Performance Impact of Data Compression on Virtual Private Network Transactions

John P. McGregor and Ruby B. Lee
Department of Electrical Engineering
Princeton University
{mcgregor, rblee}@ee.princeton.edu

Abstract

Virtual private networks (VPNs) allow two or more parties to communicate securely over a public network. Using cryptographic algorithms and protocols, VPNs provide security services such as confidentiality, host authentication and data integrity. The computation required to provide adequate security, however, can significantly degrade performance. In this paper, we characterize the extent to which data compression can alleviate this performance problem in a VPN implemented with the IP Security Protocol (IPsec).

We use a system model for IPsec transactions to derive an inequality that specifies the conditions required for data compression to improve performance. We generate performance results for many combinations of network types, data types, packet sizes, and encryption, authentication and compression algorithms. We find that compression usually improves performance when using 10 Mbps or slower networks, but compression only improves performance in systems with 100 Mbps or 1 Gbps networks when using computationally intensive encryption algorithms.

1. Introduction

As Internet usage grows exponentially, network security issues become increasingly important. Using a virtual private network (VPN), multiple hosts can communicate securely over a public network. The details of VPN protocols vary, but most consist of two major steps: the handshake and bulk data encryption and authentication. The VPN is established during the handshake step. This step involves protocol and algorithm negotiation, authentication of hosts, and secret key exchanges between the hosts. The hosts can then communicate privately by encrypting and authenticating all of the data that travels over the public network.

The IP Security Protocol (IPsec) can be used to implement encryption and message authentication in virtual private networks in a vendor-independent, application-invisible manner [11]. The encryption and message authentication services provided by IPsec, however, require significant computation time. Consequently, IPsec can degrade performance when compared to unsecured transmissions.

In other work, researchers have improved the performance of secure network transactions using a variety of techniques. By adding new instructions to conventional instruction set architectures, the number of instructions in software implementations of cryptographic algorithms can be significantly reduced [22]. In addition, the computation associated with many cryptographic protocols is highly parallelizable. When performing encryption or message authentication, a multiprocessor system can achieve nearly linear speedup by assigning individual packets or connections to single processing elements [15], [16].

In this paper, we investigate the performance benefit of compressing IP packet payloads when using IPsec to implement a virtual private network. We restrict our investigation to the performance impact of compression on bulk data encryption and message authentication. Data compression does not improve the performance of the algorithms and protocols involved in the handshake step.

The IP Payload Compression Protocol (IPComp) employs data compression algorithms to reduce the size of packet payloads at the IP layer [21]. This size reduction decreases the time required to transmit IP packets by reducing the amount of information that is physically transferred over the network. In addition, IPComp can decrease the execution times of the encryption and authentication algorithms by reducing the amount of information to be encrypted and authenticated. IPComp does not always improve performance, however. The data compression algorithms used by IPComp consume a significant number of clock cycles. As a result, compressing the IP packet payload may increase the total time needed to complete a particular transaction, and therefore IPComp can degrade performance.

Measuring the actual performance of VPN transactions is complicated and time-consuming and does not reveal

0-7695-0912-6/00 $10.00 © 2000 IEEE

how individual components of the transactions affect performance. To show the dependence of performance on different algorithm and system parameters, we model a virtual private network between two parties. This model allows us to easily change parameters such as the encryption algorithm and network speed and observe the effects. Using speedup equations that describe the performance impact of compression, we derive an inequality that specifies the conditions required for compression to improve performance in the model. This inequality depends on several parameters such as network bandwidth, packet payload sizes, algorithm throughputs, and compression ratio. By measuring the values of these parameters, we can predict whether compression will improve performance in a given system. We obtain empirical performance results by executing the security and compression algorithms on a single machine that contains one 367 MHz HP PA-8500 processor.

The paper is organized as follows. In Section 2, we discuss our system model. Section 3 discusses the details of IPsec, and Section 4 explores the performance cost of using IPsec. In Section 5, we describe IPComp and investigate the compressibility and throughput achieved by different compression algorithms. In Section 6, we present speedup equations for calculating performance and derive inequalities that predict when compression will improve performance. In Section 7, we present and analyze experimental performance results of combining IPComp and IPsec. We summarize and discuss directions for future work in Section 8.

2. System Model

When combining IPComp with IPsec, the performance depends on the characteristics of the system. For example, the time needed for an individual to retrieve banking account information over the Internet consists of three major components. These components include the computational time required by the bank's servers to prepare the IP packets, the time needed to physically transmit the IP packets, and the computational time required by the individual's computer to interpret the packets. The *prepare packet* operation consists of the procedures such as encryption and hash computation required to prepare the packet for secure transmission. The *interpret packet* operation consists of the procedures such as decryption and hash verification required to extract the contents of the packet upon receipt. As we will explain in Section 3, hash computation and verification are used to perform message authentication. We describe encryption and decryption in Section 3, and we describe compression and decompression in Section 5.

The system model includes a sender, a network, and a receiver, as shown in Figure 1. Such a system can consist of multiple processors on both the receiver and sender ends, and the packets may pass through several gateways and network types before reaching their final destination. One approach to modeling such a system would be to treat the computation and transmission segments as a pipeline. In a pipelined model, the performance would depend on the most time-consuming segment. In this paper, however, we measure the performance in terms of total latency. More specifically, we are concerned with the total amount of time required by all computation and network resources to prepare, transmit, and interpret packets. We investigate the extent to which IPComp increases or decreases the total amount of time required to complete these tasks. The performance measure, therefore, is based upon the time needed to serially compress, encrypt, hash compute, transmit, hash verify, decrypt, and decompress.

3. IP Security Protocol

The IP Security Protocol (IPsec) provides a variety of security services at the IP layer [11]. IPsec confidentiality and message authentication services are implemented using the Encapsulating Security Payload and the Authentication Header. The Encapsulating Security Payload (ESP) provides for confidentiality of the IP packet payload using symmetric key encryption algorithms [10]. Both the Authentication Header (AH) and ESP insure the authenticity as well as the integrity of the IP packet payload using symmetric key encryption algorithms or secret-keyed one-way hash functions [9], [10]. In addition, AH provides protection against IP address spoofing. IPsec allows ESP and AH to be applied to IP packets either alone or in combination with each other [11].

Requests for Comments (RFCs) describing the IPsec protocols include instructions for integration in both IPv4 and IPv6. Only minor differences exist between the IPsec integration procedures for the two IP versions, and we discuss the algorithms, protocols, and procedures as specified for IPv6 in this paper.

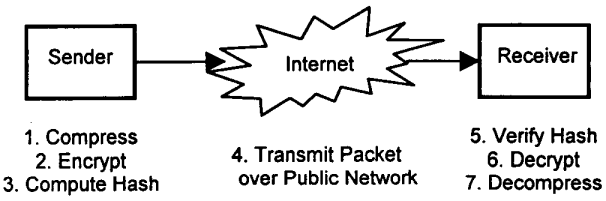

Figure 1. System model

3.1. IP Encapsulating Security Payload

ESP is used to provide confidentiality, data origin authentication, connectionless integrity, and anti-replay service (a form of partial sequence integrity) [10]. ESP employs symmetric-key encryption algorithms such as RC5 and 3DES to provide confidentiality. ESP uses keyed hash algorithms such as SHA-1 and MD5 to provide message authentication and anti-replay service.

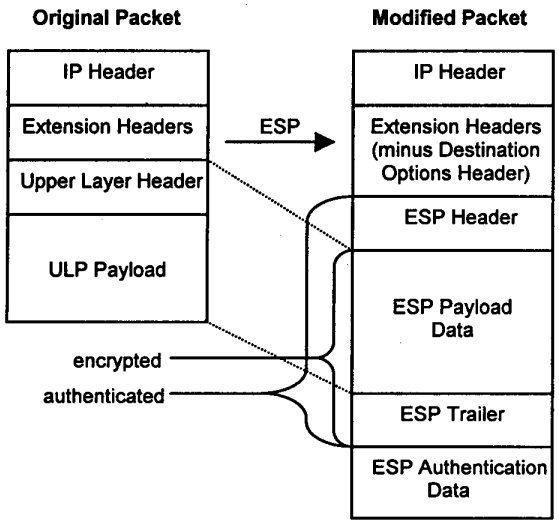

Figure 2. ESP in IPv6

ESP transforms IPv6 packets as illustrated in Figure 2 [10]. The original IPv6 packet depicted in Figure 2 consists of 4 components [4]. The IP header is the essential first component of every IPv6 packet. Following the IP header, the packet contains a variable number of Extension headers. These headers include (but are not limited to) the Routing header, the Fragmentation header, and the ESP header. The Upper Layer header immediately follows the Extension headers and is used by the protocol immediately above IPv6 (e.g., TCP, UDP, and ICMP). Last, the ULP (Upper Layer Protocol) Payload consists of the data to be sent and received by the ULP. The ESP header is inserted directly preceding the Destination Options header, if it exists. If a Destination Options header does not exist, the ESP header is inserted immediately preceding the ULP header and payload.

The first field in the ESP header consists of the Security Parameters Index (SPI), which, in conjunction with the IP destination and the security protocol, specifies the Security Association for the packet. The Security Association (SA) indicates which algorithms and modes of operation are being used to perform the encryption and authentication (see Section 3.3). The second field in the ESP header consists of the Sequence Number, which is a monotonically increasing counter value that prevents replay attacks (if authentication is employed).

The encrypted data block directly follows the ESP header. This block consists of at most three components: the initialization vector (IV), the ESP payload data, and the ESP trailer. The ESP Authentication Data field follows the ESP Trailer. This field contains a variable-length Integrity Check Value (ICV) that is computed over the ESP header, the ESP payload data and the ESP trailer. The ICV is calculated using either a secret-keyed hash algorithm or a symmetric key encryption algorithm.

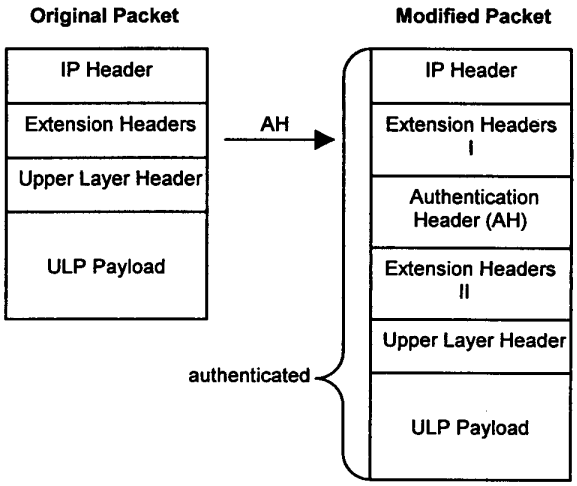

Figure 3. AH in IPv6

3.2. IP Authentication Header

The IP Authentication Header (AH) provides authentication, protection against replay attacks, and connectionless integrity for IP packets [9]. AH transforms IPv6 packets as illustrated in Figure 3 [9]. The original packet is constructed exactly as described in Section 3.1. As shown in Figure 3, the AH should be inserted immediately following Extension Headers I and immediately preceding Extension Headers II. Extension Headers I includes the Hop-by-Hop Options header, the Routing header, and the Fragment header. Extension Headers II includes the ESP header and the Destination Options header. In IPv6, the Authentication Header is also classified as an Extension header. IP packets are not required to contain any Extension headers [4].

The Sequence Number in the AH consists of a monotonically increasing counter value that is used to prevent replay attacks. Furthermore, the Authentication Data field contains the Integrity Check Value (ICV) for the packet. The ICV is the output of the chosen

authentication algorithm computed over the entire IP packet.

3.3. Encryption Algorithms

Several symmetric key encryption algorithms are defined for use in ESP [18]. In this study, we use two ciphers in CBC mode: RC5-CBC and 3DES-EDE-CBC. CBC, which stands for Cipher Block Chaining, is a mode of operation that provides more security than the ECB (Electronic Code Book) mode of operation, which is easier to implement [20].

DES, which is an acronym for Data Encryption Standard, is a symmetric-key block cipher that uses a 56-bit key to encrypt 64-bit blocks [20]. Triple DES (referred to as 3DES) provides more security than DES by encrypting a single block three times (with DES) using two or three different 56-bit keys [20]. We implement 3DES in EDE (encrypt-decrypt-encrypt) mode, as prescribed in [18]. To encrypt a block of data, we encrypt with the first key, then decrypt with the second key, and then encrypt with the third key. To decrypt a block of data, we decrypt with the third key, encrypt with the second key, and then decrypt with the first key. We apply CBC mode using the outer chaining technique as described in [20]. 3DES is cryptographically twice as strong as DES, but 3DES takes nearly three times as much computational time as DES to encrypt or decrypt blocks.

RC5 is a symmetric key block cipher that is patented by RSA Data Security, Inc. [20]. RC5 can operate over different block sizes and key lengths; in this study we use a block size of 64 bits and a key length of 128 bits. We implement RC5 in CBC mode as described in [2]. RC5 is considered to be one of the fastest secure symmetric-key block ciphers, whereas 3DES is one of the slowest.

3.4. Authentication Algorithms

Two authentication algorithms that are defined for use with AH and ESP are HMAC-MD5 and HMAC-SHA-1 [9], [10]. In this study, we shall evaluate the performance of both of these algorithms. HMAC is a mechanism that provides message authentication in which an iterative cryptographic hash function is used with a secret key [12].

MD5 is a cryptographic hash function that accepts plaintext blocks of size 64 bytes and outputs a 16-byte authentication value [13]. SHA-1, a government standard, is a cryptographic hash function that accepts 64-byte plaintext blocks and outputs a 20-byte authentication value [14]. SHA-1 is considered to be a cryptographically superior hash function, but MD5 is faster [12].

3.5. Implementing ESP and AH

When combining ESP and AH, the AH header directly precedes the ESP header in the modified IP packet. Since AH authenticates more data in an IP packet than the authentication services in ESP, we always use AH to perform authentication in this study; we only use ESP to perform encryption. In addition, hash computation and hash verification are equivalent operations: the sender and the receiver execute the same hash algorithm over identical keys and identical packets.

It is possible for ESP and AH to specify NULL for the encryption or authentication algorithms. When using the NULL algorithm, we treat the packet as if the protocol were not being employed at all. This policy reduces computation time by eliminating useless protocol processing.

4. Performance Impact of IPsec

In this section, we explore the performance costs of employing IPsec procedures. We obtain data concerning the throughput of IPsec procedures by implementing and executing the cryptographic algorithms on a HP Visualize C360 workstation. This workstation consists of a 64-bit 367 MHz HP PA-8500 processor with 1.5 MB of on-chip L1 cache and 128 MB of RAM. We implemented all four of the authentication and encryption algorithms in C. We use the HPUX C compiler (`cc`) to build the modules, and we employ full compiler and linker optimizations (i.e., `+O4`). We obtain the timing results for all of the authentication and encryption procedures using the UNIX `clock()` function. In order to avoid imprecision resulting from the relatively high granularity of the `clock()` output, we execute the IPsec procedures thousands of times. We obtain the final timing result by dividing the total time needed to complete the thousands of iterations by the total number of iterations executed.

4.1. Network Types and Payload Sizes

We consider the following network types: 56 kbps (phone line modem), 1.54 Mbps (T1, wireless), 10 Mbps (Ethernet), 100 Mbps (Ethernet), and 1 Gbps (Ethernet). Under ideal conditions, Ethernet networks can achieve a channel efficiency of over 90% [23]. As the number of network users increases, however, the channel efficiency drops to roughly 82% [23]. In this study, we assume all the network connections sustain 80% of their maximum throughput.

We evaluate the performance of the IPsec procedures using 3 ULP payload sizes: 1 kilobyte, 4 kilobytes, and 63 kilobytes. TCP, which stands for Transmission Control

Protocol, is a connection-based protocol that employs a 20-byte payload header [19]. We assume that all ULP payloads are TCP payloads. The maximum size of an IPv6 packet minus the IP header is 64 KB, so the 63 KB payload size represents the largest possible payload size but leaves room for ESP, AH, IPComp, and TCP headers. We set the network maximum transmission unit (MTU) to be 1280 bytes, the minimum MTU allowed in IPv6, and we fragment packets accordingly [4]. We assume that all the original IP packets solely consist of a 40-byte IPv6 header, an 8-byte fragmentation header (if necessary), a 20-byte TCP header, and a variable length TCP payload. The use of additional headers or a transport protocol other than TCP should not significantly affect the performance results.

Table 1. Performance of encryption and authentication algorithms (Mbps)

Algorithm Name	Payload Sizes			
	1K	4K	63K	Average
MD5	229	299	332	287
SHA-1	214	274	302	263
RC5	96	97	96	96
3DES	28	28	28	28

4.2. Performance Results and Analysis

The execution time of the encryption and authentication algorithms is a function of the input size and is independent of the statistical characteristics of the input data. Hence, for a given processing platform, the throughputs of the encryption and authentication algorithms are constant. Since hash computation and hash verification are equivalent operations, they will consume the same amount of time. Furthermore, DES is a Feistel cipher, so the encryption speed of 3DES is equivalent to the decryption speed of 3DES. Although RC5 is not a Feistel cipher, the speed of the encryption procedure is roughly the same as the speed of the decryption procedure.

We summarize the throughput of the encryption and authentication algorithms for the three packet sizes in Table 1. The algorithms are executed in the modes of operation described in Section 3. From Table 1, we see that the throughput of the authentication algorithms is much higher than that of the encryption algorithms. Furthermore, MD5 runs slightly faster than SHA-1 and RC5 runs three times as fast as 3DES.

We now calculate the performance degradation caused by different combinations of these algorithms. As described in Section 2, the system model consists of the computation required to serially compress, encrypt, compute the hash, verify the hash, decrypt, and decompress the IP packet. We quantitatively determine the level of performance degradation by calculating the speedup S as follows.

$$S = \frac{T_{NET}}{T_E + T_{HC} + T_{SECNET} + T_{HV} + T_D}$$

In this equation, T_{NET}, T_{SECNET}, T_E, T_{HC}, T_{HV}, and T_D represent the original packet transmission time, the encrypted/authenticated packet transmission time, the encryption time, the hash computation time, the hash verification time, and the decryption time, respectively. T_{NET} may not equal T_{SECNET} because of the headers that are added to the encrypted/authenticated packet. Values of S close or equal to 1.00 indicate minor performance degradation, whereas values closer to 0.0 indicate enormous performance degradation. We calculate the speedup results using a TCP payload size of 4 KB in our network model (see Section 4). The speedup results for the five network bandwidths and eight combinations of security algorithms are listed in Table 2.

From Table 2, we see that encryption and authentication never degrade performance more than 10% when using 56 kbps or 1.54 Mbps network connections. As bandwidth increases, however, the performance impact of authentication and encryption becomes more pronounced. Both encryption and authentication decimate performance when using a 1 Gbps link. The results also show that the encryption algorithms have a greater effect on the performance than the authentication algorithms, for the encryption routines require much more computation.

Table 2. Speedups (Slowdowns) resulting from authentication and encryption

Network Type	Algorithm Combination							
	MD5	SHA-1	RC5	RC5 MD5	RC5 SHA-1	3DES	3DES MD5	3DES SHA-1
56 kbps	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98
1.54 Mbps	0.99	0.98	0.97	0.96	0.96	0.92	0.90	0.90
10 Mbps	0.94	0.94	0.86	0.82	0.82	0.64	0.62	0.62
100 Mbps	0.65	0.63	0.39	0.32	0.32	0.15	0.14	0.14
1 Gbps	0.16	0.15	0.06	0.04	0.04	0.02	0.02	0.02

This performance differential between encryption and authentication becomes more pronounced as bandwidth increases. We now investigate the degree to which data compression can alleviate this performance problem.

5. IP Payload Compression

The IP Payload Compression protocol (IPComp) employs data compression algorithms to reduce the size of IP packets [21]. This size reduction decreases the time required to transmit IP packets by decreasing the amount of information that is physically transferred over the network. Furthermore, IPComp decreases the execution time of encryption and authentication algorithms by reducing the amount of information to be encrypted and authenticated. In this section, we describe IPComp and analyze the compressibility and throughputs achieved by the LZS and DEFLATE compression algorithms.

5.1. Protocol Description

IPComp reduces the size of IP packets in IPv4 and in IPv6 using data compression algorithms [21]. The protocol specification includes several restrictions and guidelines concerning these compression algorithms. To preserve the consistency of the packet payload, the compression algorithm must be lossless. Furthermore, the compression of a packet payload must be completed before any IP security processing is performed: a cryptographically secure encryption algorithm outputs ciphertext that cannot be compressed. It follows that the decompression and reassembly of IP packets must also occur after decryption. In addition, each IP packet must be compressed and decompressed independently of other packets, since IP packets may arrive out of order or may never arrive at all. Lastly, the total size in bytes of the compressed payload and the IPComp header must be smaller than the size of the original payload. Otherwise, the original payload is sent without any IPComp header. This non-expansion policy saves clock cycles at the receiver end and guarantees that network traffic will not increase when using IPComp.

IPComp transforms an IPv6 packet as shown in Figure 4 [21]. The 4-byte IPComp header is inserted immediately preceding the Destination Options header in the IP packet. If the Destination Options header does not exist, the IPComp header is inserted immediately preceding the upper-layer header (e.g., the TCP header) in the IP packet. The compression algorithm used by IPComp compresses all the data that follows the IPComp header in the IP packet. The Destination Options header, the upper layer header, and the ULP payload are all compressed.

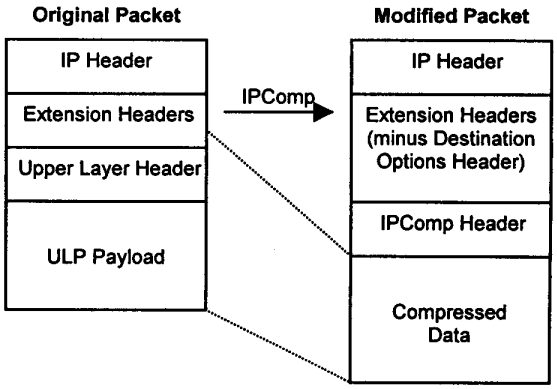

Figure 4. IPComp in IPv6

5.2. Data Compression Algorithms

We evaluate the performance of IPComp using two lossless compression algorithms: DEFLATE and LZS. DEFLATE compresses data using a combination of the LZ77 algorithm and Huffman coding [17]. In this study, we use zlib version 1.1.3, a freely distributed implementation of DEFLATE, with default compression settings [5], [6].

LZS is a lossless compression algorithm based on LZ77 that employs a sliding window of maximum size 2 kilobytes [1], [7]. LZS is an ANSI standard and is patented by Hi/fn, Inc. Our C implementation of LZS stores the compression dictionary as a quickly accessible and infrequently updated order-3 hash table. We also optimized our implementation by employing 64-bit features and by carefully reducing the compression capabilities of the algorithm. By not using the full compression power of LZS, we can obtain huge gains in throughput but pay only a small penalty in compression ratio. Our implementation requires approximately 50 KB of RAM and outperforms DEFLATE in throughput by an order of magnitude.

Unlike encryption and authentication algorithms, the performance of the compression algorithms depends on the statistical properties of the input data as well as its size. In other words, the throughput of the compression algorithms depends on the compressibility of the input data. Hence, we analyze the performance of the compression algorithms using an eclectic group of data benchmarks.

5.3. Data Benchmarks

In a real-world networking environment, a user may transmit a rich variety of data types and sizes. For example, a user may send or receive a compressed 100 KB text file, a 1.3 MB binary executable, or a 12 KB

bitmap almost entirely composed of white pixels. We chose 8 data benchmarks to obtain the performance results; their names and descriptions are listed in Table 3.

The first six are members of the Calgary corpus [3]. Researchers use the Calgary corpus to evaluate the practical performance of text compression algorithms. The remaining two benchmarks, gif and random, represent a GIF compressed image file and a randomly generated binary data file, respectively. We expect, therefore, that the gif and random benchmarks will be relatively incompressible. The 6 Calgary corpus benchmarks will exhibit different levels of compressibility, and therefore the performance will vary when using IPComp in conjunction with ESP and AH.

Table 3: Benchmark descriptions

Name	Description
obj2	Compiled code for Apple Macintosh: Knowledge support system
progl	Lisp source code for system software
paper2	A technical paper entitled "Computer (in)security" by Witten
trans	Transcript of a session on a terminal
book1	A book entitled *Far from the Madding Crowd* by Hardy
pic	Picture number 5 from the CCITT Facsimile test files (text + drawings)
gif	Compressed GIF file that contains a campus map
random	Randomly generated data

Table 4. Compression ratios

	DEFLATE			LZS		
	1K	4K	63K	1K	4K	63K
obj2	1.76	2.08	2.36	1.52	1.67	1.77
progl	2.31	3.16	4.23	1.76	2.16	2.33
paper2	1.74	2.08	2.70	1.28	1.50	1.61
trans	1.93	2.55	5.02	1.52	1.82	2.01
book1	1.68	1.94	2.34	1.22	1.40	1.46
pic	26.1	67.5	21.4	10.0	12.7	8.01
gif	0.99	1.00	1.00	0.89	0.89	0.89
random	0.99	1.00	1.00	0.89	0.89	0.89

5.4. Compressibility Results

We compute the compressibility results for both DEFLATE and LZS using the 8 benchmarks and the 3 payload sizes defined in Section 4.1. In order to avoid any compression anomalies associated with compressing the first few kilobytes of the benchmarks, we compress several different portions of the benchmarks when using the 1-KB and 4-KB payload sizes. For the 1-KB payloads, we compress the first 40 1-KB blocks of the benchmark, and the compressibility results are based upon the average compressibility of those 40 blocks. Similarly, for the 4-KB payload sizes, we compress the first 10 4-KB blocks of the benchmark. When using the 63-KB payload size, however, we only compress the first 63-KB block of the benchmark. In addition, to avoid timing imprecision resulting from the relatively high granularity of the result of the `clock()` function, we repeat the compression and decompression of the blocks thousands of times.

The eight benchmarks exhibit many different levels of compressibility. Table 4 summarizes the compression ratios achieved by DEFLATE and LZS. The size of the compressed data block includes the 4-byte IPComp header. The zlib implementation of DEFLATE achieves a higher compression ratio than LZS for each packet size and benchmark combination. DEFLATE usually outperforms LZS in compression ratio by a factor of 1.0 to 2.0, but the ratio of the compression ratios can exceed 5.0 for highly compressible data.

Table 5. DEFLATE performance (Mbps)

Benchmark Name	Compression			Decompression		
	1K	4K	63K	1K	4K	63K
obj2	14	21	23	66	121	365
progl	18	31	31	77	146	492
paper2	17	25	23	67	123	365
trans	17	29	43	70	143	492
book1	17	23	20	68	112	340
pic	34	72	76	166	299	703
gif	14	28	31	0	0	0
random	14	29	31	0	0	0

Table 6. LZS performance (Mbps)

Benchmark Name	Compression			Decompression		
	1K	4K	63K	1K	4K	63K
obj2	217	210	201	254	252	245
progl	234	245	252	253	263	263
paper2	178	179	182	224	214	211
trans	216	222	232	259	255	259
book1	172	169	169	220	207	203
pic	983	1047	735	498	537	476
gif	186	160	150	0	0	0
random	186	161	151	0	0	0

5.5. Throughput Results

The performance of the compression algorithms depends on both the size and compressibility of the benchmark. Tables 5 and 6 list the compression and decompression rates for DEFLATE and LZS, respectively. The zero values for gif and random in the

$$S = \frac{\frac{1}{R_E}N_P + \frac{1}{R_{HC}}N_P + \frac{1}{R_{SECNET}}N_P + \frac{1}{R_{HV}}N_P + \frac{1}{R_D}N_P}{\frac{1}{R_{COMP}}N_P + \frac{1}{R_E}\left(\frac{N_P}{X}\right) + \frac{1}{R_{HC}}\left(\frac{N_P}{X}\right) + \frac{1}{R_{SECNET}}\left(\frac{N_P}{X}\right) + \frac{1}{R_{HV}}\left(\frac{N_P}{X}\right) + \frac{1}{R_D}\left(\frac{N_P}{X}\right) + \frac{1}{R_{DECOMP}}N_P}$$

Figure 5. Speedup due to data compression

decompression columns are a consequence of IPComp's non-expansion policy (described earlier).

Tables 5 and 6 show that LZS always greatly outperforms DEFLATE in compression rate. LZS compresses data nearly as fast as SHA-1 authenticates data, and DEFLATE compresses data almost as slowly as 3DES encrypts data. In addition, LZS significantly outperforms DEFLATE in decompression rate for 1 KB and 4 KB payloads. DEFLATE, however, always achieves a higher decompression rate than LZS for 63 KB payloads. We observe that DEFLATE always achieves a higher compression ratio than LZS, but LZS usually compresses and decompresses data faster than DEFLATE. Which algorithm should we use to maximize performance? We address this issue in Section 7.

6. Calculating the Performance Impact of IPComp on IPsec Transactions

In this section, we present equations that describe the performance impact of combining ESP and AH with IPComp. First, we introduce the equation we use to calculate the speedup. Second, we use this speedup equation to derive an inequality that describes the conditions required for compression to improve performance.

6.1. Speedup Calculation

We calculate the speedup using a single equation. T_E, T_D, T_{HC}, T_{HV}, and T_{SECNET} represent the encryption time, decryption time, hash computation time, hash verification time, and transmission time, respectively. The variables C_E, C_D, C_{HC}, C_{HV}, and C_{SECNET} represent the time needed to encrypt, decrypt, compute the hash, verify the hash, and transmit a *compressed* packet, respectively. If encryption were not being used, for example, then T_E and C_E would both equal 0. Furthermore, the variables T_{COMP} and T_{DECOMP} represent the time required to perform the payload compression and decompression, respectively. We calculate the speedup S for the system model as follows:

$$S = \frac{T_E + T_{HC} + T_{SECNET} + T_{HV} + T_D}{T_{COMP} + C_E + C_{HC} + C_{SECNET} + C_{HV} + C_D + T_{DECOMP}}$$

A speedup greater than or equal to 1 indicates that IPComp reduces the total amount of time required by a secure transaction. Hence, a speedup greater than or equal to 1 means compression improves performance. A speedup less than 1 indicates that compression increases the total time needed; therefore compression degrades performance.

6.2. Predicting the Performance Impact of Data Compression

We now derive an inequality that describes the conditions required for data compression to improve system performance. We rewrite the speedup equation as shown in Figure 5. R_E, R_D, R_{HC}, R_{HV}, R_{COMP}, R_{DECOMP}, and N_P represent the encryption rate, the decryption rate, the hash computation rate, the hash verification rate, the compression rate, the decompression rate, and the size of the original (uncompressed) ULP payload, respectively. R_{SECNET} is the transmission rate of the network (i.e., effective network bandwidth). X is the compression ratio, which equals the size of the uncompressed payload divided by the size of the compressed payload. In order to make the relationships clear, we choose to neglect the overhead resulting from packet headers. For example, the amount of data to be hashed is not N_P as the previous equation indicates. The hash is calculated over the entire packet, so the amount of data to be hashed is at least N_P plus the size of the IP and AH headers. The total size of all the packet headers is usually less than 100 bytes, so if the payload size is relatively large, e.g., multiple kilobytes, the omission of the header overheads will not significantly affect the results.

Compression improves performance if the speedup is greater than or equal to 1. Hence, by setting the right side of the previous equation to be greater than or equal to 1, we obtain a relationship between the compression ratio and the algorithm throughputs such that compression will improve performance. N_P cancels out since we are not considering overhead from packet headers, and we now have the following equation:

$$\left(1 - \frac{1}{X}\right)\left(\frac{1}{R_E} + \frac{1}{R_D} + \frac{1}{R_{HC}} + \frac{1}{R_{HV}} + \frac{1}{R_{SECNET}}\right) \geq \frac{1}{R_{COMP}} + \frac{1}{R_{DECOMP}}$$

If the encryption rate, decryption rate, hash computation rate, hash verification rate, and network transmission rate are constant, we obtain the following expression:

$$K\left(1-\frac{1}{X}\right) \geq \frac{1}{R_{COMP}} + \frac{1}{R_{DECOMP}}, \text{ where}$$

$$K = \frac{1}{R_E} + \frac{1}{R_D} + \frac{1}{R_{HC}} + \frac{1}{R_{HV}} + \frac{1}{R_{SECNET}}$$

This inequality describes a relationship between compression rate, decompression rate, and compression ratio such that the resulting speedup will be greater than or equal to 1.

Using this inequality, one can determine whether compression will improve the performance of secure packet transactions without conducting extensive simulations. Secure network transactions in some systems involve other time-consuming steps such as bus transfer time, disk access time and protocol processing time. The speedup equations presented here could be modified to model a particular system more accurately by adding relevant latency terms.

Table 7: System parameters

Parameter	Values
Encryption algorithm	3DES, RC5, NULL
Authentication algorithm	MD5, SHA-1, NULL
Compression algorithm	DEFLATE, LZS, NULL
ULP payload size	1 KB, 4 KB, 63 KB
Data benchmark	obj2, progl, paper2, trans, book1, pic, gif, random
Network bandwidth	56 kbps, 1.54 Mbps, 10 Mbps, 100 Mbps, 1 Gbps

7. Experimental Results and Analysis

In this section, we evaluate the experimental performance results for all the combinations of parameter values listed in Table 7. We execute the security and compression algorithms on the HP Visualize C360 workstation described in Section 4. The simulations produced an enormous amount of performance information due to the large number (3240) of parameter value combinations; we only present the most valuable results here.

We describe results for 4 of the 8 benchmarks and for 2 of the 3 packet sizes. Using the compressibility results discussed in Section 5, we divide the data benchmarks into 4 groups according to level of compressibility. We select 1 benchmark from each of these 4 groups: pic, gif, book1, and trans. The benchmarks pic and gif represent highly compressible and incompressible payloads, respectively, and book1 and trans fall in between. In addition, we only present results for the 1 KB and 63 KB payload sizes. Most of the results for the 4 KB payload size closely resemble those for the 1 KB payload size.

The choice of compression algorithm depends on the network speed. We discover that compression ratio is more important than compression throughput when using slow network links, whereas the compression and decompression rates are more important than the compression ratio for fast network links. We find that DEFLATE usually yields a higher speedup than LZS for 56 kbps and 1.544 Mbps connections, and LZS usually yields a higher speedup than DEFLATE for 10 Mbps, 100 Mbps, and 1 Gbps links. It is important to note that these conclusions concerning compression algorithms and network types are highly implementation dependent. For example, if our LZS implementation were significantly slower, it would be advisable to use DEFLATE rather than LZS for 10 Mbps links. In general, fast compression algorithms should be used with fast network connections, and algorithms that achieve high compression ratios should be used with slow network connections.

Upon inspecting the speedup results, we discover that certain different algorithm combinations produce similar results. For example, consider the case where message authentication is used but encryption is not used. In almost every case, the speedups for MD5 and SHA-1 (when using compression) are both greater than 1 or are both less than 1. In other words, the decision to compress when using AH but not ESP is independent of the authentication algorithm being used. We expected this result, for the throughputs of the MD5 and SHA-1 authentication algorithms differ by only 7.5% when using 63 KB payloads.

We propose 4 algorithm combination classes: combinations that include 3DES (slow encryption), combinations that include RC5 (fast encryption), combinations that include authentication only, and the case where neither authentication nor encryption are employed (referred to as NULL). The results of the NULL case can be used to determine whether compression improves the performance of ordinary communications. We present performance results for one algorithm combination from each class. The algorithm combinations are NULL, SHA-1, RC5/SHA-1, and 3DES/SHA-1; we use SHA-1 since it is computationally more intensive than MD5. For each of these algorithm combinations, we use our earlier recommendations for the compression algorithm that yields the highest performance for a given network type.

Figures 6 through 9 illustrate the speedups obtained for the four algorithm combinations. In all four cases, compression improves performance for network bandwidths less than or equal to 10 Mbps. Figure 6

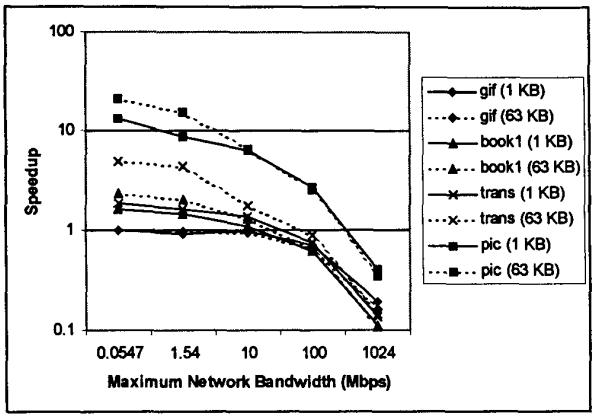

Figure 6. Speedups when using compression only

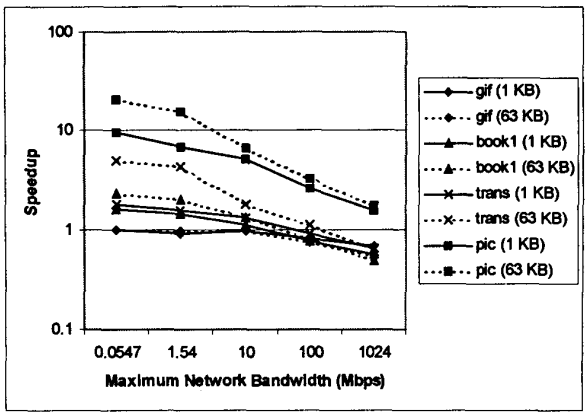

Figure 7. Speedups when using SHA-1

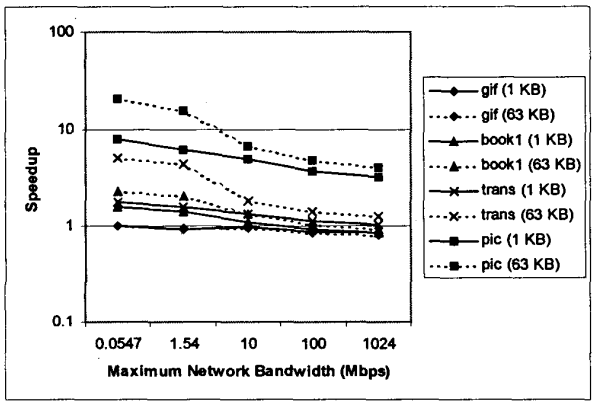

Figure 8. Speedups when using RC5/SHA-1

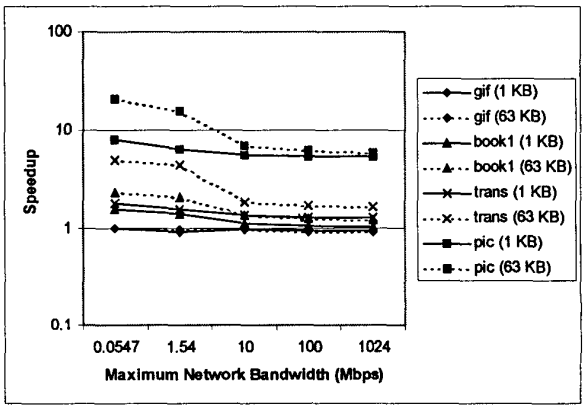

Figure 9. Speedups when using 3DES/SHA-1

depicts the speedups achieved when using compression without encryption or authentication. Compression almost always degrades performance when using 100 Mbps and 1 Gbps networks. Figure 7 illustrates the speedups obtained when using SHA-1. Compression used with message authentication only can improve performance for 100 Mbps and 1 Gbps networks if the packet payloads are expected to be highly compressible (i.e., compressible by a factor of 10). Figures 8 and 9 show the speedup results when using RC5/SHA-1 and 3DES/SHA-1, respectively. For slow encryption (3DES/SHA-1), compression almost always improves performance when using 100 Mbps or 1 Gbps network connections. For fast encryption (RC5/SHA-1), compression improves performance for 100 Mbps and 1 Gbps links if the data is reasonably compressible (i.e., compressible by a factor of 1.5).

If the security and compression algorithms are all executed on workstations similar to the one we used, compression will improve performance for network bandwidths less than or equal to 10 Mbps. In addition, compression improves performance when using network bandwidths as high as 1 Gbps if the payload data is reasonably compressible and encryption is used. We define reasonably compressible data to mean a compression algorithm can reduce the size of packet payload by at least a factor of 1.5.

If processing power increases relative to network speed, the speedups generated by compression will also increase. If processing power decreases relative to network speed, compression will be more likely to cause performance degradation. Similarly, if compression speed and ratio increase relative to encryption and authentication speed, performance speedups due to compression will increase. If compression speed and ratio decrease relative to encryption and authentication speed, compression will be less beneficial in improving system performance.

8. Conclusions

In this paper, we investigated the performance impact of combining data compression with encryption and message authentication to provide for fast, secure network transactions. We defined a system model that allows us to quickly estimate the performance of many combinations of packet, network and algortihm types. First, we demonstrate the extent to which IPsec can degrade performance. Encryption with 3DES can reduce system performance by 36% for 10 Mbps networks and 85% for 100 Mbps networks, i.e., an encrypted transaction can be 1.5 times slower than normal transactions over a 10 Mbps network and 6.6 times slower over a 100 Mbps network.

We derived an inequality that predicts whether data compression will improve performance for a given system. This inequality depends on the throughputs of the encryption, authentication and compression algorithms used, on the bandwidth of the network, and on the compressibility of the packets. We analyzed performance results for 3240 combinations of system parameter values and categorized them into four classes: compression with no authentication or encryption, compression with authentication (and no encryption), compression with fast encryption, and compression with slow encryption. In each class, compression usually improves performance for 10 Mbps or slower networks. For 100 and 1000 Mbps networks, compression only improves performance when encryption (fast or slow) is employed: compression does not improve performance for these faster networks when message authentication is used without encryption.

Future work includes exploring the impact of data compression on secure information processing using more complex models that include communications pipelines, multiple clients and servers, and multiprocessor systems.

9. Acknowledgements

We wish to thank Hewlett Packard Company for funding this research and providing equipment. We also thank Zhijie Shi for writing the encryption and message authentication routines used in this study.

References

[1] American National Standards Institute, Inc., "Data Compression Method – Adaptive Coding with Sliding Window for Information Interchange", ANSI X3.241-1994, August 1994.

[2] Baldwin, R., and R. Rivest, "The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms", RFC 2040, October 1996.

[3] Bell, T.C., et al., *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[4] Deering, S., and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[5] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[6] Deutsch, P., "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[7] Friend, R., and R. Monsour, "IP Payload Compression Using LZS", RFC 2395, December 1998.

[8] Glenn, R., and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, November 1998.

[9] Kent, S., and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[10] Kent, S., and R. Atkinson, "IP Encapsulating Security Payload", RFC 2406, November 1998.

[11] Kent, S., and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[12] Krawczyk, H., et al., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[13] Madson, C., and R. Glenn, "The Use of HMAC-MD5-96 within ESP and AH", RFC 2403, November 1998.

[14] Madson, C., and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.

[15] Nahum, E., et al., "Parallelized Network Security Protocols", *Proceedings of the Internet Society Symposium on Network and Distributed System Security (SNDSS '96)*, February 1996.

[16] Nahum, E., et al., "Towards High Performance Cryptographic Software", *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communications Subsystems (HPCS '95)*, August 1995.

[17] Pereira, R., "IP Payload Compression Using DEFLATE", RFC 2394, December 1998.

[18] Pereira, R., and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[19] Postel, J., "Transmission Control Protocol", RFC 793, September 1981.

[20] Schneier, B., *Applied Cryptography Second Edition*, John Wiley & Sons, New York, NY, 1996.

[21] Shacham, A., et al., "IP Payload Compression Protocol", RFC 2393, December 1998.

[22] Shi, Z., and R. Lee, "Bit Permutation Instructions for Accelerating Software Cryptography", *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, July 2000.

[23] Tanenbaum, A., *Computer Networks Third Edition*, Prentice-Hall, Upper Saddle River, NJ, 1996.