A QUERY UNIT FOR THE IPSEC DATABASES

Alberto Ferrante, Sathish Chandra
ALaRI, Faculty of Informatics, University of Lugano, Lugano, Switzerland

Vincenzo Piuri
DTI, University of Milano, Milano, Italy

Keywords: IPSec, IPSec Databases, IPSec accelerator, SADB, SPD.

Abstract: IPSec is a suite of protocols that adds security to communications at the IP level. Protocols within IPSec make extensive use of two databases, namely the Security Policy Database (SPD) and the Security Association Database (SAD). The ability to query the SPD quickly is fundamental as this operation needs to be done for each incoming or outgoing IP packet, even if no IPSec processing needs to be applied on it. This may easily result in millions of query per second in gigabit networks.
Since the databases may be of several thousands of records on large secure gateways, a dedicated hardware solution is needed to support high throughput. In this paper we discuss an architecture for these query units, we propose different query methods for the two databases, and we compare them through simulation. Two different versions of the architecture are presented: the basic version is modified to support multithreading. As shown by the simulations, this technique is very effective in this case. The architecture that supports multithreading allows for 11 million queries per second in the best case.

1 INTRODUCTION

IPSec is a suite of protocols that adds security to communications at the IP level. This suite of protocols is becoming more and more important as it is included as mandatory security mechanism in IPv6. IPSec is mainly composed of two protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP). The former allows authentication of each IP datagram's selected header fields or – depending on the operational mode that has been selected – of the entire IP datagram. The latter allows encryption – and optionally authentication – of the entire IP datagram or of the IP payload, depending on the operational mode that has been selected, namely the transport and the tunnel modes. The former was designed for being used in host machines, the latter is for secure gateways. In tunnel mode the entire original IP datagram is processed; the result becoming the data payload of a new IP datagram with a new IP header. In transport mode only parts of the original IP datagram are processed (e.g., the data payload for the ESP protocol) and the original IP header is kept with some small modifications. Through encryption, authentication, and other security mechanisms included in IPSec (e.g., anti-reply), data confidentiality, data authentication, and peer's identity authentication can be provided (Kent and Atkinson, 1998c) (Kent and Atkinson, 1998a) (Kent and Atkinson, 1998b) (Harkins and Carrell, 1998) (Kent, 2005a) (Kent, 2005b). IPComp, a protocol for data payload compression, is also included in the IPSec suite of protocols (Shacham et al., 1998).

Two databases are involved in processing IP traffic. These two databases are the Security Policy Database (SPD) and the Security Association Database (SAD). The former specifies the policies that determine the disposition of all IP traffic. The latter contains parameters that are associated with each SA. The SPD needs to be queried for each packet traversing the IP communication layer. Upon the conformance with the SPD, an IP datagram needs to be processed by IPSec, the SAD also needs to be queried to discover the parameters of the considered SA. Information about whether a SA has already been created or not are contained in the SPD. If a suitable SA

Ferrante A., Chandra S. and Piuri V. (2007).
A QUERY UNIT FOR THE IPSEC DATABASES.
In *Proceedings of the Second International Conference on Security and Cryptography*, pages 133-139
DOI: 10.5220/0002119901330139

for the IP datagram to be processed does not exist, it needs to be established using the Internet Key Exchange protocol (IKE) (Harkins and Carrell, 1998).

IPSec is often used to create Virtual Private Networks (VPNs). A VPN is an extension of a private network on a public network (e.g., the Internet) (Feghhi and Feghhi, 2001) (Yuan and Strayer, 2001). The extended part of the network logically behaves like a private one. Typical usage scenarios for VPNs are: remote user access to a private LAN over the Internet and connection of two private networks. In these cases a virtual secure channel needs to be created, respectively, from the user's PC to the LAN public access point or from one LAN to the other. Private network public access points are called *secure gateways*. A secure gateway is a router or a router/firewall also running a VPN-enabled software (e.g., an IPSec implementation). All the traffic within the LAN is usually not protected, while the traffic going out or coming in the LAN through the secure gateway is protected by some security mechanisms.

IPSec has proved to be computationally very intensive (Miltchev et al., 2002) (Ariga et al., 2000) (Alberto Ferrante et al., 2005b). Thus some hardware acceleration is needed to support large network bandwidths, as may be required even in small secure gateways. Cryptography is often believed to be the only part of the IPSec suite that requires a large amount of resources. In the reality, IPSec implementations also require to perform other operations, such as header processing and IPSec database querying. The latter may become a bottleneck for the system as it requires to be done at least once for each IP packet that is traversing the system. In fact, the SPD needs to be queried for each IP packet, the SAD needs to be queried only when IP packets are determined to require some IPSec processing. Considering an overall traffic of 1Gbit/s, and the worst possible case (i.e. the packets are received at the maximum possible rate and their size is the smallest possible one, that is 40bytes), the SPD needs to be queried $3,355,443$ times per second. On average, queries are usually fewer then one million per second in a normal system operating at the same speed. In any case, an efficient database query unit is therefore vital to achieve high performance

In this paper we present a study about a database query unit for the SAD and the SPD databases; in the best configuration this unit is able to perform 11 million of queries per second. Section 2 describes the different possible architectural solutions and the different techniques that can be adopted for the database query. This database unit has been taught to be used in IPSec accelerators such as the one shown in (Alberto Ferrante and Vincenzo Piuri, 2007). Section 3 presents the model for the simulations and the obtained results. Section 4 shows an improvement of the proposed architecture and the related simulations and results. Section 5 shows a study of the optimal architecture when an area–delay cost function is considered.

2 SYSTEM ARCHITECTURE AND DATABASE QUERY TECHNIQUES

As shown in Figure 1, the main databases are stored in an off-chip memory and are accessed through on-chip caches, one for the SAD and the other one for the SPD. This structure remains the same for both hardware and software implementations of the query unit. An off-chip memory for the databases provides flexibility at the cost of diminishing performance. In fact, an external memory provides ease expandability; on the opposite, an internal memory delivers performance that cannot be reached by external ones. The main goal of the cache is to mask the access to the external memory thereby reducing the access time. In our case the total query time is not only given by the physical access time of the external memory, but also by the lookup time of the records that are stored in it.

The two caches are implemented as two Content Addressable Memories (CAMs) (Kostas Pagiamtzis, nd) (Pagiamtzis and Sheikholeslami, 2003). With this kind of memory cells can be addressed by a part of their contents. Therefore, they provide a good way to implement lookup tables. For this reason they are often used in routers and network processors (see, for example, (Prashant Chandra et al., 2003)). The two databases can be implemented in external memory either in shared or unshared fashion; even if the memory is physically shared, the databases should be considered as logically separated.

When a new packet arrives, the database is first queried in the cache; if a cache miss occurs, then a query is performed in the main memory. Hence, the worst case search time for a record is the sum of the time required to perform a query in cache and the time to perform a query in the main database. The best case search time is defined as the time to do a query in the cache. Depending on the implementation of the database query unit, the memory may need more than one port. Later in this section we discuss different methods to query the databases and different cache replacement techniques.

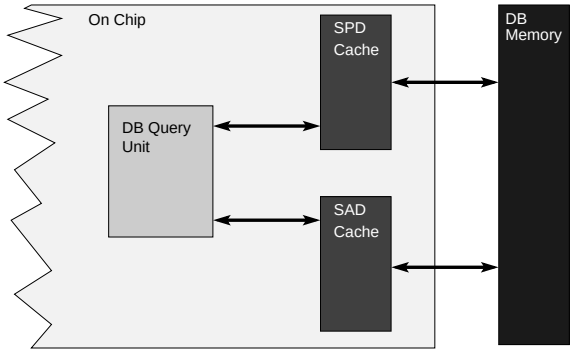

Figure 1: Core of the database query unit.

2.1 Size of the Records

SPD record size is variable. In fact, several configuration proposals can be stored in each one of them. All of these proposals are used only during the SA negotiation phase and not during the normal system operation. Therefore, the SPD records can be divided into two parts: the first one containing the essential information (i.e., the source and destination IP addresses, the direction, the policy, the pointers to the SAs, and the pointer to the first element of the list of proposals); the second one containing the list of proposals. The first part is cached; the second one – that is used only during SA negotiations – is stored into a special area of the external database memory. The size of the part of the record that needs to be cached can be estimated in 232 bits.

SAD records contain the settings of the protocols and of the algorithms, along with the keys for the cryptographic algorithms. An estimation of the size of each SAD record is of 792 bits.

2.2 Main Database Lookup Techniques

Two different techniques for database lookup have been considered in this work: the Linear LookUp Technique (LLUT) and the Partitioned LookUp Technique (PLUT) (Henry Hong-Yi Tzeng and Tony Przygienda, 1999).

By using the LLUT technique, records are queried in a linear fashion. This is the simplest technique to implement, but it is very slow.

In the PLUT technique, the search in the database is performed by using a tag that is obtained by summing up the first three decimal digits of the source IP address. When IPv4 addresses are considered, each database is divided into 10 sections. For example, a record containing a source IP address that is 192.168.8.1 should be placed in the DB space 3 ($1+9+2=12$ and $1+2=3$). Inside each subspace the search is performed linearly. This technique should provide far better speed performance than the LLUT, but it also has a drawback: packets may tend to concentrate in certain memory partitions. Therefore, some of these will remain almost empty; others will be overutilized.

2.3 Cache Replacement Policies

Two different cache replacement policies have been considered: First In First Out (FIFO) and Least Recently Used (LRU) (John Hennessy and Dave Patterson, 2002, p. 378). When the cache is full, the first loaded record is replaced, according with the FIFO policy. By considering the LRU policy, the least recently used record in the cache is replaced. Before replacing the record in the cache, the record is written back in the memory just if some modifications have occurred to it.

When the SAD is considered, it may happen that the database memory (or a memory segment) becomes full. Two different actions can be taken in this situation: no new SA creation is allowed, or the oldest SAs are discarded and the creation of new ones is allowed. In this work we adopted the second solution, but in a real life system the behavior to adopt in this case should be specified in the system security policy.

3 SIMULATIONS

The described architecture, along with the different parameters, have been simulated by using a SystemC model. The SystemC language (Sys, nd) was selected to describe our model as it allows for specification of hardware-software systems. Delays associated with the operations can be easily modeled with this language. The next subsections show the model and the results of the simulations.

3.1 Description of the Model

Our SystemC model provides the ability to simulate the different lookup techniques and cache replacement methods described above, along with different cache sizes. With this model the delays of the different operations that are performed during the database query phase are simulated.

Inputs used for the simulations are taken from network trace files, e.g., the ones provided on the Internet Traffic Archive (ITA, nd) website. These files contain long traces obtained by using the *tcpdump* tool (tcp, nd) on various networks. We have considered a

Table 1: Sizes of the caches in number of elements and in bytes.

Number of elements	SPD size [bytes]	SAD size [bytes]
64	1,856	6,336
128	3,712	12,672
256	7,424	25,344
512	14,848	50,688
1024	29,696	101,376
2048	59,392	202,752
4096	Not used	405,504

trace taken from a 2Mbit/s gateway and that contains about 3.8 million TCP packets. For our simulations we used only 1 million of these packets to avoid long simulations. Among the different parameters in the trace file we considered source and destination IP addresses thereby ignoring the timestamps and all the other information contained in the file.

During SPD query, only some parts of the SPD records need to be fetched from memory to identify a possible match; the size of this part is 203 bits. During SAD query, only some parts of the SAD records need to be fetched from memory to identify a possible match; the size of this part is 129 bits. The remaining parts of the records need to be fetched only when a match is found.

Some memory structure need to be used for implementing FIFO and LRU policies. These structures, which are saved in an on-chip memory, are updated while data are fetched from the memory; therefore, their management does not introduce any further delay.

During the simulations, an access time of 7*ns* has been considered for the CAMs as suggested in (Kostas Pagiamtzis, nd); an access time of 30*ns* has been used for the memory. A read/write time of 10*ns* has been used for all the memory transfers (on a 64-bit bus) after the first one.

3.2 Simulation Results

The simulations provide different results: the average time per query and the hit and replacement rates of the caches. The average time per query is computed as the total simulated time divided by the number of requests that have been processed in such a time.

Table 1 shows the correspondence between cache sizes in number of elements and in bytes for the SAD and for the SPD. All of these cache sizes have been simulated; they have been combined with the LLUT and the PLUT query techniques and with the FIFO and the LRU replacement methods. All the possible combinations of these parameters result in 168 different system configurations. In the first 84 configurations the LLUT technique is considered. Configurations from 85 to 168 are similar to the first 84 ones; in these configurations the PLUT technique is considered instead of the LUT one. Figure 2 shows the different SPD and SAD cache sizes that have been considered in system configurations from 1 to 84; odd configuration numbers identify the use of the FIFO cache replacement policy; even configuration numbers are used to represent the configurations in which the LRU policy is used.

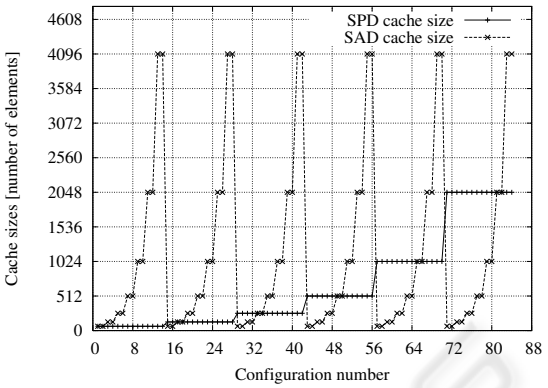

Figure 2: Cache sizes for configurations 1–84.

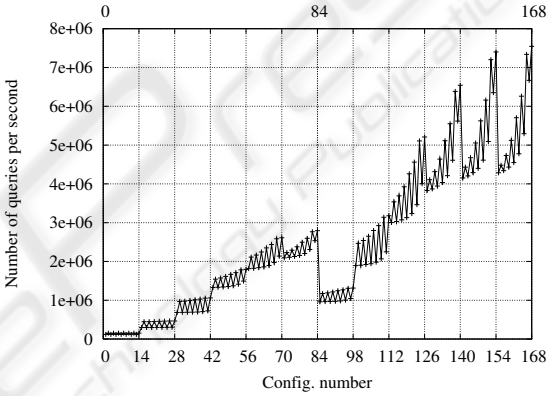

Figure 3: Average number of queries per second.

The average number of queries that can be performed in one second is the inverse of the average query time (i.e., the average time that is required to complete a query). Figure 3 shows the average number of queries that can be performed in one second for the different configurations. Some of the fastest solutions (i.e., the ones adopting the PLUT technique and 1024-2048 elements in the SPD cache) provide the capability to perform from 4 to 7.5 million of queries per second. This speed is good enough to support a total bandwidth of 7-8Gbit/s. If, as previously

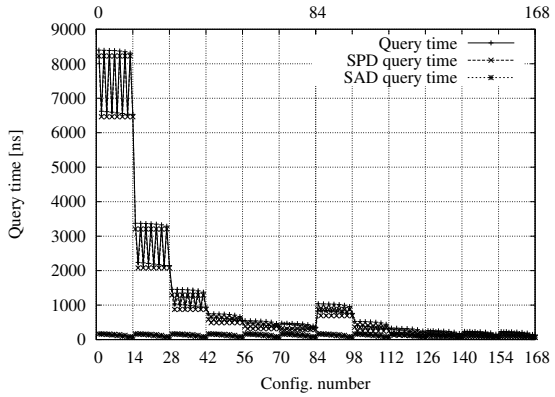

Figure 4: Average global query time and partial query time of the SPD and the SAD.

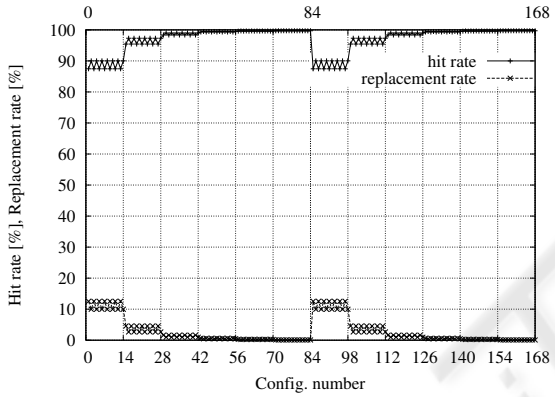

Figure 5: Hit and replacement rates for the SPD cache.

proposed, two database query units are adopted, full duplex communications at 7Gbit/s can be easily supported. Figure 4 shows the average global query time as well as the SPD and SAD average query times. SPD query time gives the major contribution to the total query time for slow configurations. In fast configurations, the query time of both SAD and SPD plays a major role in total query time. The average query time noticeably decreases for increasing sizes of the SPD cache. The adoption of the PLUT technique provides a large reduction of the average query time. The LRU replacement policy provides better results in terms of query time. The difference in performance between the FIFO and the LRU policies is quite noticeable.

Figure 5 shows the hit and the replacement rates of the SPD cache obtained by considering all the different system configurations. This graph helps in explaining the results shown previously: larger SPD caches provide a hit rate close to 100% (99.7% for the 2048-element cache), therefore the SPD query time becomes close to the query time in the cache.

A similar behavior is observed when the SAD cache is considered, larger caches provide best results in terms of performance. Caches larger than 1,024 elements provide better results (the hit rate is of 99.4% for 4096-element caches). The LRU technique gives better results over the FIFO for both the SAD and the SPD caches.

4 FURTHER IMPROVING SPEED: PARALLELIZING QUERIES

When a cache miss occurs, the DB query unit must query the database that is contained in the main memory. This is a time consuming operation that could lead to a long waiting time both for the packet that is under processing and for all the other incoming packets. One natural extension to the architecture presented in this section, is to introduce multithreading: when a query in the main database starts, other queries (one at a time) related to the subsequent packets can be started too. If these queries produce a cache hit, they will be served much faster than in the non-multithreaded system. If they produce another cache miss, their main database query requests will be queued and served after the current one. This technique has the disadvantage of requiring a memory to store the requests under processing and to queue the processing requests. This disadvantage is highly compensated by far better speed performance, especially when queries in main DB are much slower than the ones in cache.

In some cases the multithreaded queries cannot be done: when a query related to a specific SA is already being performed either in the SPD or in the SAD, no new queries related to the same SA should be started. This is due to problems that possible DB and cache modifications may cause. In this case the new request, along with all the subsequent ones, is made to wait until the current query is completed.

4.1 Simulations

The model presented above has been modified to include multithreading. Figure 6 shows a comparison of the number of queries per second that can be performed by the serial and by the parallel units. As shown in the figure, the parallel architecture always outperforms the serial one. The parallel architecture provides the ability to perform up to 11 million queries per second, 50% more than the serial one.

The query time for the SPD and the SAD, along with the global query time, are shown in Figure 7.

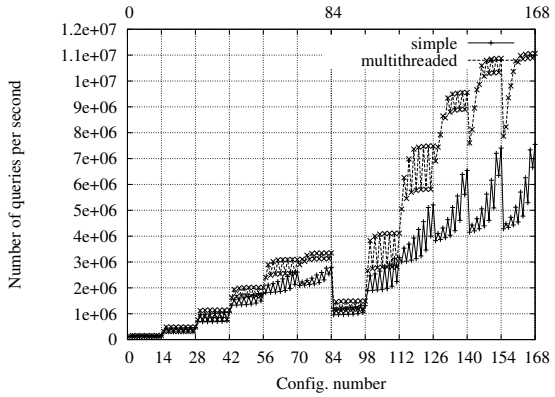

Figure 6: Average number of queries per second.

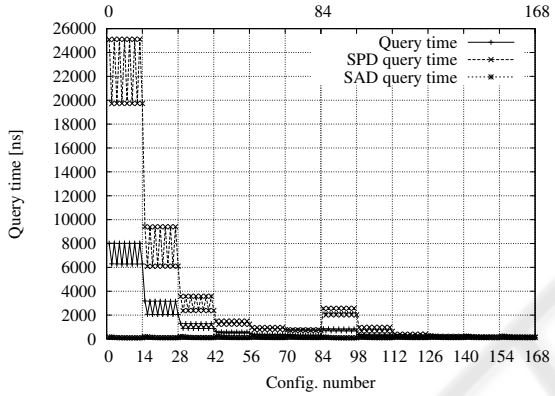

Figure 7: Average number of queries per second.

The global query time is obtained by dividing the total simulation time by the number of processed packets, hence this time is less than the sum of the query times of the SPD and of the SAD. The SPD query time is much greater than in the serial architecture. In fact, in some cases packets need to wait in the input queue before being processed. In any case, the global query time is smaller when the parallel architecture is adopted. In fact, the query time of different packets overlap in this case, thus providing better global performance.

5 OPTIMIZING THE AREA–DELAY PRODUCT

In the previous sections, the proposed architecture has been optimized only for speed. In many cases we may need to optimize the architecture by minimizing some cost functions such as the area–delay product. This approach is similar to the one adopted in (Alberto Ferrante et al., 2005a).

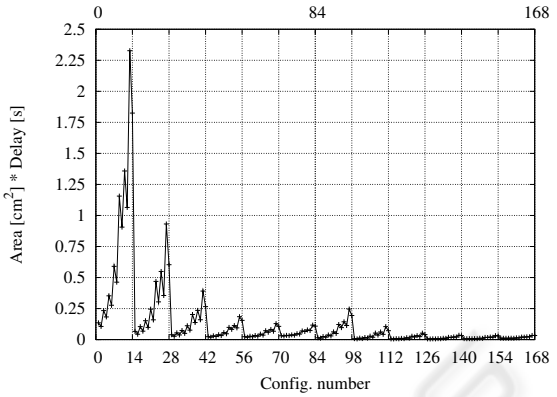

Figure 8: Area–Delay product for the multithreaded architecture.

Figure 8 shows the area–delay product we have obtained for the multithreaded architecture. Only SAD and SPD caches have been considered for area measurements. These values have been computed by means of CACTI (Norm Jouppi et al., nd), by considering a $0.13\mu m$ technology; the obtained sizes (in centimeters square) have been incremented by 50% to accomplish the fact that CAMs are used instead of standard memories. The considered delay (in seconds) is the total simulated time for processing $1,000,000$ datagrams. Our results emphasizes the fact that the configuration number 114 minimizes the area–delay product, in which SPD cache size is 256 and SAD cache size is 64 elements. The replacement policy employed is LRU and the memory lookup techinque is PLUT. This configuration allows to query 6.2 million of queries per second; the fastest configuration allows for 11 million of queries per second. In general, all the configurations with small SAD caches allows to obtain low values for the cost function. In fact, SAD records are bigger than SPD ones, therefore the area of the SAD cache has more influence on the cost function value than the SPD cache area.

Similar results have been obtained for the non-multithreaded version of the system, but in that case we have to consider that the control unit will be less complex than in the multithreaded case.

6 CONCLUSIONS AND FUTURE WORK

In this paper we presented an architecture that provides the ability to query the IPSec databases efficiently. We also developed some simulations to estimate the performance that can be obtained by us-

ing the proposed architecture. An improvement of the architecture, namely the adoption of multithreading, has also been proposed. The multithreaded architecture provides the ability to perform up to 11 million queries per second.

Future work will be put into developing better simulations and to derive more accurate results. Multithreading may be improved to allow out of order processing of the queries. This should allow to further improving the performance when more than one packet belonging to the same SA needs to be processed.

REFERENCES

(n.d.). The Internet Traffic Archive. http://ita.ee.lbl.gov/.

(n.d.). SystemC Official Website. http:/www.systemc.org/.

(n.d.). TCPDUMP Public Repository. http://www.tcpdump.org/.

Alberto Ferrante, Giuseppe Piscopo, and Stefano Scaldaferri (2005a). Application-driven Optimization of VLIW Architectures: a Hardware-Software Approach. In *Real-Time and Embedded Technology Applications*, pages 128–137, San Francisco, CA, USA. IEEE Computer Society.

Alberto Ferrante and Vincenzo Piuri (2007). High-level Architecture of an IPSec-dedicated System on Chip. In *NGI 2007*, Trondheim, Norway. IEEE Press.

Alberto Ferrante, Vincenzo Piuri, and Jeff Owen (2005b). IPSec Hardware Resource Requirements Evaluation. In *NGI 2005*, Rome, Italy. EuroNGI.

Ariga, S., Nagahashi, K., Minami, M., Esaki, H., and Murai, J. (2000). Performance Evaluation of Data Transmission Using IPSec Over IPv6 Networks. In *INET*, Yokohama, Japan.

Feghhi, J. and Feghhi, J. (2001). *Secure Networking with Windows 2000 and Trust Services*. Addison Wesley.

Harkins, D. and Carrell, D. (1998). The Internet Key Exchange (IKE) – RFC2409. IETF RFC.

Henry Hong-Yi Tzeng and Tony Przygienda (1999). On Fast Address-Lookup Algorithms. *IEEE Journal on Selected Areas in Communications*, 17(6):1067–1082.

John Hennessy and Dave Patterson (2002). *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Pub, 3rd edition.

Kent, S. (2005a). IP Authentication Header – RFC4302. IETF RFC.

Kent, S. (2005b). IP Encapsulating Security Payload (ESP) – RFC4303. IETF RFC.

Kent, S. and Atkinson, R. (1998a). IP Authentication Header – RFC2402. IETF RFC.

Kent, S. and Atkinson, R. (1998b). IP Encapsulating Security Payload (ESP) – RFC2406. IETF RFC.

Kent, S. and Atkinson, R. (1998c). Security Architecture for the Internet Protocol – RFC2401. IETF RFC.

Kostas Pagiamtzis (n.d.). CAM Primer. http://www.eecg.toronto.edu/ pagiamt/cam/camintro.html.

Miltchev, S., Ioannidis, S., and Keromytis, A. D. (2002). A Study of the Relative Costs of Network Security Protocols. Monterey, CA. USENIX Annual Technical Program.

Norm Jouppi, Glenn Reinman, Premkishore Shivakumar, and Steve Wilton (n.d.). CACTI. http://research.compaq.com/wrl/people/jouppi/CACTI

Pagiamtzis, K. and Sheikholeslami, A. (2003). Pipelined Match-lines and Hierarchical Search-lines for Low-power Content-addressable Memories. In *IEEE Custom Integrated Circuits Conference*, pages 383–386.

Prashant Chandra, Sridhar Lakshmanamurty, and Raj Yavatkar (2003). Intel Corporation – Intel IXP2400 Network Processor: A Second-Generation Intel NPU. In Crowley, P., Franklin, M. A., Hadimioglu, H., and Onufryk, P. Z., editors, *Network Processor Design*, volume 1, chapter 13, pages 259–275. Morgan Kaufmann.

Shacham, A., Monsour, R., Pereira, R., and Thomas, M. (1998). IP Payload Compression Protocol (IPComp) – RFC2393. IETF RFC.

Yuan, R. and Strayer, W. T. (2001). *Virtual Private Networks*. Addison Wesley.