USENIX Association

Proceedings of BSDCon ’03

San Mateo, CA, USA
September 8–12, 2003

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Fast IPSec: A High-Performance IPsec Implementation

Samuel J. Leffler

Errno Consulting
sam@errno.com

ABSTRACT

Fast IPsec is an implementation of the IPsec protocols [Kent & Atkinson, 1998a] for FreeBSD that was designed for high performance. In particular the protocols use the OpenBSD Cryptographic Framework, as ported to FreeBSD [Leffler, 2003], so any cryptographic hardware is automatically used to accelerate their operation. Fast IPsec, running on a uniprocessor system with a single Broadcom BCM5822 cryptographic processor, has demonstrated throughput of more than 400 megabits/second when acting as an IPsec terminator. This is more than 50% higher than any other freely available IPsec implementation.

1. Background and Introduction

The IP Security protocols (**IPsec**) are a suite of protocols [Kent & Atkinson, 1998a] standardized by the IETF [Kent, 1998; Kent & Atkinson, 1998b] for secure communication of IP datagrams. IPsec is comprised of three protocols: AH, ESP, and IPCOMP. AH provides authentication, ESP encryption and optionally authentication, and IPCOMP adds compression. Any or all of these protocols can be combined though most uses of IPsec employ only the ESP protocol (with the optional authentication). Several freely available implementations of the IPsec protocols exist: the KAME Project distributes an IPsec suite that has been integrated into FreeBSD and NetBSD, OpenBSD has their own IPsec implementation, while the FreeS/WAN Project distributes an IPsec implementation for Linux. Of these implementations only the OpenBSD software includes support for using cryptographic hardware to accelerate the protocols. Cryptographic hardware vendors sometimes provide software to integrate their products with some of these IPsec implementations but these tend to be special-purpose and only a few are freely available [Communications, 1999].

Fast IPsec is an implementation of the IPsec protocols for FreeBSD that was designed for high performance. In particular the protocols use the OpenBSD Cryptographic Framework, as ported to FreeBSD, so any cryptographic hardware is automatically used to accelerate their operation. Fast IPsec, running on a uniprocessor system with a single Broadcom BCM5822 cryptographic processor, has demonstrated throughput of more than 400 megabits/second when acting as an IPsec terminator [Ambrisko, 2003]. This is more than 50% higher than any other freely available IPsec implementation.

Fast IPsec is derived from the KAME IPsec, but integrates many of the lessons OpenBSD learned when adding hardware acceleration to their IPsec implementation. The decision to start from the KAME implementation was made for two reasons:

1) The KAME distribution is the most widely used IPsec implementation for BSD systems. Providing a successor implementation that is familiar to existing users simplifies adoption.
2) The OpenBSD IPsec is tightly integrated with other facilities that conflict with or duplicate existing facilities in FreeBSD.

While Fast IPsec has been designed as a successor to the KAME IPsec it has also been written to coexist with the KAME implementation in the source tree. This permits users to evaluate Fast IPsec without giving up their existing IPsec implementation. Further, because Fast IPsec and KAME share APIs, users do not need to learn a new set of configuration tools.

The remainder of this paper is organized as follows. Section 2 describes the Fast IPsec implementation. Section 3 discusses performance issues and the techniques used to attain its high performance. Section 4 describes performance results for FreeBSD and compares them to other freely available IPsec implementations. Section 5 outlines the the status and availability of this work and talks about future work. Section 6 gives conclusions.

2. Fast IPsec Implementation

Fast IPsec is comprised of four protocols (AH, ESP, IPIP, and IPCOMP), the security database, **PF_KEY** socket support through which applications interact with the database, and glue code that resides in protocols such as IP, TCP, UDP, and ICMP. There is no cryptographic transformation or compression code in Fast IPsec; this comes entirely from the FreeBSD cryptographic framework.

2.1. Protocols

The protocol implementations in Fast IPsec are completely different from those found in KAME. They borrow heavily from what is found in OpenBSD in several ways:

1) The protocols are structured in a ‘‘continuation style’’ that permits the decoupling of cryptographic and protocol processing. That is, each of the input and output processing paths for AH, ESP, and IPCOMP are broken up into ‘‘before’’ and ‘‘after’’ portions that invoke the cryptographic framework and then continue processing on return through a callback mechanism.
2) Many code paths are unified to handle both IPv4 and IPv6 protocols. This eliminates code duplication.
3) IP-in-IP encapsulation is implemented as a separate protocol. KAME handles the encapsulation/decapsulation of packets for tunnels as special case code in the ESP and AH protocols.
4) Header data are retrieved from packets with ‘‘m_copydata’’ instead of forcing mbuf data to be contiguous. This eliminates many assumptions about the handling of data in layers above and below. Performance measurements indicate doing this adds no noticeable cost over the techniques used by KAME.

Otherwise, notable differences in the protocols are in areas like statistics. Fast IPsec is careful to record error statistics that uniquely identify each problem; this is critical for diagnosing systems without source code.

2.2. Packet Tags

Aside from the cryptographic framework that is described elsewhere [Leffler, 2003], Fast IPsec required only the addition of ‘‘packet tags’’ to FreeBSD. Packet tags are used to associate typed variable-length data with a packet. The standard mbuf routines propagate these tags when packet headers are moved or duplicated and they are automatically reclaimed when an mbuf chain is freed. Packet tags originated in OpenBSD [Keromytis, 2003] but they were changed in two important ways:

1) OpenBSD allocates all storage with **M_NOWAIT** (do not block to wait for memory). This is too inflexible for general use. In FreeBSD routines that allocate tag storage take a parameter that specifies whether the caller is willing to block when allocating storage.
2) OpenBSD defines packet tag type values as a single 16-bit value that must be centrally administered to ensure uniqueness. In FreeBSD this 16-bit value was expanded to add a 32-bit **cookie** that specifies an ABI or module ID. By convention cookies are defined as the date and time that a module is created, expressed as the number of seconds since the epoch (e.g. using the output of ‘‘date -u +%s’’). This scheme permits software modules to be developed without the need for a central administrator [Elischer, 2002]. Developers define a unique cookie and then manage tag types privately.

When these changes were made, OpenBSD-compatible shims were provided so that cross-platform compatibility could be maintained in code that needed to be portable. Packet tags were also used to replace the KAME ‘‘auxiliary mbuf’’ mechanism.

2.3. Security Database

The security database and policy management code is derived from the KAME implementation. While numerous changes were done to reduce memory usage and improve performance, its design has been left mostly intact.

The most notable change was to replace the use of **sockaddr_storage** for recording network addresses with a **sockaddr_union** data structure that is a union of the potential address formats. This reduces the storage for a network address from 128 bytes to 28. For systems with limited memory this is significant as each entry in the security database records at least two addresses. This change also reduces the size of the runtime stack and the amount of data referenced by structure copy and zeroing operations.

Otherwise the explicit algorithm specifications and algorithm-related data were replaced with references to transformation routines through which the protocols interact with the cryptographic subsystem. This technique has been used successfully in other systems [Keromytis et al, 1997; Spencer et al, 2002].

2.4. Locking

Fast IPsec was initially developed for the 4.6 release of FreeBSD. This version of the operating system uses traditional synchronization techniques designed for a non-preemptive uniprocessor environment. Only one thread of execution is expected to be active at a time. Code that needs to synchronize access to data structures does this by blocking interrupts so that asynchronous events are disabled.

FreeBSD 5.0 is the first release of FreeBSD to have a fully preemptive kernel. In this environment it is undesirable to guard execution paths to ensure synchronization of data structures [Hsu, 2003]. Instead locks are associated with data structures and concurrent threads of execution are managed by preempting a thread when it encounters a locked data structure. This can simplify code and make proper locking more intuitive.

Fast IPsec has very few locking requirements. The protocols are expressly designed to function as separate threads that depend only on private data. The only central data structure that needs synchronization is the security database. The database is referenced by code that is executed on behalf of system calls (applications sending data), by asynchronous activities that occur because of incoming network traffic, and by timer-driven threads that do things like expire security policies. Prior to FreeBSD 5.0 it was possible to block asynchronous access to the security database by raising the processor priority to **splnet**, but with a fully preemptive kernel this no longer works. Instead a set of locks were added to guard accesses to each of the major data structures and references to data structures are safeguarded with reference counters. These changes were straightforward and sufficiently fine-grained that there is no noticeable lock contention (as measured by the lock profiling facilities). The only real issue is the checking of security associations when transmitting packets. Due to the structure of the database inherited from KAME it is necessary to lock the **ipsecrequest** structure to ensure references to security associations are safely held. This turns out to be a ‘‘hot spot’’ that limits performance for bidirectional traffic flow. A redesign of the data structures to eliminate this locking is in progress.

3. Performance Analysis

There are several major areas to study to understand the performance of Fast IPsec: the cryptographic subsystem, the protocol implementations, the security database, and the network interface drivers. (There are other components such as mbuf and memory allocators but their individual performance tends to be less critical.) By far the majority of the overhead associated with IPsec is in the cryptographic processing. This is why it is so important to accelerate and offload the work from the main CPU. However the interactions between the various software components and the system architecture can have a noticeable effect on overall performance too. The next sections discuss these issues in more detail.

3.1. Crypto Subsystem

Fast IPsec uses the FreeBSD cryptographic framework to do all encryption and authentication work. This subsystem provides general-purpose device-independent support for a variety of transformations, including the symmetric-key cryptographic operations required by the ESP and AH protocols. The crypto support is comprised of a core set of code that manages requests and a set of device drivers for cryptographic devices. In addition there is a software-only device driver that implements symmetric-key operations on the host CPU for systems that do not have hardware devices.

The payload of each packet sent and received is passed to the crypto subsystem. Prior to dispatching the crypto request all the data needed to process the packet on return are collected and associated with the request. When the operation completes the crypto subsystem invokes a callback function stored in the request. This callback method completes the processing for the packet and dispatches the packet either ‘‘up’’ (for reception) or ‘‘down’’ (for transmission).

[Leffler, 2003] describes the work that was done to optimize the performance of the FreeBSD cryptographic subsystem. Increased performance of the crypto subsystem directly affects the performance of Fast IPsec. Of particular note is the work done to reduce latency in processing cryptographic requests. Compared to other systems, the FreeBSD cryptographic support has significantly less overhead and lower latency. This is reflected in significantly higher performance, especially for embedded systems where CPU cycles spent on overhead are more noticeable and for high-end systems where keeping the cryptographic hardware busy is critical to optimal performance.

3.2. Data Handling, Alignment, and Fragmentation

The performance of network protocols is typically constrained by the efficiency with which data are moved (or not moved) and manipulated. Because IPsec requires significant computation to process each packet, inefficiencies in this area can be less noticeable. Nonetheless, while tuning the performance of Fast IPsec data handling issues frequently appeared.

One issue was the need to properly align packet data to ensure it is always processed with the ‘‘fast path.’’ For example, some network interface drivers did not properly align received Ethernet frames so that the IP header is aligned to a 32-bit boundary. This can force data to be copied by protocols as the packet is passed up the stack. Further, when this data is passed to a crypto device driver, misaligned data can require additional copying. Several drivers with problems of this sort were fixed and the Fast IPsec protocols take care to ensure data are optimally aligned for ancillary operations. Fast IPsec and the FreeBSD cryptographic framework also keep statistics on any misaligned or otherwise suboptimal data manipulations.

Optimizing the input data path is simpler than optimizing the output path. For devices with a fixed-size link-level header, drivers can set up receive buffers so that data are contiguous and well aligned. As packets work their way up the protocol stack protocol headers can be efficiently removed. The only issue is ensuring proper alignment of data that requires cryptographic processing and this happens automatically if the IP header is aligned to a 32-bit boundary.

Optimizing the output path is a bit more involved. Headers must be prepended and packets must be rewritten with cryptographic transformations. Data that comes from a stream socket typically must be copied to create a writable version that can be transformed. Other data, such as packets being forwarded by an IPsec terminator, may be writable and not need to be copied. Fast IPsec creates a writable copy of an mbuf chain with the ‘‘m_clone’’ routine. This routine uses an *aggressive coalescing* technique that tries to compact the resulting mbuf chain and linearize the data, but balances this goal against the cost of copying already writable data. Compacting an mbuf chain is good in that it reduces the number of individual segments that must be processed; especially when arranging DMA to/from cryptographic hardware devices. Linearizing data is critical to the efficient use of cryptographic hardware that does not support scatter/gather DMA; but it can also improve performance of software algorithms. The scheme employed by ‘‘m_clone’’ uses the following rules:

1) ‘‘Inline mbufs’’ are coalesced only when there is a cluster immediately preceding that has space to hold the data.

2) Mbufs with writable external storage are left untouched.

3) Mbufs with read-only external storage must be copied. If there is space in the immediately preceding mbuf, then the data are copied. Otherwise, new storage is allocated and the data are copied. Data larger than a cluster is broken into multiple mbufs.

This scheme is designed for Ethernet traffic where the maximum frame size fits in a cluster. Further it depends on the ability to identify whether or not mbufs with external storage are writable. Finally, the ‘‘busting of jumbograms’’ is required because many drivers assume packets can be directly mapped for DMA.

In practice many packets are coalesced into a single mbuf with all the data linearized. Packets being forwarded are easily identified as writable in FreeBSD 5.0 but require some special handling in 4.x versions of the system.[1] Statistics kept on the operation of this scheme indicate about 45% of the packets that require copying are coalesced into a single mbuf; but these numbers are believed to be artificially low. (Most traffic is from performance benchmarks that run on the gateway machine. In this case traffic is TCP-based and the data are received locally instead of being forwarded. Both these factors affect the results.) The issue with mapping packets for DMA is discussed below.

3.3. Network Interface Drivers

The operation of the network interface driver can strongly influence performance. Fast IPsec was tested with a wide variety of hardware and drivers. Performance varies significantly depending on load and frame size. The currently preferred device is the Intel PRO/1000 which comes in 32-bit and 64-bit PCI configurations.

Two issues with the driver in the handling of large packets were identified and fixed. First, the driver did not support the ‘‘bus dma’’ API for mapping mbuf chains on to the system bus for DMA. This meant that packets larger than a physical page had to be broken up into multiple mbufs (as described above for the ‘‘m_clone’’ function). The driver was redone to use the bus dma functions so that outbound packets may be left intact (though they presently are still broken up into clusters).

[1] In FreeBSD 5.0 read-only mbuf chains are explicitly marked, but in 4.x one must check a reference count that may be maintained in a driver-private data area.

The second issue was that received jumboframes were written to multiple segments and an mbuf chain was then constructed. An alternative approach is to allocate contiguous receive buffers. This however can require pre-allocation of a significant amount of memory because the device constrains receive buffer sizes to be a power of two (so a 9000-byte mtu requires 16 kilobyte receive buffers). The driver was changed to optionally allocate contiguous memory for receiving large frames.

This revised driver performs noticeably better than the standard one. With a 9000-byte mtu set on both interfaces of systems connected by a cross-over cable, **netperf** performance results with the standard driver drop by more than 210 Mb/s while performance for the driver with contiguous receive buffers increases by a modest amount (the increase is small because performance is already near peak and one of the machines is constrained by PCI bus bandwidth.)

3.4. System I/O Performance

Performance is influenced by many system-level issues. Cryptographic requests directed to hardware devices require two trips across the I/O bus for each operation. This means that I/O performance is very important in understanding a system's capabilities. Bus width, latency, and device configuration can significantly affect performance results. Kernel profiling indicates that most systems are limited by their ability to field and process interrupts. In an IPsec terminator configuration each packet requires four DMA operations (two for NIC DMA and two for hardware crypto DMA) and as many as four interrupts to be processed. This implies that interrupt overhead must be minimized for the highest possible performance. In particular IRQ multiplexing must be avoided as well as system overhead like harvesting ‘‘IRQ entropy’’ for the pseudo random number generator (PRNG). Polling techniques such those available in FreeBSD [Rizzo, 2001] can be useful in reducing this overhead. Some vendors of hardware crypto products claim to intelligently coalesce interrupts in their hardware but it is unclear if these are anything more than marketing hyperbole.

With regard to the PRNG, most cryptographic hardware includes a hardware random number generator (RNG) that can supply sufficient entropy to seed the system PRNG. While this can eliminate expensive techniques used to collect entropy data, care must be taken to evaluate the quality of the entropy supplied by the hardware. **Rndtest** is a kernel module that was developed to test the quality of random data sources [Wright & Leffler, 2002]. It monitors data on the way to the system PRNG to see if it complies with the FIPS 140-2 standard [Federal Information Processing Standards, 2002]. If data fail any of the FIPS 140-2 tests, they are discarded and no data from the source are passed to the system PRNG until compliant data are seen. Testing can be done continuously or periodically and there are various controls (e.g. whether to report problems to the system console) and statistics maintained. This facility has shown, for example, that the entropy data produced by a Broadcom BCM5822 device is sometimes unreliable.

4. Performance Results

The IPsec protocols are very flexible. There are three protocols that can be combined to generate a variety of packet formats. Encapsulation may also vary: there is a **transport** mode in which the IPsec protocol headers are directly encapsulated in IP and a **tunnel** mode in which an additional IP encapsulation is done when an intermediate machine acts as an IPsec **terminator** or **gateway**. Tunnel mode is typical of IPsec-based Virtual Private Network (VPN) applications.

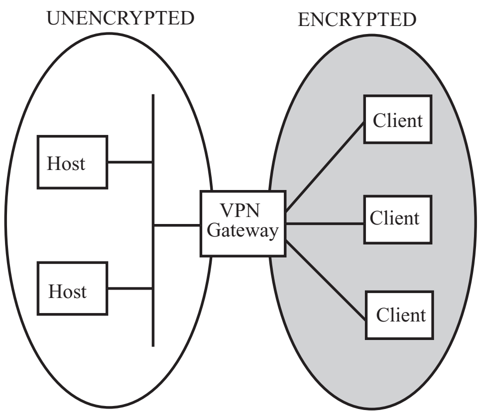

Figure 1: *Typical VPN gateway configuration*

For this paper we analyzed the performance of IPsec operating in tunnel mode using only the ESP protocol, but doing both encryption and authentication of the payload. When IPsec is deployed in this manner the network is typically configured as a VPN gateway, as shown in Figure 1. Specifically, IPsec data are received from peers, decrypted and authenticated, and forwarded in the clear to trusted clients. This means each packet requires one pass through cryptographic processing before it is forwarded. This is in contrast to a point-to-point tunnel configuration where data are cryptographically processed twice. Throughput measures for an IPsec gateway will be significantly higher than for a point-to-point tunnel. However collecting performance data for a gateway configuration is more complicated because it requires multiple clients and/or peers to saturate a Fast IPsec gateway. With multiple clients and/or peers it is also necessary to include a switch in the configuration and this device can become a bottleneck when evaluating performance. Therefore, to ensure others can easily reproduce the results presented here, we have chosen a network configuration in which two machines are physically connected with a cross-over cable. Testing done on a multi-client terminator configuration confirms the result for this simpler configuration apply directly.

The test configuration has two systems, A and T, connected by a cross-over cable. Machine A has an Asus P4B533-V Intel845G motherboard with a 1.8 GHz P4 processor. A dedicated Intel Pro/1000 (82540) NIC located in a 32-bit PCI slot was used for testing. Machine T has a Tyan S2707G2N motherboard with a 1.8 GHz P4 processor. The Tyan system has dual Intel gigabit Ethernet devices (82545 and 82551) on-board. The 82545 has a 64-bit PCI interface and the 82551 has a 32-bit PCI interface. Testing was done using the 82545. The mtu on each interface was the default, 1500 bytes.

To establish baseline performance, tests were first run on an open network. Table 1 shows data collected using **netperf** version 2.2pl3. The data were collected with a 99% confidence interval (+/- 2.5%), identical buffering parameters, and a 32 kilobyte message size.

A->T			T->A			Operating
Mb/s	%Sys		Mb/s	%Sys		System
627	75	55	799	84	78	FBSD 4.8†
519	75	24	653	48	32	OBSD 3.3‡

Table 1: *Raw netperf results for fast testbed*

Next a manually-keyed IPsec tunnel was set up between the two machines. All traffic was encrypted with 3DES and authenticated with SHA1. Table 2 shows results for several configurations. The transfer rates reported by netperf are provided as well as the percentage of system time for the sending and receiving machines, as reported by the **vmstat** program. Performance is different depending on which machine is initiating the transfer; ‘‘A->T’’ results are for when netperf was run on the Asus-based machine, while ‘‘T->A’’ is when netperf was run on the Tyan-based machine. In one configuration a Broadcom BCM5822 was placed in each machine. Broadcom data sheets claim the 5822 can do 3DES+SHA1 calculations at 500 Mb/s. Testing described in [Leffler, 2003] measured a peak performance for a 5822 of 470 Mb/s when located in a 64-bit PCI slot and about 410 Mb/s when located in a 32-bit PCI slot. In the other configurations crypto calculations were done by the host CPU; this is denoted by ‘‘sw/’’.

† FBSD 4.8 is a July 4, 2003 snapshot of FreeBSD.

‡ OBSD 3.3 is a March 17, 2003 snapshot of OpenBSD.

A->T			T->A			Operating	Crypto
Mb/s	%Sys		Mb/s	%Sys		System	Support
160	80	73	170	70	94	FBSD 4.8	BCM5822
n/a	-	-	n/a	-	-	OBSD 3.3	BCM5822
43	100	93	44	89	97	FBSD 4.8	sw/FastIPsec
42	95	95	43	100	100	FBSD 4.8	sw/KAME
27	87	94	27	88	96	OBSD 3.3	sw/OpenBSD

Table 2: *IPsec tunnel results for fast testbed*

The results show Fast IPsec is about 60% faster than OpenBSD when using software crypto and slightly faster than the KAME implementation. The higher performance relative to OpenBSD shows the value of FreeBSD’s optimized cryptographic subsystem. The comparison to KAME demonstrates that moving the crypto support out of IPsec into a general-purpose framework can be done without losing any performance.

Performance comparisons for Broadcom-accelerated configuration were not possible. KAME does not support hardware crypto acceleration and while OpenBSD claims support for the 5822 it did not work; traffic stalled and the tests never completed. However, based on the performance for raw cryptographic operations reported in [Leffler, 2003] and the relative network performance over the unencrypted link, one can assume Fast IPsec will be significantly faster than OpenBSD.

Performance in this simple network configuration is limited by the slowest component in the loop; in this case the CPU is the limiting factor. Tests with faster processors and the 5822 show performance of FreeBSD scales linearly with the CPU speed up to 218 Mb/s for 2.53 GHz P4 processors. At that point I/O bandwidth to the 5822 in Machine A becomes the limiting factor. 64-bit PCI support on Machine A would make a significant difference in the results.

Because no direct comparison to OpenBSD was possible with the Broadcom hardware a second testbed was set up with different hardware. In this configuration two slower systems were used, each with a GTGI XL-Crypt card. The XL-Crypt uses a Hifn 7811 crypto part and can do 3DES+SHA1 calculations at approximately 145 Mb/s. The two systems were purposely slower to match the performance of the cryptographic hardware (using slow crypto hardware in a fast machine yields uninteresting results because the cryptographic hardware becomes the bottleneck.) Machine D was a Dell XPS 300 with 266 MHz PII processor. Machine E was an E-Machines etower 366i with a 366 MHz PIII processor. As before the systems were connected by a cross-over cable but this time only 100 Mb/s Ethernet was used. Table 3 shows the results for running netperf over a manually-keyed IPsec tunnel on this hardware.

D->E			E->D			Operating	Crypto
Mb/s	%Sys		Mb/s	%Sys		System	Support
45.9	81	20	51.4	92	22	FBSD 4.8	Hifn 7811
37.3	96	40	33.0	81	30	OBSD 3.3	Hifn 7811
9.8	100	93	9.4	99	92	FBSD 4.8	sw/FastIPsec
8.9	100	94	9.2	100	94	FBSD 4.8	sw/KAME
5.3	100	84	7.0	89	98	OBSD 3.3	sw/OpenBSD

Table 3: *IPsec performance results for slow testbed*

FreeBSD was 23/55% faster than OpenBSD when using hardware acceleration. With crypto operations done in software the difference was 34/84%. As the system performance decreases the FreeBSD optimizations are less noticeable because other factors become significant. In this case the I/O performance of the Dell system is so poor that it is the critical factor in determining overall performance.

5. Status and Future Work

An initial version of Fast IPsec was completed September 2001. Integration of this work into FreeBSD was completed November 2002 and committed to the stable branch in January 2003. The work described here, except for the fine-grained locking, is freely available as part of the FreeBSD 5.0 and 4.8 releases. Fast IPsec is currently being integrated into the NetBSD operating system [Stone, 2002]. Several vendors have incorporated Fast IPsec in their products.

Fast IPsec lacks support for IPv6; this has yet to be done because of lack of interest. The **IPCOMP** support is fully implemented but is not working due to an issue with the **gzip** compression support in the FreeBSD kernel.

The security database implementation needs to be redone. Much of the information stored in this database is better co-located with the routing tables so that the routing table lookup algorithms can be used. In addition much of the data stored in the database requires two and three levels of indirection to access. Relocating this data will simplify the protocols and eliminate some of the locking currently required to ensure indirect pointers do not change while the protocol code follows indirect references. A redesigned database is likely to provide a noticeable performance improvement for systems of any significant scale.

There is no support for the cryptographic algorithms required by the legacy versions of the AH and ESP protocols. Correcting this requires changes to the cryptographic framework API and may not be worthwhile.

The **PF_KEY** support used to communicate with user-level applications is intertwined with the IPsec security database implementation. This needs to be changed by separating the pfkeyv2 protocol and database implementations. This would fix, for example, problems where changing aspects of **PF_KEY** break user-mode applications such as **racoon** and **setkey**.

Otherwise, the most significant deficiency in the current design is the inability to use protocol-specific hardware operations. Most vendors of crypto hardware optimize their products for use as ‘‘all-in-one’’ devices that take an IPsec packet and parse the protocol and perform the cryptographic transforms in a single request. This is incompatible with the current general-purpose API provided by the cryptographic framework. Supporting this kind of operation requires exposing IPsec state that is currently private. However adding this is the only way that some hardware devices can be used at all as they do not otherwise provide access to the cryptographic transformation hardware.

6. Conclusions

Fast IPsec is an implementation of the IPsec protocols that has been designed as a plug-compatible successor to the KAME IPsec protocols with high performance. The software has been demonstrated to have performance more than 50% better than any other publicly available IPsec implementation.

7. Acknowledgements

This work incorporates ideas, and some code, from the OpenBSD IPsec implementation; without it this work would probably not have been done. Angelos Keromytis was helpful in understanding how bits and pieces of OpenBSD worked.

Vernier Networks supported this work; their contribution is greatly appreciated. Doug Ambrisko of Vernier Networks contributed many bug fixes and was a great help in testing.

References

Ambrisko, 2003.
D. Ambrisko, *Private email: 400mbs IPsec numbers* (February 2003).

Communications, 1999.
RedCreek Communications, Inc., *RedCreek IPSec VPN Card,* Source for the driver is included in Linux starting with 2.4 (April 1999).

Elischer, 2002.
J. Elischer, “Re: CFR: m_tag patch,” *freebsd-arch@freebsd.org* (November 2002).

Federal Information Processing Standards, 2002.
Federal Information Processing Standards, “Security Requirements for Cryptographic Modules” (FIPS PUB 140-2), National Institute of Standards and Technology (December 3, 2002).

Hsu, 2003.
J. Hsu, “Reasoning about SMP in FreeBSD,” *BSD-Con 2003* (September 2003).

Kent, 1998.
S. Kent, “IP Authentication Header,” *RFC 2402* (November 1998).

Kent & Atkinson, 1998a.
S. Kent & R. Atkinson, “Security Architecture for the Internet Protocol,” *RFC 2401* (August 1998).

Kent & Atkinson, 1998b.
S. Kent & R. Atkinson, “IP Encapsulating Security Payload (ESP),” *RFC 2402* (November 1998).

Keromytis, 2003.
A. Keromytis, “Tagging Data In The Network Stack: mbuf_tags,” *BSDCon 2003* (September 2003).

Keromytis et al, 1997.
A. Keromytis, J. Ioannidis, & J. Smith, “Implementing IPsec,” *Proceedings of Global Internet (Globe-Com) ’97,* pp. 1948-1952 (November 1997).

Leffler, 2003.
S. Leffler, “Cryptographic Devices Support for FreeBSD,” *BSDCon 2003* (September 2003).

Rizzo, 2001.
L. Rizzo, “Polling versus Interrupts in network device drivers,” *BSDConEurope 2001* (November 2001).

Spencer et al, 2002.
H. Spencer, R. Briggs, D. Redelmeier, M. Richardson, S. Harris, & C. Shmeing, *Linux FreeS/WAN* (April 2002).

Stone, 2002.
J. Stone, *Private email: Re: Fast IPsec: patches for FreeBSD 4.x?* (November 2002).

Wright & Leffler, 2002.
J. Wright & S. Leffler, *rndtest* (2002). http://www.freebsd.org/cgi/cvsweb.cgi/src/sys/dev/rndtest.

Biography

Sam Leffler has been actively working with UNIX since 1975 when he first encountered it at Case Western Reserve University. While working for the Computer Systems Research Group (CSRG) at the University of California at Berkeley he helped with the 4.1BSD release and was responsible for the release of 4.2BSD. He has contributed to almost every aspect of BSD systems; most recently working (again) on the networking subsystem. You can contact him via email at `<sam@errno.com>`.