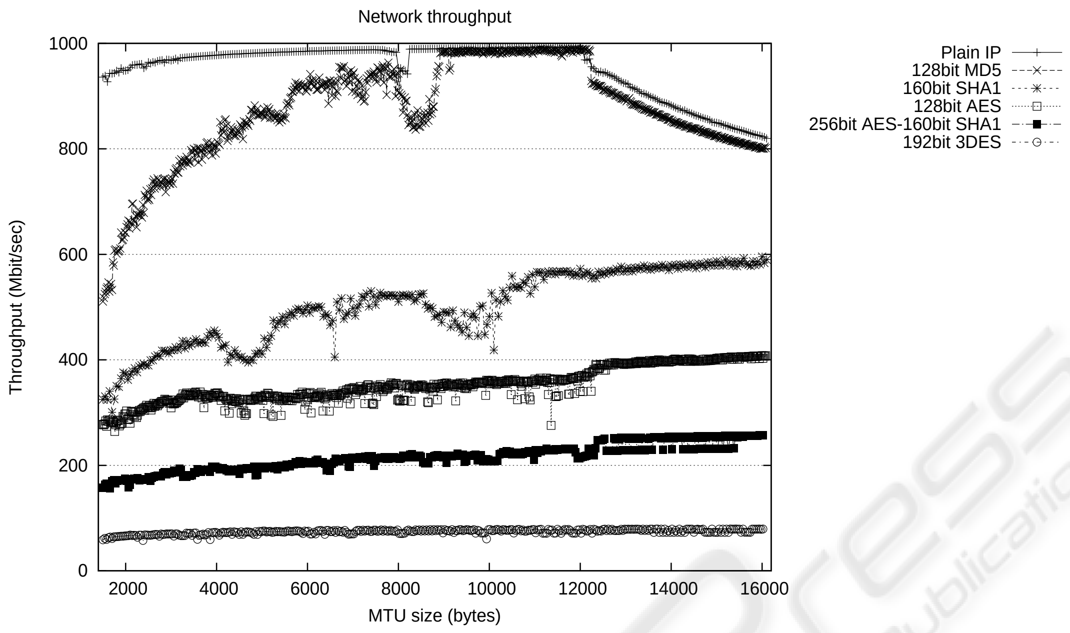

Figure 6: MTU contribution.

must be served by the appropriate interrupt handler of the kernel. The handler processes the event in IRQ context, where all interrupts are disabled. Thus, it is necessary to minimize the processing time, and delegate the rest of the network processing to a "softirq" task. This task can be scheduled for later execution. The kernel must process thousands of packets per second in a saturated link. While processing the packets, the kernel must continuously suspend and resume other processes. This interrupt "storm" has an important impact not only on the network but also on the overall system performance, even for modern, high-speed processors.

The Linux kernel implements NAPI, a heuristics-based workaround to cope with this storm (Salim et al., 2001). NAPI is a hardware agnostic, hybrid interrupts/polling mechanism. If the interrupts for the NIC reach a certain rate, the kernel disables these interrupts and processes the packets using a polling mechanism. When the rate drops below the threshold, the kernel switches back to interrupt-handling mode. This approach provides interrupt overload reduction, removes packet re-ordering issues in SMP architectures, and handles (early drop) system overloading due to network traffic better.

We run a set of experiments using the NAPI-enabled Intel e1000 network drivers in system pair (X1,X2). For packet sizes less than 1024 bytes there is a small throughput decrease of about 10 Mbps. However, for larger packets, there is a throughput increase of 50 Mbps.

The differences between transport and tunnel mode are negligible for MTU size of 1500 bytes. We further experimented with the so-called "jumbo" frames for an MTU of 9000 bytes. Our results lead to three observations:

- Six times larger MTU has an amplifying effect upon the previous results: up to 40 Mbps less for small packets and 40-120 Mbps more for larger ones.
- The increased MTU yields to more unstable results, with considerable standard deviation. Until now, the standard deviation was near zero.
- There is a significant throughput increase of 20-100 Mbps for small and up to 450 Mbps (AH, MD5) for large packets in tunnel mode. This is the only case that we observed a differentiation between transport and tunnel mode.

## 5.3 MTU

We extensively tested the effects of MTU size in the system pair (X1,X2), after enabling the TCP optimizations described above and the NAPI. We tested effects in plain IP, in IPsec in tunnel mode using AH (MD5 and SHA1), ESP (AES128, 3DES), and combined ESP and AH (AES256 and SHA1).

The performance gains are rather strong, as Figure 6 depicts: MD5 peaks at 985 Mbps, the same as plain IP and the combined ESP and AH operation mode achieves 100 Mbps more throughput.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper we analyzed the performance of the Linux native IPsec implementation, for both transport and tunnel mode. The analysis indicates that even with commodity systems, we can easily saturate links up to 100 Mbps, without any significant penalty for throughput. IPsec falls short of expectations in saturating Gigabit links. The implementation of cryptographic algorithms can be an attractive target for optimization. However, detailed system analysis revealed that the problem is not processing power per se. Rather, it is the combined effect of the IRQ storm and the softirq kernel state due to IPsec processing, even with increased MTU sizes. Once the real cause is identified, careful system engineering can lead to significantly increased IPsec throughput.

Future work in this area includes extensive testing of advances in Linux kernel network stack, and use of hardware-based cryptographic processors for offloading security operations. Another direction is the comparison with the BSD IPsec stack variants and validation of our findings in higher link speeds; 10 Gbps is a good candidate for this. Finally, it would be interesting to compare our results in scenarios with user-space based VPN solutions.

## REFERENCES

Bellovin, S. (2004). A look back at “security problems in the TCP/IP protocol suite”. In *ACSAC ’04: Proceedings of the 20th Annual Computer Security Applications Conference*, pages 229–249, Washington, DC, USA. IEEE Computer Society.

Bellovin, S. M. (1996). Problem areas for the IP security protocols. In *Proceedings of the Sixth USENIX Security Symposium*, pages 205–214.

Bellows, P., Flidr, J., Gharai, L., Perkins, C., Chodowiec, P., and Gaj, K. (2003). IPsec-protected transport of HDTV over IP.

Degabriele, J. P. and Paterson, K. G. (2007). Attacking the IPsec standards in encryption-only configurations. Cryptology ePrint Archive, Report 2007/125.

Eastlake 3rd, D. (2005). Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). RFC 4305 (Proposed Standard). Obsoleted by RFC 4835.

Elkeelany, O., Matalgah, M., Sheikh, K., Thaker, M., Chaudhry, Medhi, G., and Qaddour, J. D. (2002). Performance analysis of IPSec protocol: encryption and authentication.

Hoffman, P. (2005). Cryptographic Suites for IPsec. RFC 4308 (Proposed Standard).

Jacobson, V., Braden, R., and Borman, D. (1992). TCP Extensions for High Performance. RFC 1323 (Proposed Standard).

Jones, R. (2009). Netperf. Retrieved April 27, 2009 from http://www.netperf.org.

Levon, J. (2008). OProfile - A System Profiler for Linux. Retrieved April 27, 2009 from http://oprofile.sourceforge.net/.

Mathis, M. and Heffner, J. (2007). Packetization Layer Path MTU Discovery. RFC 4821 (Proposed Standard).

Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A. (1996). TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard).

McDonald, D., Metz, C., and Phan, B. (1998). PF_KEY Key Management API, Version 2. RFC 2367 (Informational).

Mogul, J. and Deering, S. (1990). Path MTU discovery. RFC 1191 (Draft Standard).

Postel, J. (1981). Transmission Control Protocol. RFC 793 (Standard). Updated by RFC 3168.

Salim, J. H., Olsson, R., and Kuznetsov, A. (2001). Beyond softnet. In *ALS ’01: Proceedings of the 5th annual Linux Showcase & Conference*, pages 18–18, Berkeley, CA, USA. USENIX Association.

Shue, C., Shin, Y., Gupta, M., and Choi, J. Y. (2005). Analysis of IPSec overheads for VPN servers. In *IEEE ICNPs NPSec Workshop*.

Shue, C. A., Gupta, M., and Myers, S. A. (2007). IPSec: Performance Analysis and Enhancements. In *IEEE Conference on Communications (ICC)*.