MT6D: A Moving Target IPv6 Defense

Matthew Dunlop*† Stephen Groat*† William Urbanski† Randy Marchany† Joseph Tront*
*Bradley Department of Electrical and Computer Engineering
†Virginia Tech Information Technology Security Office
Virginia Tech, Blacksburg, VA 24061, USA
Email: {dunlop,sgroat,urbanski,marchany,jgtront}@vt.edu

***Abstract*—The Internet Protocol version 6 (IPv6) brings with it a seemingly endless supply of network addresses. It does not, however, solve many of the vulnerabilities that existed in Internet Protocol version 4 (IPv4). In fact, privacy-related crimes in IPv6 are made easier due to the way IPv6 addresses are formed. We developed a Moving Target IPv6 Defense (MT6D) that leverages the immense address space of IPv6. The two goals of MT6D are maintaining user privacy and protecting against targeted network attacks. These goals are achieved by repeatedly rotating the addresses of both the sender and receiver. Address rotation occurs, regardless of the state of ongoing sessions, to prevent an attacker from discovering the identities of the two communicating hosts. Rotating addresses mid-session prevents an attacker from even determining that the same two hosts are communicating. The continuously changing addresses also force an attacker to repeatedly reacquire the target node before he or she can launch a successful network attack. Our proof of concept demonstrates the feasibility of MT6D and its ability to seamlessly bind new IPv6 addresses. We also demonstrate MT6D's ability to rotate addresses mid-session without dropping or renegotiating sessions. Since MT6D operates at the network layer of the protocol stack, it provides a powerful moving target solution that is both platform and application independent.**

***Index Terms*—moving target defense, IPv6, security, privacy**

I. Introduction

Information compromise is a serious issue for military communications systems. Due to infrequently changing, or static, network addresses, an attacker can eavesdrop on information exchanges or attack specific hosts. For example, static nodes are susceptible to Denial-of-Service (DoS), man-in-the-middle (MITM), replay attacks, and other similar attacks. If an attacker can disrupt a target user's availability, time-sensitive communications could be lost. Similarly, attackers able to hijack sensitive data exchanges can glean information or even modify network traffic.

Many security devices exist to defend against these types of attacks (e.g., firewalls, etc.). No matter how effective the protection mechanism, an attacker has unlimited time to attempt to find a weakness if the hosts it protects are static. Dynamic addressing, on the other hand, limits an attacker's time to find a vulnerable attack vector. If an attacker targets a node, the attack is viable only for the time it takes for the address to rotate again. Combining dynamic addressing with current defense in depth strategies provides a powerful moving target defense that grants an attacker little gain for massive amounts of effort.

Dynamic addressing also adds privacy and anonymity to communicating hosts. Since addresses rotate, it is difficult for attackers to determine the locations and identities of communicating hosts. If addresses rotate often enough, attackers cannot even piece together enough network traffic to determine what information is being passed. The problem with implementing a robust dynamic addressing strategy in IPv4 is that the address space is small and densely populated. IPv6 solves these issues by increasing the address size to 128 bits.

Our strategy, the Moving Target IPv6 Defense (MT6D), leverages the vast address space of IPv6 to implement dynamic addressing. The remainder of this paper describes the MT6D protocol. Section II provides the motivation for our work. The design of MT6D is presented in Section III. Section IV introduces some implementation possibilities. Other research related to modifying network addresses, is discussed in Section V. Section VI describes the prototype test configuration, while Section VII presents the analysis and results of our testing. We discuss some limitations of MT6D in Section VIII. In Section IX, we discuss future work and conclude.

II. Motivation

We mentioned the threats that can result from information exposure in Section I. Adoption of IPv6 brings with it new threats in addition to all the existing Internet Protocol (IP) threats. For example, StateLess Address AutoConfiguration (SLAAC) in IPv6 exposes hosts to privacy and anonymity related attacks. This is especially true with 64-bit Extended Unique Identifier (EUI-64) addresses that assign a static host address, or interface identifier (IID), to nodes that remains static across different subnets. Static IIDs provide third parties with the ability to globally track users on the Internet [1]. Address tracking is not the only concern. Hosts are also susceptible to traffic correlation, meaning that a third party can piece together packets from multiple sessions to map human users to network traffic. Potentially more damaging is that an attacker can easily find and attack hosts regardless of where they connect to the Internet. MT6D was designed to prevent all of these types of attacks.

Certain network attacks are prevented by MT6D due to the nature of dynamically rotating network addresses. These attacks include, but are not limited to, address-based DoS and MITM attacks. MT6D prevents these attacks by rotating the addresses an attacker would use to target the victim. Each of these attacks is accomplished by targeting a specific network address. When the host address changes, the targeted victim disappears.

MT6D also preserves hosts' privacy. This is accomplished primarily through dynamic obscuration of sender and receiver addresses. Dynamic obscuration prevents a third party from pinpointing the identity of either communicating host. Since a third party cannot identify the host, tracking is prevented. By tracking, we mean that a third party is capable of pinpointing the geographic location of a node based on the network it associates with. The dynamic obscuration of addresses also prevents network traffic correlation. A third party may be able to capture a few packets communicated between two unknown hosts, but not enough to determine the nature of the network traffic. Since both communicating host addresses can change multiple times within the course of a single session, a third party will have difficulty linking previously captured packets with newly observed packets. To further hide the nature of network traffic, each packet can optionally be encrypted using a symmetric key shared by the two communicating hosts. The incorporation of encryption prevents any third party from using payload analysis in an attempt to correlate network traffic. Encrypting packets in this fashion also protects the identities of hosts when authenticating network traffic.

III. Design

MT6D provides a means for hosts to communicate with each other over the public Internet while protecting from targeted network attacks and maintaining anonymity. This section describes the system design.

A. Key Exchange and Usage

Two hosts using MT6D share a symmetric key for the purpose of address obscuration and encryption. The most secure way to generate and exchange symmetric keys is out-of-band. Symmetric keys can also be generated in-band using public key encryption. There is a risk that a third party may observe an in-band exchange, which could tip off a potential attacker as to the identities of the two communicating hosts.

B. Dynamic Addressing

Dynamic addressing modifies the network and transport layer addresses of the sender and receiver nondeterministically. MT6D is capable of dynamically changing these addresses to hide identifiable information about a host. A key feature of MT6D is that this obscuration can be made mid-session between two hosts without causing the additional overhead of connection reestablishment or breakdown. Changing addresses mid-session protects communicating hosts from an attacker being able to collect all packets from a particular session for the purpose of traffic correlation.

1) Dynamic IID Obscuration: MT6D IIDs are computed using a host's EUI-64 IID [2], a shared session key, and a timestamp. These three values are concatenated and hashed. The obscured IID is constructed from the leftmost 64 bits of the hash (bits 0-63) and has the form:

$$IID'_{x(i)} = H[IID_x || K_S || t_i]_{0\to 63} \quad (1)$$

where $IID'_{x(i)}$ represents the obscured IID for host x at time t_i, IID_x represents the unobscured IID of host x, K_S represents the shared symmetric key, and t_i represents the time at instance i. The leftmost 64 bits of the hash value are denoted by $H[\cdot]_{0\to 63}$. The MT6D IPv6 address is then formed by concatenating the host's subnet with $IID'_{x(i)}$.

In addition to dynamically obscuring IIDs, ports must also be obscured. MT6D includes two techniques to dynamically obscure port numbers. The first technique allows the host to specify a port range that more closely mimics normal network traffic. The second technique obscures port numbers using the unused bits of the hash calculation in Equation 1.

2) Time Incrementation: Time T will increment at t_i intervals. The frequency at which t_i increments can be no smaller than twice the single-trip time $(2 \cdot STT)$ of a packet sent between a sender and receiver. It is necessary to have $t_i > (2 \cdot STT)$ in order for the sender of the packet to be able to predict with reasonable accuracy the value of t_i when the receiver receives the packet. By making the minimum rotation twice the STT, a packet created at the end of the first STT can be sent using t_i. A packet sent within one STT of the next address rotation will be sent using the t_{i+1} address. This is to ensure the packet arrives at the destination using the proper time required to accurately compute the original addresses.

3) IID Rotation: Hosts using MT6D will rotate to the next dynamic address at every increment of t_i. MT6D uses Equation 1 to recalculate both the sender and receiver IIDs of each communicating pair at each time increment . MT6D will also purge the IIDs for t_{i-1} to prevent any connection attempts from malicious third parties. Purging past IIDs also prevents replay attacks.

4) IID Notification and Lifetime: Each time a host recalculates its obscured IID, it must notify the local router of its new IPv6 address so that packets can be properly forwarded. This notification occurs through the use of the Neighbor Discovery Protocol (NDP) [3]. The NDP serves two purposes. First, Neighbor Solicitation and Advertisement Messages verify the new address does not conflict with a preexisting address on the subnet by performing Duplicate Address Detection (DAD) [4]. The second purpose is to ensure that the router has the new MT6D IPv6 address for proper delivery of incoming packets.

At any given time, routers maintain multiple IPv6 addresses that correlate with a single obscured host. This is beneficial to minimize packet loss. Future obscured IPv6 addresses are precalculated and advertised so that routers add the new host address to their neighbor cache [3] prior to packets arriving using that address. To accommodate this requirement, host x will notify the router of $IP'_{x(i+1)}$ at time t_i. Addresses for previous time increments will be purged from MT6D tables to prevent replay attempts. Purging must be done at the host since previously discarded addresses will remain in the router's neighbor cache until no response is received after MAX_UNICAST_SOLICIT solicitations [3].

The host will store two obscured IPv6 address states. The two states are $IP'_{x(i)}$ and $IP'_{x(i+1)}$. The state $IP'_{x(i)}$ corresponds to the current computed obscured IPv6 address. This is considered the active state. The state $IP'_{x(i+1)}$ corresponds to the obscured IPv6 address that will be utilized at the next time increment. The state at t_{i+1} is stored but not used until t_i increments to the next time interval. It is precalculated to verify the validity of $IP'_{x(i+1)}$ within the subnet prior to time t_i. As already discussed, the state $IP'_{x(i-1)}$ is purged from the host. Any packets delayed past the active state will be discarded and handled according to the appropriate transport layer protocol of the original packet.

C. MT6D Tunneling

Rather than rewriting each original packet using the obscured addresses, the original packet is encapsulated in a MT6D tunnel. The benefit of retaining the original packet is that any established end-to-end connections between the source and destination are retained. Not only does the application of MT6D become transparent to the host, but also the MT6D connection can have different settings.

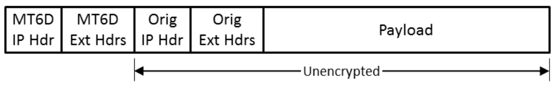
A MT6D packet is formed by first overwriting the source and destination addresses from the original packet. The Ethernet frame is also overwritten to anonymize the Media Access Control (MAC) addresses. The entire packet is then prepended with a new IPv6 header, which we refer to as a MT6D header. The MT6D header is formed using the dynamically obscured source and destination addresses discussed in Section III-B. The MT6D packet is also configured to use a transport layer protocol that supports connectionless communication.

In our implementation, each packet is encapsulated using the Unreliable Datagram Protocol (UDP) to prevent Transmission Control Protocol (TCP) connection establishment and termination from occurring every time a MT6D address rotates. Encapsulating packets as UDP has a minimal effect on the transport layer protocol of the original packet. Since transport layer protocols are end-to-end, decapsulation will occur before the host processes the original packet. A session using TCP will still exchange all required TCP-related information. This information will simply be wrapped in a MT6D UDP packet. Any lost packets that were originally TCP will be retransmitted by the end host after a retransmission timeout occurs.

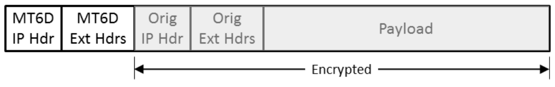
1) Unencrypted Tunnel: By default, MT6D tunnels the original packets unencrypted as illustrated in Fig. 1(a). Since the source and destination addresses are stripped from the original packet header, address tracking is not feasible. Unencrypted tunnels do not prevent traffic correlation since the remainder of the original headers and payloads stay intact. Traffic correlation does, however, require deep packet inspection since any relevant header fields are embedded within the MT6D packet payload.

The default implementation of MT6D also prevents a number of network attacks. For example, a host implementing MT6D is protected against targeted network-layer DoS attacks. Since source and destination addresses change constantly, an attacker cannot pinpoint a victim. If an attacker is able to pinpoint a victim, the duration of the attack is limited by the address rotation interval.

2) Encrypted Tunnel: MT6D provides the option of encrypting each original packet before appending it with the MT6D header. An example of a MT6D encrypted packet



(a) Unencrypted tunnel



(b) Encrypted tunnel

Fig. 1. Original IPv6 packet encapsulated within a MT6D tunnel.

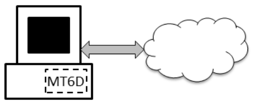
is shown in Fig. 1(b). By encrypting the original packet, a third party is unable to glean any useful information from the packet. For example, if the original packet is sent using TCP, the header gets encrypted so that a third party cannot attempt to correlate network traffic using the TCP sequence numbers. Additionally, the nature of the network traffic is also kept private through encryption. Encrypted tunnels also provide authentication privacy by wrapping the original authenticated packet inside an encrypted MT6D packet. Similar to unencrypted tunnels, MT6D using encrypted tunnels provides protection from targeted network attacks.

D. Architecture

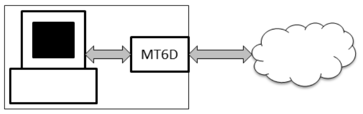
The architecture for a single MT6D host is comprised of an encapsulator and decapsulator. This architecture is mirrored by the host at the other end of the MT6D tunnel. Each packet is transmitted by the sender and sent to an inward-facing MT6D Network Interface Controller (NIC), which directs all incoming packets to the MT6D encapsulator.

The MT6D encapsulator transmits all outbound packets. Upon receipt of a packet, the encapsulator checks to see if a MT6D profile exists for the sender/receiver pair. The encapsulator maintains a table of all valid MT6D destinations that a sender trusts. These entries are referred to as profiles. Each profile includes the shared symmetric key that is valid between the host and each receiver. If no profile exists, the packet is treated as non-MT6D traffic, also referred to as unsupported traffic. Unsupported traffic is immediately forwarded to the nearest gateway device. It is necessary to allow unsupported traffic to facilitate communication with hosts that do not use MT6D (e.g., some web servers). Packets that match profiles have the source and destination network addresses overwritten in the packet header. They are then checked to see if encryption is desired. The final step is to pass the packet to the outward-facing NIC and transmit it.

The MT6D decapsulator receives all inbound packets via an outward-facing NIC. Each packet is checked for a MT6D profile. Those packets that do not match profiles are considered unsupported and delivered immediately to the host. Packets that match MT6D profiles have the tunnel header stripped off and are decrypted, if necessary. The source and destination addresses are rewritten to the original packet header, and the packet is delivered to the host via the inward-facing NIC.



(a) MT6D embedded on the host device in software



(b) MT6D implemented as a stand-alone gateway device

Fig. 2. MT6D implementation examples

E. Flow

Each MT6D host runs two concurrent processes. One process controls the generation and advertisement of IPv6 addresses. The other process listens for and handles packets. Both processes rely on a thread-safe shared memory resource, which we refer to as the Shared Routing Table (SRT).

We refer to the process that generates and advertises IPv6 addresses as the address obscurer. The address obscurer is used to supply obscured IPv6 addresses to the SRT. The first step of the address obscurer is to initialize the SRT. For initialization, it is necessary to compute and store entries for the current and next time increments for all sender/receiver profiles. The MT6D host will also advertise its MT6D source addresses for each profile entry to the external network using NDP. After the SRT is initialized, the address obscurer waits for the next time increment. When t_i increments, the current MT6D addresses are overwritten by the MT6D addresses at t_{i+1}. New MT6D addresses are then calculated for t_{i+1} and stored in the SRT. These new MT6D source addresses are again advertised to the gateway devices.

The listener process continually listens for packets. Outbound packets are sent to the encapsulator for obscuration and transmission. The encapsulator fetches the MT6D addresses and symmetric key from the SRT to use in constructing the MT6D packet. Inbound packets are send to the decapsulator for processing and transmission. The decapsulator fetches the original addresses and symmetric key from the SRT and delivers the packet to the host.

IV. Implementation

The two most common implementations of MT6D are either as embedded software on the host or as a separate gateway device. Both implementations adopt a trust model that assumes trust only between the sender and receiver. In a trust model where all insiders are trusted, the gateway implementation can be expanded to the border of a trusted network.

A. Embedded Software

One possible implementation of MT6D is to embed it onto the host device as illustrated in Fig. 2(a). This option has a number of advantages. The biggest advantage is mobility. By hosting MT6D on the host device, MT6D can be implemented on handheld devices. Another advantage is cost. Since MT6D is loaded directly onto the host device, there is no requirement to purchase additional hardware. By having MT6D on the host device, managing the configuration is also easier. There is no need to transfer keys or preferences to a separate device.

B. Gateway Device

Implementing MT6D on a separate gateway device, as shown in Fig. 2(b), is also an attractive option. In this implementation, MT6D is transparent to the user. This is especially useful if the user has devices running different operating systems since it becomes platform independent. Probably the biggest advantage of implementing MT6D in a gateway device is that the computational complexity of using MT6D is offloaded to the MT6D gateway.

As mentioned, MT6D can be implemented on a border device in a trusted environment. Implementing MT6D in this fashion mimics a Virtual Private Network (VPN). There are two benefits to implementing MT6D in this fashion over the already stated benefits of a gateway implementation. First, internal hosts can communicate internally without any performance degradation. Second, network administrators can manage internal host activities, which would be obscured in a host-based MT6D implementation.

V. Related Work

There have been other proposals that attempt to obscure network addresses. There are those that obscure network addresses for the purpose of privacy and those that obscure addresses to prevent certain network attacks. We present the most relevant proposals and discuss how MT6D differs.

A technique by Sheymov [5] was designed with the goal of dynamically obscuring cyber coordinates. Cyber coordinates can represent any piece of information in cyber space. Sheymov's objective is to provide intrusion protection from certain network attacks. MT6D also prevents these attacks, while providing the additional benefit of anonymity. Sheymov's design does not provide anonymity due to the use of the Domain Name System (DNS) to assign permanent names to devices. An attacker will have little problem correlating traffic using hosts' DNS names. Sheymov also uses a management unit to distribute addresses. In MT6D, communicating hosts are able to independently calculate their own addresses.

Fink et al. [6] also proposed a technique for dynamically obscuring host addresses called Adaptive Self-Synchronized Dynamic Address Translation (ASD). ASD is similar to MT6D in that its objective is to hide the location of communicating hosts. It does this through a handshake process between a trusted sender and receiver enclave to assign source and destination addresses. An enclave, in its preferred embodiment, encompasses a subnet. Obscured addresses are selected from those available to the ASD enclave. MT6D improves upon ASD by allowing MT6D hosts to communicate without the need to reauthenticate each time an address rotates. Reauthentication minimally gives away the identity of the trusted enclaves communicating. In MT6D, there is no need for an authentication handshake to occur, which provides further protection for communicating hosts.

Two other proposals obscure addresses to achieve anonymity. The two proposals are privacy extensions [7] and Cryptographically Generated Addresses (CGAs) [8]. Privacy extensions were designed with the intent of protecting IPv6

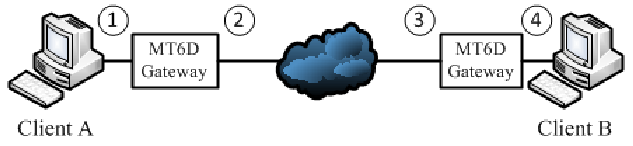


Fig. 3. MT6D testbed. The numbers 1-4 indicate measurement points.

addresses that use SLAAC. CGAs were designed to securely associate IPv6 addresses with public keys for use with the SEcure Neighbor Discovery (SEND) protocol. This association also hides the senders' addresses. Neither of these schemes dynamically obscure addresses. Once an address is assigned, it remains constant, minimally, until the network session is terminated. A third party monitoring the connection can accomplish both address tracking and traffic correlation. These techniques also only obscure the source address. MT6D not only rotates addresses multiple times within a single session, but also does so for both the source and destination addresses.

VI. Test Configuration

A proof of concept prototype software implementation of MT6D was developed to prove the validity of the design. The MT6D prototype was developed using the Python programming language. Since the optional packet encryption has minimal impact on the validity of the design, it was not implemented in the prototype. The prototype software was installed on a GuruPlug Server containing a 1.2 GHz ARM CPU with 512MB DDR2 800MHz RAM running 32-bit Debian Linux. The GuruPlug also contains two Gigabit Ethernet NICs, which act as the inward and outward facing NICs. The configuration used for testing was the gateway implementation described in Section IV-B. This configuration supported isolation of the MT6D prototype from the clients. It also allowed us to verify the MT6D prototype's ability to pass and accept network traffic from different operating systems.

Testing was conducted on a production IPv6 network. Virginia Tech has a fully functional IPv6 network, providing globally unique addresses through SLAAC to every wireless and wired node on the network. The production network provides us with results that account for the effects of actual network traffic on MT6D packets. The test network is depicted in Fig. 3. Network monitoring was accomplished by directing network traffic through two Cisco 2960 Fast Ethernet switches configured with four SPANs. The four collection points are illustrated by the numbers 1-4 in Fig. 3. A network monitoring machine was configured with four identical NICs to simultaneously listen to all four SPANs.

Our test scenario was primarily aimed at demonstrating the functionality of MT6D. For ease of traffic analysis, we set a static address rotation interval of 10 seconds. To measure basic functionality, we sent 1000 ping packets from *client A* to *client B* at a rate of one packet per second. To test MT6D under a high traffic volume of connectionless traffic, we sent a 10,000-packet ping flood from one client to the other. To test how MT6D handles connection-oriented traffic, *client A* used the Hypertext Transfer Protocol (HTTP) over TCP to download files ranging from 1KB to 500MB from *client B*.

VII. Analysis of Results

We analyze four metrics using the results of the tests described in Section VI. These metrics are address entropy, overhead, packet loss, and latency.

A. Address Entropy

Due to changing addresses, MT6D hosts are more difficult to locate on a subnet than static hosts. This claim can be supported using Shannon's explanation of entropy [9]. In terms of network addresses, the lower the number of addresses there are on a subnet, the more certain an attacker is of a target host's identity. Static addresses never change. Thus, the number of addresses on a subnet can be represented by h.

Hosts using MT6D generate multiple addresses over time. The total number of addresses observed can be represented by $hT/\Delta t$ where T is the total time MT6D is in operation and Δt is the time between address rotations. The total observed addresses increase with the number of active hosts and the duration MT6D is in operation. Decreasing the address rotation interval also results in more observed hosts on the subnet. The greater the host entropy on a network, the more uncertain an attacker is of locating a target host.

Our results confirm that the MT6D address changing scheme is feasible for preserving anonymity in IPv6 networks. During our tests, it was infeasible to locate a specific host through pinging or scanning from points 2 or 3 on Fig. 3. It was only possible for us to identify the original payload of a packet through deep packet inspection of a packet capture.

B. Overhead

The overhead of each MT6D packet is 62 bytes. Since we do not remove anything from the original IPv6 packet, the overhead is a result of adding additional headers. The new IPv6 header incurs 40 bytes. All MT6D traffic is sent via UDP in our current implementation. The UDP header adds 8 bytes. The remaining 14 bytes are from the Ethernet frame header. We were able to verify the 62-byte overhead by observing captured packets.

C. Packet Loss

The first set of tests we performed were designed to evaluate MT6D's ability to pass connectionless traffic. To measure this we sent 1000 pings from *client A* to *client B* at one ping per second. The packet loss was 2.1%. We then sent a 10,000 packet ping flood to *client B* and observed 2.6% packet loss. Analyzing packet loss on captured traffic showed that all packets lost by a MT6D gateway were lost simultaneously. We traced the packet loss pattern to address rotations. When a new address is added to the NIC, the adapter is temporarily disabled while the operating system (OS) is configured to accept the new address. While this configuration occurs, all the packets sent through MT6D are buffered by the kernel. With the standard ping test, the buffer was no longer able to keep up after 95% of the packets had been transmitted. With the ping flood test, this occurred at approximately 82% of packets transmitted. Native implementation of MT6D in the network stack could address this issue. Expanding the kernel buffer might also address this problem, but would likely result in increased latency.

The next set of tests were designed to evaluate MT6D's ability to pass connection-oriented traffic. We transmitted files ranging in size from 1KB to 500MB and measured packet loss. There was no packet loss for files 100KB and below. This is because the files were small enough to not fill the kernel buffers. Files greater than 1MB experienced the same group packet loss that we observed with connectionless traffic. These lost packets were successfully retransmitted by the endpoints using TCP, regardless of the number of address rotations.

D. Latency

To measure latency, we calculated the amount of time it took for packets received by a MT6D gateway at points 1 or 4 to be retransmitted at points 2 or 3, respectively. The latency incurred during periods where no address rotation occurred was approximately three milliseconds. During an address change the latency was greater, approximately 12 milliseconds, due to the temporary disabling of the NIC during address binding.

The latency added to MT6D-enabled communications can be traced to three sources. First, the GuruPlug computers acting as the MT6D gateways are severely underpowered for the network bridging that they were required to perform during our tests. Our implementation required that all packets be decoded and routed in the application's memory. Second, Python is an interpreted language and is ill-suited to perform the kind of real-time network switching that is required by MT6D. We acknowledge that ultimately MT6D would be best implemented in a lower-level language or hardware. Finally, rotating addresses causes the OS to temporarily disable the NIC and drop packets. Despite the latency experienced during our tests, large TCP sessions were able to recover seamlessly during address rotations without any need for the MT6D device to reconfigure or restart a session. As a case in point, the 500MB file transfer rotated through over 500 address pairs without losing the session.

VIII. Limitations

Despite its strengths, MT6D does have some limitations. One limitation is that MT6D is designed to operate on an IPv6 network. The concept of address rotation would work in IPv4; however, the majority of IPv4 subnets do not contain enough free space to facilitate it. Even if a pool of IPv4 addresses were reserved for address rotation, it would be statistically feasible for an attacker to locate a host through simple scanning. An exhaustive scan of a /16 subnet in IPv4 could be accomplished by a single host in three hours or less [10].

Other than packet loss described in Section VII, there are a few scenarios where MT6D could drop packets. The first, and most likely, scenario is when a packet gets delayed past the address rotation time. A packet arriving past an address rotation will be dropped by design. A second issue can occur if there is an address collision with an advertised address. The other communicating host has no way of knowing about this collision. Therefore, all packets sent during the rotation interval will be dropped. The likelihood of an address collision, however, is very small. The probability can be written as $P_c = h/2^{64}$ where P_c represents the probability of a collision and h represents the number of other hosts on the subnet. To minimize the number of packets lost in the unlikely event of an address collision, the address rotation interval can be made shorter. Reducing the interval between address rotations will also increase the computational requirement.

IX. Conclusion and Future Work

Static network addresses pose a security and privacy risk to network hosts. MT6D solves this problem by presenting a solution that dynamically rotates addresses, regardless of the end-to-end connection state. Dynamic addresses prevent an attacker from tracking a host, conducting network traffic correlation, and targeting a host for attack. We developed a proof of concept MT6D prototype that validates the soundness of our design. Testing was conducted on a production IPv6 network. Our results show that MT6D seamlessly tunnels both connectionless and connection-oriented network traffic.

The next phase of MT6D development is to optimize the design and produce a production-ready system capable of wide-scale deployment. The initial prototype was built to demonstrate the validity of our dynamic addressing concept under multiple network traffic scenarios. The production-ready system will be optimized and include all of the optional capabilities not required for validity testing.

References

[1] M. Dunlop, S. Groat, R. Marchany, and J. Tront, "IPv6: Now you see me, now you don't," in *the Tenth International Conference on Networks (ICN 2011)*, Jan. 2011.

[2] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006.

[3] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), Internet Engineering Task Force, Sep. 2007.

[4] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862 (Draft Standard), Internet Engineering Task Force, Sep. 2007.

[5] V. I. Sheymov, "Method and communications and communication network intrusion protection methods and intrusion attempt detection system," Patent, Feb. 2010, US 2010/0042513 A1.

[6] R. A. Fink, M. A. Brannigan, S. A. Evans, A. M. Almeida, and S. A. Ferguson, "Method and apparatus for providing adaptive self-synchronized dynamic address translation," Patent, May 2006, US 7,043,633 B1.

[7] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941 (Draft Standard), Internet Engineering Task Force, Sep. 2007.

[8] M. Bagnulo and J. Arkko, "Cryptographically Generated Addresses (CGA) Extension Field Format," RFC 4581 (Proposed Standard), Internet Engineering Task Force, Oct. 2006.

[9] C. E. Shannon, "A mathematical theory of communications," *The Bell Systems Technical Journal*, vol. 27, 1948.

[10] S. Groat, M. Dunlop, R. Marchany, and J. Tront, "Using dynamic addressing for a moving target defense," in *the 6th International Conference on Information Warfare and Security*, Mar. 2011.