Prompt Lightweight VPN Session Resumption for Rapid Client Mobility and MTD Enablement for VPN Servers

如何 实现的，快速
VPN连接恢复？

Abdullah Alshalan*‡, Dijiang Huang*
* Arizona State University, Tempe, AZ, USA
‡King Saud University, Riyadh, Saudi Arabia
e-mail: Abdullah.Alshalan | Dijiang.Huang@asu.edu

***Abstract*—TLS-based VPN are increasingly used to establish a secure communication channel between VPN clients and server. However, they are not designed to handle the mobility VPN clients in efficient manner. OpenVPN, a widely deployed TLS VPN, binds VPN sessions with the clients and server IP addresses. A vertical handover will require an inactivity timeout to be triggered and full TLS handshake thereafter for the mobile client to resume the VPN session. Moreover, A VPN server that changes its IP address frequently as part of an MTD strategy will require the VPN clients to reconnect after their inactivity timeouts trigger with yet full TLS handshake. In this work, we developed and evaluated a lightweight VPN session resumption protocol that allows a VPN client or server to request an IP address update on-demand, maintaining the original TLS/VPN session. We implemented our protocol as part of MobiVPN which is a variation of OpenVPN. Our evaluation shows that VPN sessions can be maintained and resumed after an IP address change with an average of 97.19% decrease in time required to resume the VPN session in MobiVPN compared to the original OpenVPN.**

***Index Terms*—VPN, OpenVPN, Mobile VPN, Security, Mobility, MTD, Availability**

I. Introduction

In today's computing world, remote workers rely extensively on VPN technology in order to do their job remotely accessing private company resources securely [1]. However, traditional VPNs were designed for stationary devices. Both the VPN client and server were expected to maintain the same IP address during the entire VPN session. This assumption no longer holds. Nowadays, the use of mobile devices such as smart phones and tablets, is very prevalent. A VPN client's IP address will most likely change after each vertical handover. These devices experience continual IP address change due to roaming from one cellular network to another, switching either from a cellular network to a Wifi network or from a Wifi network to another Wifi network. A VPN server's IP address can also change. This could happen as a result of employing a moving target defense (MTD) technique on the VPN server. An MTD framework can migrate a VPN server from one virtual machine to another with a different IP address, or performs IP hopping by frequently changing the IP address of the VPN server.

In both cases, the change of the IP address must be handled gracefully maintaining the original VPN session. This is an essential objective since the IP address changes quite frequently in these two cases. In TLS-based VPNs, such as OpenVPN, the security association, unlike IPsec, is not tied to the IP address of the client or the server. Therefore, in theory an established TLS session between two endpoints will not suffer from a change of the IP address, and it can be used after an IP address change. However, an OpenVPN server maintains a list of VPN sessions (instances), one per client, and identifies them by the UDP address of the clients i.e. $< IPaddress, portno. >$. Hence, when the client's IP address changes, any packets the clients send over the VPN tunnel will be ignored by the VPN server since it will not be able to locate the correct VPN session using the new IP address. As a result, the client will have to time out before it reconnects with the VPN server using a whole new VPN handshake.

When the IP address of the VPN server changes, the VPN server performs nothing to preserve or resurrect the VPN sessions of the connected clients as it is designed to be passive in the relation between the VPN client and server. Therefore, all VPN clients will have to time out first, before they can attempt to reconnect with the VPN server. The attempts to reconnect will again require a full handshake and may even fail if the the VPN client is configured to connect to the VPN server's old IP address.

In our work, we developed a light-weight VPN session resumption protocol as part of the MobiVPN project, presented in [2], which is built on top of OpenVPN to make it a mobility-friendly VPN. Our design allows both the VPN client and server to be active participants, allowing them to initiate a signaling protocol which informs the other party of the IP address change. We utilized the TLS session ID to assist in locating the VPN sessions. In this paper, how we developed our protocol by modifying OpenVPN 2.2.2 is described, and our work was evaluated compared to the same original OpenVPN version.

The conducted experiments showed that VPN sessions can be maintained and resumed after an IP address change with an average of 97.19% decrease in time required to resume the VPN session in MobiVPN compared to the original OpenVPN.

The remainder of this paper discusses the related work in section II. It presents a background about OpenVPN in section III. Sections IV and V discuss respectively the model, design of our protocol. The implementation was omitted due to space constraints. The evaluation of our protocol is presented in section VI, followed by the conclusion in section VII.

II. Related Work

The problem of VPN client mobility has ignited several researches to solve this problem. In IPsec-based VPNs, MobileIP (MIP) protocol has been utilized to address the mobility problem in which each mobile node has a fixed home agent that can be reached at, while packets are routed to a foreign agent in the newly visited network. The foreign agent ensures the delivery of these packets to the mobile node [3]. This work was succeeded by [4] which utilizes two home agents to allow MIP to traverse multiple VPN gateways. [5] proposed the dynamic assignment of the external home agent to reduce the delay caused by a handover.

The IPsec security association needed to be renegotiated with an IP address change until MOBIKE was developed which enabled the IPsec in tunnel mode from preserving the security association during a layer 3 handover [6]. This essentially allowed for multi-homing in mobile nodes. Likewise, our work, although differently, preserves the TLS session across layer 3 handover events.

A mobile VPN was developed by [7] to support real-time applications using the Session Initiation Protocol (SIP). A VPN session here is identified by the SIP session address instead of an IP address. The VPN server has a SIP proxy that maintain a binding between a SIP session ID and mobile node IP address. The mobile node after roaming can send a request to update the address binding. Our work resembles the SIP-based mobile VPN in the fact that we identify VPN sessions using TLS session IDs and allow mobile nodes to update the VPN server with the new IP address.

Koponen et al. extended TLS and SSH to support mobility. Their extension allows for a TLS session renegotiated in abbreviated manner after an IP address change [8].

A solution to solve mobility in OpenVPN was presented in [10]. Similar to our work, when a change of the IP address is detected, they send ping messages accompanied with the TLS session ID in order to update the UDP address of the client's VPN session information at the VPN server. Our work differs from this work in the following: 1) they detect the IP address change of the tun interface, whereas we preserve the IP address of the tun interface (virtual IP), and we detect the change of the IP address of the physical interface, 2) the signal to update the UDP address binding is sent over the control channel which is a reliable layer, instead of sending it through the unreliable data channel, 3) because OpenVPN uses TLS which does not protect the IP layer, we send and encrypt the new IP address in the update signal message to prevent a middle-man from performing IP spoofing, 4) we allow the VPN server to use the update message signal to inform all connecting clients of a server's IP address change. Our work is the only work as of now that allow a TLS-based VPN server from preserving the VPN sessions of connection clients when its IP address changes.

The work of [11] introduced a VPN framework that hides the VPN server from attacker using Mobile-IPv6 based MTD. This work in fact is motivating to our work as our work enables for an MTD-protected VPN server without the need for Mobile-IPv6.

III. Background

Before we present our solution, we introduce in this section a brief background on how OpenVPN operates.

OpenVPN is implemented as a user-space process that interacts with the TCP/IP stack through a virtual network interface; a TUN interface for a switched network mode (Layer 3), or a TAP interface for a bridged mode (Layer 2). Our work is focused on the former mode where the TUN interface is assigned a virtual IP by the VPN server. For a setup with multiple VPN clients, OpenVPN uses a TLS-based mode to authenticate the connecting VPN clients and subsequently negotiate session keys to encrypt and/or authenticate data packets.

Two communication channels are set up when a VPN client connects with a VPN server. The control channel is first set up starting with a full TLS handshake followed by several messages exchanged for the configuration of the VPN tunnel. Our experiment of OpenVPN 2.2.2 showed in Wireshak 132 packets exchanged between OpenVPN client and server to establish the VPN tunnel with 33 RTTs. Control channel implements a reliability layer by acknowledging control packets are required by the TLS protocol. A data channel is then constructed after negotiating session encryption keys. This channel is not reliable and therefor no acknowledgment are sent back upon the reception of a data packet.

Figure 1 shows the format of data and control packets. One key difference is that data packets, unlike control packets, do not contain the TLS session ID. A UDP socket is created between the VPN client and server to transport both the control and data packet. The UDP address of the client is used by the VPN server is the identifier of the VPN session. Data packet as indicated by its format in figure 1 first go through the TUN interface and then get encapsulated and sent out using the physical interface using the aforementioned UDP socket.

Once a VPN client changes its physical IP address, the VPN server will not be able to locate the VPN session information and therefore it will not be able to decrypt any packets sent from the client. The VPN client as being the only active party will have to wait for an inactivity timeout to be triggered, at which, a hard reset signal will be thrown to trigger a full VPN connection handshake exactly similar to the initial connection.

IV. System Models

In this section we discuss the models of our mobility-aware variant of OpenVPN.

A. Lightweight VPN resumption model

We model our lightweight VPN resumption as a finite state machine in both the VPN client and server.

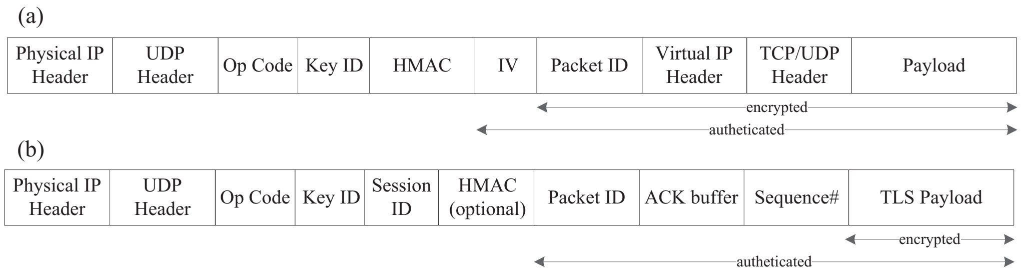

Fig. 1. The format of OpenVPN packets: a) data packet, b) control packet.

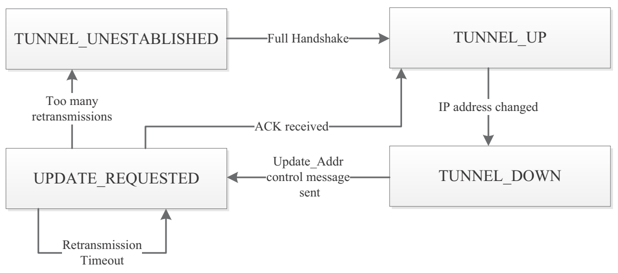

Fig. 2. Lightweight VPN resumption finite state machine of the VPN client

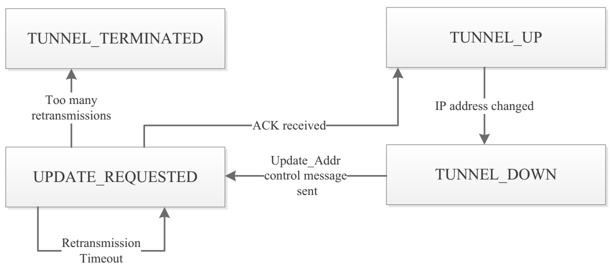

Fig. 3. Lightweight VPN resumption finite state machine of the VPN server

1) VPN client model: The VPN client model is illustrated in figure 2. The client starts with in the `TUNNEL_UNESTABLISHED` state. After a full TLS/VPN handshake the VPN process enters the `TUNNEL_UP` state. When the physical IP address changes due to mobility, the VPN enters the `TUNNEL_DOWN` state.

The VPN then sends a control message requesting the VPN server to update the UDP address binding with the new IP address. After that, the client enters the `UPDATE_REQUESTED` state. A reception of an acknowledgment from the VPN server indicates that it was able to find the appropriate VPN session information as it was able to decrypt the control message. The VPN client can then move to the `TUNNEL_UP` state.

The absence of acknowledgment from the VPN server will take the VPN client back to the `TUNNEL_UNESTABLISHED` state in which the VPN will attempt to perform a full handshake as the original OpenVPN does.

2) VPN server model: The VPN server model is very similar to the VPN client model as shown in figure 3. The difference is that the VPN server cannot initiate a full handshake as this is the responsibility of the VPN client. Therefore,

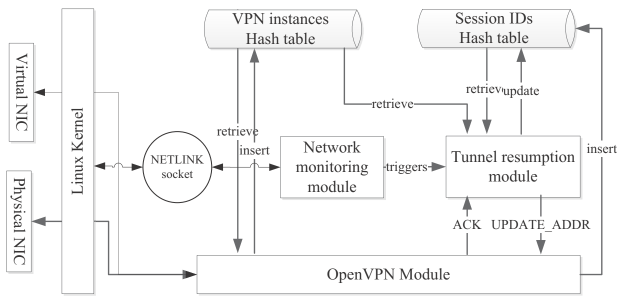

Fig. 4. The design of our system

the VPN server starts in the `TUNNEL_UP` state. Moreover, when the VPN server's IP address is changed deliberately as a result of an MTD mechanism, unacknowledged request to update the IP address binding at the VPN client will result in the VPN server terminating the VPN session. We note that the VPN server maintains a unique state per VPN session (VPN client).

B. Attack model

We take in consideration a man-in-the-middle attack model, in which the attacker can passively monitor the VPN traffic. The attacker can identify an update address control message via inspecting the unencrypted $OpCode$ and the length of the packet. Once succeeded, the attacker can alter the IP address of outer IP header. The receiving endpoint of the update message may bind the VPN session with the altered IP address. The attacker at this point has successfully denied service from the update address requester.

V. System Design

Our design introduces two new modules to OpenVPN: a networking monitoring module and tunnel resumption module who interacts with OpenVPN which is considered in our design as a one module.

These three modules interact with each other as presented in figure 4. The current tunnel state as described in section IV is stored in a variable as part of a VPN instance's context. We now describe the modules we added.

A. OpenVPN module

This module is responsible for fully establishing the tunnel in addition to sending and receiving packets in encrypted and authenticated manner. In the VPN server case, a hash table is to save VPN session information. Each entry of this hash table consist of a hash of the UDP address as the entry key, and the entry value is a $VPN\_instance$ record that contains the the VPN session information including cryptographic keys and the IP of the client for verification purposes.

We make a modification to this module by introducing another hash table where the key of entries is the TLS session ID, and the value is the hash of the UDP address. After a full handshake is concluded between a VPN client and server, the VPN server adds a new entry into the session IDs hash table. This modification is added to the VPN server. The VPN client maintains a single $VPN\_instance$ and thus does not need a lookup table.

B. Network monitoring module

This module is designed to detect any changes to the IP address of the physical network interface, or if the packets are being routed through a different network interface. Once such event is detected, this module sets the tunnel state variable to `TUNNEL_DOWN`, and triggers the tunnel resumption module. This module helps eliminating the idle time until the inactivity timeout triggers.

C. Tunnel resumption module

This module is the brains of this project. Once it is triggered, it checks whether this VPN process works in a client mode or a server mode.

In the client case, it sends an `UPDATE_ADDR` control message to the VPN server. We define the format of the update message as illustrated in figure 5. The VPN server will not at first be able to locate the appropriate $VPN\_instance$ in order to process the packet since the packet is sent from a new IP address. The VPN then inspect the `OP_CODE` which is unencrypted. If it determines that this is a control message, it uses the packet's session ID to perform a lookup on the session IDs hash table. If found, the old UDP address is retrieved and used to find the correct $VPN\_instance$. At this point, the TLS payload is decrypted and checked for 1) the existence of the command `"UPDATE_ADDR"` and 2) followed by an IP address that matches the IP address in the IP header. The second check is to account for the attack model presented in section IV-B. If any of the two checks fails, the update message is ignored and dropped without acknowledgment. If the two checks hold, the VPN server updates the updates the session IDs hash table with the new IP address. It also updates the $VPN_instances$ hash table with the new IP address as well as updating the relevant information in the $VPN\_instance$ with the new IP address. This is done because OpenVPN always checks when processing a packet that the IP address of the packet matches the IP address in the $VPN_instance$. Finally, an acknowledgment is sent to the VPN client indicating a successful IP update.

In the server case, it goes through the $VPN\_instances$ hash table sequentially, and sends an `UPDATE_ADDR` control message to every connecting client. Upon receiving the message at the client side, the client unlike the server does not perform any lookups and instead try to decrypt the packet with the only $VPN\_instance$ it maintains. If the message is found to be an `"UPDATE_ADDR"` and the two checks from above holds, the IP address information in the $VPN\_instance$ record is updated with the new IP address. An acknowledgment is then sent to the VPN server to confirm the success of the IP address update.

In both cases, `"UPDATE_ADDR"` messages are retransmitted if not acknowledged. After a configured number of retransmission times, the pursuit to update the IP address stops. The VPN server just terminate the $VPN\_instance$ in question, while in the case of a VPN client, it returns the `TUNNEL_UNESTABLISHED` state where it attempts to perform a full handshake.

VI. Evaluation

A. Security Evaluation

Here, we evaluate our work against the attack model presented in section IV-B. The MITM attacker, can definitely intercept an update packet, and replace the IP address in the IP header with his or her own IP address, or with a bogus IP just to deny the update receiver from updating its context to the correct address of the update sender. Our protocol is resilient to such an attack since the new IP address is appended to the `UPDATE_ADDR` command in TLS payload. This IP address is protected by TLS from tampering and only the real owner of the TLS keys can create such a packet.

The attacker can keep a copy of a legitimate update message and replay it at a later time when the IP in this packet is no longer the current IP of the VPN sender. Our system is also resilient to this attack as OpenVPN already implements a replay attack counter measure using a monotonic packet ID and an acceptable receive window at the receiver side.

B. Performance Evaluation

1) Testbed Setup: The testbed was set up with two configurations: a local testbed and a distant testbed. In both, the VPN server and the application server are installed in virtual machines running on VMware Fusion for Macbook. These VMs run Ubuntu 16.04 with 2GB RAM. The client VM and runs Ubuntu 12.04 with 2GB RAM. In the local testbed, the VM is hosted in the same servers' Macbook, whereas in the distant testbed, it is hosted on a separate Macbook.

The VPN server is connected to two networks, a private network (10.0.100.0/24) along with the application server. The VPN client communicates with the VPN server through another local network (192.168.100.0/24) in the local testbed, or through the Internet in the distant testbed.

In the local testbed, we emulated a client mobility event or a VPN server migration event by changing the IP address of their respective physical network cards using the $ifconfig$ command. In the distant testbed, we triggered a mobility event by turning off the Wifi interface and turning on the cellular interface. The cellular connection is obtained by USB

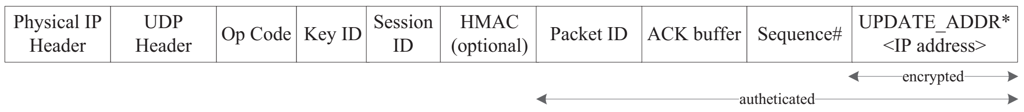

Fig. 5. The format of UPDATE_ADDR control message

tethering. In all of our experiments, we took the average of 5 trials.

2) VPN Tunnel Resumption at the VPN client: In order to evaluate the performance effect of our work, we conducted our experiment over the local and distant testbed setup. The experiment was conducted by sending data using iperf through the VPN tunnel between the mobile client and the application server. After 7 seconds of data transfer, a one mobility event is triggered. We sent 200MB in the local setup, and 40 MB in the distant setup. The mobility event causes the mobile client to get a new IP address.

Figure 6 shows the amount of data transfer over time using OpenVPN with the timeout set to the recommended default value of 60 seconds denoted as (OpenVPN-60s). The same was done with the timeout of OpenVPN set to the minimum value possible of 3 seconds denoted as (OpenVPN-3s). We also ran the same test on MobiVPN.

Figure 6 shows one trial of data transmission over the local testbed. After the mobility event, OpenVPN-60s took 64.123 seconds to re-instantiate the VPN session. The data transfered was delayed even further due to TCP retransmission backoff. The data transfer took 130.1 seconds. OpenVPN-3s took 6.907 seconds to re-instantiate the VPN session. The data transfer took 33.1 seconds. MobiVPN was the fastest with 222 milliseconds to resume the VPN session including the time to detect the change of IP address. The data transfer took 24 seconds. MobiVPN was able to decrease the time to resume the VPN by 99.65% compared to OpenVPN-60s, and by 96.79% compared to OpenVPN-3s.

Performing a similar experiment over the distant testbed resulted in MobiVPN surpassing both OpenVPN-3s and OpenVPN-60s. Figure 7 shows one trial of data transmission over the remote testbed. We noted that these savings in data transfer time would increase if multiple network switching occurs. Figure 8 shows a comparison of the time it took to resume the VPN tunnel after the mobility event. MobiVPN clearly outperformed OpenVPN in both of its configurations.

During the experiment, we observed that sometimes the data transfer in the OpenVPN-3s was delayed for about 3 seconds due to the aggressive timer of 3 seconds as the tunnel was unavailable during an unnecessary attempt at restarting the tunnel.

Performing an aggressive timeout such as 3 seconds is not a practical solution for two reasons. 1) During a period of inactivity, the VPN client will perform unnecessary full VPN handshakes every three seconds. MobiVPN achieves faster tunnel resumption without consuming the tunnel during inactivity. This is quite an important feature especially for battery-constrained mobile phones. Sending unnecessary data

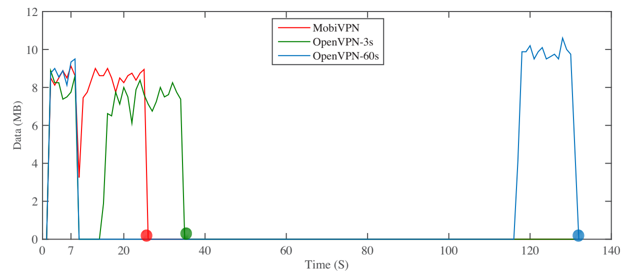

Fig. 6. Effect of Fast VPN Resumption on Data Transfer - Local Testbed

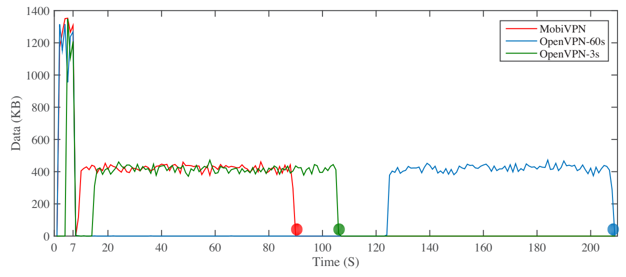

Fig. 7. Effect of Fast VPN Resumption on Data Transfer - Distant Testbed

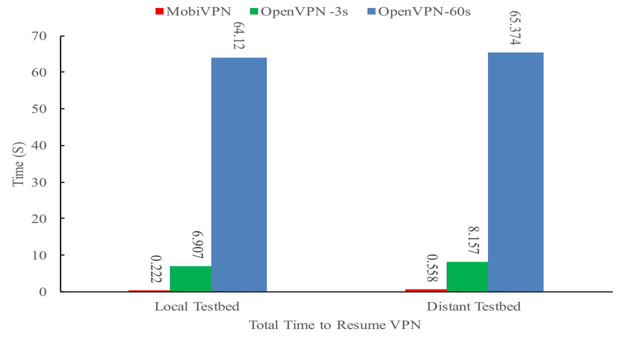

Fig. 8. Total Time To Resume the VPN

over the tunnel prevents the radio module of mobile devices from going to sleep mode during inactivity, which, as a result, consumes more power. 2) Even if the mobile client sets an aggressive timeout, the VPN server can overwrite that, and pushes its default settings onto the mobile client.

3) VPN Tunnel Resumption at the VPN server: In this experiment, we aimed to measure how long it took all connecting OpenVPN clients to resume the VPN tunnel after changing the IP address of the VPN server. We performed our experiment on the local testbed. We used one client machine but ran multiple OpenVPN client processes as each process is considered by the OpenVPN server as a unique VPN client. We configured

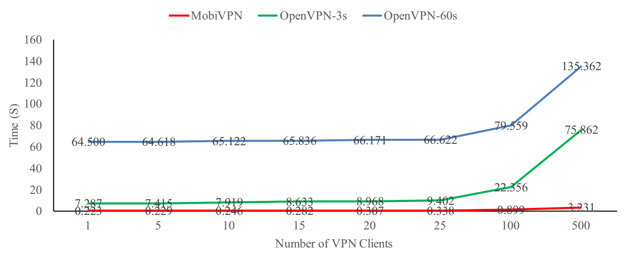

Fig. 9. Evaluation of MobiVPN vs OpenVPN when the VPN server changes its IP address

the VPN server to accept the same certificate from multiple clients.

Since OpenVPN server cannot resume/re-instantiate a VPN session, as this is the role of the VPN client in OpenVPN's design, we emulated a mobility event at the OpenVPN server by issuing a *kill* command that terminates the VPN session of all connected clients since they have the same certificate's common name. This made all VPN clients re-instantiate the VPN session after their configured timeout.

Figure 9 shows the outcome of the experiment in the same three scenarios we explained in the VPN client experiment. The figure shows how MobiVPN server was able to inform the clients of its IP address change and resume the the VPN tunnel in a much smaller fraction of time than that of original OpenVPN. As more clients were connected, the time it took OpenVPN in both configurations to resume all VPN sessions was increasing in a much higher rate than that of MobiVPN.

In an MTD framework where servers IP addresses change frequently, resuming VPN sessions in OpenVPN becomes impractical. The VPN server's IP address may be due for a new change of IP address even before all clients have reconnected. In addition to the costly time it requires to reestablish the VPN sessions with the clients, packet loss can be another problem which we highlight in the next section.

C. VPN Resumption Impact on Packet Loss

In this experiment, our goal was to measure the packet loss when using our light-weight VPN session resumption as opposed to a full VPN handshake. We eliminated the timeout effect in this experiment by issuing a `SIGUSR1` signal through the management interface to perform a full VPN handshake. Our light-weight handshake was similarly triggered by issuing a `SIGUSR2` signal.

We performed the experiment by streaming UDP packets over the VPN tunnel using *iperf* at 1Mbps sending rate for 60 seconds. We performed 5 cases, where in each case the number of signals triggered is increased by one. Figure 10 shows the significance reduction of packet loss in MobiVPN compared to OpenVPN.

VII. Conclusion

In this paper, we presented a new protocol for a light-weight VPN session resumption that allows both MobiVPN client and server to resume an already-established VPN tunnel.

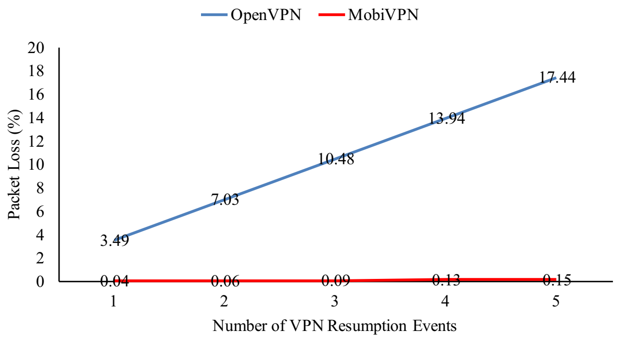

Fig. 10. Data Loss Caused by OpenVPN's Full Handshake vs. MobiVPN Lightweight Handshake

The evaluation we performed on our implementation of the protocol showed the feasibility of employing it for mobile VPNs and MTD-enabled VPN servers. The time it took to resume a VPN tunnel after a mobility event was decreased in MobiVPN by an average of 97.19% compared to the time it took OpenVPN to re-instantiate the VPN tunnel in both timeout configurations.

Therefore, we believe our light-weight VPN resumption will improve the mobile user experience as well as enable VPN servers from utilizing MTD protection without negative effect on the connected clients.

VIII. Acknowledgments

This research is based upon work supported by the NSF under Grants 1642031 and 1528099, and NSFC under Grants 61628201 and 61571375.

References

[1] A. Alshalan, S. Pisharody, and D. Huang, "A survey of mobile vpn technologies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1177–1196, 2016.
[2] ——, "Mobivpn: A mobile vpn providing persistency to applications," in *Computing, Networking and Communications (ICNC), 2016 International Conference on*. IEEE, 2016, pp. 1–6.
[3] F. Adrangi and H. Levkowetz, "Problem statement: Mobile ipv4 traversal of virtual private network (vpn) gateways," Internet Requests for Comments, RFC 4093, August 2005.
[4] S. Vaarala and E. Klovning, "Mobile ipv4 traversal across ipsec-based vpn gateways," Internet Requests for Comments, RFC 5265, June 2008.
[5] J.-C. Chen, J.-C. Liang, S.-T. Wang, S.-Y. Pan, Y.-S. Chen, and Y.-Y. Chen, "Fast handoff in mobile virtual private networks," in *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*. IEEE Computer Society, 2006, pp. 548–552.
[6] P. Eronen, "Ikev2 mobility and multihoming protocol (mobike)," 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4555.txt
[7] S.-C. Huang, Z.-H. Liu, and J.-C. Chen, "SIP-based mobile VPN for real-time applications," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 4. IEEE, 2005, pp. 2318–2323.
[8] T. Koponen, P. Eronen, M. Särelä *et al.*, "Resilient connections for ssh and tls." in *USENIX Annual Technical Conference, General Track*, 2006, pp. 329–340.
[9] J. Yonan, *OpenVPN 2.2 man page*, 2008. [Online]. Available: https://community.openvpn.net/openvpn/wiki/Openvpn22ManPage
[10] A. Zúquete and C. Frade, "Fast vpn mobility across wi-fi hotspots," in *Security and Communication Networks (IWSCN), 2010 2nd International Workshop on*. IEEE, 2010, pp. 1–7.
[11] V. Heydari, S.-M. Yoo, and S.-i. Kim, "Secure vpn using mobile ipv6 based moving target defense," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.