Failure preventive mechanism for IPsec gateways

Daniel Palomares, Daniel Migault, Maryline Laurent

Failure Preventive Mechanism for IPsec Gateways

Daniel Palomares*, Daniel Migault*, Maryline Laurent†
*France Telecom, {firstname.lastname}@orange.com
†Mines-Télécom, Télécom SudParis, CNRS Samovar UMR 5157, maryline.laurent@telecom-sudparis.eu

***Abstract*—Operators are mainly using IPsec Virtual Private Networks (VPNs) to extend a security domain over untrusted networks. A VPN is usually established when an End-User (EU) and a Security Gateway (SG) negotiate security associations (SA). For a better QoS, the SGs are geographically distributed so they are as close as possible to EU. As such, the higher is the level of responsibility of the SG, the higher is the risk to be overloaded and to break down.This paper presents a mechanism for extracting and reinstalling security associations as well as a mechanism to transfer a given IPsec traffic from one SG to another. We also propose an additional mechanism for solving the mis-synchronization of IPsec anti-replay counters and IKEv2 Messages ID counters. Finally some performance measurements are provided in terms of delays, and packet loss, and prove feasibility of the approach. Results obtained through real implementation showed that the system time to extract an IKEv2/IPsec session is in a range of $5ms$ up to $15ms$ whereas the system time to restore an IKEv2/IPsec session can take $2ms$ up to $22ms$.**

***Index Terms*—IPsec, IKEv2, Security Gateway Handover, IPsec Clustering, Failure-Preventive, QoS.**

I. Introduction

IPsec is a security framework at the network layer allowing IP and upper-layer protocols to benefit from encrypted and authenticated communications. They are mostly used tu secure peer-to-gateway or gateway-to-gateway communications, also called VPN (Virutal Private Networks). When providing VPN services based on IPsec, an End-User (EU) commonly makes use of the IKEv2 protocol [1] to negotiate IKEv2 security associations (IKE_SAs) and further IPsec security associations (IPsec_SAs) towards a security gateway (SG). In order to reduce delays and improve communications, the path between an EU and the SG should be as short as possible. As such, operators may rely on geographically distributed pool of SGs to provide a high QoS. On the other hand, an operator that offers fail-over capacities should be able to transfer VPN sessions from one SG to another. Motivation may be to spread the workload among different SGs, as well as to provide high availability features in case a SG fails. The scenario we are considering is an EU that sets up a VPN towards an SG (E.g. SG1) which gives access to a trusted network. The EU is first authenticated by using IKEv2 protocol with either pre-shared keys, EAP-AKA, EAP-SIM, certificates, etc. Then, further IKEv2 exchanges allow to set up the VPN (i.e. IKE_SAs and IPsec_SAs). At a given time, SG1 is overloaded, whereas some other SGs are not (E.g. SG2). The operator wants to move this VPN session from SG1 towards SG2, so it transfers all the IKEv2/IPsec context. The main goal of transferring all IKEv2/IPsec context is to avoid re-authentication against SG2. In fact, re-authentication would increase signaling and would interrupt the communication which could be critical for real-time based applications. Note that if SG1 and SG2 do not have the same IP address, then the EU MUST also update its IKE_SAs and IPsec_SAs. Transferring the IKEv2/IPsec context introduces the following constraints: (1) identification of the IKEv2/IPsec parameters so it can be extracted and reinstalled into another SG. Notice that some parameters of the context requires perfect timing, which is much more difficult to get than the context itself, (2) two gateways with different IP addresses bring complexity as the hosts should consider the management of their tunnel and their SAs (mobility aspect), as well as the distance between SGs introduces network delays.

Section II describes details of an IKEv2/IPsec state. Section III introduces a proposed mechanism to perform IPsec Context Transfers (referred to as IPsec-CXT throughout this paper), a description of **GET** and **PUT** functions and a step-by-step description of an IPsec-CXT. Finally, sections IV and V describe our implementation as well as the testbed results.

II. IKEv2/IPsec Context Detailed Description

The IKEv2/IPsec context contains all the information negotiated through IKEv2 protocol as well as all the information contained in the IPsec databases. Most of the current operating systems implement the SAD and SPD in the kernel while the PAD is mostly a database that exists in the userland space. The SAD, SPD and PAD are defined as follows:

- **Security Association Database SAD**: contains all the information related to each unidirectional IPsec_SA (spi, concerned protocols, SA's lifetime, algorithms, keys, etc.).
- **Security Policy Database SPD**: defines how the packets that match the IPsec policies are handled by the device: PROTECTed, BYPASS or DISCARD IPsec.
- **Peer Authorization Database**:links the SPD with the key management protocol (i.e. IKEv2). This database determines if an identity is allowed or not to establish a VPN with the device.

The Security Associations (SAs) contained in the SAD can be configured manually, but this is obviously not scalable when big amounts of IPsec connections are established with different devices. Thus, IKEv2 (see [1]) is the protocol that sets a secure channel (a.k.a. IKE_SA) between two end-points in order to negotiate and continuously update all the IPsec related information (a.k.a. CHILD_SA or IPsec_SA) between them. First, during IKEv2 initial exchanges (known

as *Phase 1*), one of the end-points (INITIATOR) triggers the communication by sending an IKE_INIT request. The other end-point (RESPONDER) replies back with an IKE_INIT response. At this point, both nodes have a shared secret to perform encryption and integrity protection for further IKEv2 exchanges, and have agreed on the following parameters of their IKE_SA:

- *Cryptographic algorithms*: algorithms to protect further IKE exchanges, a Diffie-Hellman Group and a pseudo-random function.
- *SKEYSEED*: secret key from which all keys are derived for that IKE_SA (SKe encryption key to ensure confidentiality, SKa authentication key to ensure integrity and SKd derivation key master secret that will be used to compute further CHILD_SAs keys).
- *IKE_SPI*: IKE_SPI stands for IKE Security Parameter Index. It uniquely identifies an IKE_SA.
- *Lifetime*: duration of an IKE_SAs.
- *Nonces*: Ni (INITIATOR nonce) and Nr (RESPONDER nonce). These are randomly generated values to reinforce the security (freshness to the key materials).
- *Message ID counters*: the ID counters provide anti-replay for IKEv2 exchanges by increasing the ID counter by one for every emitted IKEv2 message.
- *IKEv2 window size*: if the window size has a value of "N", it implies that there can be N un-acknowledged IKEv2 requests at any given time during communication.

During the *Phase 2* of IKEv2, the INITIATOR sends a IKE_AUTH request and the RESPONDER replies with an IKE_AUTH response. Now, the nodes consult their PAD in order to authenticate each other. As mentioned, the PAD determines if the identity of a given node is allowed or not to establish a CHILD_SA. The PAD is composed of the following information:

- *Identifiers*: IDi (INITIATOR ID) and IDr (RESPONDER ID). Usually an IPv4/IPv6 address, a fully-qualified domain name, an email address,etc.
- *Authentication Method*: pre-shared key, EAP, certificates, RSA, etc.

The establishment of the CHILD_SA (negotiation of parameters stored in the SAD and SPD) is piggybacked during the IKE_AUTH exchange. It is done just once the authentication and the authorization are performed. When *Phase 2* is done, both nodes agree on the following parameters of their CHILD_SAs: **Concerning the SAD**

- *CHILD_SA SPI*: a 32 bits unique identifier of the CHILD_SA.
- *IP addresses*: source/destination IP address of the IKEv2 compliant nodes.
- *IPsec Protocol*: AH (Authentication Header) or ESP (Encapsulating Security Payload).
- *Sequence number counter*: value to control every incoming/outgoing IP packet protected with IPsec, preventing replay or unauthorized re-injection of already processed IPsec traffic.
- *Anti-replay window size N*: any packet with the sequence number $X + N$ is discarded, where X is the awaited sequence number.
- ESP/AH information: Encryption and/or authentication algorithms, keys, initialization values, key lifetimes.
- Lifetime: time interval or byte count after which a SA must be replaced with a new SA (and new SPI).
- IPsec Protocol Mode: tunnel or transport mode.
- Path MTU: maximum size of an IPsec packet that can be transmitted without fragmentation.

Concerning the SPD

- *IPsec Protocol Mode*: tunnel or transport mode.
- *Header's IP Address*: source/destination IP addresses of the IKEv2 compliant nodes.
- *Source/Destination IP addresses of the communications*: if transport mode is being used, these addresses are the same as those used for routing purposes, whereas in tunnel mode these IPs concern the end-points of the communications (which could be an internal IP address of a protected subnet).
- *Upperspec*: upper-layer protected protocol.
- *Policy rules*: DENY, BYPASS or PROTECT the targeted traffic.

III. Protocol Description

We propose a framework (fig. 1) to dynamically transfer an IKEv2/IPsec context from one SG to another. The transfer between two SGs is performed with CXTP protocol (see [2]). In order to improve the IPsec-CXT, we defined two operations: **GET** and **PUT**. **GET** extracts the IKEv2/IPsec context for a given tunnel, whereas **PUT** makes installation of it. Details of an IKEv2/IPsec state are also described in [3].

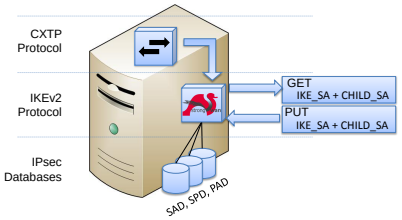


Fig. 1. Schema of an IKEv2/IPsec Context Transfer Mechanism in a Gateway

A. Context Transfer Protocol (CXTP)

The IETF's experimental protocol, CXTP [2], enables transferring contexts for various services (E.g. QoS, Security, Authentication, etc.) between different SGs. A Context Transfer (CT) can be either *Mobile Controlled* (initiated by End-Users) or *Network Controlled* (initiated by the newly active SG **nSG** or the previously active SG **pSG**). Once the CT is activated, it may happen that one of the end-points already knows towards which destination the context will be transferred. This case is known as **predictive** mode, and it is the most favorable scenario in terms of packet loss and performance, mostly because the context transfer takes place before the handover is

done. On the other hand, if the context transfer occurs abruptly, it is known as **reactive** mode. Obviously this mode is less beneficial for an IKEv2/IPsec connection in terms of packet loss and performance.

B. *Functions Added:* **GET** *and* **PUT**

We implemented two functions in order to successfully recover a whole IKEv2/IPsec context. By now, we **GET** a context and store it into a plain text file. We also implemented a function to **PUT** (re-install) an IKEv2/IPsec context into a SG directly from the previously stored plain text file. See figure 1 to observe the graphical representation. Both **GET** and **PUT** functions are implemented in *strongSwan*, which communicate with the kernel through the Netlink XFRM API. This API has recently been modified in order to solve issues concerning the IPsec replay sequence counters synchronization. Netlink XFRM allows to modify these counters at any time in order to synchronize them. This feature is very useful when clustering security gateways. On the other hand, the lookup of the IKE_SA and CHILD_SA inside *strongSwan* is done through its connection name (we refer to it as "`<conn-name>`" in the examples): the command `ipsec get <conn-name>` performs **GET** whereas the command `ipsec put <conn-name>` performs **PUT**.

C. *RFC5685 - Redirect Mechanism*

The RFC5685 describes a method (see VI and [4]) that allows to redirect an IKEv2/IPsec session from a previously active SG towards a newly active SG. This mechanism does not pre-authenticate an EU against the newly active SG, actually the EU needs to renegotiate the cryptographic information from scratch. Our proposal aims to combine this redirect mechanism with the transfer of the IKEv2/IPsec context, therefore the EU does not need to re-authenticate its session. The step number 5 in figure 2 corresponds to the IKEv2 exchanges concerning redirect mechanism during an active session.

D. *Step-by-Step IKEv2/IPsec Context Transfer*

This section presents the sequence of steps of an IPsec-CXT using predictive mode with CXTP. Figure 2 represents all the steps to perform :

1) The EU establishes a tunnel with SG1 and indicates support of redirect mechanism.
2) A trigger incident happens (E.g. SG1 overloaded). The SG1 initiates IPsec-CXT and sends a CTD message (wihch contains IKE_SA+SAD+SPD+PAD information).
3) SG2 receives the context and performs **PUT**.
4) SG2 acknowledges the IPsec-CXT and sends *Context Transfer Data Received* CTDR message to SG1. SG1 sends a REDIRECT message to EU. REDIRECT tells to EU that the SG1 is moving from @SG1 to @SG2. The [NO_REAUTH_IPSEC_CXT_TRANSMITTED] notify payload advises the EU that its session is already migrated to the newly active SG. Finally, the [IPSEC_SYN] notify payload synchronizes both the message ID and the IPsec replay counters.
5) EU receives the REDIRECT request. [NO_REAUTH_IPSEC_CXT_TRANSMITTED] notify payload confirms the IPsec-CXT from @SG1 to @SG2. EU updates its IKE_SA and CHILD_SAs and the IKEv2/IPsec counters are synchronized through [IPSEC_SYN] payload.
6) Finally, EU performs the handover and acknowledges with an INFORMATIONAL response to SG1.

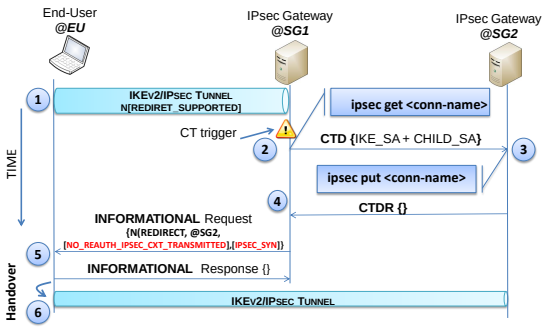


Fig. 2. Exchanges during an IKEv2/IPsec Context Transfer

IV. Implementation - StrongSwan 4.5.0

Our implementation is based on *StrongSwan 4.5.0* [5], a widely used IKEv2 open-source implementation for linux environments. A daemon called **starter** initializes the IPsec framework and launches the different daemons defined through configuration files (i.e. *ipsec.conf* where parameters are stored). The daemon responsible of IKEv2 is called **charon**. It intercepts the IP packets at the IP stack and applies the security policies (whether to perform encryption/authentication or not). This daemon is initiated as a thread, a master daemon creates simultaneous processes and feeds them with different information. This way, *strongSwan* works as a highly parallelized application. As stated in section III-B, we added two new functions: `get` and `put` commands. These functionalities allow to dynamically manage the traffic of a concerned IKEv2/IPsec session. The `ipsec get <conn-name>` command recovers all the information concerning the IKE_SA, PAD, SAD and SPD of an already established IKEv2/IPsec session. Then, a SG can re-install the previously recovered context with the command `ipsec put <conn-name>`.

A. *Testbed description*

Our platform is composed of two desktop stations. Both operating systems run over Ubuntu (v12.04LTS in the IPsec EU, and v11.10 in the SG). We did not use the newest version of Ubuntu in the SG as the v11.10 was the recent one when we implemented **GET** and **PUT** functions. Our tests are performed locally on the same SG, which avoids the interactions and delays introduced by CXTP. In other words, these tests are especially focused on the interaction with the IPsec stack.

First of all, we focused on measuring the time to establish a single VPN on a SG by varying the number of VPN connections. Then, we measured the impact for upper-layers caused by the interruption during the functions **GET** and **PUT** at the IPsec layer (refer to figure 5a to see the exchanges during a **GET** and **PUT**). All the tests are performed with HTTP traffic over Ethernet links. Measurements show statistical results and quartiles. During these tests, we used Traffic Control in order to vary the bandwidth limit during the HTTP download and thus study the impact over different transmission rates. To generate statistical results, 100 measurements were done.

The tests have been measured by using the command `time`, which writes a message to standard output giving timing statistics about this program run. The outputs of `time` are (i) the *elapsed real time* between invocation and termination of the command, (ii) the *user CPU time* and (iii) the *system CPU time*. We considered the value of the *elapsed real time* in order to evaluate our stats results. For simplicity, we will refer to *elapsed real time* as `time`. This is how we measured the time it takes for a given command to run.

V. Performance tests & Results

Our graphs are represented in box-and-whiskers style. This kind of representation is mostly used to plot statistical data. For each measurement (based on 100 samples) the box-and-whisker plot indicates: (i) the smallest observation, (ii) lower quartile, (iii) the upper quartile and (iv) the largest observation. The space between the lower and upper quartile represents 50% of the samples. For more clarifications, refer to [6]. The different testbed measurements are performed over Ethernet links (some of them with different bandwidths). We also loaded the SG by establishing different VPN tunnels for each TCP connection granulated by its port number. For example, a TCP connection using port 80 will not traverse the same tunnel as a TCP connection using port 81. They will use different cryptographic material and thus different IKEv2/IPsec tunnels.

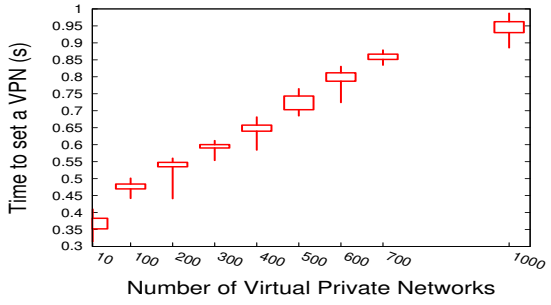
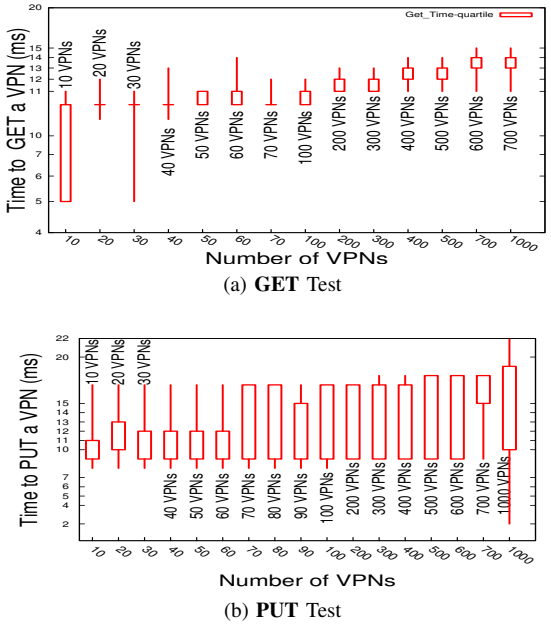


Fig. 3. VPN Establishment Time

A. First Test - VPN Establishment Time

The first test measures the time for establishing IKEv2/IPsec tunnels between two end-points. The tunnels were initiated one after the other to avoid half-opened IKE-SAs. This is not a stress test but a load test. Figure 3 gives the results for initializing 10, 100, 200, 300, 400, 500, 700 and 1k tunnels. We can clearly see that the time measured is proportionally higher as the amount of VPNs rises. Establishment of 10 tunnels takes 0.31s to 0.41s, whereas 1000 tunnels may take 0.89s up to 1s for a single VPN establishment. All the other measures (i.e. 100, 200, 300 tunnels ...) show results between 0.3s and 1s for a single VPN establishment.



(a) **GET** Test

(b) **PUT** Test

Fig. 4. Experimental IKEv2/IPsec Context Management

B. Second Test - GET and PUT Times

Our second test consists in measuring the time to **GET** and to **PUT** an active IKEv2/IPsec context. First, we established different scenarios (different amount of active tunnels): 10, 20, 30, 40, 50, 60, 70 , 100, 200, 300, 400, 500, 600, 700 and 1k active VPN tunnels. Then, we proceed to **GET** ten (10) randomly chosen VPNs over each scenario (10,100,200...). The resulting time to **GET** 10 randomly chosen VPNs is divided by 10 in order to obtain the average time to perform a **GET** over a single VPN connection. Detailed results are shown in figure 4a. We can observe that the values are quite similar for all the scenarios (we consider the case of 10 and 30 VPNs isolated variations, which is normal). The command `ipsec get` performs its task in a range of $5ms$ up to $15ms$. By the way, by observing figure 4a we can realize that a loaded SG (600-700 VPNs) takes more likely $13ms$-$15ms$ to perform a **GET** over a single VPN whereas for a relieved SG these times are around $5ms$ and $12ms$. Then, we proceed to **PUT** the ten (10) previously randomly chosen VPNs for each scenario (10,100,200...). The time that the script takes to **PUT** these connections is divided by ten (10). This results in the average time to **PUT** a single VPN active session. Figure 4b shows the measurements concerning the **PUT** function for all the scenarios. It represents the time that a SG takes to install a single IKEv2/IPsec context from a file. We observe

that *strongSwan* takes a range of $2ms$ up to $22ms$ to perform a **PUT** of a single VPN.

C. Third Test - Upper-layer's Reactivity

We observed the reactivity of upper-layer protocols facing interruptions at the IPsec layer on SG's side. The test consists to **GET** and **PUT** a single VPN connection during an HTTP download. We perform this with different traffic rates (controlled with Traffic Control implementation for linux [7]). Figure 5a illustrates a protocol representation of what happens during this test. Initially, a VPN is established between an EU and the SG. Then, we start an HTTP download by using the command `wget`. The source file is placed on the EU's side whereas the SG is configured as the destination. Even if in real life an EU (and not the SG) is usually the destination when downloading files from the Internet, we chose this methodology in order to analyze the impact of a SG facing interruptions at the IPsec layer. Thus, after the download has been performed during five (5) seconds, we **GET** the whole IKEv2/IPsec context on the SG, causing it to loose connectivity with the peer. The EU continues to send ESP packets as it is unaware of the **GET** function performed on the SG side. We considered different interruption times (T_S Time_Sleeps of 10, 30 and 60 seconds) before performing a **PUT** function. Finally, once the IKEv2/IPsec context is reinstalled, we observed the time to finish the HTTP download. As we carry HTTP traffic over an IPsec protected communication, we set the HTTP parameters through `wget` to be as flexible as possible facing interruptions. Also, as we measured different traffic rates during the downloads, we changed the size of the file being downloaded (because big size files would take too much time to download at lower rates). Table I shows the different file sizes used for each traffic rate. Figure 5 represents the download time concerning all file sizes and rates. Each figure (5b, 5c and 5d) shows the download times for each interruption T_S. We observe that at lower traffic rates (less than 3000kB/s), the measures are very stable because the quartiles are very thin. For higher traffic rates (more than 3000kB/s), the download time is unstable and so the quartiles are more spread and thicker. On the other hand, the three graphs have a similar behavior. Even though they are all offset by their corresponding interruption time. Finally, this test represents the impact of the interruption at the IPsec layer during an active session.

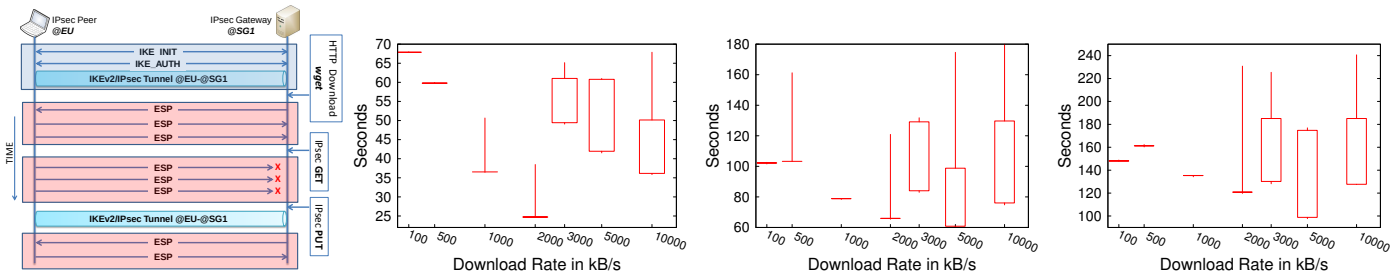
Interruption Time T_S	Rate	File Size
10s, 30s and 60s	10 kB/s	1MB
	100 kB/s	5MB
	500 kB/s	20MB
	1 MB/s	20MB
	2 MB/s	20MB
	3 MB/s	100MB
	5 MB/s	100MB
	10 MB/s	100MB

TABLE I
THIRD TEST - INTERRUPTION TIMES, TRAFFIC RATE AND FILE SIZES

VI. POSITION OF OUR WORK & RELATED WORK

Concerning IPsec-CTX there are different mechanisms that have similar approaches:

- **RFC6311: Protocol Support for High Availability of IKEv2/IPsec [8]**. This RFC proposes an extension to the IKEv2 protocol. It aims to solve the refreshing of both IKEv2 (IKE_SA) and IPsec (CHILD_SA or IPsec_SA) counters due to a mismatch caused by a failure take-over process. Although it involves changes to the IKEv2 protocol, this extension handles the renegotiation of the IKEv2/IPsec counters in an efficient manner.
- **RFC4067 Context Transfer Protocol [2]** This protocol introduces a generic mechanism that allows context transfer between SGs. In our scenario, Context Transfer Protocol (CXTP) is the protocol we could use to transfer an IKEv2/IPsec context between Security Gateways. One issue when using CXTP under this scenario is that the EU is actually involved during the SG migration, thus increasing the number of messages exchanged between the EU and the IPsec SG. CXTP can be triggered by one of the SGs (network controlled) or by the EU itself (mobile controlled, or controlled by the EU). In both cases, a message named CTAR (Context Transfer Activate Request message) must be sent from the EU towards one of the IPsec SGs during the context transfer.
- **Mobility Related Documents**: [9] and [10] addresses EU's IP mobility and multihoming. When an EU changes its IP address, the IKEv2/IPsec contexts are not valid anymore and the EU looses connectivity. An INFORMATIONAL exchange allows all concerned SAs to be updated and thus let the node remain connected and protected with IPsec.
- **Allard [11]** addresses the IPsec context transfer between two Security Gateways with an implementation using IKEv1. In contrast, our work is positioned using IKEv2, which involves differences compared with IKEv1 (see section 2.3.1 in [12]): less signalization, mobile friendly ([9] [10]), better performance and less complexity.
- **RFC5685 Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2) [4]** proposes an IKEv2 extension to redirect an IKEv2/IPsec session from one gateway to another. It does NOT pre-authenticate the EU prior to the connection towards a new Security Gateway (SG). The EU needs to renegotiate a new SA with the new SG.
- **Yu [13]** proposes to solve availability issues on IPsec by simulating a cluster mechanism for IPsec gateways. Even if we do not implement a cluster, we consider it a similar work, as *seamless switching* mechanism spreads SAs among both active and standby SGs. The author does not recommend a High-Reliable link between the gateways in order to communicate their SAs. However, it mentions that the members of a cluster could be deployed



(a) Detail of the test (b) Dowload Time with T_S=10s (c) Dowload Time with T_S=30s (d) Dowload Time with T_S=60s

Fig. 5. Upper-layer's Reactivity

in different network segments. The results are based on simulations and not a real implementation, which could differ from reality.

VII. Conclusion

We proposed a mechanism to dynamically manage IPsec-CXT. Our proposal aims to solve the reliability of overloaded IPsec SGs. By performing IPsec-CXT, a SG can manage the traffic and distribute efficiently the different IPsec tunnels between different servers of a cluster. Additionally, we studied through a real implementation the time to establish an IKEv2/IPsec session, as well as we measured the time to extract and re-install a security context. Finally, we measured the impact for upper-layers through different interruption times at the IPsec layer during an HTTP-based download. Future works will consider measuring the impact of the CXTP protocol and the communication between physically different SGs. Additionally to **GET** and **PUT** operations, we will consider the added network delays when performing an IPsec-CXT. We will also consider other mobility mechanisms [9] [10], instead of Redirect Mechanism [4]. Regarding a classic network behavior, we consider the results of **GET** and **PUT** functions as optimistic. Based on the results in section V-B, the worst case to perform a **GET** and a **PUT** are $15ms$ and $22ms$ respectively. We can even discard network delays since predictive mode performs IPsec-CXT prior to mobility operations, and thus the reestablishment of the tunnel could be done within $37ms$ ($15ms$ to GET + $22ms$ to PUT) plus the time to update the Security Associations at the EU's side.

References

[1] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 5996 (Proposed Standard), Internet Engineering Task Force, Sep. 2010, updated by RFC 5998.

[2] J. Loughney, M. Nakhjiri, C. Perkins, and R. Koodli, "Context Transfer Protocol (CXTP)," RFC 4067 (Experimental), Internet Engineering Task Force, Jul. 2005.

[3] D. Palomares, "Mechanisms to Ensure Continuity of Service for *IPsec/IKEv2* Based Communications," in *ICSNA-2011: International Conference on Secure Networking and Applications ,24-25 October, Paris, France*. FT - France Télécom, Division R&D, Issy Les Moulineaux (France Télécom), 2011.

[4] V. Devarapalli and K. Weniger, "Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)," RFC 5685 (Proposed Standard), Internet Engineering Task Force, Nov. 2009.

[5] A. Steffen., "the OpenSource IPsec-based VPN Solution: *StrongSwan*." [Online]. Available: http://www.strongswan.org

[6] Wikipedia, "Boxplot — Wikipedia The Free Encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/Box_plot

[7] M. Devera, "HTB Linux queuing discipline manual - user guide." [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

[8] R. Singh, G. Kalyani, Y. Nir, Y. Sheffer, and D. Zhang, "Protocol Support for High Availability of IKEv2/IPsec," RFC 6311 (Proposed Standard), Internet Engineering Task Force, Jul. 2011.

[9] P. Eronen, "IKEv2 Mobility and Multihoming Protocol (MOBIKE)," RFC 4555 (Proposed Standard), Internet Engineering Task Force, Jun. 2006.

[10] D. Migault, "MOBIKE eXtension (MOBIKE-X) for Transport Mobility and Multihomed IKE_SA," (Work in Progress), Internet Engineering Task Force, Sep. 2009.

[11] F. Allard, J.-M. Bonnin, J.-M. Combes, and J. Bournelle, "IKE Context Transfer in an IPv6 Mobility Environment," in *MobiArch'08, 22 août, Seattle (WA), USA*. FT - France Télécom, Division R&D, Issy Les Moulineaux (France Télécom), RSM - Dépt. Réseaux, Sécurité et Multimédia (Institut Mines-Télécom-Télécom Bretagne-UEB), 2008.

[12] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071 (Informational), Internet Engineering Task Force, Feb. 2011.

[13] L. Yu, S. Jia, C. Xu, J. Guan, and D. Gao, "An ipsec seamless switching mechanism with high availability and scalability by extending ikev2 protocol," *IET Conference Publications*, vol. 2011, no. CP588, 2011.