Fast VPN Mobility Across Wi-Fi Hotspots

André Zúquete
IEETA / Instituto de Telecomunicações
Dep. de Electrónica, Telecomunicações e Informática
Universidade de Aveiro
Campus Universitário de Santiago
3810–193 Aveiro, Portugal
andre.zuquete@ua.pt

Carlos Frade
Instituto de Telecomunicações
Campus Universitário de Santiago
3810–193 Aveiro, Portugal
carlos.frade@av.it.pt

***Abstract*— Wi-Fi hotspots deployed by domestic networks are an attractive approach to foster ubiquitous Internet access for mobile computers. However, it raises several security issues, since the owners of such hotspots cannot be trusted to be honest; therefore, visitors should consider them as potential Man-in-the-Middle attackers. On the other hand, the owners of such hotspots should not be legally accounted for illegal or unethical activities performed by visitors. We defend that both issues can be tackled by using a VPN-based access architecture for visitors. However, current VPN solutions do not handle gracefully the mobility of clients, an interesting possibility to *navigate in a sea* of user-provided hotspots. In this paper, we propose a modification of OpenVPN, a very effective and secure, open-source VPN solution, to support the seamless mobility of VPN clients. The proposed modification allows OpenVPN tunnels to be reconfigured as soon as possible after a client handover, in order to minimize outage delays and traffic losses. Performance evaluation with an implemented prototype in different test scenarios showed that we can reduce the handover delay imposed at the VPN level down to 1.7% of the current value, at least.**

I. Introduction

Mobility and/or roaming capability are fundamental requirements of current and future Internet clients. To address these requirements, it is fundamental to provide widespread connectivity and handover support for Internet clients. At an absolute limit, clients should be able to use any available mean, and possible several means (e.g. multi-homing), to get seamless access to the Internet.

User-provided networks [16] go towards this goal. By empowering each current Internet subscriber to share its network subscription with others, through domestic wireless hotspots using unregulated bands (e.g. IEEE 802.11), we can implement a grassroots approach for installing an ubiquitous Internet access infrastructure. With this model, everyone with a broadband subscription can become a *micro-provider* [16] of Internet access.

A. Motivation

However, in terms of trust, user-provided networks (UPNs) make a critical shift relatively to the traditional Internet architecture. Traditionally, Internet clients access the Internet through an Internet Service Provider (ISP), which mediates their access to the Internet backbone. Clients (implicitly) believe that an ISP is an honest organization, following well-known guidelines for protecting the privacy of costumers, although keeping enough records for legal liability in case of abuse. In particular, clients believe that an ISP will not deeply inspect their traffic for a purpose other than to fulfil legal obligations. In short, clients believe that an ISP protects their privacy to some extent.

On the contrary, such beliefs no longer stand when one considers UPNs. In fact, UPNs are provided by ordinary people, not by organizations, and they do not have to follow any legal rules or ethical guidelines to protect the privacy of clients (visitors). Furthermore, a UPN stands between visitors and the Internet, which is the perfect location to conduct Man-in-the-Middle (MitM) attacks against the traffic of visitors.

Therefore, by default, visitors should not ever trust on the honesty of the owners of UPN hotspots. Consequently, they should protect all their traffic flowing through such hotspots. Such protection should encompass integrity control (to prevent traffic forgery and mangling by a MitM), data and traffic confidentiality (to prevent the identification of actions performed and their actual details) and anonymity (to prevent attacks using the identity of visitors). A way to fulfil all these security requirements is to use a strong VPN between the client and Internet services, throughout a UPN.

On the other hand, people running UPN hotspots will have contracts with ISPs, which held them responsible for the traffic they generate. Therefore, they need to protect themselves from abuses perpetrated by hotspot visitors. An absolutely minimum requirement for protecting an hotspot owner from such abuses is to empower his ISP to differentiate the traffic initiated by the subscriber (hotspot owner) from the traffic initiated by visitors.

In this paper we defend that both these security problems – privacy and protection for the traffic of UPN visitors and traffic differentiation at the ISP level – can be solved by using a VPN for encapsulating the traffic of UPN visitors (see Fig. 1). UPN visitors have a *Virtual ISP*, or *virtual operator* [16], which essentially is an organization that allows authorized people to create VPNs. Virtual ISPs can also be provided by traditional ISPs, as a *virtual* complement to their physical Internet access provisioning.

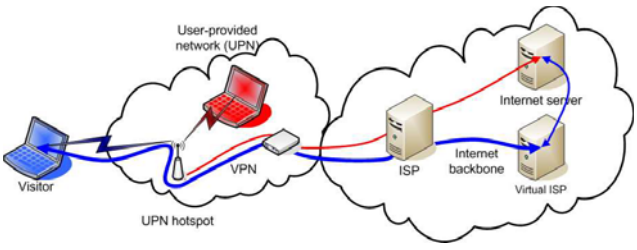

Fig. 1. Proposed architecture, using a VPN (thick blue arrow) for enforcing the privacy of UPN visitors and for enabling proper accountability, by Virtual ISPs, of traffic generated by UPN visitors

978-1-4244-6939-0/10/$26.00 ©2010 IEEE

A VPN between a visitor and his Virtual ISP (VISP) will naturally enforce the protection of the visitor's traffic regarding any eavesdropping, tampering and forging attempts by UPN owners. On the other hand, UPN owners may only authorize traffic of their visitors to well-known VISPs. This way, they are able to transfer legal liabilities of visitors' traffic to VISPs, which will then have to deal with abuses perpetrated by their clients. From the ISP point of view, they do not have to deal with abuses initiated by any visitors of UPNs ran by their subscribers. Finally, both the ISP and the VISP can easily implement several mechanisms and policies for regulating the QoS provided to UPN visitors, such as traffic shaping.

Note, however, that this approach is the opposite of the one proposed in [16]. They argue that all these security and trust problems, identified as *short-term challenges*, do not *require tight security mechanisms* but, instead, should be tackled with *innovative trust managements schemes*, following the *human trust behaviour* and using *reputation schemes*. However, eavesdropping leaves no trace, thus it is very difficult, or even impossible, to build a reputation scheme providing the honesty of someone regarding eavesdropping. Concluding, we argue back that reputation schemes are not enough, we still need strong security mechanisms to effectively deal with all these security issues.

B. Contribution

Above we defended the use of VPNs between UPN visitors and their VIPS to tackle two security issues raised by UPNs. But we want to go further ahead, as we want UPN visitors to be able to move seamlessly among UPN hotspots, possibly seeking for the best offer regarding costs or QoS.

In this paper we studied and implemented a fast mobility solution for a VPN, namely for OpenVPN. Our goal was to enable OpenVPN sessions to be maintained across modifications of the IP address of the client host. For our work we assumed no more than OpenVPN already does about the network between the client and the server (see details in Section II). Furthermore, we tried to change OpenVPN as little as possible, to prevent the involuntary inclusion of security vulnerabilities. Namely, we completely maintained the existing control protocol: no control messages where changed and no new control messages were added.

The solutions here discussed encompass both a lazy approach and an aggressive approach regarding the reconfiguration of an OpenVPN tunnel. The lazy approach simply reconfigures the OpenVPN when the server gets client traffic from a new address. The aggressive approach does the same but the client proactively generates traffic when it detects a change in its IP address. This last approach is a best effort attempt to redirect the server-client traffic as soon as possible after an handover, in order to minimize the amount of traffic sent by a server to the former location of a client.

This paper is structured as follows. Section II overviews OpenVPN in order to justify our contribution. Section III explains the mobility problem of OpenVPN visitors. Section IV presents our contribution. Section V shortly presents the implementation of the proposed OpenVPN modification. Section VI presents the practical evaluation of our contribution. Section VII presents some related work, namely the reasons for using OpenVPN instead of other VPN solutions. Finally, Section VIII concludes the paper.

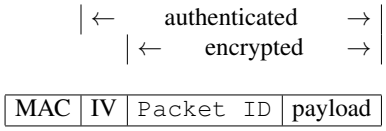

Fig. 2. DATA message format

II. OpenVPN Overview

OpenVPN is an open-source VPN solution[1]. It operates at the application layer and interacts with the TCP/IP protocol stack through the TUN/TAP virtual interface[2].

OpenVPN implements two channels, control and data, on top of UDP datagrams (using the same ports for addressing). On top of the control channel it also implements a reliable transport protocol, required for supporting SSL/TLS [3] authentication and key agreement protocols. Optionally, OpenVPN can also run on top of TCP, but we will not address it here.

OpenVPN supports two authentication modes: (i) static key mode, with pre-shared keys, or (ii) TLS mode, with X.509 public key certificates. We will not discuss the advantages and disadvantages of either mode; all that matters to our work is that both modes required the exchange of several control messages for setting up a new security context for a VPN tunnel.

OpenVPN can use several types of triggers in a VPN to renegotiate new, security-related session contexts (shared keys, sequence counters, etc.). In our work we had not the goal of adding a new trigger to negotiate new session contexts; instead, our goal was to add triggers to reconfigure only the end-point addressing information of a VPN tunnel, while keeping all existing key material (session contexts) and rekeying policies.

A. OpenVPN messages

OpenVPN encapsulates messages in independent UDP datagrams. Each message contains a header, identifying the message type (`OpCode` field) and the set of session keys being used, and the payload, which depends on the message type (CONTROL, ACK or DATA)

The structure of a DATA message, presented in Fig. 2, is divided as follows:

- A header, with 3 optional fields: a Message Authentication Code (MAC), an Initialization Vector (IV) for a cipher mode and a sequence number (`Packet ID`);
- The payload.

The MAC is a full-message authenticator, computed with HMAC [9], either with MD5 [13] or SHA-1 [1]. IV is a random 32 or 64-bit array, required by the CBC, CFB and OFB cipher modes[3] supported by OpenVPN. `Packet ID` is a monotonic sequence number, with 32 or 64 bits, for preventing message replay attacks. Receivers use a sliding-window mechanism to accept or discard messages given their `Packet ID`. Messages with an already received `Packet ID` (belonging to a valid message) are also discarded.

The DATA channel is also used to exchange a few extra *control* messages. This is the case of Ping messages, which are DATA messages starting with a pre-defined, 16-byte payload,

[1] `http://openvpn.net`
[2] `http://vtun.sourceforge.net/tun`
[3] Cipher Block Chaining, Cipher FeedBack and Open FeedBack

	time →			
Delays	L2 handover	L3 handover	VPN termination	VPN instantiation
Causes	802.11 authentication & association 802.1X authentication & key distribution	DHCP	Inactivity timeout	Authentication Key distribution

Fig. 3. Connectivity outages created by mobility: delays and causes

	time →		
Delays	L2 handover	L3 handover	VPN reconfiguration
Causes	802.11 authentication & association 802.1X authentication & key distribution	DHCP	Update of server-side session context

Fig. 4. Our goal: reduction of the overall outage delay replacing the termination and re-instantiation of the VPN by its reconfiguration

used as keep-alive beacons. These Ping messages have no reply.

During the setup of an OpenVPN tunnel, client and server exchange configuration information using the OpenVPN Configuration Control (OCC) protocol. OCC messages are DATA messages with a pre-defined set of initial 16 bytes (OCC magic) and an OCC request or reply.

B. VPN contexts and session's contexts

A OpenVPN server may keep several VPN tunnels, one for each active client. For each VPN tunnel, both client and server manage a related context, which can have several session contexts (set of shared keys, etc.), each with a unique session identifier (`Session ID`).

The server always uses the client UDP address (IP address + UDP port) to lookup for its VPN context; session identifiers are never used for this purpose. Session identifiers are only exchanged within CONTROL messages, on behalf of key agreement and setup protocols; they are never exchanged on DATA messages.

III. OpenVPN mobility problem

The OpenVPN was conceived to protect the traffic flowing in all kinds of IP networks, namely those containing Network Address Translation elements (NAT boxes). The only assumption that OpenVPN makes about the network is that the client of a VPN tunnel always uses the same IP address and UDP port number (UDP address), possibly provided by a NAT box along the client-server routing path. Fortunately, this is the normal behaviour of NAT boxes, otherwise it would be impossible to establish dialogs (sequences of requests and responses) using UDP through NAT boxes (possibly many).

When the IP address of the client changes, because it moved to another network, the server is no longer capable of finding the correct VPN context from the client's datagrams, and silently discards them. Note that the change of the client's IP address is relevant from the server's point of view (i.e. in the datagrams that arrive at the server). Thus, even when a client keeps its IP address when moving to another network (using, for instance, the same private IP address), the IP address that will be observed by the server will naturally be different.

On the other hand, when the IP address of the client changes, because it moved to another network, the server is no longer capable of sending data to it, as it does not know the new UDP address of the client and datagrams sent to the old UDP address are not likely to be rerouted to the new one (unless using MIP on the client side, for instance). Furthermore, the OpenVPN server ignores ICMP error messages caused by undelivered UDP datagrams, to prevent Denial of Service (Dos) attacks, therefore it will keep the (now) useless VPN until its garbage collection (for instance, after a threshold inactivity period).

Summarizing, when an OpenVPN client moves between networks, the VPN becomes entirely inoperative, as client messages become unacceptable to the server and server messages cannot be routed to the new client address.

In theory, this problem could be solved with IP routing mechanisms designed to handle mobility, such as MIP [12]. This approach, however, has two major drawbacks: (i) it would require a widespread use of MIP or other similar protocols, which is far from being a reality and (ii) re-routed IP datagrams should continue to be routed through the initial sequence of NAT boxes used when the VPN was established. Otherwise, even with MIP the client IP address would change, from the server's point of view. Nevertheless, it was proposed for commercial solutions for mobility of VPN clients (e.g. Motorola's Mobile VPN [11]).

Concluding, the actual OpenVPN implementation forces a mobile client to negotiate a new VPN tunnel when its IP address changes (from the server's point of view), which is a natural occurrence when the client moves across networks. This means that, in a mobility scenario, OpenVPN clients would have to, repeatedly, wait for inactivity timeouts after an IP reconfiguration and renegotiate a new VPN tunnel afterwards. The consequences of all this are connectivity outages created by mobility far longer than the ones created by ordinary reconfiguration delays within layer 2 and layer 3 handovers (e.g. 802.11 authentications/reassociations and DHCP IP address reservation, see Fig. 3).

IV. Support for clients' mobility

In this section we present our solution for enabling a seamless and fast reconfiguration of an OpenVPN tunnel when the client IP address changes. Our final goals were the following: (i) VPN reconfigurations should occur in the minimum possible time, after the change of the client IP address, (ii) reduce to the minimum the amount of traffic loss during a reconfiguration action and (iii) involve only the VPN client and server, and not any network element the in path(s) between them.

After studying the current OpenVPN protocol and implementation, we established the following guidelines for our contribution:

- Do not update the control protocol, to minimize the risk of introducing new security breaches by adding extra complexity to the control state machine.
- Do not increase the size of ordinary DATA messages, for keeping the original performance of OpenVPN.

Regarding legacy compatibility, our goal was to keep backward compatibility without extra signalling. This means that an updated server should be able to interact with a legacy client without any extra configuration, and, vice-versa. In this last case, however, it would be useful if updated clients could learn the mobility support from the server they are using. This is still an open issue.

A. Lazy reconfiguration approach

Client DATA messages originated in a different UDP address cannot be handled by the server because it has no means to find the related VPN context. Therefore, as a first approach we considered adding extra information to DATA messages that could help servers to find out the correct context upon the change of the client's IP address. An obvious choice for this extra information would be the (current) session identifier being used in the VPN.

From the observation of the structure of a DATA message we conclude that the IV field could be used, in some cases, for carrying a (constant) session identifier, instead of a random value. The specific cases depend on the relevance of the randomness of this field value. For CBC, for instance, it does not need to be random for improving security. On the contrary, for CFB, and mainly for OBF, its randomness is of utmost importance.

Session identifiers are 64-bit TLS session identifiers. With ciphers with a 64-bit data block, the session identifier could only replace the IV for the CBC cipher mode, and both client and server would have to exchange explicitly a (constant) IV in all DATA messages, which would be a waste of time and bandwidth. With ciphers with a 128-bit data block, the session identifier could replace only half of the random IV, which could then be used with all cipher modes (CBC, CFB and OFB). However, CBC can be used with a fixed IV, which does not need to be transmitted in all DATA messages, and piggybacking the session identifier in the IV would force the exchange of an IV in all DATA messages.

Summarizing, we can fully or partially replace the IV by the session identifier but usually with some restrictions (we cannot use CFB and OFB) or performance penalties (we are obliged to transmit a constant or useless IV always). The only cases where we do not loose in performance and we do not affect security significantly is when using CFB or OFB with 128-bit data block ciphers.

In spite of being easy and straightforward to implement this solution, it does not solve the problem of server messages that get lost until some client datagram reaches the server (either with a Ping or with an ordinary DATA message). This problem is addressed in the next section.

B. Aggressive reconfiguration approach

The aggressive configuration approach attempts to reconfigure the server's VPN context as soon as the client changes its IP address. Furthermore, it creates no restrictions on the use of cipher modes and adds no extra traffic overhead during the normal operation of VPNs.

In the aggressive mode, the VPN client works normally as it does now. When the client detects a modification in the network IP address of the TUN interface, it enters a special reconfiguration state. During this state, the client aggressively sends (modified) Ping messages to the server until receiving some DATA message from the server. Such reception is the required evidence that the server was correctly reconfigured; consequently, the client can abandon the reconfiguration state and resume normal operation.

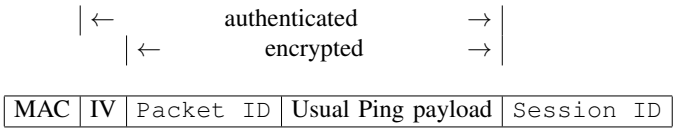

Fig. 5. Modified Ping message format, used for server-side VPN reconfiguration

The modified Ping messages are Ping messages that include an extra cleartext session identifier at the end of the payload (see Fig. 5). The server should now be able to recognize two types of Pings: (i) the current keep-alive Ping and (ii) the reconfiguration Ping.

When a reconfiguration Ping reaches a server, there are two possibilities: (i) the VPN context was not yet reconfigured or (ii) the VPN context was already configured. In the first case, the server cannot find the correct VPN context from the source UDP address. In this case, it checks the message size to validate if it can be a reconfiguration Ping and, in case of success, it fetches a session identifier from the tail of the payload, uses it to find the correct VPN context and validates the Ping as usually. If valid, the VPN context is updated with the new client UDP address, and a new context index is created based on that UDP address, replacing the old one. In the second case, the server is already able to validate and identify the reconfiguration Ping using its source UDP address, thus no further configuration is required.

In both cases, immediately after checking the validity of a reconfiguration Ping, the server sends a DATA message to the new client UDP address. This message can be either an ordinary DATA message already in the queue to be sent to the client, or a normal Ping message otherwise. The faster the reaction of the server, the less the client floods the network with reconfiguration Pings.

Note that the session identifier at the end of a reconfiguration Ping does not need to be authenticated; its goal is to replace the client's UDP address in the lookup of the VPN context, and the UDP address is not authenticated as well.

We decided to use reconfiguration Pings instead of modified, ordinary DATA messages, because we did not find a way of adding a reconfiguration tag to them without changing the format of all DATA messages. We even considered the possibility of adding such tag to other fields of the UDP datagram, but UDP has no option fields, similar to the ones of TCP, and kernels, by default, do not provide the means to add arbitrary options to IP headers.

C. Comparison between the two modes

The lazy reconfiguration mode provides a minimal reconfiguration approach, as it only reconfigures the server side of the VPN when the client sends a DATA message after an IP change. Consequently, all traffic sent by the server in the meanwhile gets lost. This mode also imposes limitations in the possible cipher suites used in the VPN and may create overheads in DATA messages by forcing them to contain always an IV with a piggybacked session identifier.

The aggressive reconfiguration mode provides a just in time, as fast as possible reconfiguration approach, but it depends on the client's ability to detect the modification of the IP address

of its TUN interface. This mode may also temporarily flood the network during the reconfiguration phase, but it reduces the loss of traffic sent by the server after the change of the client IP address. Finally, this mode has no impact on the cipher suites used by VPNs and does not create any extra overhead during the normal operation of VPNs.

In both cases, the modified DATA messages do not create new security problems, since the receiver validates them using the same methodology. Thus, attackers should not be able to send spoofed DATA messages that could reconfigure the VPN, because it requires them to send a DATA message with both a valid sequence number, which is already infeasible in the current OpenVPN implementation.

An attacker sitting between a client and a server (e.g. the UPN hotspot owner) may intercept a DATA message with a session identifier and change its source UDP address before dispatching it to the server. In this case, the server would be fooled by the tampered DATA message and would reconfigure the VPN in a defective way. Nevertheless, an attacker using this strategy can only deploy a DoS attack, and even this would be limited in time, because after a given period without a reply, the client would destroy the VPN and the attacker no longer could be able to continue the attack, eventually leading to the destruction of the VPN on the server side as well. Finally, OpenVPN is already vulnerable to this kind of attacks, both during the initial setup of VPN contexts and during its normal exploitation. Therefore, we did not introduce a new security vulnerability; it simply cannot be solved because OpenVPN, deliberately, does not authenticate source addresses (in order to handle NAT boxes in the client-server path).

V. Implementation

We implemented the aggressive reconfiguration mode in Linux using version 2.0.9 of OpenVPN (the current stable one).

The server modifications included a new hash table for finding VPN contexts given a session identifier, the detection and validation of reconfiguration Pings, the immediate dispatching of a reply DATA message after the validation of a reconfiguration Ping, the reconfiguration of the client UDP address of a VPN context and the update of the hash table that finds VPN contexts from source UDP addresses.

The client modifications included the detection of the modification of the IP address of the TUN interface, the switching to and from the reconfiguration state and the aggressive transmission of reconfiguration Pings. For detecting IP address modifications, we used a NETLINK ROUTE socket and we added its descriptor to the client's socket polling main loop.

VI. Practical evaluation

For evaluating the effectiveness of our VPN reconfiguration protocol, we created an OpenVPN tunnel between two machines and we continuously changed the IP address of the client TUN interface while maintaining a continuous, bidirectional client-server data flow. Due to the reconfiguration latencies, some traffic from the server to the client was lost during reconfigurations, but the server was always able to reconfigure itself.

We evaluated the performance of the OpenVPN modification in terms of overhead introduced during the normal operation of the VPN and traffic loss during reconfigurations. From our measurements we observed that the overhead is negligible and mostly due to extra socket readings on the client side (to process events on the NETLINK ROUTE socket other that the RTM_NEWADDR event).

Traffic losses and outage delays depend on at least four factors: computational speed of client and server hosts, network distance, in terms of hops and latency, between client and server hosts and data throughput between server and client. Since these parameters can vary a great deal in different operational scenarios and with different kinds of traffic flowing through the VPN, results obtained for a specific test scenario cannot be easily extrapolated to other scenarios.

Nevertheless, we present some performance evaluation results to enable readers to get an idea of the performance gains and limits of our proposal for OpenVPN reconfiguration.

The first is a throughput penalty evaluation in a very simple environment: two Linux computers, interconnected by a 100 Mbit/s switched, wired network, one being the VPN client and the other the server. We evaluated the throughput performance, both from the client to the server and vice-versa, using a very aggressive UDP traffic injection (with the `iperf` tool[4]). The throughput was measured in four different scenarios: (i) with the current OpenVPN, (ii) with the modified OpenVPN without mobility and (iii and iv) with the modified OpenVPN and frequent IP reconfigurations, in each direction. Each evaluation was performed during 60s, with 3 different injection ratios that create packet drops (35, 36 and 37 Mbit/s), and for the 3th and 4th scenarios we changed the IP address each 5 seconds (11 times during 60s). For each measure we took three samples for calculating average and standard deviation values. Table I summarizes the results.

Note that in these evaluation scenarios we intentionally removed all throughput disturbances created by reconfiguration outages due to L2 or L3 handovers (secure reassociations, DHCP, etc.). Such disturbances are independent of the VPN management, and vice-versa, therefore it would be meaningless to consider them when evaluating the performance gain of our VPN reconfiguration mechanism.

OpenVPN	Injection rate (Mbit/s)	Average recv rate (Mbit/s)	σ (Mbit/s)
Original	35	33.52	0.20
	36	34.46	0.14
	37	34.66	0.77
Modified	35	33.15	0.31
	36	34.23	0.29
	37	33.98	0.37
Modified w/ mobility (C→S)	35	31.47	1.0
	36	33.74	0.73
	37	31.30	1.05
Modified w/ mobility (S→C)	35	25.81	0.92
	36	27.86	0.92
	37	27.00	0.58

TABLE I

Throughput evaluation of the original OpenVPN, the modified OpenVPN and the modified OpenVPN with mobility (both for the C→S and for S→C directions)

The results above presented show that the throughput penalty imposed by the modifications introduced in OpenVPN vary from 0.7 to 2%, thus almost irrelevant. When mobility is considered, and for the evaluated scenario, penalties vary from 2.1 to 9.7% in the C→S direction and 19.2 to 23% in the S→C direction. This last direction provides worse results because

[4] `http://iperf.sourceforge.net`

when the client IP changes, the entire pipeline of datagrams flowing from the server to the client, inclusively already in the client's operating system, are lost. On the contrary, when the flow is from the client to the server, such pipeline will eventually reach the server and be accepted, thus yielding a higher throughput.

The other performance evaluation was relative to the delay of VPN reconfiguration, upon handovers, when compared with the delay of VPN terminations and re-instantiations. However, VPN termination delays depend a great deal on two unpredictable factors: (i) the maximum inactivity period configured by the user and (ii) the inactivity period observed before the handover. Consequently, we will not include it in the presented figures, which, therefore, represent our worst case. In the evaluation of VPN reconfigurations, we considered the time the client is not able to communicate with the server, i.e. the time from the modification of the client's IP until receiving a valid message from the server. Again, this represents out worst case, since the outage time from the server's perspective is smaller.

For evaluating the delays we used two architectures, one where the server is in the same network of the client and another where the server is in a network far from the client (11 hops away). In both cases we used OpenVPN with TLS mode for mutual authentication based on pre-distributed, X.509 public key certificates. Table II shows average and standard deviation values of delays of 10 VPN re-instantiation and reconfiguration tests.

	Server in the same network (ms)		Server in a distant network (ms)	
	average	σ	average	σ
VPN re-instantiation	2203	148	2259	37
VPN reconfiguration	4	0.9	39	7.4

TABLE II

DELAYS EXPERIENCED DURING THE RE-INSTANTIATION AND RECONFIGURATION OF OPENVPN SESSIONS UPON HANDOVERS USING A SERVER IN THE SAME NETWORK OR A SERVER IN A DISTANT NETWORK

The results of Table II show that the delay of the VPN reconfiguration is, at most, 1.7 % of the VPN re-instantiation delay. Naturally, this percentage would be much smaller if we had included the inactivity timeout in the value of the re-instantiation delay.

The performance of the lazy approach was not explicitly evaluated, but we can reason about its boundaries. Unlike the aggressive approach, the lazy one does not need to monitor the TUN interface, therefore it achieves a slightly higher throughput performance, equal to the one of the original OpenVPN. However, after an handover, the VPN reconfiguration delay depends on the client-server (upload) traffic rate, namely on the delay between the completion of the L3 reconfiguration and the next uploaded message. This delay may vary from zero (with intense upload traffic) up to the interval between keep-alive beacons (with reduced upload traffic). In any case, this delay does not depend on the download traffic. Concluding, the upload throughput performance of the lazy and aggressive approaches should be very similar. On the other hand, the download throughput performance of the lazy approach depends on the upload traffic load, while in the aggressive approach it always reaches the maximum possible value, independently of traffic load.

VII. RELATED WORK

In this section we provide a short motivation for choosing OpenVPN for adding client mobility instead of other VPN solutions. Considering only open-source solutions, since commercial solutions could not be adapted, we could consider IPSec-based VPNs [8] or PPTP [6].

IPSec was not considered a good option since it does not natively supports NAT boxes; it requires the help from some sort of a NAT transversal protocol [7]. Nevertheless, there is already an IETF standard for overlay VPN mobility, Mobike [4]. This standard addresses the fast reconfiguration of IPSec IKE Security Association in mobility scenarios, where peers can change their IP address. However, Mobike is much more complex than our reconfiguration protocol, as it takes 8 control messages while we only require 2 messages (and one them can even contain useful data).

There is also some research work and commercial products combining both MIP and IPsec, such as the work of Ruppelt *et al.* [14] and the Motorola's Mobile VPN product [11]. However, as we stated in the introduction, we considered from start that MIP is not likely to be widely available and, consequently, that we had to choose a different approach for handling scenarios without MIP.

PPTP, on the other hand, was also not considered a good option for two reasons. First, it uses a TCP connection for keeping the VPN alive. Therefore, adding mobility support would require adapting an established TCP connection in both the client and the server, and that would require a major modification in the PPTP management of live VPNs. Second, PPTP is not very strong in terms of effective security [15], therefore it should never be used to get proper protection from possibly very dangerous, MitM attackers hiding behind UPN hotspots. The Internet Encyclopedia [2] describes it as being "*good for low-threat environments*", offering "*medium security for moderate threat environments*", which is certainly not the case of UPNs. Graham & Cook, in [5], also say that "*PPTP is inherently insecure because there are too many unauthenticated control packets that are readily spoofed*", which is trivial for MitM attackers.

To conclude the related work, we can also refer the work of Lee *et al.* [10], describing a solution for supporting VPN mobility in a MPLS (Multi Protocol Label Switching) network using Costumer Edge Routers. The deployment of this solution is limited, as it depends on the availability of MPLS, so it is not particularly suitable for a widespread exploitation for accessing IP-based UPNs.

VIII. CONCLUSIONS

In this article we presented a modification of OpenVPN that allows clients to move across networks and, simultaneously, reconfigure their VPN tunnels in a very efficient and secure way. The proposed modification was implemented and tested in a real environment, using Linux machines. This work was motivated by the fundamental idea that the use of VPNs could solve two security issues of UPNs: protection of visitors' traffic and accountability of visitors' traffic. Therefore, our work may help visitors to move seamlessly across UPN hotspots, enabling them to move aggressively to the best hotspots in terms of costs and QoS.

ACKNOWLEDGEMENTS

This research work was supported by the Portuguese QREN (Quadro de Referência Estratégico Nacional) project n. 3144 (PANORAMA).

REFERENCES

[1] D. E. 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, IETF, Sept. 2001.

[2] H. Bidgoli. *The Internet Encyclopedia*. John Wiley & Sons, 2004.

[3] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, IETF, Jan. 1999.

[4] P. Eronen. IKEv2 Mobility and Multihoming Protocol (MOBIKE). RFC 4555, IETF, June 2006.

[5] D. J. S. Graham and M. Cook. *Secure Virtual Private Networks: Technical Guide*. ja.net, 2009.

[6] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol. RFC 2637, IETF, July 1999.

[7] A. Huttunen, V. Volpe, L. DiBurro, and M. Stenberg. UDP Encapsulation of IPsec ESP Packets. RFC 3948, IETF, Jan. 2005.

[8] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, IETF, Dec. 2005.

[9] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, IETF, Feb. 1997.

[10] Y. Lee, H. Choi, J. Na, and S. Sohn. A Mobility Support Mechanism for IP VPN on the MPLS Network. In *IASTED Int. Conf. on Communications, Internet, and Information Technology*, St. Thomas, US Virgin Islands, Nov. 2002.

[11] I. Motorola. Mobile VPN, Secure Connectivity on the Move. White Paper, 2008.

[12] C. Perkins. IP Mobility Support for IPv4. RFC 3344, IETF, Aug. 2002.

[13] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, IETF, Apr. 1992.

[14] R. Ruppelt, A. Pelinescu, C. Constantin, J. Floroiu, D. Sisalem, and B. Butscher. Building ALL-IP Based Virtual Private Networks in Mobile Environment. In *Int. Works. on Informatik and Mobile communication over wireless LAN: Research and applications*, Austria, Sept. 2001.

[15] B. Schneier, Mudge, and D. Wagner. Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2). In *In CQRE '99*, pages 192–203. Springer-Verlag, 1999.

[16] R. Sofia and P. Mendes. User-provided networks: consumer as provider. *IEEE Communications Magazine*, 46(12):86–91, Dec. 2008.