A Study of the Relative Costs of Network Security Protocols*

Stefan Miltchev
miltchev@dsl.cis.upenn.edu
University of Pennsylvania

Sotiris Ioannidis
sotiris@dsl.cis.upenn.edu
University of Pennsylvania

Angelos D. Keromytis
angelos@cs.columbia.edu
Columbia University

Abstract

While the benefits of using IPsec to solve a significant number of network security problems are well known and its adoption is gaining ground, very little is known about the communication overhead that it introduces. Quantifying this overhead will make users aware of the *price* of the added security, and will assist them in making well-informed IPsec deployment decisions.

In this paper, we investigate the performance of IPsec using micro- and macro-benchmarks. Our tests explore how the various modes of operation and encryption algorithms affect its performance and the benefits of using cryptographic hardware to accelerate IPsec processing. Finally, we compare against other secure data transfer mechanisms, such as SSL, `scp(1)`, and `sftp(1)`.

1 Introduction

The increasing need for protecting data communications has led to the development of several protocols that provide very similar services, most notably data secrecy/integrity and origin authentication. Examples of such protocols include IPsec, SSL/TLS, and SSH[8, 2, 11]. While each of the protocols is based on a different set of assumptions with respect to its model of use, implementation characteristics, and supporting applications, they all fundamentally address the same problem, namely to protect the secrecy and integrity of data transferred over an untrustworthy network such as the Internet.

Securing the data while in transit is not sufficient by itself in building a secure network: data storage, key management, user interface, and backup security must also be addressed to provide a comprehensive security posture. These are often overlooked, yet are an essential part of a secure system. In this paper, we aim to quantify the costs of specific mechanisms and clarify the options available to system and network architects. In particular, we wish to quantify the performance implications of using various security protocols that are either widely used (*e.g.,* SSL and SSH) or are expected to be in wide use (*e.g.,* IPsec).

Compared to other network security mechanisms, IPsec offers many architectural advantages. Firstly, he details of network security are usually hidden from applications, which therefore automatically and transparently take advantage of whatever network-layer security services their environment provides. More importantly, IPsec offers a remarkable flexibility not possible at higher or lower network layers: security can be configured end-to-end (protecting traffic between two hosts), route-to-route (protecting traffic passing over a particular set of links), edge-to-edge (protecting traffic as it passes between "trusted" networks via an "untrusted" one, subsuming many of the current functions performed by network firewalls), or in any other configuration in which network nodes can be identified as appropriate security endpoints. However, a perception of complexity[1] and reduced performance have acted as deterring factors in its deployment and use. The former point is being addressed by new APIs and refinement of administrative interfaces that make configuration and use of IPsec easier. However, the performance issue has not received adequate examination.

In this paper, we investigate the performance of IPsec using micro- and macro-benchmarks. Our tests are designed to explore how the various modes and encryption algorithms affect its performance, the benefits of using hardware accelerators to assist the IPsec cryptographic framework, and finally compare against other secure transfer mechanisms, such as SSL, `scp(1)`, and `sftp(1)`. We use the OpenBSD operating system as our experimental platform, because of its support for

*This work was supported by DARPA under Contract F39502-99-1-0512-MOD P0001.

[1] In particular with respect to configuration tools, and PKI support.

cryptographic hardware accelerators and its native IPsec implementation[9].

2 System Architecture

In this section we briefly describe the OpenBSD IPsec and Kernel Cryptographic Framework architecture. Since the goal of this paper is not to discuss the implementation details, we refrain from going into too much depth.

2.1 IPsec

The IP Security architecture [8], as specified by the Internet Engineering Task Force (IETF), is comprised of a set of protocols that provide data integrity, confidentiality, replay protection, and authentication at the network layer. This positioning in the network stack offers considerable flexibility in transparently employing IPsec for different roles (*e.g.,* building Virtual Private Networks, end-to-end security, remote access, *etc.*). Such flexibility is not possible at higher or lower levels of the network stack.

The overall IPsec architecture is very similar to previous work [5] and is composed of three modules:

- The data encryption/authentication protocols [6, 7]. These are the "wire protocols," used for encapsulating IP packets to be protected. They simply provide a format for the encapsulation; the details of the bit layout are not particularly important for the purposes of this paper.

  Outgoing packets are authenticated, encrypted, and encapsulated just before being sent to the network, and incoming packets are decapsulated, verified, and decrypted immediately upon receipt. These protocols are typically implemented inside the kernel, for performance and security reasons. A brief overview of the OpenBSD kernel IPsec architecture is given in Section 2.2.

- A key exchange protocol (*e.g.,* IKE[4]) is used to dynamically establish and maintain Security Associations (SAs). An SA is the set of parameters necessary for one-way secure communication between two hosts (*e.g.,* cryptographic keys, algorithm choice, ordering of transforms, *etc.*). Although the wire protocols can be used on their own using manual key management, wide deployment and use of IPsec in the Internet requires automated, on-demand SA establishment. Due to its complexity, the key management protocol is typically implemented as a user-level process.

- The policy module governs the handling of packets on their way into or out of an IPsec-compliant host. Even though the security protocols protect the data from tampering, they do not address the issue of which host is allowed to exchange what kind of traffic with what other host. This module is in fact split between the kernel (which decides what level of security incoming or outgoing packets should have) and user space (making higher-level decisions, *e.g.,* which user is allowed to establish SAs and with what parameters).

For more details on their implementation in OpenBSD, see [3].

2.2 OpenBSD IPsec Implementation

In the OpenBSD kernel, IPsec is implemented as just another pair of protocols (AH and ESP) sitting on top of IP. Thus, incoming IPsec packets destined to the local host are processed by the appropriate IPsec protocol through the protocol switch structure used for all protocols (*e.g.,* TCP and UDP). The selection of the appropriate protocol is based on the protocol number in the IP header. The SA needed to process the packet is found in an in-kernel database using information retrieved from the packet itself[2]. Once the packet has been correctly processed (decrypted, authenticity verified, *etc.*), it is re-queued for further processing by the IP module, accompanied by additional information (such as the fact that it was received under a specific SA) for use by higher protocols and the socket layer.

Outgoing packets require somewhat different processing. When a packet is handed to the IP module for transmission (in `ip_output`), a lookup is made in the Security Policy Database (SPD) to determine whether that packet needs to be processed by IPsec. The SPD in OpenBSD is implemented as an extension to the standard BSD routing table. The decision is made based on the source/destination addresses, transport protocol, and port numbers. If IPsec processing is needed, the lookup will also specify what SA(s) to use for IPsec processing of the packet (even to the extent of specifying encryption/authentication algorithms to use). If no suitable SA is currently established with the destination host, the packet is dropped and a message is sent to the key management daemon through the **PF_KEY** interface [10]. It is then the key management's task to negotiate the necessary SAs. Otherwise, the packet is processed by IPsec and passed to `ip_output` again for transmission. The packet also carries an indication as to what IPsec processing has already occured to it, to avoid infinite processing

[2] Specifically, the destination IP address, the 32-bit SPI field from the IPsec header, and the IPsec protocol (ESP or AH) number.

loops.

2.3 OpenBSD Cryptographic Framework

To improve the performance of the cryptographic operations in IPsec, we developed a framework for cryptographic services in OpenBSD that abstracts the details of specific cryptographic hardware accelerator cards behind a kernel-internal API. The details of the framework are beyond the scope of this paper. However, we give a brief description here so the reader has the proper context within which to consider our measurements.

Besides abstracting the API for accessing these cards, the framework was designed with these goals in mind:

- **Asynchronous operation:** The kernel should not have to wait until the hardware finished the requested operation.
- **Load balancing:** If multiple cryptographic accelerators are present, they should be utilized such that throughput is maximized.
- **No dependence on hardware:** If no hardware accelerators are present, the system should offer the same services (albeit at lower performance).
- **Application independence:** Although the framework was initially developed for use with IPsec, it should be possible to use it to accelerate other kernel operations (*e.g.,* filesystem or swap encryption) and user-level applications (*e.g.,* the OpenSSL library).
- **Support for public key operations.** This is currently work in progress.

Work on the framework is still in progress, but the main skeleton is present and has been in use with IPsec since OpenBSD 2.8.

The framework presents two interfaces: one to device drivers, which register with the framework and specify what algorithms and modes of operations they support; and one to applications (*e.g.,* IPsec), which create "sessions" and then queue requests for these.

Sessions are used to create context in specific drivers (selected by the framework based on a best-match basis, with respect to the algorithms used) and can migrate between different cryptographic accelerators (*e.g.,* if a card fails or is plugged out of the system, as may be the case with PCMCIA adaptors, or if a higher-priority session arrives). This is achieved by requiring that all necessary context is provided with every request, regardless of the fact that a session has been created (the context is kept at the application and inside the accelerator cards and is not cached by the framework itself).

Applications queue requests on sessions, and the cryptographic framework, running as a kernel thread and periodically processing all requests, routes them to the appropriate driver. Once the request has been processed, a callback function provided by the application is invoked, which continues processing (in the IPsec case, passes the packet to `ip_output()` for transmission). A software pseudo-driver registers with the framework as a driver of last resort (if any other driver can process the session, it will be preferred).

User-level applications (*e.g.,* the OpenSSL library or the SSH daemon) can access the hardware through the */dev/crypto* device, which acts as another kernel application to the framework, using the same API. Public key operations are modeled in the same way.

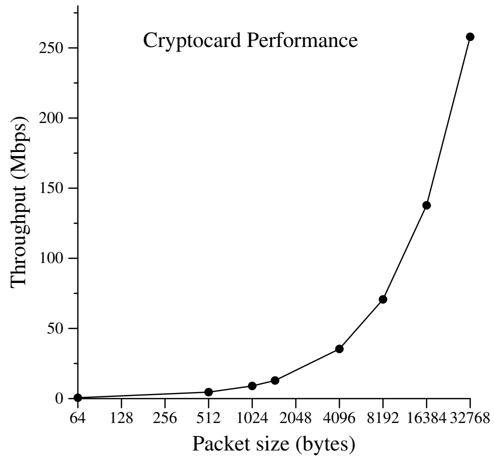

Figure 1: **Cryptographic card performance.**

Smart ethernet cards Although the cryptographic framework does not directly take advantage of ethernet cards that support IPsec processing offloading (since they are not general-purpose cryptographic accelerators), we extended the IPsec stack to use them. Unfortunately, at the time of writing this paper, driver support for these cards was not completed and thus we could not measure their performance. The cards of this type we are familiar with are 100Mbps full-duplex, and it seems reasonable (given our results with dedicated cryptographic processors) to assume that they can achieve that performance. Unfortunately, at the time this paper was written, we did not have enough information to write a device driver that could take advantage of such features.

3 Evaluation

Our test machines are x86 architecture machines running OpenBSD 3.0. More specifically, they are 1 GHz

Intel PIII machines with 256 MB of registered PC133 SDRAM, 10 GB Western Digital Protege IDE hard drives, Intel PRO/1000 F network adapters and some 3Com 3c905B 100Mbps network adapters. We chose Supermicro 370DE6 motherboards based on the ServerWorks Serverset III HE-SL chipset with dual PCI buses. Thus we were able to place our gigabit cards and cryptocards on separate PCI buses. For some of our experiments we used the Broadcom 5820 crypto-cards. The manufacturer of these cards advertises 300Mbps 3DES; our own evaluation showed a peak measured performance of around 260Mbps, probably due to operating system overhead. We summarize our results in Figure 1. Notice that even in the best case (host-to-host, large socket buffers), we only get slightly over half the nominal throughput. We believe this is a deficiency in the device driver, but did not investigate in great detail. However, given that (a) the performance of all the security protocols we measure is dominated by the cost of encryption, (b) the throughput of those protocols is markedly lower than the unencrypted protocols (*ftp, http,* and unencrypted *ttcp*[1]), and (c) we present absolute performance numbers, this should not affect the validity of our experiments: better-performing ethernet cards/drivers would only improve the throughput numbers of the unencrypted protocols.

3.1 Benchmark Variables

In order to understand the performance trade-offs of using IPsec as well as how it compares to other approaches we designed a set of performance benchmarks. Our experiments were designed in such a way as to explore a multitude of possible setups.

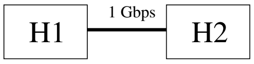

Figure 2: **Host-to-Host topology.**

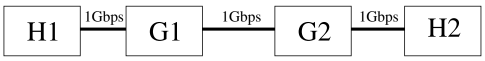

Figure 3: **Host-to-Gateway-to-Gateway-to-Host topology. In this case experiments that use IPsec form a tunnel between gateways.**

Our experiments take into consideration five variables: the type of utility used to measure performance, the type of encryption/authentication algorithm used by IPsec (or other applications), the network topology, use of cryptographic hardware accelerators, and the effects that the added security has on the performance of the system. For the IPsec experiments, we use manually configured SAs; thus, the performance numbers do not include dynamic SA setup times. For SSL, scp, and sftp, bulk data transfers include the overhead of session setup; however, that overhead is negligible compared to the cost of the actual data transfer.

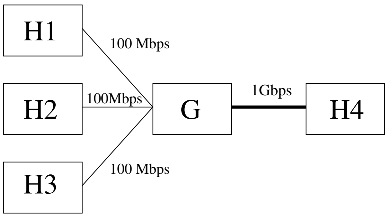

Figure 4: **3 Hosts-to-Gateway-to-Host topology. We use two IPsec tunnel configurations, end-to-end (where the 3 hosts form tunnels to the end host) and gateway-to-host (H4).**

Large filetransfer experiments were repeated 5 times, all other experiments were repeated 10 times and the mean was taken. Error bars in our graphs represent one standard deviation above and below the mean. Graphs presenting ttcp measurements do not show error bars to avoid clutter, however the standard deviation is small in all cases.

We will go into more detail about each experiment in the following section.

3.2 Micro-benchmark Results

In Figures 5, 6, 7 and 8, we explore different network configurations using the ttcp benchmarking tool. We explore how the various encryption algorithms affect performance and how much benefit we get out of hardware cryptographic support. The host-to-host topology is used as the base case, and should give us the optimal performance of any data transfer mechanism in all scenarios. The other two topologies map typical VPN and "road warrior" access scenarios.

The key insight from our experiments is that even though the introduction of IPsec seriously worsens performance, our crypto hardware improves its performance (relative to pure-software IPsec) by more than 100%, especially in the case of large packets. For the host-to-host experiment, we see that throughput over IPsec varies from 40% of the unencrypted transfer (for small packet sizes) to 30% (for 8KB packets [3]). We notice a similar situation in the VPN configuration (host-gateway-gateway-host). In the last two scenarios, the difference

[3]This is the size of the buffer that the *ttcp* benchmark is using for reading and writing to the network.

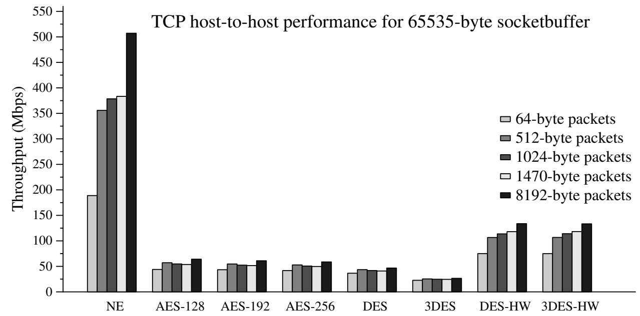

Figure 5: **The ttcp utility over TCP, for the host-to-host network configuration with 65535 bytes of socket buffer. *NE* means No Encryption. We measure the AES algorithm with three different key sizes (128, 192, and 256 bits), as well as DES (56 bits) and 3DES (168 bits). The suffix "-HW" indicated use of a hardware accelerator for that cryptographic algorithm. In all cases where IPsec is used, we use HMAC-SHA1 as the data integrity/authentication algorithm; when hardware acceleration is used, HMAC-SHA1 is also accelerated.**

in performance is less marked between the unencrypted and the hardware-accelerated cases, since the aggregate throughput of the three hosts on the left is limited to at most 300 Mbps (due to the topology).

In our experiments, we also noticed some anomalous behavior with 512 byte packet sizes, we believe that this has to do with buffer mis-alignments in the kernel and will investigate further in the future using profiling.

In our previous experiments we stress-tested IPsec by maximizing network traffic using ttcp. In our next set of experiments, we investigate how IPsec behaves under *"normal"* network load and how it compares with other secure network transfer mechanisms like `scp(1)` and `sftp(1)`. Our tests measure elapsed time for a large file transfer in two different network configurations, host-to-host and host-to-gateway-to-gateway-to-host. In the first case, IPsec is used in an end-to-end configuration; in the second case, IPsec is done between two gateways.

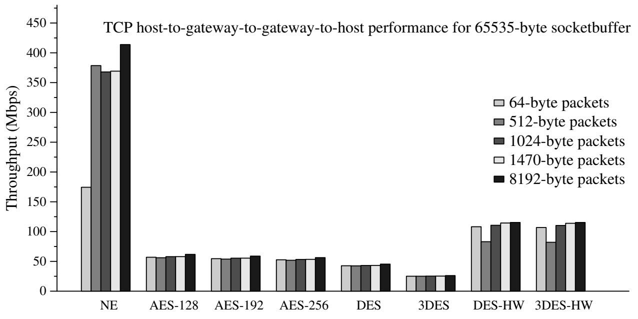

Figure 6: **The ttcp utility over TCP, for the host-to-gateway-to-gateway-to-host network configuration with 65535 bytes of socket buffer. IPsec is used between the two gateways.**

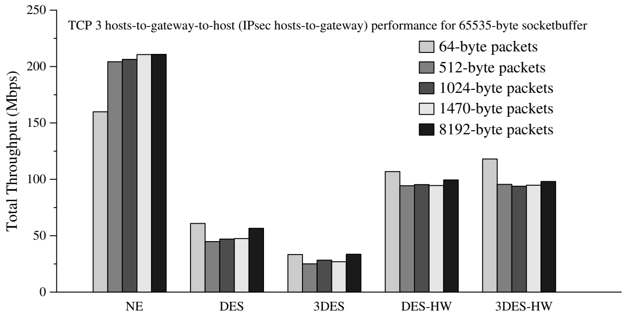

Figure 7: **The ttcp utility over TCP, for the 3 hosts-to-gateway-to-host network configuration with 65535 bytes of socket buffer. In this case we create an IPsec tunnel between hosts H1, H2, H3 and the gateway.**

Figures 9 and 10 present our results. Since we are doing large file transfers, we easily amortize the initialization cost of each protocol. Comparing the two figures, we notice that most of the time is actually spent by the file system operations, even after we normalize the file sizes. Another interesting point is that when we use IPsec the file transfer is quicker in the gateway network topology compared to the direct link. At first this might seem counter-intuitive, however it is easily explained: in the gateway case, the IPsec tunnel is located between the gateways, therefore relieving some processing burden from the end hosts that are already running the ftp program. This leads to parallel processing of CPU and I/O operations, and consequently better performance, since the gateway machines offload the crypto operations from the end hosts. Note that IPsec is not used for the plaintext ftp, scp, and sftp measurements.

Figures 11 and 12, compare IPsec with `ssl(3)` as

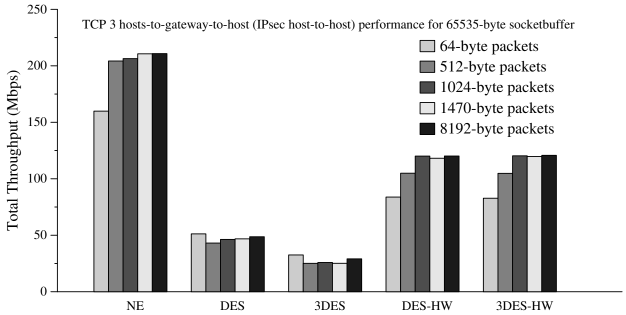

Figure 8: **The ttcp utility over TCP, for the 3 hosts-to-gateway-to-host network configuration with 65535 bytes of socket buffer. In this case, all 3 hosts on the left form IPsec tunnels to the end host.**

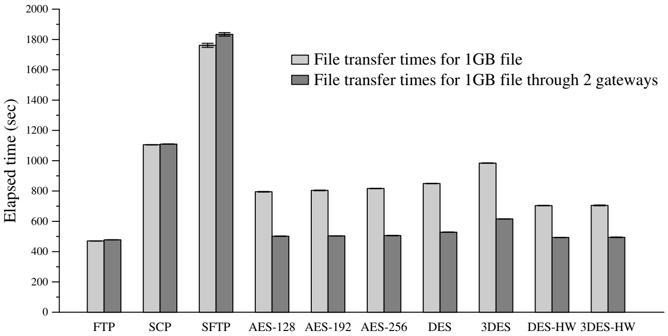

Figure 9: **Large file transfer using ftp, scp, sftp, and ftp over IPsec, over two different network topologies. The file is read and stored in the regular Unix FFS. IPsec is not used for the plaintext ftp, scp, and sftp examples, in either setup.**

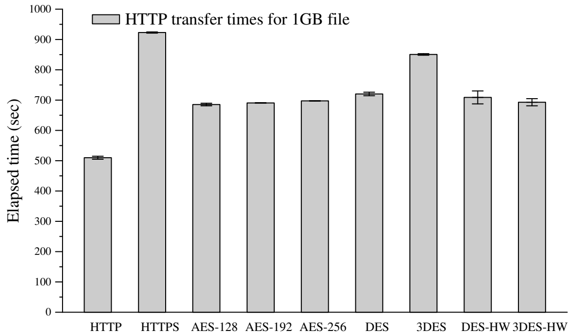

Figure 11: **Large file transfer using http, https, and http over IPsec, on a host-to-host network topology. The file is read and stored in the regular Unix FFS.**

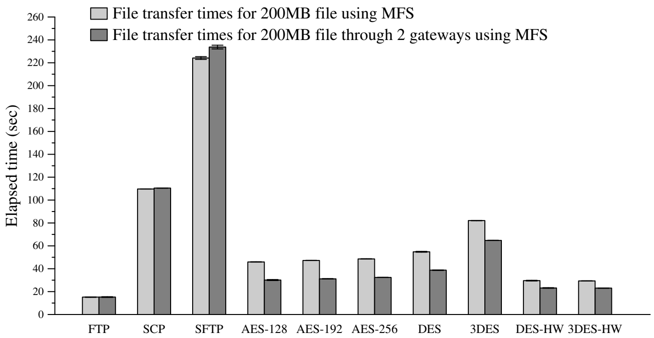

Figure 10: **File transfer using ftp, scp, sftp, and ftp over IPsec, over two different network topologies. The file is read and stored in the Unix memory file system (MFS).**

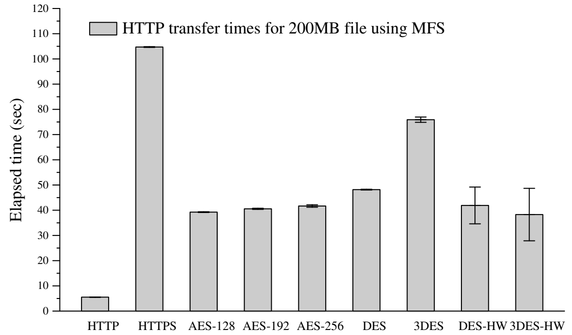

Figure 12: **Large file transfer using http, https, and http over IPsec, on a host-to-host network topology. The file is read and stored in the Unix memory file system (MFS).**

used by HTTPS, the network configuration is host-to-host. We used `curl(1)` to transfer a large file from the server to the client. Once again IPsec proves to be a more efficient way of ensuring secure communication.

Figure 13 provides insight on the latency overhead induced by IPsec and HTTPS. We used `curl(1)` to transfer a very small file from the server to the client. The file contained just an opening and closing html document tag. We timed 1000 consecutive transfers. The latency overhead introduced by IPsec over cleartext HTTP is only 10%. There was practically no difference between using manual keying and *isakmpd*, as the cost of key and security association management gets amortized over many successive connections. The need to perform a handshake for each connection clearly hurts performance in the case of HTTPS.

In our final set of experiments, we explore the impact IPsec has on the operation of the system. We selected a CPU-intensive job, *Sieve of Eratosthenes* [4], which we run while constantly using the network. We tested the impact of a number of protocols to the performance of other jobs (in this case, the sieve) running on the system. In Figure 14, we present the execution times of our CPU intensive job while there is constant background network traffic. To understand the results of Figure 14, one needs to understand how the BSD scheduler works. In BSD, CPU intensive jobs that take up all their quanta have their priority lowered by the operating system. When executing the sieve while using ftp, the sieve program gets its priority lowered and therefore ends up taking more time to finish. In the case where it is run with `scp(1)` or `sftp(1)`, which are themselves CPU intensive because of the crypto operations, the sieve finished faster. When the sieve is run with IPsec traffic, the crypto operations are performed by the kernel and therefore the sieve gets fewer CPU cycles. With hardware cryptographic support, the kernel takes up less CPU which leaves more cycles for the sieve. In the case of HTTPS background network traffic, the CPU cycles spent in crypto processing were not enough to affect the priority of the sieve.

[4] Sieve of Eratosthenes is an algorithm for computing prime numbers. We run `primes(6)`, a program that uses this algorithm which is CPU intensive, to emulate a loaded gateway machine

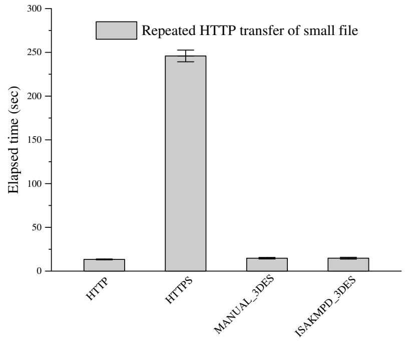

Figure 13: **Small file transfer using http, https, and http over IPsec (using manual and automatic keying via *isakmpd*), on a host-to-host network topology. We timed 1000 transfers of the file. The 3DES algorithm was used for encryption.**

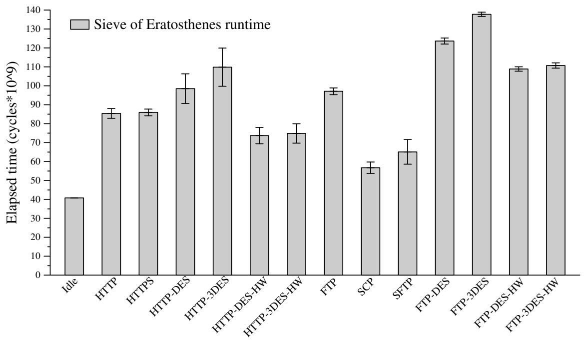

Figure 14: **IPsec introduced overhead on the normal performance of a system. Impact on the execution time of CPU intensive job (sieve) on a system that uses IPsec.**

3.3 Macro-benchmark Results

All the experiments we run so far were designed to explore specific aspects of the security protocols, under a variety of configurations. In this section we present benchmarks that reflect a more realistic use of these protocols.

For our first macro-benchmark, we created a local mirror of the `www.openbsd.com` site, 728 files and a total of 21882048 bytes, to a server machine. We then used `wget(1)` from a client machine to transfer the whole tree hierarchy over the Intel PRO/1000F network adapters. We used `wget(1)` instead of `curl(1)` because of its support for recursive web transfers. Four different ciphers/modes were used for HTTPS. The HTTPS tests used server certificates. The IPsec tests were conducted using manual keying with DES, 3DES, AES and hardware accelerated DES and 3DES. Finally, for completeness, we also included ephemeral Diffie-Hellman results for HTTPS. We present the results in Figure 15.

Our second macro-benchmark is the compilation of

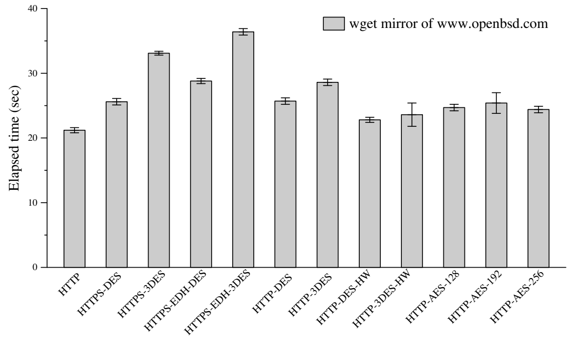

Figure 15: **Mass transfer of a web tree hierarchy using wget.**

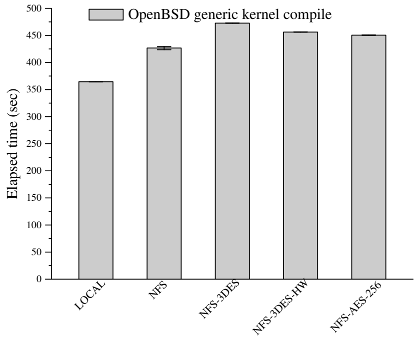

Figure 16: **Compilation of the OpenBSD kernel over NFS, with and without use of IPsec.**

the OpenBSD source over NFS (see Figure 16). We present results for 3DES with and without hardware support, as well as AES. As expected, using hardware support for the encryption is particularly useful when the system is burdened with intensive CPU and filesystem loads.

4 Discussion

One lesson that can be drawn from our experiments is that the current generation of hardware cryptographic accelerators is not sufficient to support ubiquitous use of encryption. Figure 1 points to one problem: the nominal performance of crypto cards is only achieved for large buffer/packet sizes. Since a large percentage (up to 40%) of the packets in a TCP bulk-transfer is 40 bytes, we can see that much of the benefit of such hardware is lost: the cost of card and DMA initialization, PCI transfers, and interrupt handling is roughly comparable to the cost of pure-software encryption, especially as processor speeds increase. This observation suggests that one possible solution is a hybrid approach, where the system uses software encryption for small packets, and hardware encryption for large ones. Another possible solution could be integrating cryptographic functionality with the network interface, which would also improve CPU utilization by offloading the encryption.

One argument against this is the versatility of separate cryptographic components, which allows their use by many other applications (*e.g.,* filesystem encryption, database and other user-level processes that do crypto, *etc.*). While this may be a valid argument in the case of user-level processes, we believe that cryptographic accelerators can be integrated with other I/O devices that can use them more efficiently (in particular, disk and tape controllers). The declining cost of high-performance cryptographic chips makes this a viable alternative to dedicated processors.

5 Concluding Remarks

In this paper, we investigated the costs of network security protocols. We used a variety of benchmarks to determine how IPsec performs under a wide range of scenarios. Our experiments (and in particular our macrobenchmarks) have shown that IPsec outperforms all other popular schemes that try to accomplish secure network communications. Even though this safety comes at a price, which is present no matter which protocol one uses, it is possible to get enough performance for practical use by using dedicated cryptographic hardware. This price may easily be acceptable for many applications and environments, given the remarkable flexibility and transparency offered by IPsec.

References

[1] TTCP: a test of TCP and UDP Performance. USNA, 1984.

[2] T. Dierks and C. Allen. The TLS protocol version 1.0. Request for Comments (Proposed Standard) 2246, Internet Engineering Task Force, January 1999.

[3] Niklas Hallqvist and Angelos D. Keromytis. Implementing Internet Key Exchange (IKE). In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*, pages 201–214, June 2000.

[4] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[5] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network-Layer Security Under Unix. In *Fourth Usenix Security Symposium Proceedings*. USENIX, October 1993.

[6] S. Kent and R. Atkinson. IP Authentication Header. Request for Comments (Proposed Standard) 2402, Internet Engineering Task Force, November 1998.

[7] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). Request for Comments (Proposed Standard) 2406, Internet Engineering Task Force, November 1998.

[8] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, November 1998.

[9] A. D. Keromytis, J. Ioannidis, and J. M. Smith. Implementing IPsec. In *Proceedings of Global Internet (GlobeCom) '97*, pages 1948 – 1952, November 1997.

[10] D. McDonald, C. Metz, and B. Phan. PF_KEY Key Management API, Version 2. Request for Comments (Informational) 2367, Internet Engineering Task Force, July 1998.

[11] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft, Internet Engineering Task Force, February 1999. Work in progress.