Paper

A new approach to header compression in secure communications

Christoph Karg and Martin Lies

Abstract—The paper presents a new header compression mechanism for the IPv6 protocol. Its main benefit is the reduction of the overhead caused by IPSec tunnel mode which enlarges datagrams in order to provide security services such as authentication and secrecy.

Keywords—IPv6, header compression, IP security, restricted bandwidth.

1. Introduction

In contrast to civil networks, tactical military computer networks lack the broad data rate links. This is especially true for those tactical networks where the deployment is dependent on a high mobility. Because of the necessity to utilize wireless techniques or even open ISPs as transfer-nets, the topic of security becomes ever more important. The chosen security mechanism has to provide strong encryption of the transported data regardless of the deployed applications, has to support an authentication mechanism to limit the access and provide protection against a variety of attacks. These requirements are fulfilled by IPSec [2], as well as the need for a matching key exchange mechanism, the Internet key exchange (IKE) [6]. With these components, the communication of complete subnets can be secured using IPSec gateways, building virtual private networks (VPNs). In essence, every IP packet is completely encrypted inside another IP-packet, thereby hiding every information about the application, the transmitted data and also about the topology of the secured area.

The cost of the usage of IPSec tunnel mode is the growth in the required data rate for the same amount of transmitted data. To give an example, an Internet protocol Version 6 (IPv6) packet holds a header of 40 byte. IPSec tunnel mode adds 40 byte IP-header for the transfer in the black, i.e., unsecured network, then depending on the chosen authentication algorithm around 22 bytes for the authentication header (AH) and again depending on the chosen encryption algorithm another 22 bytes with a certain amount of padding to obtain the necessary length. This still ignores all application data.

Especially in narrow-bandwidth environments, e.g., communication across HF, GSM or satellite, the overhead caused by IPSec is not negligible. To enhance to performance of IPSec, a header compression mechanism was described in [5, 8]. The proposal resulted in the modification of an existing IPSec implementation. The release supported the following type of header compression. After a connection between two hosts inside two distinct secure subnets is established, the black header is replaced by a small index tag. Then the modified datagram is encrypted and encapsulated by the sending IPSec gateway. The receiving IPSec machine verifies the mandatory authentication. On success, it decrypts the payload and restores with the original red IPv6 header by replacing the index tag. Comprehensive tests substantiated the good performance of the header compression approach. Alas the approach has an major drawback. It is strictly link orientated and thus does not work in a network topology consisting of various (i.e., more than two) secure subnets.

This paper presents an enhancement of the above mechanism which works for arbitrary VPNs. It is developed to work within the network resource manager (NRM) [7]. An NRM is a security gateway with extended functionality. Besides enabling secure communications via IPSec, it gathers information on the current network traffic. This statistical data can be used to estimate the available bandwidth as well as the reliability of the existing connections. Furthermore, the NRM supplies an enhanced connection management in order to support secure multicast-based group communication and the corresponding key exchange mechanism (MIKE) as presented in [3].

The paper is organized as follows. In Section 2, the idea behind the header compression mechanism is described. The algorithms and data structures needed by header compression are presented in Sections 3 and 4, respectively. Section 5 provides information on how to integrate header compression into the IPSec framework. The performance of the compression mechanism is discussed in Section 6. The paper closes with concluding remarks in Section 7.

2. Header compression

The IP tunneling mechanism (see Fig. 1a for details) is an integral part of VPN solutions such as IPSec. A drawback is the enlarged datagram size, since two IP headers must be transmitted. In order to decrease the packet size, we propose to replace IP tunneling by an alternative header compression mechanism (see Fig. 1b). The idea is as follows. The sending gateway replaces the inner IP header by a compression header of smaller size. The compression header contains among other an unique identifier. This information enables the receiving gateway to reconstruct the original IP header and forward the datagram to its destination. The benefit comes from the fact that the identifier can be used often while the assignment between IP and compression header must be transmitted only a few times.

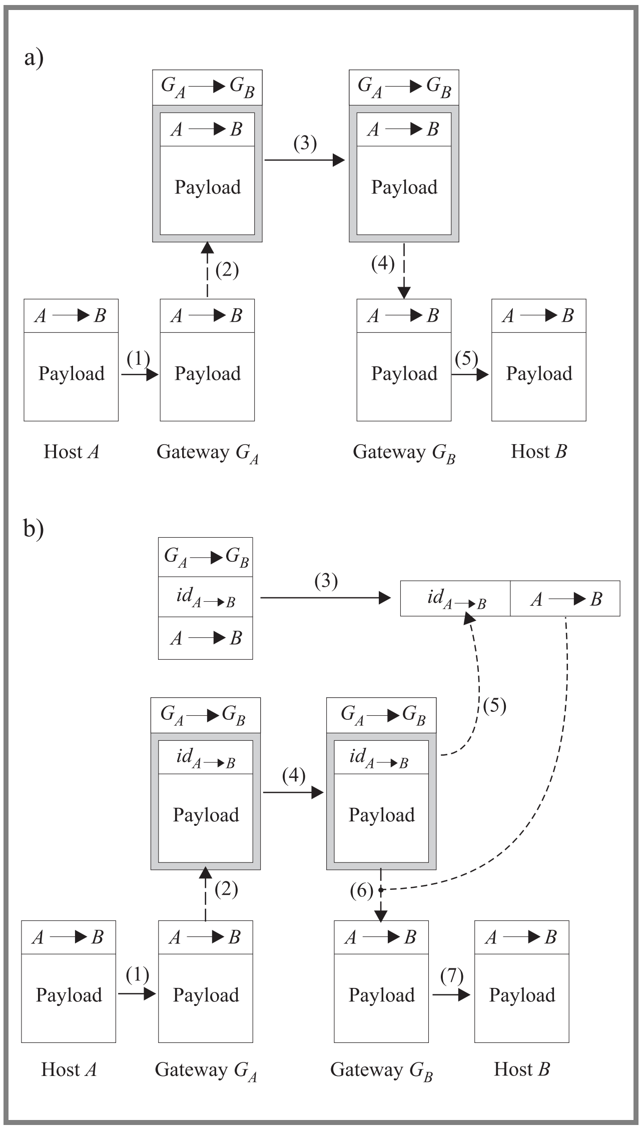

Fig. 1. (a) IP tunneling: (1) Host A sends an IP packet to host B via the gateways G_A and G_B, whereat the link between G_A and G_B is an IP tunnel. (2) On receipt, G_A stores the IP packet in a new one. This mechanism offers several opportunities to manipulate the IP packet to be tunneled. For example, IPSec encrypts the packet before transmission. (3) G_A sends the resulting packet to G_B. (4) G_B unfolds the incoming packet, this is, it removes the tunnel header and does some post-processing. For example, IPSec decrypts the payload and checks it't integrity. (5) Finally, G_B forwards the content to host B.
(b) Header compression: (1) Host A sends an IP packet to host B via the gateways G_A and G_B. Instead of tunneling the packet directly, gateway G_B performs the following manipulation. Based on the IP header $A \to B$, G_A computes an unique identifier $id_{A\to B}$. (2) G_A sends the pair $(A \to B, id_{A\to B})$ to gateway G_B, which stores the information in an internal lookup table. (3) G_A replaces the header $A \to B$ by the corresponding identifier $id_{A\to B}$ and sends the result to G_B. This replacement is done for all following IP packets with header $A \to B$. (4) On receipt, G_B removes the tunnel header and the identifier. (5) G_B uses $id_{A\to B}$ to reconstruct the original header $A \to B$. (6) G_B combines $A \to B$ with the payload. (7) G_B forwards the original IP packet to host B. If A sends another datagram to B, then the same identifier can be used. Hence, Step 3 can be omitted.

In the following we describe the header compression in more detail. There are several questions to be discussed.

- How do we compute the compression header and the identifier within?
- What is the structure of the header replacement?
- Which algorithms and data structures are required on sender and receiver side?
- How is header compression integrated in the security environment?

The following sections provide answers to these questions.

3. Computation of the compression header

The reduction of the datagram size is achieved by the replacement of the original IP header by a smaller compression header. Its structure depends on the algorithm to be used. There are two requirements to be met. Firstly, the information provided by compression header must be unique such that the receiver can reconstruct the original header. Secondly, the identifier should be usable for many different IP headers so that the amount of assignment messages is minimized.

A standard way to derive an unique identifier is universal hashing with open addressing (see [4] for an excellent discussion of this topic). If the range of the hash function fits into a 4-byte integer, there is a possibility of 2^{32} simultaneous outgoing transmissions. If the hash function is chosen uniformly at random then a good average case performance is guaranteed.

To determine the input of the hash function, let's take a look on the structure of the IPv6 header depicted in Fig. 2. The header size is 40 bytes. The largest part consists of the source and destination address each with a size of 16 bytes. Consider for a moment only datagrams sent from the host A to the host B. In this case, the source and destination address are constant. The contents of the fields payload length, next header, and hop limit may change frequently. The frequency of change of the contents stored in the fields traffic class and flow label is difficult to estimate. It depends on the quality of service features of the underlying network. As a rule of thumb, we assume a low modification rate.

Based on the above assumption, the fields version, flow label, traffic class, source address and destination address are the input of the hash function. With other words, the hash function maps 32 bytes to values of 4 byte length. Open addressing guarantees unique hash keys given the number of different inputs is smaller than the range (i.e., 2^{32}). Since the payload length, hop limit and next header entries do not influence the outcome of the hash function, they additionally must be stored in the compression

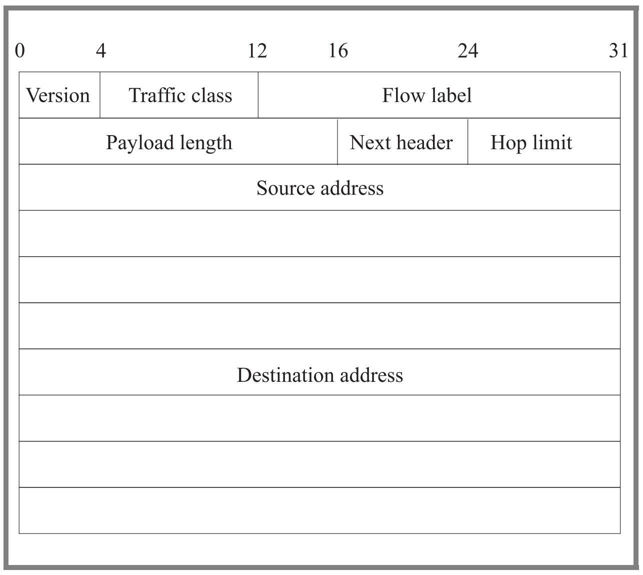

Fig. 2. Format of the IPv6 header.

header. This leads to the following structure of the compression header:

1) hash key to determine version, flow label, traffic class, source address and destination address;

2) payload length;

3) next header;

4) hop limit.

The size of the compression header is 8 bytes. Hence, it's size is 20% of the IP header.
Additional costs arise from the fact that the assignment between hash key and according IP header parts must be transmitted to the receiving gateways. The cost in terms of data rate depends on the way, the assignment information is distributed. Among others, two following methods are conceivable.

- If the compression starts, the first datagrams are not compressed. Instead, they are extended with an optional header which contains the hash key. On receipt, the destination gateway stores this identifier together with the associated parts of the IP header. On this way, it learns how to decompress future datagrams.

- In the beginning of compression, the first datagrams are transmitted without modification. Additionally, the assignment information is sent in a separate datagram.

Note, the transmission of the compression information is critical for the whole communication. Hence, the spread of the assignment information must be redundant to guarantee that the destination gateway gets the assignment before the compressed data.

The computation of the index is strictly sender oriented. This is, the sending gateway computes the identifiers of all outgoing datagrams independently on it's own. As an advantage, the gateways do not need to coordinate the identifier creation. This keeps things simple and there is no additional communication between the gateways required. Furthermore, compression header assignment messages are receivable even by those gateways which are in emission control (emcon) condition.

4. Data structures

The mapping between identifiers and according IP header fragments must be stored in an appropriate data structure. The requirement to be met is the fast lookup of a given index. The best performance is achieved by an hash table. However this is not feasible because of the large range of the deployed hash function and its many memory consumption. A good compromise between space and performance is the usage of a balanced search tree such as red-black trees [4]. Such a data structure guarantees time $0(\log_2 n)$ for search, insertion and deletion where n is the number of elements stored in the tree.
In order to compress a datagram the sending gateway acts as follows. It computes the hash index of the IP header fragment. Then, it replaces the header by the compression header. If it is the first time, that this header fragment is compressed, then the assignment information is sent the receiving gateway. Additionally, the pair consisting of index and header fragment is stored in the search tree.
On the receiving side, the data keeping is slightly different. Since the identifier is correlated with the gateway which created it, the receiver must manage a search tree for each sending gateway. These trees are stored in another search tree with the senders' IP addresses as key.
The contents of the search trees change dynamically. An entry may be deleted if the respective hash index was not used for a certain time. This behavior increases the performance of both running time and memory usage.

5. Integration into security services

The header compression mechanism is thought to be used in combination with a secure transmission method. An obvious approach is the IP security protocol suite [2]. Particularly, the compressed datagrams are sent via IPSec transport mode. The usage of IPSec has the advantage of well-developed administrative tools such as automated key negotiation. However, the header compression must be integrated in the IPSec processing chain. There are two possible solutions. The first one is the extension of the operating system's kernel. The second one consists in the packet filtering and manipulation in user space via the TUN/TAP interface.
The integration of header compression into the operating system results in optimal performance since the code runs

in kernel space. However for modification of the kernel the underlying source code is required. Even if the code is available, the compression algorithm must be adapted to the respective conditions of the operating system. Hence, the work is not portable to other operating systems. Another drawback is the large amount of maintenance required to keep the software up to date with new kernel releases.

The second approach uses the TUN/TAP interface for packet manipulation. TUN/TAP is a standardized mechanism for manipulation of IP packets and Ethernet frames. It is supported by major UNIX systems such as BSD, Linux and Solaris and even by Windows XP. Hence, the code of the header compression extension is portable between the different operating systems with moderate effort. A potential disadvantage is a loss in performance since the compression runs in user space and has to share the system's resources with the other processes.

Finally, we remark that IPSec is not the only way to deploy header compression. An alternative is OpenVPN [1], a VPN solution which uses the SSL/TLS protocol for a secure data tunneling. Compared to IPSec, OpenVPN is a lightweight VPN solution. It can be set up easily. Additionally, OpenVPN provides a traffic shaping functionality to restrict the rate of data flowing to the tunnel.

6. Performance estimates

To evaluate the performance of the header compression mechanism, two aspects have to be considered:

- **Network delay**. The compression of the IP header is an additional processing step in the data flow which increases the delay between sending and receiving host.
- **Data rate improvement**. Header compression is deployed with the goal of reducing the amount of data to be transmitted. Hence, this criterion has to analyzed.

The influence on the network delay can be estimated by the amount of time needed to compute the compression header. The expensive part is the lookup of the hash index in the balanced search tree.

A well-known result concerning universal hashing is the following [4]. If the hash function h is randomly chosen from a universal collection of hash functions and is used to hash n keys into a table of size m, where $m \leq n$, the expected number of collisions involving a particular key is less than 1. Hence, if the outgoing connections result in n different hash indices, where $n \leq 2^{32}$, then there are no collisions in the average case. Since the keys are stored in a balanced tree, the expected running time per lookup operation is logarithmic in n.

What does this mean in practice? Assume, that the search tree is based on red-black trees. Then a tree storing n hash indices as height at most $2\log_2(n+1)$. This value is an upper bound for the cost of a lookup operation. Each entry consists of at least 40 bytes (4 bytes for the index and 36 for the IP header fragment). This is a lower bound for the memory needed to store the search tree. If any host in the subnet has 10000 concurrent outgoing data connections then the number of different hash indices is at most 10000. Given the total number n of hosts, the upper bound of different indices is $10000n$. Using this bound, we estimate the size of the tree for concrete values for n (Table 1).

Table 1
Tree size estimations

Hosts	Indices	Tree height	Size [MB]
10	100000	34	3.815
20	200000	36	7.629
50	500000	38	19.073
100	1000000	40	38.147
200	2000000	42	76.294
500	5000000	45	190.735

Because the number of steps to compute a table lookup is small even for a large number of hosts, the delay caused by header compression is negligible on modern computer hardware.

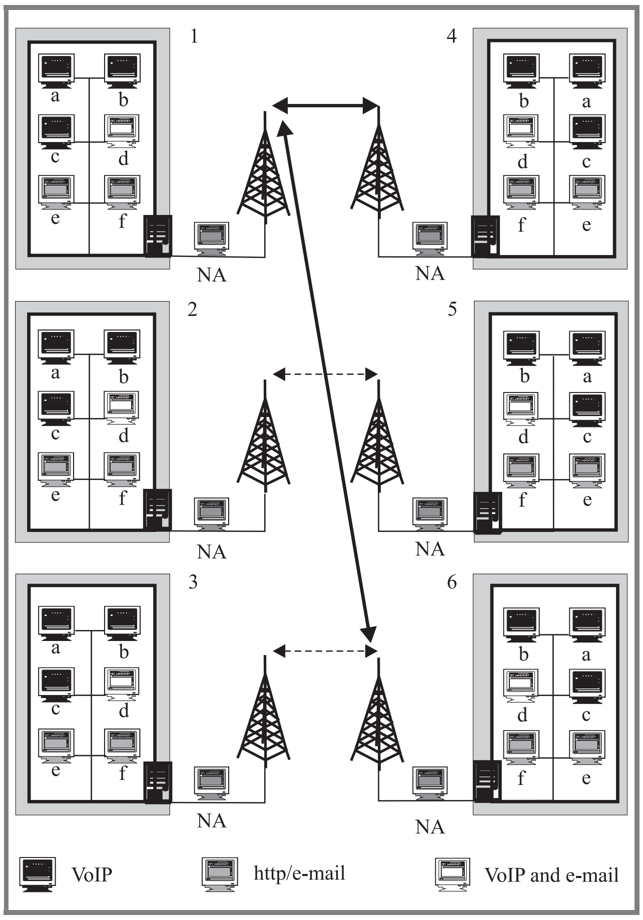

Fig. 3. The topology of a tactical military network at batallion level.

The improvement of the data rate is quite difficult to estimate. It depends on both the topology of the underlying network and the type of traffic and its distribution within the network. The topology can be created based on existing military tactical networks. For an example, we refer to Fig. 3. The determination of network traffic structure in terms of distribution and service type is a difficult or even impossible venture.
A promising approach consists in a statistical analysis via a network simulation which is beyond the scope of this paper. As an obvious rule of thumb, the gain of header compression is high for network services which send datagrams in an high frequency. A good example is voice over IP (VoIP).

7. Conclusions and future work

This paper presents a new approach to header compression to be used in combination with the IP security framework. The benefit is a reduction of the overhead caused by IPSec tunnel mode in terms of enlarged datagrams. Furthermore, it can be integrated in common existing operating systems with moderate effort. Currently the header compression is a concept. Future steps are simulations based on network simulator 2 and the implementation in a testing environment.

References

[1] OpenVPN. Project webpage, http://www.openvpn.net

[2] R. Atkinson and S. Kent, "Security architecture for the Internet protocol", RFC 2401, Nov. 1998.

[3] T. Aurisch and C. Karg, "A daemon for multicast Internet key exchange", in *IEEE Conf. Loc. Comput. Netw.*, Bonn, Germany, 2003, pp. 368–376.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge: MIT Press, 2003.

[5] R. Göbel, "Beschreibung eines QoS-unterstützten Netzadapters für schmalbandige Subnetztypen", FKIE-Bericht 38, FGAN, Jan. 2002.

[6] D. Harkins and C. Carrel, "The Internet key exchange (IKE)", RFC 2409, Nov. 1998.

[7] M. Lies, P. Sevenich, C. Karg, and C. Barz, "Resource management in tactical military networks", in *NATO/RTO IST Symp. Milit. Commun.*, Roma, Italy, 2005.

[8] P. Sevenich and G. Beling, "Multiplexing time-critical data over heterogeneous subnetworks of low bandwidth", in *Reg. Conf. Milit. Commun. Inform. Syst. RCMCIS*, Zegrze, Poland, 1999.

Christoph Karg is a Professor of computer science at the University of Applied Sciences at Aalen, Germany. Doctor's Karg research interests are computer networks, cryptology, algorithms and complexity theory.

e-mail: christoph.karg@htw-aalen.de
Fakultät Elektronik und Informatik
HTW Aalen
Beethovenstraße 1
73430 Aalen, Germany

Martin Lies is a senior scientist of the communications department of FGAN/FKIE. His research interests are computer networks with special emphasis on security and resource restrictions, robotics and cryptology.

e-mail: lies@fgan.de
Department Computer Networks
FGAN
Neuenahrer Straße 20
53343 Bonn, Germany