IEEE Copyright Notice

Introducing QoS mechanisms into the IPsec packet processing

Lars Völker
Institute of Telematics
Universität Karlsruhe (TH)
Karlsruhe, Germany

Marcus Schöller†
NEC Europe Ltd.
Heidelberg, Germany

Martina Zitterbart
Institute of Telematics
Universität Karlsruhe (TH)
Karlsruhe, Germany

***Abstract*— The deployment and use of IPsec has consistently increased in recent years. IPsec is a protocol that allows, besides other things, secure branch offices connectivity and secure VPN access for road warriors. The limitations of IPsec are much better understood today, and efforts to improve IPsec are still underway. One aspect of improvement is the integration of IPsec with other functions and protocols of the network. Quality of Service (QoS) is one example. QoS is used to prioritize demanding traffic like Voice over IP, network control messages, and traffic for other mission-critical systems. QoS can be used to mitigate risks of DoS attacks, ill-behaving hosts, and other attacks by separating traffic classes and treating packets according to the respective class. In order to facilitate all the advantages QoS can offer, an IPsec implementation must not only be superficially changed, but needs thorough modifications or, even better, should be designed with QoS support as an objective. The current IPsec standard does hardly offer any guidance to do this. In this paper, we detail our QoS-capable IPsec and compare it with a widely-used regular IPsec implementation. Furthermore, we show that these QoS extensions prove to be valuable, even in difficult scenarios, e.g. using host CPUs for packet processing.**

I. Introduction

IPsec is the current protocol of choice in the Internet if a flexible and comprehensive security solution is needed. When securely connecting two networks, it allows to tunnel packets from one network to another in a secure fashion. When giving access to Road-Warriors it is able to support all applications based on IP.

The current version of IPsec was defined in [1] and implementations for most current operating systems are available. However, the standard only offers little advice on QoS: it only requires the TOS field in the IP header to be copied during the encapsulation of packets to keep the QoS marking for the following network hops. Unfortunately, the influence of the IPsec packet processing on the packets latency and jitter are not discussed. Therefore, the IPsec developer has to carefully design and integrate an IPsec implementation into the operating system in order to achieve QoS.

Nowadays, networks often transport several different kinds of traffic including real-time and network-control traffic. Traffic generated by applications such as Internet telephony (Voice over IP), video conferencing, SNMP, SSH, telnet, and other mission-critical protocols fall into this category. Regular data traffic often includes HTTP traffic, mail, and other bulk data transfers. Networks transporting traffic mixes need particular methods to ensure that the different traffic classes do not have a negative impact on each other. For instance, backup traffic should not interfere with a video conferencing stream. These goals can be achieved by using Quality of Service (QoS) mechanisms.

Extending QoS-aware networks with IPsec seems to be infeasible, considering that most IPsec implementations were not designed with QoS as an objective, do not differentiate different packets according to their QoS requirements, and do introduce latency due to the packet processing. Such implementations often do not perform very well in QoS scenarios. A simple approach would be to treat IPsec tunnels as virtual wires. However, this does not take into account that the IPsec processing adds latency depending on the packet length, chosen cryptographic algorithms, current load, and of course backlog of the incoming IP queue. Since processing times may vary, additional jitter is introduced. Without means to differentiate between packets, all aggregates are being treated equally bad. We show later in this paper that especially in overload situations the level of jitter is high.

A. Additional advantages

We identified two further aspects of the current IPsec packet processing as a result of the absence of QoS mechanisms:

- protecting IPsec from denial of service attacks
- fair shared VPN access

Implementing QoS support in an IPsec system does not only allow the network to better handle a challenging traffic mix, but it can also minimize the impact of Denial of Service (DoS) attacks and misconfiguration. Since the IPsec processing speed of current CPUs is a magnitude slower than current network technology, IPsec may constitute a bottleneck in the network. This makes IPsec gateways an easy target for DoS attacks. In order to close this window of opportunity for the attackers, packets need to be treated according to their Differentiated Services classes and must not affect other packets adversely. It is obvious that marking packets so that an IPsec system can fulfill the previous requirements is necessary. Treating the packets accordingly makes it possible to limit the influence between packets of different classes, and therefore can help to make DoS attacks against an IPsec system much harder or even impossible. In detail, an IPsec system may restrict the

†Former affiliation: Lancaster University, Lancaster, UK.

amount of resources committed to low priority packets if a lot of high priority packets have to be processed. The scarcest resource, CPU cycles, will preferably be used on high priority packets. Thus important traffic is protected and the amount of bogus traffic is limited.

When using IPsec to secure access connections like 802.11, using QoS mechanisms may improve the fairness between users. Users that do not use the wireless link in a fair way may reduce its quality of service for other users and they might not be able to use applications that depend on low latency (like VoIP) anymore. By integrating QoS mechanisms into the IPsec processing the problem that IPsec packets cannot be distinguished by regular QoS mechanisms can be overcome. Otherwise, IPsec packets could not be dropped nor shaped in order to improve the overall traffic situation.

B. Related Work

Flexible mapping of connections to QoS classes is required for a potent system. The IPsec standard [1] requires the DiffServ Field in the IP header to be copied to the outer IP header during the encapsulation of packets. For IPv6 packets transported over IPv4 IPsec tunnels, the Class field is adapted before inserting it into the IPv4 DiffServ field. Futhermore, the IETF IPsec working group discussed the idea of mapping QoS classes to IPsec Security Associations (SAs). This could allow the IPsec system on the egress site to prioritize packets even if the TOS field got reset in transit.

The motivation for making IPsec QoS-aware is presented in [2]. The authors conclude that latency and loss affect real time traffic over IPsec. [3] describes a scheduler strategy for use in IPsec systems using hardware accelerators and presents some first simulation results. This strategy could be implemented in our system for possible further improvements. An architecture for QoS and the possibility of using the SPI field in IPsec headers for classification is presented in [4].

[5] compares the service quality of VoIP and different security protocols, while [6] discusses performance and quality improvements due to header size optimizations. These changes could possibly complement our solutions.

One of the first commercial products understanding the need for QoS aware IPsec implementations was [7], a hardware-only IPsec solution that supports the prioritization of real time data. The hardware crypto unit has a FIFO queue for best effort traffic an an additional Low Latency Queue (LLQ) for real time traffic. The LLQ is limited to hardware crypto units only and must be used in combination with Class Based Weighted Fair Queueing (CBWFQ).

II. Introducing a specific IPsec implementation

The IPsec Construction Kit (IPseCK) is an IPsec implementation for the Linux operating system with focus on a highly modular and decoupled design. The support of different cryptographic accelerators and processing mechanisms was a main design goal from the very beginning on.

In every IPsec implementation the packet processing follows a simple plan. After matching the packet to a specific Security Association by using the Security Policy Database (SPD), the packet will be sent to a cryptographic processing unit, which in turn encrypts the packet using cryptographic material delivered by the Security Association Database (SAD). Depending on the Security Association an ESP or AH header will be prepended and the Integrity Check Value (ICV) field will be calculated by a cryptographic unit.

The SPD is a (linear) list of rules specifying selectors to match the current IP packet and a target security association for that class of packets. Netfilter and Iptables [8] were chosen to implement the SPD for IPseCK—providing more than the needed matching functionality to IPsec. Additional matching functions can be used to implement IPsec policies for specific scenarios, like the usage of QoS. They allow the IPsec SPD to look at headers and fields that the IPsec standard did not envision. Furthermore, these functions have been tested exhaustively and provide a good basis for the SPD.

Blocking of IP queues is a highly undesireable behavior because IPsec processing is time intensive and the forwarding of non-IPsec packets should be independent of the IPsec packet processing. This constraint resulted in the architecture shown in figure 1 which does not block the processing of non-IPsec packets. The *incoming main queue* and the *outgoing main queue* decouple the Netfilter adapter and IPseCK. Results are presented in chapter V.

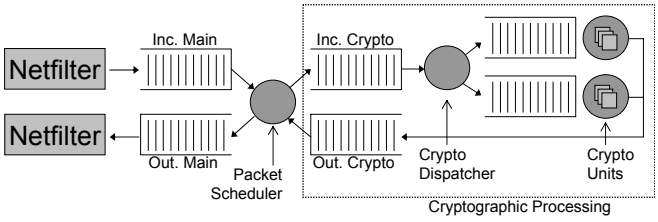

Fig. 1. The structure of IPseCK

The packet scheduler looks up the security association of an IPsec packet in the security association database and forwards the packet and this information to the *Cryptographic Processing* using two similar queues: the *incoming crypto queue* and the *outgoing crypto queue*. Inside the Cryptographic Processing, the *Crypto Dispatcher* is located which distributes packets to one or more cryptographic units (*Crypto Units*). These units implement cryptographic and header processing.

The original IPseCK facilitates a pure push architecture. Packets are pushed through the system to the Crypto Units. Packets are accumulated in the queue in front of the Crypto Unit, while the Crypto Unit is busy processing. These queues are on the right in figure 1.

The Packet Scheduler follows a simple design. In order to be resource preserving it tries to dequeue the Outgoing Crypto queue first and the Incoming Main queue afterwards.[1] Packets that need processing are sent to the Incoming Crypto queue

[1]The order of dequeuing is important. A work preserving system needs a scheduler which prefers the packets whose processing did already start, so the processing of these packets can be finished.

and finished packets are sent to the Outgoing Main queue. The Crypto Units process a packet before returning it to the Packet Scheduler. This allows processing the packet on different units with different algorithms. In order to process a complete Security Association (SA) bundle, a packet has to cycle in the inner loop for each SA. (Most SA bundles have just one SA.) The Packet Scheduler also starts the SAD (Security Association Database) lookup in order to attach the SA meta data to the packet. The meta data includes cryptographic keys, initialization vectors, replay windows, and other data needed to cryptographically process a packet.

The Crypto Dispatchers procedure is slightly more complex since it has to check if a certain Crypto Unit supports the algorithm needed for the processing of the packet. Especially the support of hardware- and software-based packet processing adds to this complexity since hardware is usually called asynchronously while software is mostly called synchronously. With more than one unit in the system, the Crypto Dispatcher also needs to schedule the packets in order to achieve a high throughput. During this procedure, it needs to check the SAD in order to schedule that packet to a certain cryptographic unit.

We used the Linux kernel 2.6 IPsec implementation and the newly introduced IPseCK implementation to perform our QoS experiments. The remainder of the paper shows how we enhanced IPseCK to cope with different QoS requirements. Finally, the results of this implementation are further compared to the Linux kernel implementation.

III. Design of QoS enhancements

Designed and implemented in a modular way, IPseCK can be augmented with QoS support. The goal was to introduce a flexible architecture for QoS. This way, adding of a specific QoS configuration is possible by just loading and configuring QoS components at runtime.

The main classes of such QoS components are behaviors, queues, enqueue behaviors, dequeue behaviors, and dequeue disciplines. A fifo queue, for example, would consist of fifo enqueue, a fifo queue, and a fifo dequeue. The enqueue behavior inserts packets into the queue, and the dequeue behavior gets packets out of the queue, and the fifo queue is used for storage.

Behaviors include classifiers, meters, shapers, and many more. These components meter, change, or redirect packets. Similar to the behaviors are the dequeue disciplines, which represents strategies of choosing packets from queues. They include Round Robin, Weighted Fair Queueing and others. A classifier would select a queue for a packet to be inserted while a dequeue discipline/scheduler would pick the queue which should be dequeued next.

When analyzing the IPseCK design, it is obvious that queues and dispatchers/schedulers can effectively be replaced with QoS modules. Packets are stored in queues and need to be prioritized and reordered. Dispatchers and schedulers make differentiations, which influence timing and order of packets. Further examination revealed that modifications of the Packet Scheduler does not significantly help implementing QoS support in IPsec, whereas the modifications of the Crypto Dispatcher are indispensable, especially if multiple Crypto Units are used. The major reason that the Packet Scheduler cannot significantly contribute in delivering QoS is that its decision are fairly simple and predictable; the decisions are orthogonal to QoS. The Packet Scheduler just checks if packets need more cryptographic processing and puts these packet into the Incoming Crypto queue. The Crypto Dispatcher, on the other hand, has to assign the packet to a suitable crypto unit, so further differentiation between packets needs to be done. It must also be able to prioritize packets to support priority traffic flows. The Crypto Dispatcher can control the length of the queues in front of the Crypto Units, and therefore the waiting time of already committed packets. This determines the minimal latency of a packet, since the already committed packet will be processed first.

Compatible interfaces (section III-A) had to be introduced into IPseCK which was done for the queues (section III-B) and the new Crypto Dispatcher (section III-E). Section III-C explains changes of IPseCK in order to enable the use of QoS components, while section III-D describes the design of the IPsec classifier that is used to make a decision based on the IPsec meta data and IPsec headers of a packet.

A. Connection for QoS modules

In order to allow for a modular and flexible QoS concept, our design is based on the idea of specific attachment hooks. Attachment hooks are used to insert a chain of software components at a given location in the packet processing, thus allowing to modify the packet processing. These modifications cannot only be done at startup time but also at runtime to allow adjustments to the implementation by connecting, disconnecting, and exchanging QoS modules. An administrator could, for example, decide that a new QoS class needs to be integrated into her domain. She could add needed functionality like a different scheduler, a certain queue type, and queueing behaviors, while the system can keep running. Thus, not causing downtime.

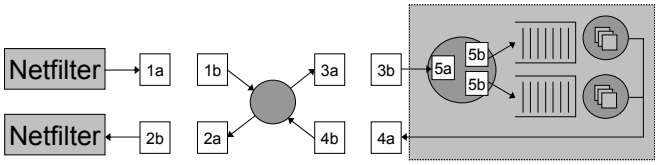

Fig. 2. IPseCK with hooks for QoS modules

Figure 2 presents the hooks, which were added to our IPsec implementation. In the figure hooks are represented by white squares and are differentiated by number. These hooks can be divided into two categories: hooks to replace queues[2] are presented in section III-B and the hooks to replace policy[3] in

[2]The IPsec queues are not necessarily replaced by just simple queues. The common case is to attach functionality, like rate limiting, shaping, classification, and one or more queues at the queue hooks.

[3]These hooks, numbered 5a and 5b are inside the cryptographic scheduler and can be used to attach different scheduling behaviors.

section III-E.

B. Queue modifications

A feature of the original IPseCK is the ability to easily exchange queues. Adding flexibility for future requirements and providing a mechanism to decouple different parts of the packet processing are results of this decision. In order to attach QoS components, the queue interface has to be changed by providing two attachment hooks per queue: one for enqueuing and one for dequeuing. By attaching components for classification, enqueuing policy, dequeuing policy, and storage, a virtual QoS queue emerges that can handle traffic according to its traffic class.

A sample usage of the new queue hooks is detailed in figure 3. The simple fifo queue is attached to the hooks of the first queue (labeled with 1a and 1b). This is basically the simplest queue configuration and does hardly differ in functionality to the former Incoming Main Queue, as shown in figure 1. The only important difference is the limitation of queue length, which is presented in section III-C.

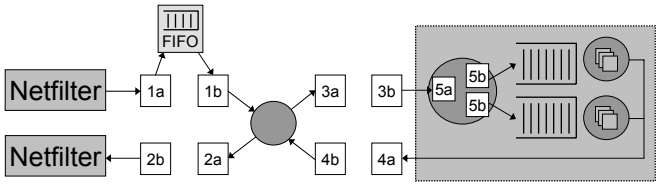

Fig. 3. A simple queue configuration

The creation of the queue hooks requires additional changes: a function to drop packets and mechanisms to limit the length of a queue, which are needed for the changes proposed in section III-C. The QoS modules cannot drop packets themselves: packet state is held by IPseCK and cannot be accessed by the QoS modules. Therefore, the ability to drop is provided by introducing a new primitive, that QoS components can send to attachment hooks. When receiving a packet marked with this drop primitive, the hook will drop the packet for the component. The queue interface has to be changed to limit the length of the queues. A length function has to be created in addition to a blocking call for queuing and dequeuing packets. These functions permit blocking and non blocking packet operations which allow for a very flexible implementation.

C. Rate Limitation

One of the modification's major goals was to limit the latency of IPsec packets. The latency of an IPsec packet is basically determined by two factors: the start time of packet processing and the actual duration of processing. Changes that can be made to influence the packets processing time are limited and are discussed in section III-E. However, the start time of the packet processing can be influenced. The latency introduced by this is mainly the sum of the packet processing times of packets which are being processed before the new packet.

Two different techniques are combined to lower this latency. The obvious first solution is to limit the length of the queues used in the system and only admit packet into the system as long as queue space is left. This effectively introduces an upper bound of latency and is explained in this section.

The other technique is to rearrange the packet order to reduce latency of one packet by delaying another. Such a decision will be made in most cases by scheduling resources of different traffic classes/flows. Section III-E gives further details on this.

Reducing jitter and latency of the queues cannot be achieved by simply limiting the length of the used queues. In fact, rate limitation has to be applied to every traffic aggregate or class separately. To achieve this, the incoming queues have to be replaced by a length limited queue for each aggregate or class. Also a classifier and dispatcher are needed to assign packets to their appropriate queue or drop them if this queue is already full. This access control mechanism allows the system to limit the packets it accepts. Using length limited queues instead of just a simple static token bucket allows the system to automatically self-tune the rate of packets, which are accepted for a certain aggregate.

An implementation option would be to build this rate limitation outside the IPsec system, instead of the incoming packet processing. While this could be sufficient in simple scenarios, it is obvious that in complex system using two different cryptographic units this can never work effectively, because no information about the capabilities of different cryptographic units and their current work load are known. Furthermore, the classification must be based on IPsec parameters not available outside the system, for example matching the dynamically assigned IPsec Security Parameter Index to a QoS class can not be done without information of the IPsec system.

D. The IPsec Packet Classifier

To enhance IPseCK with QoS components, an option is needed to differentiate between packets based on IPsec information. Regular packets are classified considering fields in the header, but for the classification of IPsec packets certain headers might not be available because they are encrypted and other information are not stored in headers at all. The only traditional source of information would be the Differentiated Service Code Point field, not allowing to differentiated on the IPsec processing properties, like cryptographic algorithms used. In order to gain enough data to classify packets, one can lookup the Security Association of a packet and the attached data in the SAD[4]. This is done by the IPsec classifier.

The new IPsec classifier allows to classify packets according to matching SAD entries. The important entries are the algorithms and keys used, the index of the SA in the bundle, the security parameter index (SPI), and the kind of IPsec headers used.

This classifier can not only work on incoming packets, which are IPsec protected, but also on outbound packets

[4]The Security Association Database is used to store cryptographic material and other meta data for every IPsec security association.

that will be protected with IPsec by this system. A uniform classification mechanism of IPsec packets is the result.

Adding an IPsec classifier to the QoS components allows for mapping of QoS classes/aggregates even if packets are protected. It also allows for a fine-tuned processing of IPsec packets based on QoS requirements and, if used in conjunction with a special dispatcher, as presented in section III-E, it even makes it possible to map packets to cryptographic units based on QoS requirements.

E. Modular Crypto Dispatcher

In order to allow modifications to the Crypto Dispatcher, a new dispatcher was built. It internally implements an attachment hook through which packets enter as well as a hook to pass packets to each crypto unit. It is possible to connect modules to the hooks and implement specific functionality. Thus, one can build a dispatcher to meet specific needs.

The dispatcher relies heavily on the classifier implemented for IPseCK. Without the classifier (see Figure 5 for an example) it would not be possible to assign packets based on the capabilities of the crypto units, thus packets might be assigned that could not be processed.

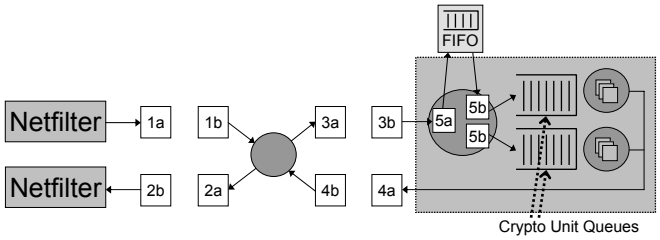

Fig. 4. A simple dispatcher configuration

A major problem of the dispatcher construction was the necessity of changing the invocation. The original design pushed the packets along the processing path into the crypto units. This behavior is good for reaching a high throughput, but it reacts adverse during DoS attacks, for example on only one aggregate. Furthermore, it adds latency depending on the number of packets already committed to the processing queue. In order to mitigate the risk of an attack and to better schedule QoS packets, it is crucial to limit the number of packets already assigned to a Crypto Unit and hereby allow for better reactivity. This allows expedited traffic packets to outrun packets with lower priority. In order to implement these changes, the crypto unit queues (see figure 4) were equipped with high and low water marks that can trigger Crypto Scheduler events. A scheduler can push packets in these queues if wanted and can also wait for a queue to signal that it falls below a defined lower threshold.

Figure 4 shows a simple Crypto Dispatcher configuration that works without QoS. A more complex configuration is shown in figure 5. Here the dispatcher and the classifier are combined and service two different aggregates. In order to distinguish between two aggregates an IPsec classifier is connected to hook 5a and distributes the packets in two queues. Packets will be eventually dequeued using hook 5b and the Round Robin scheduler in front of this hook will be instructed to dequeue the queues in a round robin manner. This configuration is the smallest possible configuration to implement differentiation of service classes.

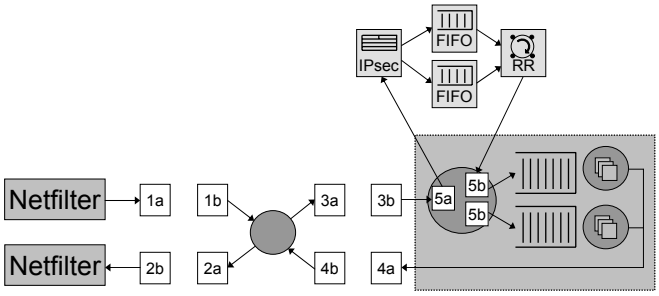

Fig. 5. A dispatcher configuration using the IPsec Classifier

IV. Implementation

In the previous chapter, a design to dynamically add QoS components to an IPsec implementation was presented. Since a framework of QoS components exists in form of KIDS [9], these components were reused as much as possible.

Implementing the presented changes in IPseCK required additional modifications. Components need to be registered at hooks and deregistered if no longer needed. Therefore, the queues were replaced by a special fifo queue, which allows components to replace it by register themselves at the queue's hooks. This allows the system to reconfigure the attached components without interruption of IPsec processing.

The special dispatcher to be used with QoS components needs a different call semantic. The original design pushed packets into the dispatcher, which in turn sent the packets to the crypto unit queues. The new QoS dispatcher needs a different approach; instead of pushing into the queues, the cryptographic units now need to send a pull request to the dispatcher. This was realized by creating a new callback interface.

Several changes have been made to achieve the three benefits:

- QoS support
- more resistance to DoS attacks
- fair share between users

The configurations for the benefits do differ in detail.

For these benefits it is important that a limited number of aggregates is differentiated and treated accordingly. The incoming main queue and the incoming crypto queue have to differentiate between the traffic classes and must therefore use classifiers, different queues, and schedulers. The configuration of the crypto dispatcher is based on two dimensions: the traffic aggregates and the number of the crypto units as well as their configuration. For IPsec systems at the domain edge, rate limitation and/or shaping needs to be included in addition.

User fairness is especially important because an IPsec concentrator is a limited resource. When using IPsec to

secure wireless network access, achieving user fairness in the concentrator will also achieve fairness for the wireless link. Instead of adding QoS components for each user it is possible to use components with support for different user profiles.

More DoS resistance is already achieved by implementing just the decoupling of IPsec and non-IPsec processing. This will be presented in chapter V. Adding the mechanisms outlined for user fairness more DoS resistance is achieved. IPsec users can neither influence non-IPsec users nor other IPsec users.

A. A sample QoS configuration

In chapter V-B results with different IPsec connections are presented showing how aggregates can be processed independently. The used configuration is very simple and basically the smallest possible configuration for two aggregates. The aggregates used are Expedited Forwarding (EF) and Best Effort (BE). The rate of the BE aggregate is limited inside the IPsec system, while the packet rate of EF is not limited at all.

Both incoming queues (incoming main queue and incoming crypto queue) and the crypto dispatcher must differentiate between both aggregates and need a classifier each. In addition fifo queues for storage and the simple round robin scheduler were used. Alternatively a weighted fair queuing or strict priority scheduler could be used to select packets. This configuration is represented by figure 6, which only leaves the legacy FIFO queues at the outgoing queues 2 and 4.

V. Performance Results

Performance and stress tests were conducted to demonstrate the effects of the proposed improvements. The goal was to confirm that the improvements solve the problems as stated before.

The tests were conducted on a set of Linux systems. These are equipped with 2.8 GHz Pentium 4 CPUs and 2 GB of RAM each. The onboard Gigabit network interfaces were used to connect the systems using a Gigabit Switch. Each system is fast enough to generate about 50–60 Mbit/s of IPsec traffic[5].

A. IPsec's influence on non-IPsec packets

The first stress test focuses on the dependence of IPsec and regular network processing. In section II, we detailed that the major goal of the IPseCK system is a decoupled implementation. The first stress test is supposed to compare the two approaches. On a well-designed system, the high IPsec load would not influence the regular packet processing, hence the measured round-trip time would be the same as without IPsec.

The setup for the test consists of four systems generating 25Mbit/s of IPsec traffic each, on the right of figure 7. This traffic is declared as best-effort traffic and does not have latency requirements. Another system is receiving the combined load of 100Mbit/s, this is the IPsec gateway in the top of the figure. Since each system is only able to handle about 50–60Mbit/s of traffic, the setup results in an overload situation of this system[6], which had to drop packets in such a situation. The probing system is used to test the reactivity of the target system, it is depcited on the left side of figure 7. Probing was done by sending unprotected ICMP packets to the target system, these packets were considered to have a low latency requirement. We measured using the Linux 2.6 kernel IPsec implementation and our IPseCK implementation for the different IPsec nodes.

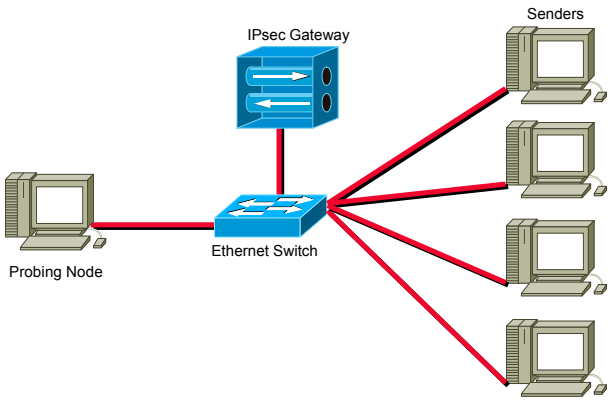

Fig. 7. The setup of the stress tests

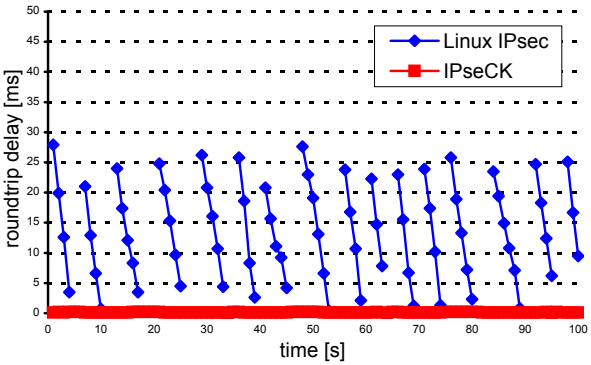

Fig. 8. Stress test with non-IPsec probes

In figure 8 the round trip times of 100 consecutive probes are displayed. The dots between 1 and 30 ms are the results of the test using the Linux 2.6 IPsec implementation. The squares at the lower end of the graph represent the IPseCK implementation's results, which have a much lower latency, no loss, and just a usual amount of jitter. Testing results are summarized in table I. Looking at the figure and the table it is rather obvious that some packet loss and a rather high amount of jitter occured when using this system.

The Linux 2.6 implementation cannot cope with the high load of the IPsec system. The system does prioritize neither

[5]IPsec ESP with 3DES encryption and HMAC-MD5-96 for authentication was used.

[6]Note that a Gigabit Ethernet network was used in these tests, making sure that the network does not act as a bottleneck.

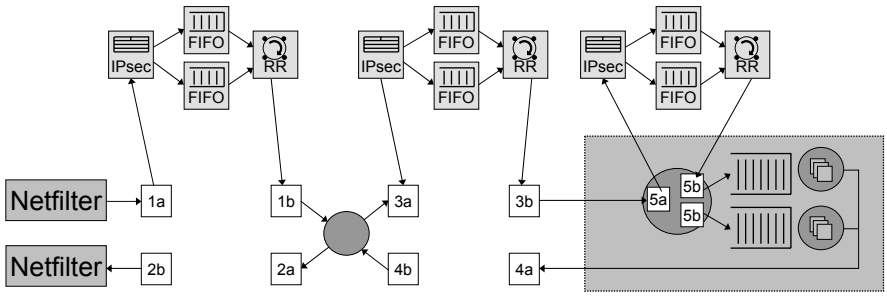

Fig. 6. The configuration used in the benchmarks.

	RTT min.	RTT avg.	RTT max.	loss
Linux 2.6	0.2 ms	13.8 ms	28.0 ms	29%
IPseCK	0.0 ms	0.1 ms	0.2 ms	0%

TABLE I
BENCHMARK WITH NON-IPSEC PROBES

IPsec nor regular traffic and does not decouple the two. This means that IPsec traffic can have a negative impact on regular traffic and vice versa. Especially the high amount of jitter is a result of the queuing strategy used in the kernel, as clearly seen in figure 8. As soon as the incoming packet queue of the network stack is full, all incoming packets will be dropped until this queue is empty again. In the mean time the queue will be emptied by processing the packets which are still in the queue.

Just the contrary are the results of the IPseCK implementation. The decoupling of the non-ipsec and ipsec packets allows IPseCK to achieve very good results in this scenario. The latency, jitter, and loss are all close to the possible minimum values. Best effort IPsec packets do not influence high priority non-IPsec packets.

The results show that a decoupled system can handle the combination of high IPsec load and traffic much better. While the IPseCK system shows the desired behavior, the Linux 2.6 implementation disappoints in this scenario.

B. IPsec's influence on IPsec packets

IPsec must be able to differentiate between different IPsec connections and between different traffic streams within a IPsec connection if QoS and IPsec are to be successfully combined. The second stress test focuses on the handling of different traffic aggregates inside IPsec. This should give us feedback if the modifications to IPsec allow us to offer QoS inside IPsec.

The setup for this stress test is equivalent to the setup described in section V-A. The only difference between both setups is that this test uses IPsec protected probe messages. The probe messages are assigned to an IPsec protected expedited effort (EF) QoS aggregate and the load traffic to the IPsec best effort aggregate. This means that the probe messages have a higher priority than the load traffic.

A perfect system should show minimal latency, loss, and jitter for the probe aggregate by preferring this traffic over the other traffic. Below are results of the Linux 2.6 kernel IPsec, our IPseCK, and IPseCK using the required QoS enhancements.

Figure 9 shows the round trip times of the test probes for each IPsec implementation. The Linux 2.6 IPsec still produces a high amount of jitter, loss, and delay. IPseCK also drops almost 50% of the probes. Its jitter is very low, but the average delay is higher than the delay of Linux 2.6 IPsec. By reducing the queue size in the IPsec system or similar tuning, the latency could be reduced, but this would still not lead to acceptable results, because loss would still occur.

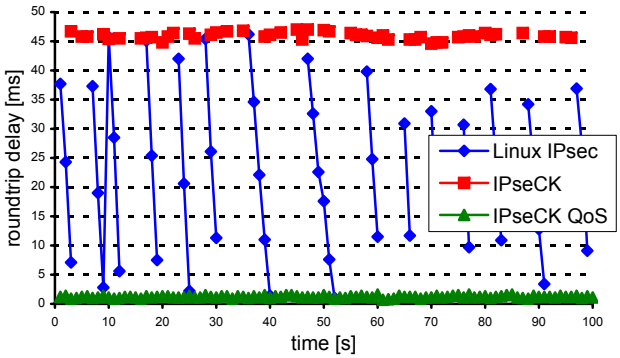

Fig. 9. Stress test with IPsec probes

Only the IPseCK with QoS enhancements produces satisfying results. Very low delay, jitter, and loss are encountered. In table II, the exact results are presented. These show that, with the QoS enhancements, the loss was reduced to 0%, which was the primary goal. In average the RTT delay was reduced to only 1.1 ms.

	RTT min.	RTT avg.	RTT max.	loss
Linux 2.6	1.0 ms	22.8 ms	46.2 ms	52%
IPseCK	44.6 ms	45.9 ms	47.0 ms	47%
IPseCK QoS	0.7 ms	1.1 ms	1.6 ms	0%

TABLE II
BENCHMARK WITH IPSEC PROBES

The results of this stress test demonstrate that, by using QoS mechanisms, a good handling of different traffic aggregates can be achieved even when using only software-based cryptography. This enables the required flexibility for further applications and reduces the impact of QoS attacks.

VI. Applying the results to other implementations

Our results conclude that QoS is achievable with IPsec if certain conditions are met. A decoupled and flexible IPsec implementation can much easier be modified to support QoS-enabled processing than others. For more static implementations deeper modifications have to be done, which will be detailed in this section.

Once considering that the cryptographic processing takes up more time than the other processing steps, like classification, it is important to optimize the decision of packet assignments to the cryptographic processing. To do this, classification and structured queuing is needed. Furthermore, the packets need to be scheduled to the processing. Different strategies could be used, a simple priority-based scheduling already achieves good results in many scenarios.

When an implementation counters an overload situation, packets need to be dropped so the system does not run out of queue space. The strategy on how to drop packets is an important matter. Some implementations, e.g. the Linux 2.6 IPsec, drop all packets until the incoming queue is empty. This strategy leads to a burst of lost packets and large jitter in overload situations. Applications like Video Streaming and VoIP might prefer to have a more distributed drop of packets, since the used multimedia codecs cope with such situation much better.

More important than the drop strategy is the place in the processing to drop packets. The earlier a packet is dropped, the less work was wasted on the dropped packet. But it is critical to not drop packets before classification took place, since important traffic should not be dropped as much as best effort traffic.

An IPsec implementation should classify packets at the beginning of the IPsec processing. This should be combined with rate limitation. The decoupling of IPsec and non-IPsec processing leads to better DoS-resistance—a property a good network stack surely tries to achieve.

VII. Conclusion and Outlook

The standard Linux 2.6 Kernel IPsec implementation has an unacceptable handling of stress situations. A simple flood of IPsec packets can render the systems useless, since network control traffic cannot reach the system anymore even if no IPsec protection for these packets is used.

A better design decouples the processing of IPsec and regular traffic and allows for a balance between both. An implementation following such principles—called IPseCK—is being presented here. Its capability to better cope with DoS attacks and demanding applications, especially if QoS support is used, is being shown. These results show that it is possible to use demanding applications such as VoIP together with IPsec.

At least the decoupling of IPsec and non-IPsec processing should be implemented in IPsec implementations. Additional QoS support is definitely a good improvement for IPsec implementations.

It is questionable whether a CPU-intense task as IPsec packet processing should be synchronously coupled with network processing. We demonstrate that this can lead to massive problems and this does not even depend on the IPsec usage scenario. Therefore, we cannot recommend to use such coupled IPsec implementation in productive systems, since the possible vulnerability is quite massive and makes it easy for attackers to mount a DoS attack on the system.

Further work includes a more complete QoS configuration, including the Assured Forwarding (AF) class and Random Early Detection (RED). Our current solution is focused on transporting the packets, while signaling and setting up the QoS mechanisms for IPsec is left aside; Domain resource management would be a further improvement.

It should also be investigated, whether the local generation of IPsec packets makes a difference to forwarding and encapsulating packets by IPsec.

Acknowledgment

The authors would like to thank Roland Bless, Michael Conrad, Stefan Mink, and Christoph Sorge for valuable input and contributions.

References

[1] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301 (Standard), Dec. 2005.

[2] J. Perez, V. Zarate, A. Montes, and C. Garcia, "Quality of Service Analysis of IPSec VPNs for Voice and Video Traffic," in *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, 19-25 Feb. 2006, pp. 43–43.

[3] A. Ferrante, V. Piuri, and F. Castanier, "A QoS-enabled packet scheduling algorithm for IPSec multi-accelerator based systems," in *CF '05: Proceedings of the 2nd conference on Computing frontiers*. New York, NY, USA: ACM Press, 2005, pp. 221–229.

[4] V. Fineberg, "A practical architecture for implementing end-to-end QoS in an IP network," *Communications Magazine, IEEE*, vol. 40, no. 1, pp. 122–130, Jan. 2002.

[5] H. Xiao and P. Zarrella, "Quality effects of wireless VoIP using security solutions," in *Military Communications Conference, 2004. MILCOM 2004. IEEE*, vol. 3, 31 Oct.-3 Nov. 2004, pp. 1352–1357Vol.3.

[6] R. Barbieri, D. Bruschi, and E. Rosti, "Voice over IPsec: analysis and solutions," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, 9-13 Dec. 2002, pp. 261–270.

[7] Cisco, "Voice and Video Enabled IPSec VPN (V3PN) Solution Reference Network Design," http://www.cisco.com/application/pdf/en/us/guest/netsol/ns241/c649/ccmigration_09186a00801ea79c.pdf, Jan. 2004.

[8] Netfilter-Group, "The netfilter/iptables project homepage," Website, http://www.netfilter.org/.

[9] K. Wehrle, "An Open Architecture for Evaluating Arbitrary Quality of Service Mechanisms in Software Routers," *Proceedings of IEEE International Conference on Networking (ICN 2001), Colmar, France. Springer, Juli*, 2001.

[10] L. Berger and T. OMalley, "RSVP Extensions for IPsec Data Flows," RFC 2207 (Standard), Sept. 1997.