MACHETE: Multi-path Communication for Security

Diogo Raposo, Miguel L. Pardal, Luís Rodrigues, Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

***Abstract*—Communication through the Internet raises privacy and confidentiality concerns. Protocols such as HTTPS may be used to protect the communication, but occasionally vulnerabilities that may allow snooping on packet content are discovered. To address this issue, we present MACHETE, an application-layer multi-path communication mechanism that provides additional confidentiality by splitting data streams in different physical paths. MACHETE has to handle two challenges: sending packets over different paths when Internet's routing imposes a single path between pairs of network interfaces; splitting streams of data sent over TCP connections. MACHETE is the first to exploit MultiPath TCP (MPTCP) for security purposes. It leverages overlay networks and multihoming to handle the first challenge and MPTCP to handle the second. MACHETE establishes an overlay network and scatters the data over the available paths, thus reducing the effectiveness of snooping attacks. Mechanisms are provided to select paths based on path diversity.**

***Index Terms*—Multi-path Routing, Communication Confidentiality, Eavesdropping, Communication Privacy, MultiPath TCP**

I. Introduction

Sending information over the Internet has the disadvantage of making it vulnerable to eavesdropping by unauthorized third parties. This problem is especially important for organizations that handle critical data, such as governments, military, or healthcare. Communication protocols based on cryptographic mechanisms such as HTTPS and IPsec are the common solution to this problem. However, recent events show that it may be possible to break these protocols under certain conditions, and suggest that powerful adversaries may be able to do it if they access the encrypted data. For example, Adrian *et al.* presented a flaw in the Diffie-Hellman key exchange that allows downgrading the security of a TLS connection for a specified 512-bit group [1]. They claim that a nation-state may have the computational power to attack 1024-bit groups, which would allow decryption of many TLS channels over the Internet that implement this method.

We present MAChETe (*Multi-pAth Communication for sEcuriTy*), a means to mitigate the impact of such vulnerabilities. This system consists on using *MultiPath TCP* (MPTCP) [2]–[4] and *overlay networks* [5], [6] to split communication flows on different physical paths, possibly over a multihomed subnetwork [7], [8], as a defense-in-depth mechanism.

The rationale is that more effort is required to eavesdrop data split over several flows in comparison to a single flow. The problem addressed in this paper is, therefore, achieving additional communication *confidentiality* for critical data while still assuming confidence in the cryptography mechanisms.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

MACHETE has to handle two challenges. The first consists in sending packets over *different paths* when Internet's routing imposes a single path between a pair of source and destination network addresses. Overlay routing enables doing *application-layer routing*, allowing packets to deviate from the routing imposed at network level, by the Internet's routers and routing protocols. Overlay networks, in combination with multihoming, are used to create path diversity, allowing flows to be split over physically disjoint paths. Using a topology-aware decision algorithm, several *overlay nodes* are chosen, according to their location. Each node will create a single-hop overlay path to the destination, generating an overlay network.

The second challenge is to split the stream of data sent over a TCP connection. MPTCP is a recent extension of the TCP protocol that has the ability to distribute and send data among the different network interfaces of a device, e.g., the IEEE 802.3 ("wired", "Ethernet") and the 802.11 ("wireless", "WiFi") interfaces of a personal computer. However, MPTCP neither ensures the use of different physical paths, nor their diversity, as it was created mostly with performance in mind. The paths used by a MPTCP connection are imposed by the network interfaces of the source and destination hosts.

The combination of MPTCP with application-layer routing is itself a third challenge. Our objective is that MACHETE works at the *application layer*, without modifications to lower layers, but it has to route packets sent at transport layer under the control of MPTCP. MPTCP is a transport-layer protocol, so applications provide it source and destination IP addresses and ports. However, the overlay nodes have their own IP addresses and ports, unrelated to the previous ones. Therefore MACHETE has to play with the destination IP addresses and ports for communication to be possible.

The paper has three main contributions: (1) MACHETE is a system that improves communication confidentiality by splitting TCP data streams over diverse physical paths leveraging MPTCP, overlay networks, and multihoming; (2) It is the first work that leverages MPTCP for security and the first to combine MPTCP with application-layer routing and overlay networks; (3) Provides an experimental evaluation of MACHETE over the Internet in a wide-area deployment.

II. Background and Related Work

This section covers background and related work on the mechanisms used in MACHETE: MultiPath TCP, overlay

routing, and multihoming.

A. MultiPath TCP

MultiPath TCP (MPTCP) is an extension of the TCP protocol that enables endpoints to use several IP addresses and interfaces simultaneously when communicating [2], [3]. The protocol discovers which interfaces are available to use, establishes a connection, and splits the traffic among them. It presents the same programming interface as TCP, however the data is spread across several flows. The option field in the regular TCP protocol is filled with MPTCP data structures in order to inform the other end-point about the capability of implementing this protocol and to add flows to the communication. MPTCP has two important components on its configuration: path manager and packet scheduler.

The *path manager* is the module that handles how the flows are created in an MPTCP connection. The implementation of the protocol in Linux currently provides four schemes [4]: `default` does not create new flows, but accepts incoming; `fullmesh` creates a full-mesh of flows with all available interfaces/addresses in the device; `ndiffports` takes only one pair of IP addresses and modifies the source port to create the number of flows set by the user; `binder` uses the loose source routing algorithm of the Binder system [9].

The *scheduler* handles the distribution of the TCP packets (segments) over the flows, in close collaboration with TCP's congestion control mechanism [10]. MPTCP does not use a single congestion window as TCP, but one per flow. Similarly to TCP, the congestion control mechanism manages the size of each congestion window based on the round-trip time (RTT) of the flow and other factors (timeouts, reception of acknowledgments). The implementation of MPTCP for Linux by default fills the flow congestion window before starting to schedule packets on the next flow. Although in terms of performance it is important to take advantage of the throughput of the channels, in terms of splitting data for confidentiality it may be a disadvantage. In a communication composed by two flows where one has twice the throughput of the other, that flow will tend to send twice the amount of data of the other, which leads to a higher amount of data is susceptible to being spied upon. Linux's MPTCP implementation provides three scheduling modes: `default`, the one we just presented and the one with best performance; only uses another flow if the window of the flow in use does not allow sending data that is pending; starts sending using the flows with lower RTT; `fast round-robin` which uses sequentially all flows but fills the congestion window of a flow before starting with the next; `strict round-robin`, does real round-robin by sending the same amount of data through all the flows in sequence; waits for a flow to have free space to send a packet before scheduling the next one.

MPTCP is very similar to TCP in terms of security. Specifically, the RFC says that "The basic security goal of Multipath TCP (...) can be stated as: provide a solution that is no worse than standard TCP" [2]. There are a few works concerned with the security of MPTCP [11], [12].

B. Path Diversity

Path diversity can be achieved in multiple ways in a Multipath Communication. *Overlay Routing* and *Multihoming* are among some of the options.

Overlay routing allows the creation of a virtual network (an overlay network) on top of an already existing network infrastructure, like the Internet, without modifying it. The nodes of the network are hosts, i.e., machines that implement the network stack up to the OSI application layer. An overlay link may connect two nodes either directly or indirectly, through other nodes. These nodes route (forward) the packets at the application layer to the next or final node of the link. This adds a level of indirection in relation to the underlying OSI network layer topology. At the network layer the packets travel through the routes imposed by the Internet routing protocols, namely the Border Gateway Protocol v4 (BGPv4) [13]. At application layer the traffic may be deviated from these routes by sending it through overlay nodes in other locations. The original motivation for overlay routing is resilience [5], [6]. In case a network layer route is congested or faulty, routing done at the application layer may allow passing the traffic through other routes.

Another method of achieving path diversity is to use *multihoming*. This approach consists simply in having a customer network linked to two or more ISPs. Resilience and performance are the main advantages of this approach [14]. Different providers offer different performance levels to different parts of the network, so choosing the "right" provider will result in a performance increase [7], [8]. Akella *et al.* [8] evaluate the use of multihoming, using Hand-shake Round Trip Time (HRTT) as a measurement unit for data centers and enterprises. These studies [7], [8] conclude that simply by using two providers the performance is increased by at least 25% and that the improvements are very small beyond four providers.

III. MACHETE

MACHETE is an application-layer mechanism for improving communication confidentiality by splitting packets in different paths. It uses MPTCP over an overlay network to create multi-path communication. By setting up a network composed by several nodes it is possible to implement an overlay network, which consists of several links between the source and destination of a communication. MPTCP will use these overlay links to split the data to be transferred. Path diversity is sought by exploiting diversity between Autonomous Systems (ASs). MACHETE uses single-hop overlay routing as there seems to be no gain in using more hops, both in terms of performance and diversity [15].

The architecture of MACHETE is represented in Figure 1 and has three main components: Multi-path devices which communicate using MACHETE, that also can play the role of server (wait for connections) or client (start connections), similarly to TCP; Overlay nodes which are the nodes of the overlay network that forward messages on behalf of multi-path devices and create alternative communication paths; Multi-

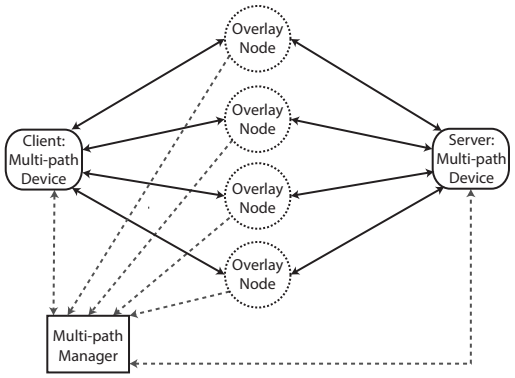

Fig. 1. The architecture of MACHETE. The solid lines represent data communication flows and the dashed lines control communication (e.g., node registration in the overlay network).

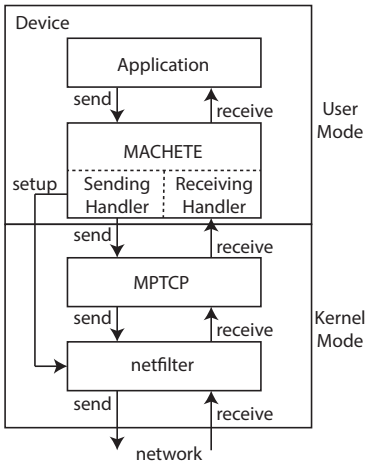

Fig. 2. MACHETE multi-path device architecture. MACHETE proper is a user level process running in a computer, which connects the application to the communication stack in the kernel (MPTCP) and netfilter (using `iptables`). MACHETE only establishes new rules when it creates a new connection (essentially an MPTCP connection).

path manager which is the component in charge of keeping track of the nodes that compose the overlay network.

This section presents MACHETE abstractly but in some places delves into the details of its implementation in Linux.

A. Threat Model

MACHETE is concerned about attacks against the confidentiality of data exchanged, so it considers passive attackers that eavesdrop on communication at certain physical locations. We assume that the attackers can eavesdrop on all packets at those locations, so confidentiality has to be achieved by reducing the locations where all traffic passes.

We assume that communications between the manager and each other component of MACHETE use default secure channels with a configuration which maximizes security, e.g., TLS using a 2048 bit key with elliptic curve Diffie-Hellman key exchange. Therefore the attackers can not compromise the manager's communication integrity and confidentiality.

We assume that the devices and nodes of the system are trustworthy, i.e., that they follow the protocol. This assumption has to be assured using proper security mechanisms, such as hardening, sandboxing, and access control. MACHETE is, however, prepared to recover from node crashes.

The multi-path manager might be a single-point of failure of the architecture, so it is replicated. We assume that a subset of the replicas can be compromised by an attacker or crash and we use a specific scheme to make overall multi-path manager tolerate these issues.

It is also important to notice that MACHETE has the objective of dealing with the most resourceful adversaries such as a nation-states. An adversary this resourceful can acquire control over several ASs in his area. Therefore it is important that the nodes are placed in a wide geographic area. Beyond that, it can also be stated that the closer an attacker is to the source or destination of the communication, the easier it is to find a single point of interception[1]. This being, multihoming is a very important component of MACHETE's deployment as it is discussed further ahead in this document.

[1]In this case, a autonomous system routing many or all of the data streams in a multi-path communication is considered a single point of failure

B. Multi-path Manager

The multi-path manager is the component that contains information about every entity in the network. Its function is to register every node and device addresses and to provide that information to devices that aim to communicate.

The multi-path manager was not developed from scratch but instead is a *tuple space* that implements Linda's generative coordination model [16]. A tuple space is a repository of data items called *tuples* and provides mainly three operations: insert tuple (`out`), read tuple (`rd`), and remove tuple (`in`).

MACHETE uses a specific tuple space called DepSpace [17], [18]. DepSpace is replicated in order to tolerate faults in some of the replicas. Specifically, it continues to operate correctly despite the failure of up to f out of $3f + 1$ replicas (typically 1 out of 4). DepSpace is Byzantine fault-tolerant, so it provides its service correctly even if f replicas are compromised or fail arbitrarily. Whenever server multi-path devices and overlay nodes start to run, they register with the multi-path manager by inserting a tuple on the tuple space.

C. Multi-path Device

A multi-path device is designated as a computer that uses MACHETE to communicate. The architecture of such a device is represented in Figure 2. This component dynamically establishes paths and splits the packets among them.

After a device registers itself on the multi-path manager, the process of transferring a stream of data (e.g., sending a file) is composed of three steps: *path setup*, *data transfer* and *path tear down*. Figure 3 represents this process. Next we describe each of these steps, dividing the first in two substeps.

MPTCP requires devices to have several network addresses to create more than one flow. If the device has several physical

interfaces, possibly connected to more than one provider – *multihoming* –, each one has an IP address. If that is not the case or that number of addresses is not enough, more than one address can be assigned to each interface, e.g., using Linux's virtual network interfaces [19]. Having two addresses (in total) on each device is enough to establish a network composed of four paths, which in general is enough to achieve the objective.

1) Path setup – choosing overlay nodes: The process starts by querying the multi-path manager about the available overlay nodes. Although the manager replies with all nodes available in the network, the number of nodes to be used by a certain connection, N_n, is a configuration parameter.

The overlay nodes are chosen taking into consideration the path *diversity* they provide. If there are several paths with the same diversity, the path with best *performance* (e.g., lowest RTT) is chosen. In the current version of MACHETE, the metric of diversity among two paths used is the number common ASs on both paths (higher number means worse diversity). For a path, the ASs are obtained using layer-4 traceroute [20], which provides precisely the ASs of the nodes along a path. The metric of performance is the RTT, measured using the `tokyo-ping` tool, which avoids some anomalies in `ping` [21]. When available, multihoming tends to improve diversity as the first ASs along the path will already be different, whereas with single-homming the opposite is true.

In MACHETE the path manager is set to `fullmesh`, to allow defining the number of flows in a way that makes MPTCP use the number of overlay nodes defined (N_n). This manager will create a network mesh composed by all the available interfaces/addresses in both the source and destination.

To balance the data among all nodes and obtain the expected confidentiality, the best packet scheduler is `strict round-robin`. This scheduler is configured with the number of packets sent in each flow before passing the turn to the next flow. To reduce the information sent in each flow (thus in each path), this parameter is set to 1. The `fast round-robin` scheduler can also be used if the communication is encrypted and the amount of bytes sent is high enough to ensure that not all communication passes in the same node, as it is not possible to decrypt data if it is not complete. The amount of bytes sent being enough or not to make the communication pass in more than one node is analyzed in Section IV-C2.

2) Path setup – managing addresses and ports: As already pointed out, the combination of MPTCP with application-layer routing is challenging. MACHETE works at application layer but it has to route packets sent by, and under the control of, a lower layer protocol: MPTCP, at transport layer.

Similarly to what happens with TCP, in MPTCP all packets sent over a connection take two pairs of IP addresses and port numbers, one for the source device, another for the destination (the difference in relation to TCP is that source and destination may have more than one address/port pair). However, in MACHETE the destination address and port may have to be different: (1) if the packet is leaving the sender device, the destination address/port should be those of the overlay node for the packet's flow; (2) if the packet is leaving an overlay node, the destination address/port should be those of the destination device and the source address/port should be those of the overlay node; (3) if the packet is returning to the overlay node, the destination address/port should be those of the source device and the source address/port should be those of the overlay node.

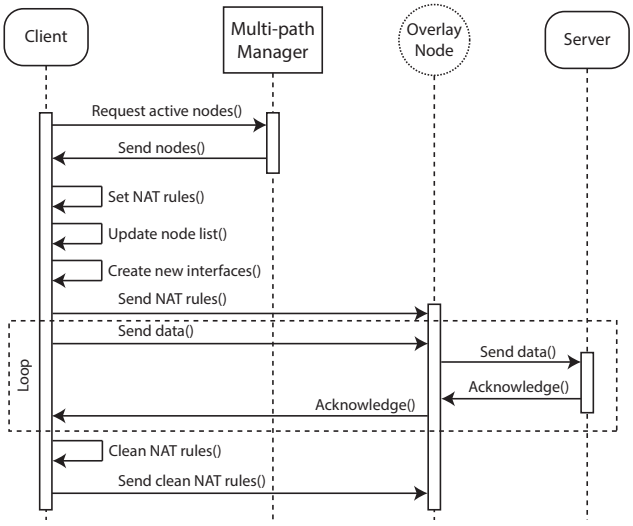

Fig. 3. MACHETE communication setup and data transfer example with a single overlay node.

The application requires MACHETE, thus also MPTCP, to send packets to the destination address and port. When a device does the setup of a path, it has to force these alternative addresses and ports to be used. To do it MACHETE leverages Linux's *netfilter* framework and the `iptables` command [22]. This framework allows doing network address translation (NAT), packet filtering, and other forms of packet handling. MACHETE uses it for network address translation.

When a path is setup, the `iptables` command is used to tell netfilter to change the destination IP address and port by those of an overlay node, depending on the flow (case (1) above; arrow *setup* in Figure 2). MPTCP inserts the destination IP address/port in the packets, but netfilter exchanges them before they are transmitted into the network. The `iptables` command inserts NAT rules for that purpose in the `output` chain, which is the set of rules applied to traffic being sent by a computer. For each link, a NAT rule is set[2].

Once this is done, the device informs each node about the rules they have to establish. In the overlay node it is necessary to route the traffic in both directions: when forwarding to the server (case (2) above) and when returning to the client (case (3) above). As soon as all nodes confirm that the rules are set, the data transfer may begin.

Figure 3 shows a time diagram that represents this process with a single overlay node.

[2]The format of the `iptables` rule is: `iptables -t nat -A -p tcp -s <source address> -d <destination address> -j DNAT --to-destination <new destination address>`

3) Data transfer: MACHETE uses MPTCP to establish a connection to the destination device. The client application will create a socket and provide it one of the server's address/port pairs; the MPTCP protocol will handle the passive creation of the flows. Despite the fact that netfilter modifies the destination addresses to deviate the connection's packets through the overlay nodes, the connection and each of its flows end up established similarly to what would normally happen with MPTCP.

This connection has two data streams, one in each direction, so that the client and server can send data to each other. This is represented in Figure 2 through the send and receive arrows. Notice again that the scheduler should be set to `strict round-robin`, otherwise MPTCP will fill each flow until its congestion window is full instead of sending packets using all flows, which is not desirable from the confidentiality point of view.

Each packet will suffer changes on its source and destination address twice: first in the source device, second in the overlay node. The same will happen to the acknowledgement packets.

4) Path tear down: To terminate a connection, the client device notifies the nodes that compose the overlay paths to remove the rules. The overlay device is listening on a specific port for receiving this indication, so that the packets destined to the node itself are never re-routed. Again, this device waits for all nodes to reply before removing its own rules. After all the steps are done the communication can be declared as finished. If the client fails to inform the nodes about the rules removal, the rules can stay established, since it is specific for a pair of source and destination addresses and, therefore, does not modify other connection's correct behavior or the possibility for the same source to create an identical connection.

D. Overlay Node

The overlay node is the component that plays the role of application-layer router, i.e., which forwards the packets received from the client device to the server device and vice-versa.

Overlay nodes receive from clients NAT rules and add/remove them from netfilter. These rules are set, again, with the `iptables` tool, this time using the `prerouting` and `postrouting` chains. The first chain leverages the changes on the traffic immediately after it was received by and interface and the second leverages the changes right before it leaves that interface. For each overlay network, four rules are established, two to change the source and destination when forwarding to the destination and two when forwarding to the source, as mentioned above in cases (2) and (3).

IV. Experimental Evaluation

This section presents the evaluation of MACHETE. We placed hosts in the Amazon AWS EC2 service [23] in nine different regions (Ireland, Frankfurt, North Virginia, California, Oregon, Tokyo, Seoul, Singapore and Sydney) and one in Portugal. We used up to 8 overlay nodes, one in each of the AWS regions, except for Ireland that contains the server.

TABLE I
Least diverse pair of paths in terms of number of common ASs with the single-home configuration. The paths are designated by the location of the overlay node.

Pair of paths	Common ASs	Common ASs (except first 7)
Singapore, Tokyo	13	6
Frankfurt, Seoul	12	5
Frankfurt, Tokyo	12	5
California, Seoul	12	5
California, Tokyo	12	5
Oregon, Tokyo	12	5
Seoul, Tokyo	12	5

TABLE II
Least diverse pair of paths in terms of number of common ASs with the dual-home configuration, in comparison to the single-home configuration. The paths are again designated by the location of the overlay node. In the dual-home configuration the left path uses the original connection and the right the 4G connection.

Pair of paths	Common ASs single-homed	Common ASs dual-homed
Oregon, Sydney	10	3
Oregon, Tokyo	12	3
Oregon, Seoul	11	2
Sydney, Tokyo	10	2
Frankfurt, California	9	1
Frankfurt, Oregon	10	1
Frankfurt, Seoul	12	1

Moreover we placed the client in the Portugal node. Therefore, between the client and server there are 8 single-hop overlay paths: one per overlay node.

Recall that the objective is to provide confidentiality by splitting communication over physically diverse paths with an acceptable performance. Therefore, the evaluation provides an assessment of the diversity in our scenario, presents a performance benchmark of the system, and analyses the confidentiality achieved.

A. Diversity

As stated before, confidentiality is only achieved if the paths are topologically disjoint, as attackers eavesdrop on traffic at certain locations (Section III-A). The approach used to verify the topology of the paths is to trace each route's chain of ASs from the source device to each node and from that same node to the destination. For that purpose we use layer-4 traceroute (i.e., the `lft` tool).

Table I shows the number of common ASs in the pairs of paths with highest value, between the 8 single-hop overlay paths, where each is designate by the location of the overlay node. There are at least 7 ASs in common in all paths leaving the client (Portugal). The reason for this lack of diversity is the fact that we did not use multihoming. Moreover, several ASs belong to Amazon, as also expected.

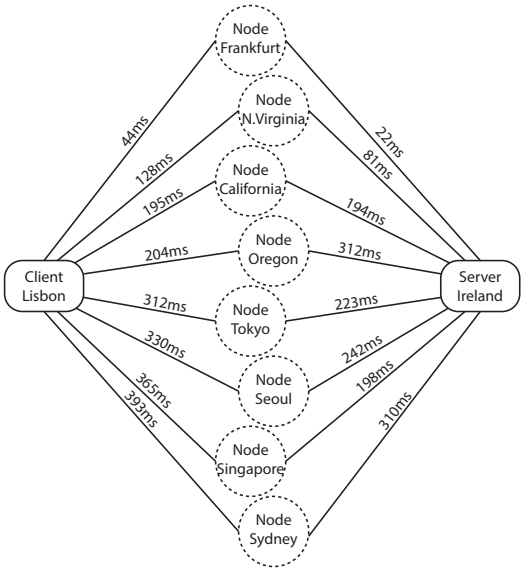

Fig. 4. Latencies between the Portugal host and the EC2 hosts used for the experimental evaluation.

We did an additional experiment to confirm that multihoming is beneficial in terms of diversity. We connected a second interface of the client device to a public provider of 4G service through a smartphone, then we used `lft` to obtain the ASs traversed by the paths. As shown in Table II, using multihoming provides an evident diversity, where the common nodes are again part of AWS's network. Notice that we used this multihoming configuration only for this test; the single-home configuration was used in all the experiments presented in the following sections. Multihoming is revealed to be a key component for MACHETE to achieve path diversity.

B. Performance

The performance evaluation considers three different aspects: the impact of adding paths on the delay of transferring files, the performance with diverse paths when transferring files, and the performance of path set up and tear down.

Figure 4 provides some insight on the network by showing the latencies between the hosts in the different locations, obtained with the `tokyo-ping` tool.

MACHETE forced MPTCP to use all the paths defined for every experiment by changing the number of IP addresses at the client (i.e., Portugal): 2 addresses for 2 paths, 3 addresses for 3 paths, etc. The client had a single network interface; the server had a single interface and a single IP address. All measurements were repeated 30 times.

1) Impact of adding paths: The evaluation consisted in observing the performance when paths (equivalently, nodes) were added one by one based on latency to the cluster: first in Frankfurt, next in N. Virginia, California, Oregon, Tokyo, Seoul, Singapore and finally Sydney (that has a the highest latency, as observed in Figure 4). The size of the files varied from 1 Byte to 1 GByte. In this experiment we used the `fast round-robin` scheduler to improve performance (and the `fullmesh` path manager which is fixed for MACHETE).

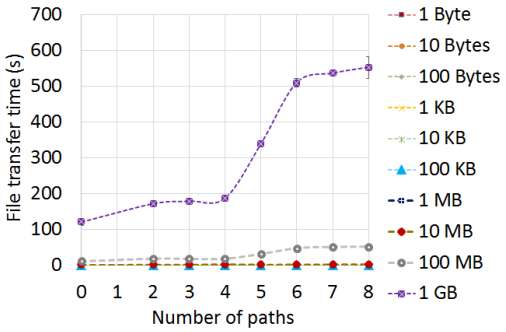

Fig. 5. Time to transfer a file versus number of paths. 0 paths means a normal TCP connection.

TABLE III
AVERAGE TIME OF SENDING FILES USING TWO NODES WITH TWO TYPES OF ROUND-ROBIN SCHEDULERS, COMPARED TO A NORMAL TCP CONNECTION. ALL VALUES ARE PRESENTED IN MILLISECONDS. EACH EVALUATION WAS REPEATED 30 TIMES.

File size	TCP	Fast r.-r.	Strict r.-r.
1 B	49	81	97
10 B	49	95	94
100 B	49	87	98
1 KB	49	94	90
10 KB	49	181	122
100 KB	49	265	201
1 MB	304	382	409
10 MB	1644	2295	3106
100 MB	11556	18836	23452
1 GB	121069	172215	218332

Figure 5 shows the values obtained for the time to transfer files of all sizes and from 2 to 8 paths, plus using a standard TCP connection (with no overlay nodes), and includes 95% confidence intervals, although most are too small to be visible.

The figure shows that splitting the packets in up to four different paths does not generate considerable overhead on the communication. From the fifth the duration increases due to the overlay nodes that compose the network at that point being farther away from both the source and destination. It is important to note that to achieve a physically disjoint network, using more nodes in the overlay network will result in selecting these further away from the source and destination.

2) Performance with diverse paths: Considering the diversity achieved in each of the eight regions used on the previous tests, this evaluation considers the two paths with highest diversity, i.e., those with overlay nodes at Frankfurt and California.

Table III, shows the overhead of using these two nodes, in comparison to a normal TCP stream. As shown, there is an overhead of 42% when using the `fast round-robin` scheduler and of 80% when using the `strict round-robin` scheduler. This overhead is the result of sending traffic through a node that is geographi-

cally distant from the source and the destination, California. When using a round-robin packet scheduler the whole multi-path connection is conditioned to each path's throughput. In fact, for the `strict round-robin` scheduler, the whole throughput is highly dependent of the path with the smallest bandwidth or highest congestion, since it waits for this channel to have free window space before sending to the next one.

3) Path set up and tear down: Figure 6 shows the time for setting up and tearing down the overlay paths. The current MACHETE implementation is suboptimal in the sense that both the setup and tear down phases are executed sequentially. According to the location of the node, this time will vary, however, as it can be seen, it always takes longer than one second, but never more than two in our scenario.

C. Confidentiality

The usual way of considering confidentiality in the security and cryptography literature is by relying on cryptographic protocols. For example, in protocols like IPsec AH/ESP or TLS, confidentiality is guaranteed as far as no vulnerabilities exist in the protocol design, implementation, and configuration. In this work we do not aim to provide such guarantees but to improve confidentiality in case communication is eavesdropped, (1) either it is not encrypted or (2) if it is encrypted but there is a vulnerability. This means that confidentiality was not studied in an absolute perspective, even though it is possible that in the second scenario, it might mitigate cryptography vulnerabilities to provide full confidentiality.

This different way of considering confidentiality led us to transmit a visual intuition of the MACHETE approach: an image is transmitted over MACHETE where an eavesdropper has access to one of the flows and reconstructs the image with that data. For the figures we did the reconstruction assuming the adversary managed to guess the metadata (figure size, color depth, etc.) even if the captured flow did not contain it.

When evaluating the confidentiality that MACHETE offers it is necessary to remember the operation of MPTCP. The most important factor is the scheduling that is used. MPTCP implements different types of scheduling, however, splitting data in packets in a round-robin fashion is the best approach to achieve confidentiality. The multi-path protocol implements two types of round-robin: `fast round-robin`, which takes advantage of the whole throughput of that channel,

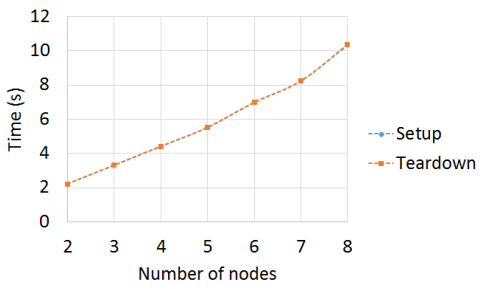

Fig. 6. Time for setting up and tearing down the overlay paths versus number of overlay nodes used.

and `strict round-robin`, that waits for the next channel to have window space before sending the packet. The former is expected to perform faster, but the second to provide access to less data to an eavesdropper.

We evaluate two aspects of confidentiality: the effect of the scheduling algorithm, and the effect of the file size.

1) Effect of the scheduling algorithm: Figures 7b and 7c show the different amount of data captured by two of four channels when sending the bitmap picture (the Linux penguin) shown in Figure 7a, with both types of round-robin scheduling. The channels shown in each figure are the ones that receive the most distinct amount of data, i.e., the one that receives the most (on the left) and the one that receives the least (on the right).

As shown on Figure 7b, using `strict round-robin` it is possible to notice that both channels receive approximately the same amount of data, resulting of the even distribution of packets. However, a pattern can also be noticed on its reconstruction.

Figure 7c shows the results of capturing the data when the `fast round-robin` scheduler is used. As expected, the flow where less data was transferred was the one passing through Sydney's node, the one with lowest throughput. As mentioned before, this scheduler depends on the throughput of each channel when distributing data, since a channel with better throughput has a larger congestion window to be filled.

In the standard MACHETE configuration, the result is the balance between flows observed in Figure 7b. The performance in the two cases was different, though. By filling the congestion windows with `fast round-robin` scheduling, the file that had 17MB was sent in 4 seconds. By using the strict round-robin the file took 88 seconds to be transferred, which is much slower. The first mode achieves a throughput of 34 Mb/s, whereas the second a mere 1.9 Mb/s.

In short both approaches have their advantages and disadvantages: the first one takes longer and might be susceptible to easier data reconstruction, but provides a good control on how the packets are distributed; the second has its packet distribution dependent of each flows' throughput, but transfers the files faster.

2) Effect of the file size: Another factor to take into account is the size of the files sent. At this point it is important to remember MPTCP's behavior when creating new flows. The first flow does not wait for the creation of new flows to start transferring data. This means that for very small files ($<$10KB) MPTCP does not split the packets through any new flows, since this data is sent before any new flow can be established for the stream. Regarding larger files, it is only necessary to experiment with the fast round-robin mode, since the strict round-robin is not influenced by congestion window sizes and, therefore, the sizes are not a factor to take into account.

Figure 8 shows the results of capturing the data transferred in the flow with highest throughput (which is the same as mentioned above of 34 Mb/s), when sending the same image with different sizes: from 1 MB to 17 MB. As it can be

(a) The original image sent.

(b) The data sent over the flows is balanced (strict round robin).

(c) The data sent over the flows is unbalanced (fast round robin).

Fig. 7. Original image and two reconstructions considering the eavesdropper has access to 2 of the 4 flows in each case.

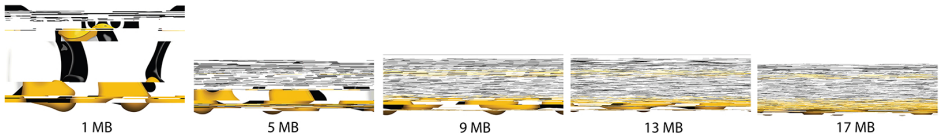

Fig. 8. Results of capturing the data transferred by the flow with most throughput. The same image was sent with different sizes.

observed, larger files have stronger resistance to eavesdropping as data is better split among the paths.

V. Conclusions

MACHETE is a first effort on providing confidentiality to communications by splitting the packet flows among different physical paths. By establishing dynamic overlay networks, composed by several paths with a single overlay node it was possible to provide physical path diversity. Using MPTCP it was possible to develop a system that transfer data streams (instead of isolated packets) without compromising performance. We evaluated the performance and confidentiality achieved by our implementation, showing that, not only it prevents the attacker from accessing considerable amounts of data, in the case it is trying to spy on the communication, but also provides different tradeoffs between confidentiality and performance. We believe, that efficiently splitting the communication over physically disjoint channels is the key to maintain confidentiality.

Acknowledgements

This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. Vandersloot, E. Wustrow, and S. Z. Paul, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[2] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development, IETF RFC 6182," 2011.

[3] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: from theory to practice," in *Networking 2011*. Springer, 2011, pp. 444–457.

[4] C. Paasch, S. Barre *et al.*, "Multipath TCP in the Linux kernel, available from http://www.multipath-tcp.org."

[5] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 131–145.

[6] Y. Amir and C. Danilov, "Reliable communication in overlay networks," *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 511–520, 2003.

[7] A. Akella, B. Maggs, S. Seshan, and A. Shaikh, "On the Performance Benefits of Multihoming Route Control," *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 91–104, 2008.

[8] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A measurement-based analysis of multihoming," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 353–364.

[9] L. Boccassi, M. M. Fayed, and M. K. Marina, "Binder: a system to aggregate multiple Internet gateways in community networks," in *Proceedings of the 2013 ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access*, 2013, pp. 3–8.

[10] D. Wischik and C. Raiciu, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 99–112.

[11] J. Díez, M. Bagnulo, F. Valera, and I. Vidal, "Security for multipath TCP: a constructive approach," *International Journal of Internet Protocol Technology*, vol. 6, no. 3, pp. 146–155, 2011.

[12] O. Bonaventure, "MPTLS: Making TLS and multipath TCP stronger together," *IETF, Individual Submission, Internet Draft draft-bonaventure-mptcp-tls-00*, 2014.

[13] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4), RFC 1771," 1995.

[14] J. He and J. Rexford, "Toward Internet-wide multipath routing," *IEEE Network*, vol. 22, no. 2, pp. 16–21, 2008.

[15] J. Han, D. Watson, and F. Jahanian, "Topology aware overlay networks," *Proceedings of the IEEE INFOCOM*, vol. 4, pp. 2554–2565, 2005.

[16] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programing Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[17] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Apr. 2008, pp. 163–176.

[18] "DepSpace." [Online]. Available: https://github.com/bft-smart/depspace

[19] J.-S. Kim, K. Kim, and S.-I. Jung, "Building a high-performance communication layer over virtual interface architecture on Linux clusters," in *Proceedings of the 15th ACM International Conference on Supercomputing*, 2001, pp. 335–347.

[20] "Layer four traceroute (lft) project." [Online]. Available: http://pwhois.org/lft/index.who

[21] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From Paris to Tokyo: On the suitability of ping to measure latency," in *Proceedings of the 2013 ACM Internet Measurement Conference*, 2013, pp. 427–432.

[22] R. Russell, "Linux 2.4 packet filtering howto, revision 1.26," Jan. 2002.

[23] "Amazon Web Services." [Online]. Available: https://aws.amazon.com/