Implementing an IPv6 Moving Target Defense on a Live Network

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

Abstract

The goal of our research is to protect sensitive communications, which are commonly used by government agencies, from eavesdroppers or social engineers. In prior work, we investigated the privacy implications of stateless and stateful address autoconfiguration in the Internet Protocol version 6 (IPv6). Autoconfigured addresses, the default addressing system in IPv6, provide a third party a means to track and monitor targeted users globally using simple tools such as ping and traceroute. Dynamic Host Configuration Protocol for IPv6 (DHCPv6) addresses contain a static DHCP Unique Identifier (DUID) that can be used to track and tie a stateless address to a host identity. Our research focuses on preventing the issue of IPv6 address tracking as well as creating a "moving target defense." The Moving Target IPv6 Defense (MT6D) dynamically hides network and transport layer addresses of packets in IPv6 to achieve anonymity and protect against certain classes of network attacks. Packets are encrypted to prevent traffic correlation, which provides significantly improved anonymity. MT6D has numerous applications ranging from hosts desiring to keep their locations private to hosts conducting sensitive communications. This paper explores the results of implementing a proof of concept MT6D prototype on a live IPv6 network.

1 Introduction

More networks are deploying the Internet Protocol version 6 (IPv6) due to the lack of available addresses in the Internet Protocol version 4 (IPv4). IPv6 contains a number of enhancements over IPv4; key among those is its 128-bit address space. The techniques used in IPv6 to form addresses, however, expose a host to targeted network-layer attacks and address tracking [2, 6]. Previous research by Dunlop et al. [3] designed and developed a Moving Target IPv6 Defense (MT6D) that dynamically rotates network and transport layer addresses to preserve a host's security, privacy, and anonymity. These address rotations occur regardless of the state of any ongoing sessions. A proof of concept prototype was developed and tested on a full-production IPv6 network. The prototype demonstrates: a) that the MT6D concept is valid, and b) MT6D is capable of achieving throughput and packet loss that results in negligible visible performance impact even with real-time streaming video and Voice over IPv6 (VoIPv6) traffic.

The remainder of the paper is organized in the following way. Section 2 provides a brief overview of MT6D. Section 3 describes the test scenario, while Section 4 demonstrates the results of testing MT6D on a live IPv6 network. Future work and concluding thoughts are provided in Section 5.

2 MT6D Overview

MT6D was developed to protect communicating hosts from targeted network attacks as well as address tracking. This protection is accomplished by dynamically rotating the addresses of both the sender and receiver without disrupting the state of ongoing sessions. Another feature of MT6D is that there is no handshaking required to rotate addresses. Hosts are able to predict each other's addresses from an IPv6 interface identifier (IID), a shared secret, and a changing nonce. The nonce controls when addresses rotate. In the MT6D preferred implementation, the nonce is a randomized time interval.

MT6D tunnels packets within a specialized MT6D packet to preserve end-to-end communications between communicating hosts. Packets can be tunneled either encrypted or unencrypted. Encrypted tunnels prevent traffic correlation since any identifiable information (e.g., payload, sequence numbers) is hidden from a third party. Encrypted tunnels are the default mode for MT6D. Results in Section 4 demonstrate that performance is not significantly degraded by the use of encryption. A more detailed description of MT6D is provided in a publication by Dunlop et al. [3].

3 Test Configuration

A proof of concept prototype software implementation of MT6D was developed to validate the MT6D design. The MT6D prototype was integrated as software within the host machine. The prototype was written in the Python programming language. Python was chosen for speed of development and ease of modification.

A series of tests were conducted using a live IPv6 network in order to evaluate the prototype MT6D. All live network testing was conducted on the Virginia Tech production IPv6 network. The live network hosts over 30,000 IPv6 nodes. Testing over varying number of hops is possible due to the size and structure of the network. The production network provides results that account for the effects of actual network traffic on MT6D packets.

The test scenario demonstrates the functionality of MT6D and its ability to successfully pass different kinds of traffic. A static address rotation interval was established to maintain consistency. To measure basic functionality, 1,000 ping packets were sent from the client to the server at a rate of one packet per second. A high traffic volume of connectionless traffic was tested by sending 10,000-packet and 50,000-packet ping floods from the client to the server. The client used the Hypertext Transfer Protocol (HTTP) over Transmission Control Protocol (TCP) to download files ranging from 500KB to 1GB from the server to test how MT6D handles connection-oriented traffic. Ten iterations of each test were conducted. More iterations would have produced greater precision in test results, but in the non-optimized prototype, precision was not critical. Multiple iteration were performed mainly to identify outliers. Default settings for MT6D were chosen for consistency across different network configurations. The default settings were an address rotation interval of 10 seconds and Advanced Encryption Standard (AES) encryption on all MT6D-tunneled traffic.

4 Implementation Results on a Live IPv6 Networks

The prototype software version of MT6D was installed on two Dell Optiplex 960 machines. One machine was labeled as the client and the other machine was labeled as the server. Each machine has 3.0 GHz Intel Core2 Duo CPUs with 4GB DDR 800MHz RAM running 32-bit Ubuntu 11.10 Linux. It is important to note that the software implementation was not written to take advantage of multiple processors. Additionally, each packet gets encapsulated by the sending host before

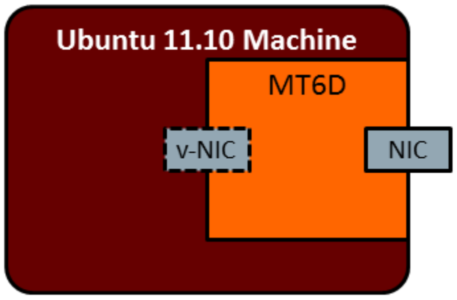


Figure 1: Diagram of MT6D integrated into the test machine

being transmitted. The speed at which this occurs is greatly dependent on the computing power of the sending host.

MT6D requires two Network Interface Controllers (NICs). An internal NIC interfaces with the unprotected host and is not accessible from the network while an external NIC transmits all MT6D-encapsulated traffic to the network. For testing, the external NIC was assigned to the integrated hardware Fast Ethernet card. The internal NIC was configured virtually on each machine. A conceptual diagram of this configuration is illustrated in Figure 1. The external hardware NIC in the diagram is labeled “NIC” while the internal virtual NIC is labeled “v-NIC”.

The prototype was tested under different network conditions. The first set of tests was conducted with all test nodes on the same subnet. These conditions tested MT6D with minimal interference from other network devices. The next set of tests had test nodes communicating from one subnet to another. These tests routed traffic through the core network, which routes traffic for over 30,000 nodes.

4.1 Same Subnet Testing

This set of tests was conducted with two MT6D hosts connected to the same subnet so that direct delivery of packets could occur without router involvement. In IPv6 intra-subnet communications, a host wishing to communicate with another host sends a neighbor solicitation message to the destination if the destination host is not stored in the sender’s destination cache. The destination host replies with a neighbor advertisement which the source adds to its destination cache for future communications [11]. The neighbor discovery process needs to occur after every MT6D address rotation since the destination’s new address will not be in the sender’s destination cache. While it is possible to artificially populate the sender’s destination cache with new destination addresses, doing so would only be useful in the isolated same subnet scenario. The neighbor discovery process is performed by routers when communicating hosts are located on different subnets.

Tests on the same subnet were conducted to establish performance baselines independent of the effects of intermediate routers. Under these conditions, MT6D was tested with different settings to gain an understanding of how each setting impacts performance. The different settings were compared against the default test settings previously described.

4.1.1 Benchmarking Default Test Settings

The first set of tests was aimed at determining the performance impact the MT6D prototype has on network traffic. The test scenario was run on the network without MT6D running to measure this. No routers were involved with the exchange of test traffic since both hosts were located on the same subnet. This configuration provides the fastest speeds and least packet loss since network factors outside the local subnet do not impact the client or server.

In all three ping tests, the average round trip time (RTT) when not using MT6D was 0.113 msec as shown in Figure 5(a). The standard deviation was 0.047 msec. A low RTT and standard deviation were expected since communications were contained within the subnet and sheltered from outside network interference. Low packet loss was also expected as illustrated in Figure 5(b). The packet loss was zero for all three connectionless tests without MT6D.

The performance impact of rotating addresses within the MT6D integrated software configuration was tested using an address rotation interval of 10 seconds with connectionless traffic. These tests also determined the impact of neighbor discovery resulting from address rotations. As illustrated in Figure 5(a), the average RTT of a 1000 packet standard ping was 5.800 msec while the average RTT over the two ping flood tests was 2.744 msec. The ping flood tests provide a better indication of the performance impact due to software than the standard ping tests do. Since the ping floods send packets as fast as possible, more packets get processed between address rotations. To illustrate this point, addresses rotate on average 11 times during a 50,000 packet ping flood. A standard ping sends an echo request every second resulting in a 1,000 packet standard ping test rotating addresses 100 times on average. The sender initiates the neighbor discovery process every time addresses rotate. During this process, the sender buffers the packets to be sent until the receiver is added to the destination cache [11]. There is more variability in the RTTs during the standard ping test since the standard ping test incurs more address rotations. It was not uncommon to see a maximum RTT over 150 msec during address changes. These high RTTs resulted in an average standard deviation of 17.853 msec for the standard ping tests and 8.632 msec for the ping flood tests.

An additional set of tests was conducted where MT6D did not rotate addresses. The purpose of these tests was to determine the impact of the software implementation on the processing of packets. In all three connectionless traffic tests, the average RTT was just over 2 msec as illustrated in Figure 5(a). This result demonstrates that an average latency over 2 msec is a result of address rotation. In situations where the ratio of packets sent to addresses rotated is high, such as the ping flood tests, address rotations have minimal impact. In situations where the ratio is low, such as the standard ping, address rotations have a much higher impact. Much of the added latency is due to queuing of packets while neighbor discovery occurs. This can be seen by comparing the standard deviations from the standard ping test where MT6D is rotating addresses and again where it is not. The average standard deviation was 17.853 msec when MT6D was rotating addresses every 10 seconds, as opposed to 0.099 msec when MT6D did not rotate addresses.

A certain amount of packet loss was expected (see Figure 5(b)) due to a few factors related to address rotations. The first factor is due to address rotation within the MT6D device itself. 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. This configuration may take less than a few milliseconds, but any packets sent through the NIC while this configuration occurs would be dropped. A native implementation of MT6D in the network stack could address this issue. Another factor that contributes to packet loss can be due to the strict address rotation rules enforced by MT6D. As soon as an address rotates, the previous address is no longer valid. It is possible for a packet to be sent just prior to address rotation. Any packets arriving for a previously bound address will be rejected. A third factor that can cause packet loss is clock drift. Even though all MT6D hosts run Network Time Protocol (NTP), clocks may drift between poll intervals. Address rotation causing packet loss is validated by the tests using MT6D with a fixed address. As demonstrated in Figure 5(b), the only test where packet loss occurred was the test with MT6D rotating addresses.

There is also a condition that can cause all packets within a rotation interval to be dropped. This will occur when the address MT6D chooses is already in use on the subnet. IPv6 contains a mechanism, called Duplicate Address Detection (DAD) that is part of Neighbor Discovery Protocol (NDP) to detect address collisions. In the event of an address collision, the host attempting to bind a duplicate address will have to select another address. If this occurs, the other communicating host has no way of knowing about this collision and will still use the address that caused the collision. The result is that all packets sent during the rotation interval will have mismatched addresses and 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 host addresses on the subnet. This condition did not occur during any of the test iterations.

Connection-oriented tests were also conducted to verify MT6D's capability of maintaining negotiated sessions while changing addresses. Additionally, these tests validated the successful retransmission of dropped packets. Without MT6D running, the average transfer speed was between 54-108 MB/s as depicted in Figure 6(a). This is consistent with the speed limitations of the media and the NICs. The average throughput was a bit under 2 MB/s with MT6D running under the default test settings. Comparing the throughput of MT6D running with a 10 second address rotation interval to MT6D running without rotating addresses, the latency incurred due to address change is less than 200 KB/s. This result indicates that the throughput limitations of MT6D are due to packet processing in software.

The number of TCP retransmissions was used to compute packet loss for connection-oriented traffic. Figure 6(b) illustrates that packet loss is caused by address rotations. There was virtually no packet loss when there were no address rotations. There was no packet loss for file transfers under 10 MB even when addresses were rotating at 10 second intervals. The reason for no packet loss on smaller file transfers was that the files transferred within a single address rotation interval. The packet loss percentage was approximately 0.33% for 10 MB file transfers. This loss was a result of half of the file transfers occurring within the span of a single address while the other half had an address rotation occur mid-transfer. The packet loss rate for large file transfers was under 2%. Figure 2 illustrates that all the packets lost were lost during address rotations. The black line on the graph shows normal TCP traffic while the red line shows the number of retransmitted packets. Note that the number of retransmissions spikes every 10 seconds in conjunction with an address rotation. Between rotations, there are no retransmissions.

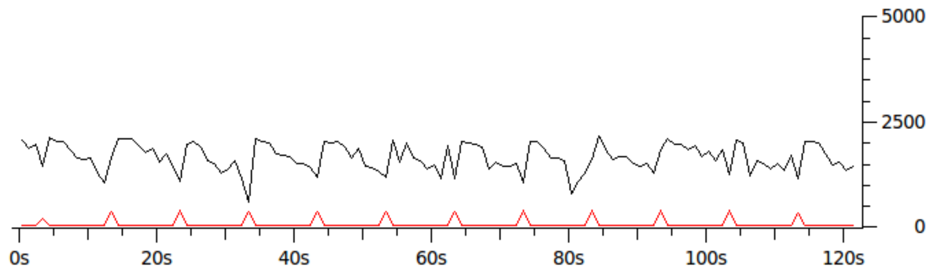


Figure 2: Graph of packet loss during address rotations. The x-axis counts the number of seconds since the packet capture was started while the y-axis counts the number of packets per second. The black line represents TCP traffic. The red line shows the number of TCP retransmissions.

4.1.2 Varying Address Rotation Intervals

A series of tests were conducted where the address rotation interval was varied. The goal of these tests was to determine how much of an impact the address rotation interval had on performance. The different rotation intervals tested were five seconds, 10 seconds (default), 15 seconds, and 30 seconds. The expectation was that throughput would increase and packet loss would decrease as the address rotation interval became larger. A larger rotation interval results in less address rotations. In the previous section, it was demonstrated that address rotations impact performance.

The 1,000 packet standard ping test illustrated in Figure 7(a) demonstrates the extent to which address rotation could impact performance. When using a five second address rotation interval, the average RTT was 8.496 msec with an average standard deviation of 23.588 msec. The standard deviation of this rotation interval is approximately 30% higher than that of the default 10 second address rotation interval. Since the five second address rotation interval incurs twice as many address rotations as the ten second rotation interval, it is expected that twice as many high RTTs will be observed. The higher standard deviation during the evenly distributed standard ping tests confirms this. The approximate doubling of RTTs over 150 msec also accounts for the RTT increasing to 8.496 msec for the five second rotation interval. The increase of the packet loss percentage from 0.28% for a 10 second address rotation interval to 0.34% for a five second address rotation interval (see Figure 7(b)) also matches expectations for doubling the number of address rotations. It is more likely for a packet to get lost with more address rotations due to one of the previously described conditions. The impact of address rotations on the RTTs for the ping flood tests were not as pronounced due to a much smaller increase in the total number of address rotations compared to the total number of packets sent.

In contrast, throughput increased and packet loss decreased when address rotation intervals were greater than 10 seconds. Using the 30 second address rotation interval as an example, the average RTT decreased with similar magnitude between a 10 second rotation interval and a 30 second rotation interval as it did between a five second rotation interval and a 10 second rotation interval. The average packet loss experienced a comparable decrease. In fact, there was no packet loss with a 30-second address rotation interval. Once again the observed decrease in average RTT and packet loss was not as pronounced for the ping flood tests due to a much lower change in the ratio of address changes to total packets transmitted.

Connection-oriented tests produced results similar to ping flood tests in terms of magnitude of change between the different address rotation intervals. Figure 8(a) illustrates that smaller file sizes have slightly higher throughput than larger files. This matches expectations since smaller files take less time to transfer and thus experience a lower number of address rotations.

Connection-oriented traffic, as with connectionless traffic, had higher packet loss with smaller address rotation intervals. Figure 8(b) illustrates the relationship of packet loss to address rotation interval for the different sized file transfers. File transfers below 10 MB again experienced little to no packet loss. Only the 500 KB file transfer with a five second address rotation interval saw any packet loss; this loss occurred in one of the 10 iteration where an address rotated mid-transfer. Packet loss was again seen in file transfers of 10 MB and larger. As witnessed during baseline testing, the packet loss for 10 MB file transfers was considerably lower than that of larger file transfers due to some of the file transfers occurring during the span of a single address while others had an address rotate mid-transfer.

4.1.3 Comparing MT6D Tunnel Modes

A series of tests were conducted with MT6D operating in the default encrypted tunnel mode and then again in unencrypted tunnel mode. The goal of these tests was to determine how much of a performance impact is caused by encryption. Packets in encrypted tunnel mode are encrypted with 128-bit AES encryption using cipher-block chaining (CBC) mode implemented through the Python PyCrypto package. CBC mode was chosen for the prototype over counter mode mainly to avoid having to synchronize counters between all communicating host pairs. Tests for both tunnel modes were conducted with MT6D rotating addresses every 10 seconds.

Connectionless traffic tests illustrate that there is not a significant performance impact due to encrypting packets in MT6D (see Figure 9(a)). With the 1000-packet standard ping test, encryption added 0.61 msec to the average RTT. The ping flood tests incurred approximately 0.2 msec RTT increase due to encryption. As demonstrated in Figure 9(b), there was also not a significant difference in packet loss caused by encrypted tunnels. For each of the three connectionless tests there was less than 0.01% difference in packet loss between the two tunnel modes.

The connection-oriented tests depicted in Figure 10(a) provide further evidence that encryption accounts for only a small decline in performance. There was less than 200 KB/sec difference between encrypted tunnels and unencrypted tunnels for all file transfers. A small decrease in performance is expected due to the added overhead of the encryption header in the MT6D packet. The 1 GB file transfers, for example, required an additional 2% more packets to be sent in encrypted tunnel mode versus unencrypted tunnel mode. The increased number of packets also caused a increase in packet loss for encrypted tunnel mode as shown in Figure 10(b). Since more packets had to be sent, the file transfers took longer to complete. Larger file transfers rotated through more addresses, which caused the additional packet loss.

4.2 Different Subnet Testing

While testing between hosts on the same subnet provides insight into how different parameters affect the performance of MT6D, it is much more likely that communicating hosts will be located on different subnets. It was, therefore, important to gain an understanding of how performance changed due to packets having to pass over routers. The same set of tests was used to test this network configuration as those used in benchmarking default test settings: testing without MT6D, with MT6D rotating addresses every 10 seconds, and with MT6D not rotating addresses. The results of these tests were then compared with those of Figures 5 and 6 to determine first, how network conditions differ, and next, how MT6D performs when crossing subnets.

A short explanation of how neighbor discovery differs in IPv6 when a router is involved is required before evaluating the test results. As previously discussed, a host communicating with another host on the same subnet will buffer outbound packets until the neighbor discovery process is complete. The process is different with hosts on different subnets. A source sending a packet destined for a host on another subnet immediately forwards the packet to the gateway device. Once the packet arrives at the subnet that contains the destination, the destination gateway initiates the neighbor discovery process if it does not have the destination host in its neighbor cache. Senders typically queue packets until neighbor discovery completes, however, routers cannot maintain large queues for all connected hosts. According to RFC 2461, routers are required to queue at least one packet [11] per potential neighbor. This is, however, infeasible because storing one packet per possible connected neighbor would require a router to maintain an available queue of at least 671,088,640 TB. This storage requirement does not take into consideration the computational power the router would have to have to send and receive all the corresponding neighbor solicitations and advertisements.

The 1000 packet standard ping tests illustrate the router queuing problem. There was no packet loss for network communications without MT6D and with MT6D operating without rotating addresses. This matches expectations since tests within the same subnet proved that packet loss results from address rotation. What did differ in these tests was the amount of packet loss due to address change. In Figure 5(b), the observed packet loss was 0.28% for two hosts on the same subnet running MT6D with a 10 seconds address rotation interval. Figure 11(b) shows a packet loss of 10.16% for two hosts located on different subnets. This 10% increase in packet loss matches the number of address rotations. Ten pings are sent per rotation interval. If one ping per address rotation is lost, the resultant loss is 10%. What is interesting is which packets are lost. Examining packet captures after an address rotation shows that the following events occur, which are also illustrated in Figure 3:

1. The client sends an echo request to the server, which is received by the router.
2. The router sends a neighbor solicitation since the server's new address is not in its neighbor cache.
3. The server replies to the router with a neighbor advertisement.
4. The router delivers the echo request to the server.
5. The server sends an echo reply to the client, which is received by the router.
6. The router sends a neighbor solicitation since the client's new address is not in its neighbor cache.
7. The client replies to the router with a neighbor advertisement.
8. The router drops the echo reply to the client.

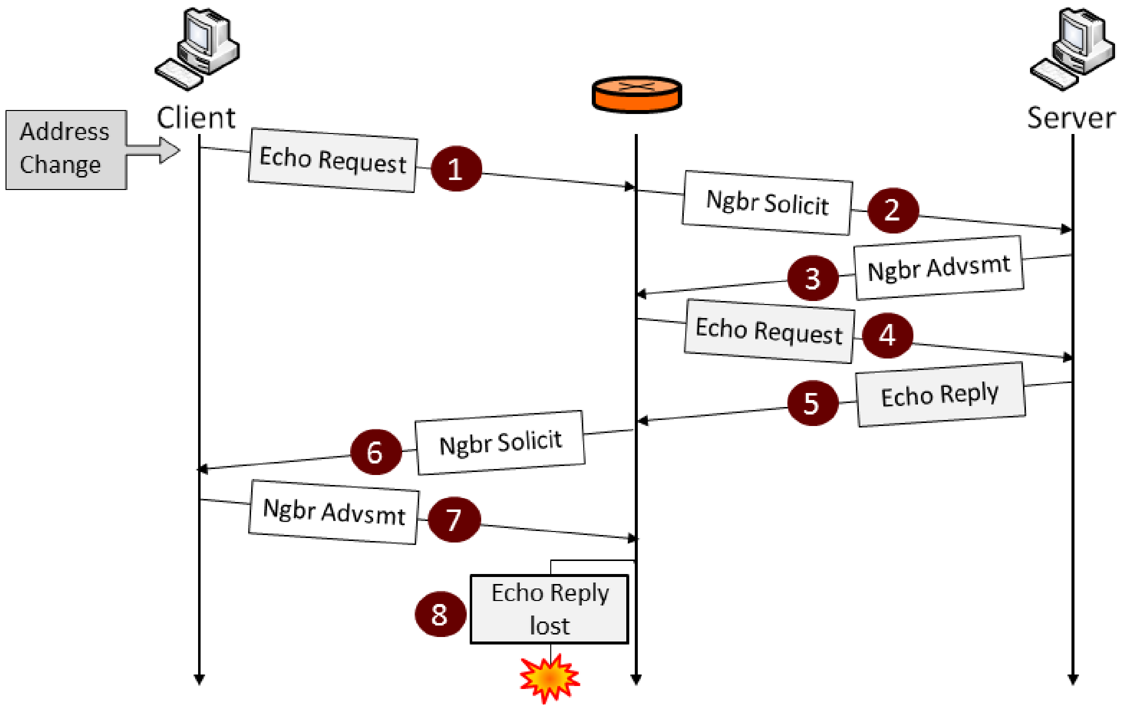


Figure 3: Ping flow after an address rotation. The first ping reply after an address rotation is erroneously dropped by the router.

Figure 3 is labeled with numbers that corresponds to the steps listed above. It is not clear whether step eight occurs before or after step seven. What is clear is that the router is queuing the first echo request after an address change, but not the echo reply. This behavior is demonstrated in Figure 4. The graph in the figure was produced from a packet capture taken from the client's "v-NIC". Each black vertical line represents a single echo request sent from the client to the server. The thick blue horizontal line represents the echo replies received by the client. The breaks in the blue line occur every 10 seconds. These lost replies occur at the router and are the first expected replies after an address change. Since the router is receiving the echo reply, this behavior is particular to how the router is configured to handle queuing of packets. A router with more queuing capability or different queuing rules would not experience the same packet loss.

Figure 11(a) shows the effect this dropped ping has on average RTT. When the ping was being sent to the server on the same subnet, the average RTT was 5.800 msec with an average standard deviation of 17.853 msec. As discussed during same subnet testing, the high standard deviation was a result of the hosts queuing packets after an address change while neighbor discovery occurred. Since the router is dropping the first ping after an address change, the average RTT is reduced to 4.985 msec with a standard deviation of 14.831 msec.

Although address rotation results in more packet loss when packets are sent to a different subnet, the impact is minimal when packets are sent at a high volume. Comparing the ping flood loss in Figure 5(b) to Figure 11(b) demonstrates a less than 0.02% difference. A comparison of average RTTs (see Figures 5(a) and 11(a)), illustrates a difference of less than 0.3 msec. Tests with MT6D not rotating addresses show a similar increase in average RTT. A slight increase is expected since more hops are required for each packet to reach its destination, which in turn means that more processing is done on each packet.

Connection-oriented tests results provide further evidence that rotating addresses when crossing subnets does not significantly impact performance as compared to operating MT6D between hosts on the same subnet. As with the ping flood tests, the average transfer speed for each of the different file sizes was nearly the same (see Figures 6(a) and 12(a)). The packet loss for connection-oriented tests was less than 0.12% higher for the 10 MB file transfers and larger as seen in Figures 6(b) and 12(b). The packet loss for file transfers below 10 MB was the same as those on the same subnet (0%) because no address rotations occurred.

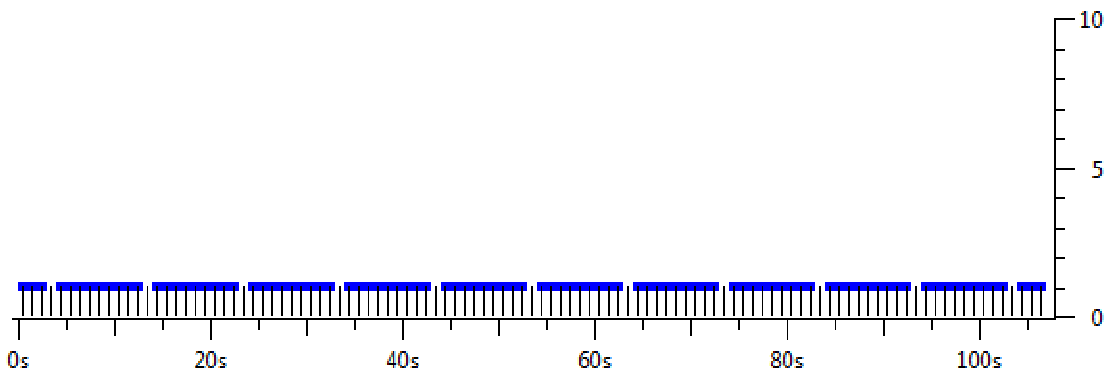


Figure 4: Dropped echo replies from the IPv6 router to the client. The black vertical lines each represent a sent echo request. The blue horizontal line represents echo replies received. The x-axis shows the time, in seconds, since the packet capture was started while the y-axis shows the number of packets sent per second.

The majority of packet loss occurs during address rotations. This is due to routers not queuing packets during neighbor discovery. Preemptively sending neighbor advertisements prior to address changes was implemented to see if this reduced packet loss. Preemptive advertisements had no impact on packet loss. Packet captures showed that, despite preemptively advertising new addresses, the router (or sender when testing on the same subnet) would still send neighbor solicitations. This is consistent with RFC 2461 [11] that says a neighbor solicitation is sent to verify a neighbor is "still reachable via a cached link-layer address." Since the router has not previously communicated with the intended destination, verification is required to ensure the link-layer address is still valid. As a result, preemptive advertisements were disabled.

Preemptive advertisements also have the second-order effect of acting as an oracle to identify that hosts on the subnet are using MT6D. An adversary sniffing a network would be able to see unsolicited neighbor advertisements being sent at regular intervals. Since this is abnormal behavior, the adversary could surmise that hosts on the network are using MT6D. An adversary would not, however, know any host identities nor necessarily how many hosts are using MT6D.

The series of tests conducted between different subnets, made it evident that the biggest impact on MT6D performance is a result of the MT6D code itself. As observed during connection-oriented testing, there was not a significant difference between the performance results of running MT6D between two hosts connected to the same subnet and two hosts on different subnets. The difference in throughput when MT6D was not running, however, was significant. Non-MT6D communications between two hosts on the same subnet had an average throughput of approximately 85 MB/sec as shown in Figure 6(a). The same two hosts communicating without MT6D across different subnets had an average throughput of approximately 11 MB/sec as shown in Figure 12(a). Passing non-MT6D traffic over a router caused near an 800% decrease in throughput. This is significant because despite a sharp decrease in throughput with non-MT6D traffic, MT6D performance remained consistent, regardless of network conditions. Optimizing MT6D code or even moving MT6D into hardware should increase performance.

4.3 Real-time Traffic Testing

The connectionless and connection-oriented tests outlined earlier in this Section are aimed at establishing the performance impact of using MT6D. None of the tests thus far have been focused on the impact MT6D has on real-time traffic. Although real-time traffic typically uses either Unreliable Datagram Protocol (UDP) or TCP, the testing of these protocols already conducted does not take user tolerance to latency and packet loss into account. Testing was, therefore, conducted using VoIPv6 and streaming video. These two applications are currently two of the more popular real-time traffic applications.

VoIPv6 testing was done to evaluate MT6D's ability to handle connectionless real-time traffic. The VoIPv6 connection between the two test hosts was established using Mumble [10]. Mumble was chosen due to its ease of setup and IPv6 support. Mumble runs a voice codec called speex [12] instead of the more standard G.711 [9]. Tests were conducted using a two minute audio clip for consistency. The first set of tests was conducted without MT6D. The next set used MT6D configured to rotate addresses on a 10-second interval in encrypted tunnel mode. Packet loss and latency were determined using packet captures, since no open-source tools were discovered that measure VoIPv6 performance. Jitter was unable to be measured. Mumble does have an internal measurement capability that measures latency and packet loss, but comparisons against results from packet captures found Mumble's statistics to be inaccurate. In all tests, the voice quality was set to the maximum setting available of 96 kb/sec.

A comparison of test results showed that, although MT6D does impact latency, the impact is within acceptable limits. Latency of voice traffic without MT6D was 0.18 msec while latency with MT6D operating was, on average, 1.45 msec. Although this may seem like a large increase in latency, the recommended Quality of Service (QoS) for acceptable latency according to G.114 [8] is less than 150 msec. MT6D is well within this acceptable limit. There was only a slight delay from a qualitative standpoint between the tests without MT6D and those with MT6D. The delay did not impact the ability to communicate. Tests without MT6D resulted in negligible packet loss. Tests with MT6D resulted in a packet loss of 0.52%, on average. Maximum packet loss for most network service-level agreements (SLAs) is between 0%-0.5% packet loss [4, 5, 7]. The MT6D packet loss is slightly higher than this, but would likely be lower in an optimized version of MT6D. There are no statistics on jitter, as previously mentioned, because no good tool to measure jitter that works with IPv6 was discovered.

Streaming media was used to test connection-oriented real-time traffic. A high resolution open animated movie titled "Big Buck Bunny" [1] was streamed in 720p from the server to the client machine to perform the tests. Streaming was done using the open-source VLC media player [13]. Other than packet loss and latency, quantitative performance measurements were not possible due to the lack of open-source IPv6-capable performance measurement tools. A packet loss of 0.10% was determined from the number of retransmissions seen in a packet capture. This was much lower than other connection-oriented testing. An analysis of the packet capture showed that, like in other testing, packet loss occurred during address rotations. An average latency of 3.34 msec was determined by comparing the time packets left the server to the time the corresponding packets arrived at the client. From a qualitative perspective, there was little noticeable degradation of video quality due to MT6D. There were on average four times during the 597 second movie that the movie froze. When the movie did freeze, it was usually for less than one second.

These results demonstrate that MT6D is able to successfully pass both connectionless and connection-oriented real-time traffic. Qualitatively, there is little degradation in performance from the user's perspective. This is a positive indication that MT6D can be used for secure VoIPv6 communications. As mentioned, optimized versions of MT6D are likely to improve performance.

4.4 Robustness Testing

A test of MT6D running over an extended period was conducted to determine the robustness of the prototype. MT6D was set to rotate addresses every 10 seconds during this test. The test ran for a week span. A standard ping between the client and server was also running to generate traffic. Both the client and server were able to successfully communicate during the test period without losing the MT6D connection. To quantify these results, each host rotated through over 60,000 addresses.

5 Conclusion and Future Work

This paper described how the prototype MT6D performed on a live IPv6 network. A test network was built utilizing the Virginia Tech production IPv6 network. The integrated software configuration [3] of MT6D was used during all testing. The test results demonstrate that the MT6D concept is valid. Even with the non-optimized prototype, MT6D is capable of successfully transmitting various types of network traffic, to include real-time traffic.

There is clearly the potential for future research with MT6D. The most important future research direction is optimizing MT6D. Now that a working prototype has been developed and the concept has been validated, effort can be shifted toward developing a more efficient implementation usable on various platforms. Two different directions can be pursued for optimization – software or hardware. An optimized software version would use a more efficient programming language. Performance would also increase if the optimized version was designed as a loadable kernel module that ran in kernel-space. The ultimate goal for optimization is developing a hardware version of MT6D. Along with optimizing MT6D to run faster, research can be done to make it more power efficient. Optimizing MT6D to use less power would make it more feasible to use with power-constrained devices, such as mobile devices.

The results of this research take an important step toward providing security, anonymity, and privacy for any communicating hosts on the network. Certain applications would greatly benefit from this technology. One potential application of MT6D is securing sensor networks, such as the SmartGrid. Attacks against sensors could deny critical information communications or expose sensitive information. MT6D prevents an attacker from locating and subsequently targeting sensors. This technology can also be used to provide secure communications for military/intelligence agents, corporate entities, or for e-commerce. Virtual Private Network (VPN) technologies can benefit from MT6D as well. Current VPNs are susceptible to Denial-of-Service (DoS) at the endpoints. A VPN using MT6D will have moving endpoint that an attacker statistically cannot target. Moving target defenses are the next critical addition to the defense in depth model. MT6D takes advantage of the features of IPv6 to provide a functioning moving target defense.

References

[1] Big buck bunny. Available at: `http://www.bigbuckbunny.org/` accessed on 12 October 2011.

[2] 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.

[3] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6D: a moving target IPv6 defense. In *the 2011 Military Communications Conference (MILCOM)*, pages 1321–1326, Baltimore, Maryland, Nov. 2011.

[4] Expert Service Providers hosted voice over ip service level agreement. Available at: `http://www.espdelivers.com/esp/esp.nsf/site/voip-sla`, accessed on 10 January 2012.

[5] P. Garg and R. Singhai. QoS in VoIP, n.d.

[6] S. Groat, M. Dunlop, R. Marchany, and J. Tront. What DHCPv6 says about you. In *the World Congress on Internet Security (WorldCIS-2011)*, Feb. 2011.

[7] IP SLA manager thresholds page. Available at: `http://www.solarwinds.com/netperfmon/SolarWinds/wwhelp/wwhimpl/common/html/wwhelp.htm#context=SolarWinds&file=OrionVoIPMonitorPHPageThresholds.htm`, accessed on 10 January 2012.

[8] ITU-T recommendation G.114. One-way transmission time, May 2003.

[9] ITU-T recommendation G.711. Pulse code modulation (PCM) of voice frequencies, Nov. 1988.

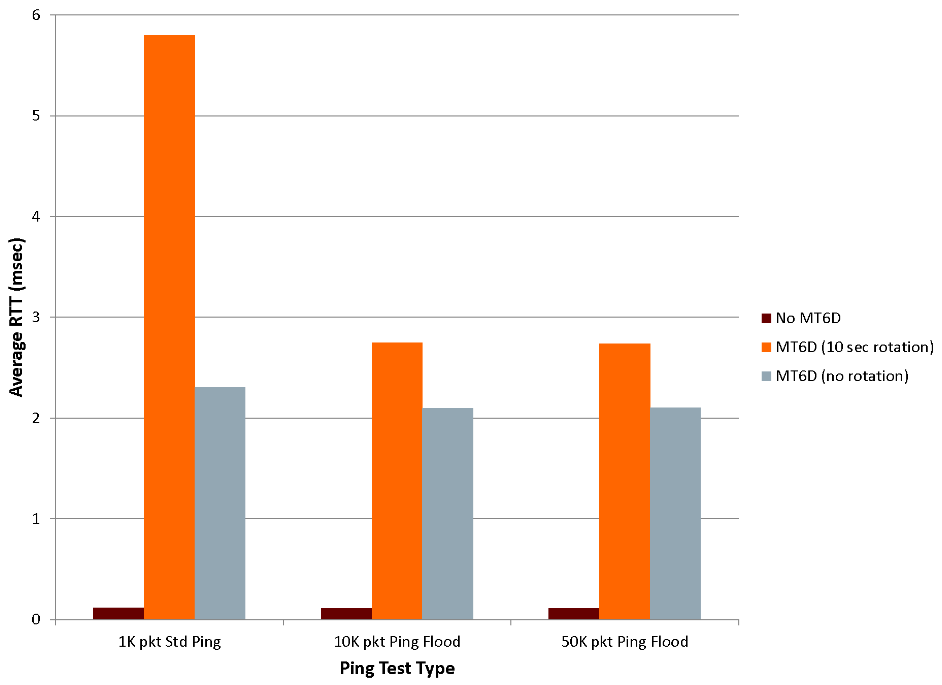
[10] Mumble. Available at: `http://www.mumble.com/` accessed on 30 January 2012.

[11] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461 (Draft Standard), Dec. 1998. Obsoleted by RFC 4861, updated by RFC 4311.

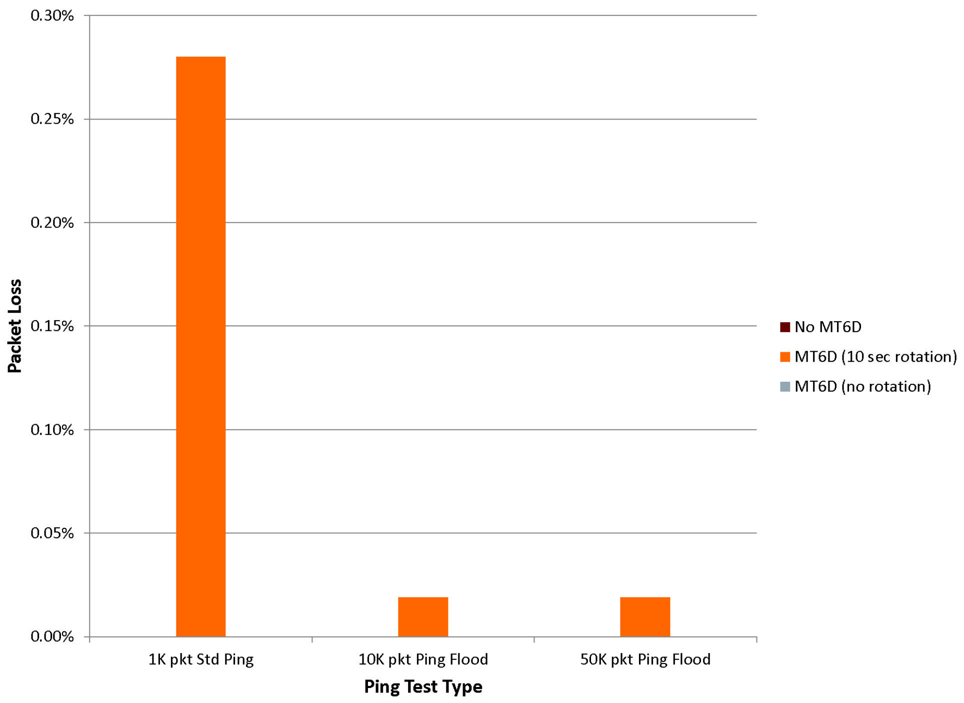
[12] Speex codec. Available at: `http://www.speex.org/` accessed on 30 January 2012.

[13] VideoLAN Organization. VLC media player. Available at: `http://www.videolan.org/vlc/` accessed on 12 October 2011.

A Live Test Supporting Figures

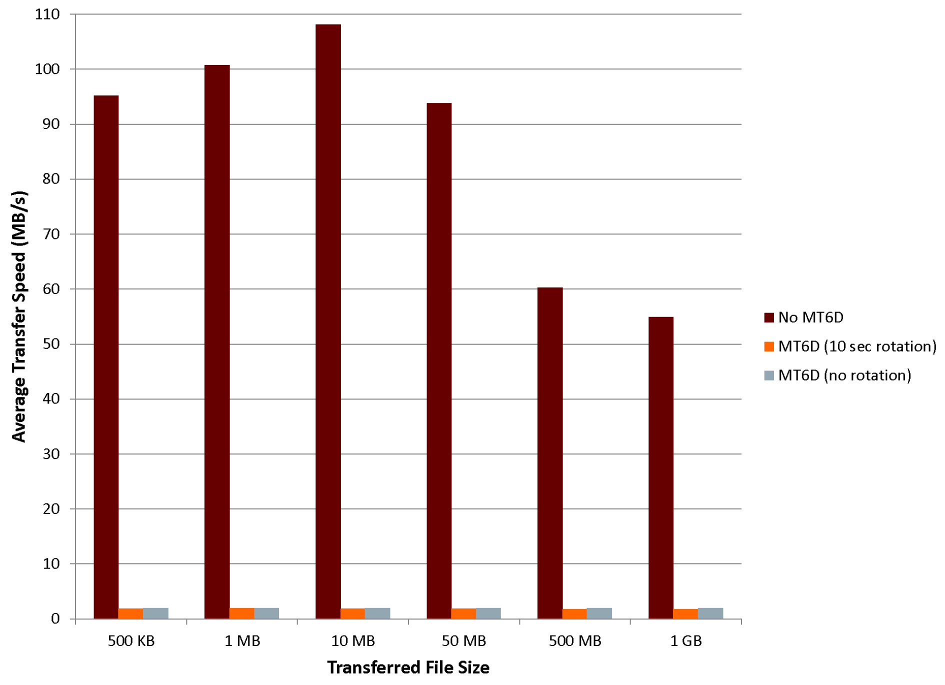


(a) Average RTT

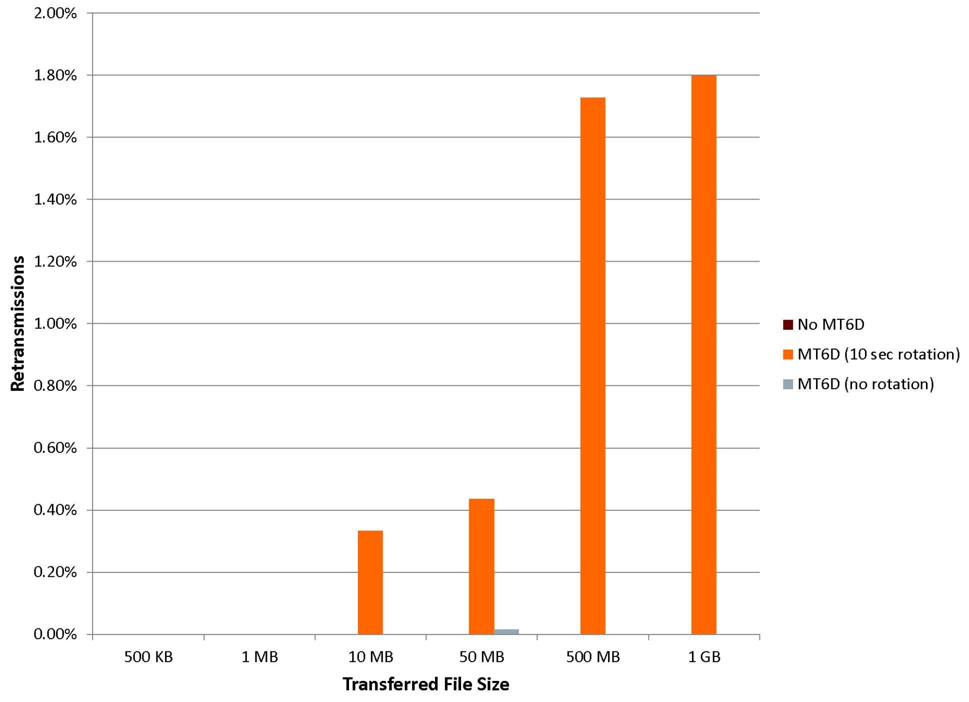


(b) Average packet loss

Figure 5: Comparison of the average speed and packet loss of connectionless traffic without MT6D, with MT6D using the default test settings, and with MT6D using a fixed address

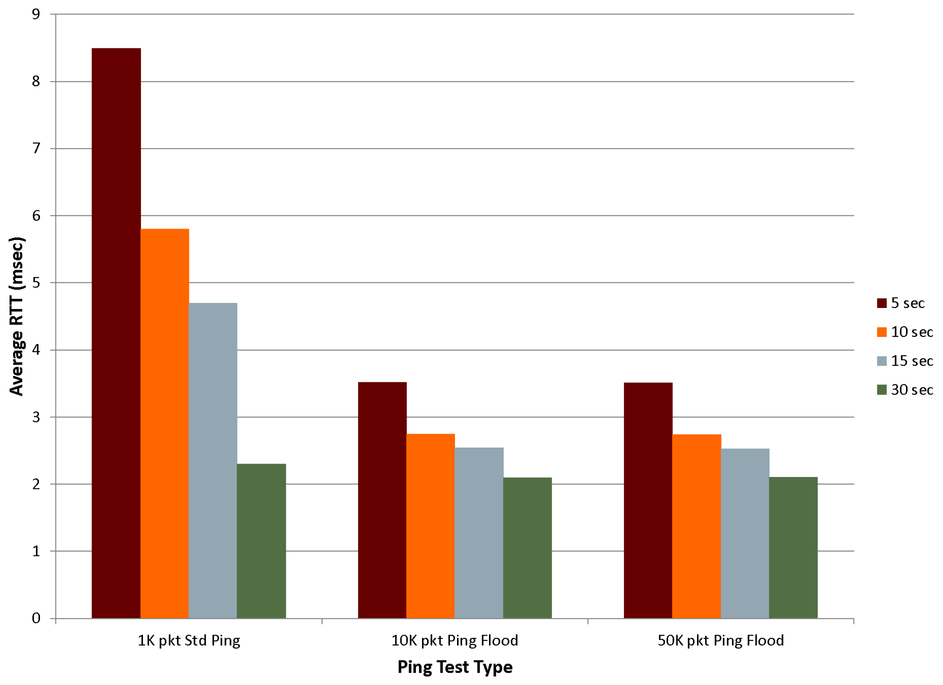


(a) Average transfer speed

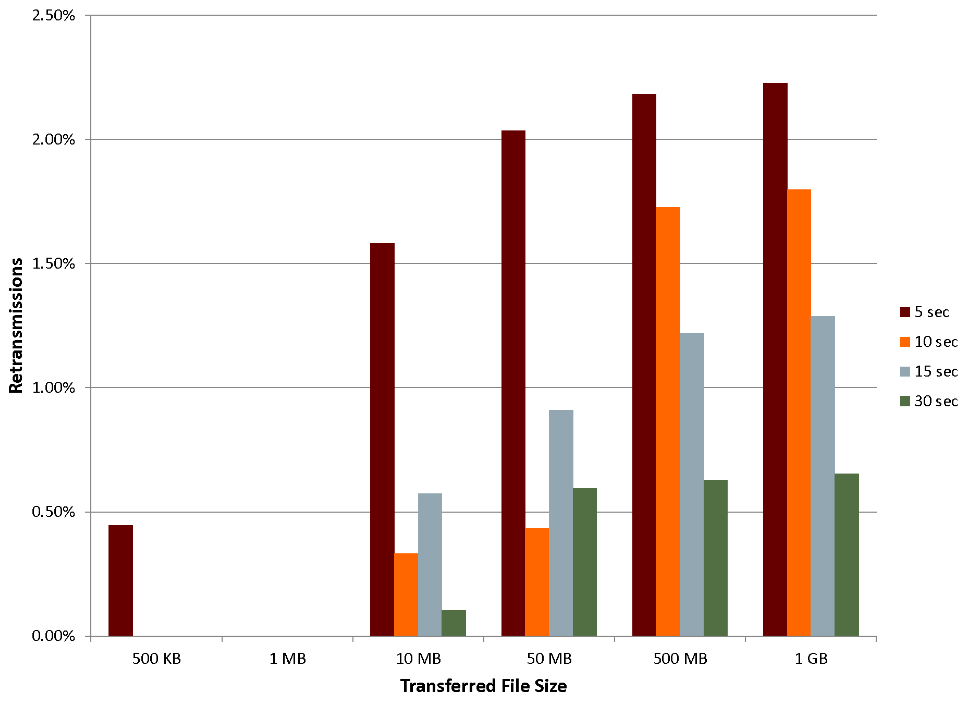


(b) Average percentage of retransmitted packets

Figure 6: Comparison of the average transfer speed and packet loss of connection-oriented traffic without MT6D, with MT6D using the default test settings, and with MT6D using a fixed address

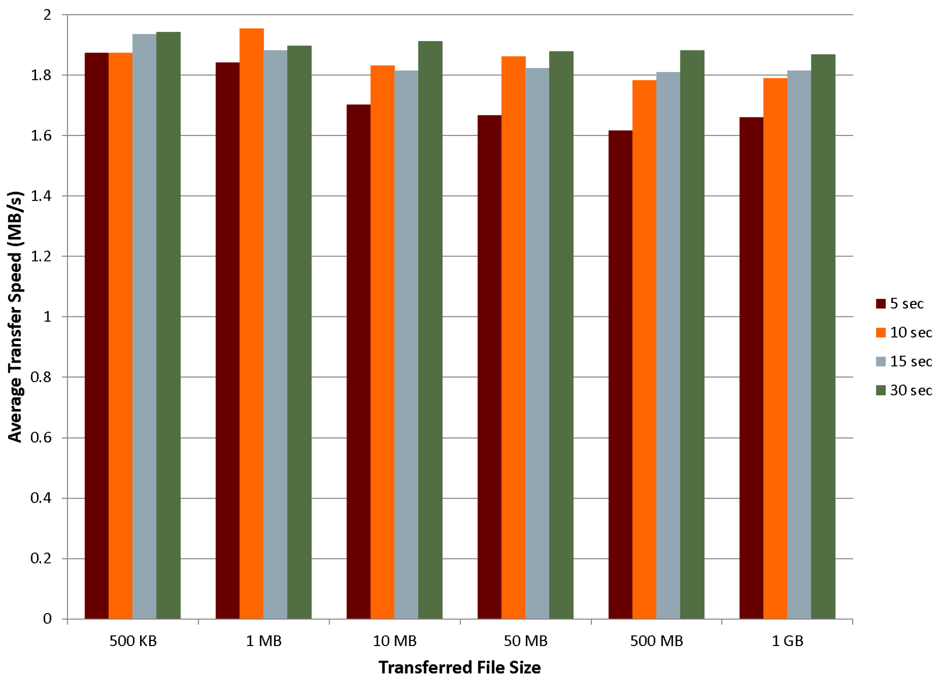


(a) Average RTT

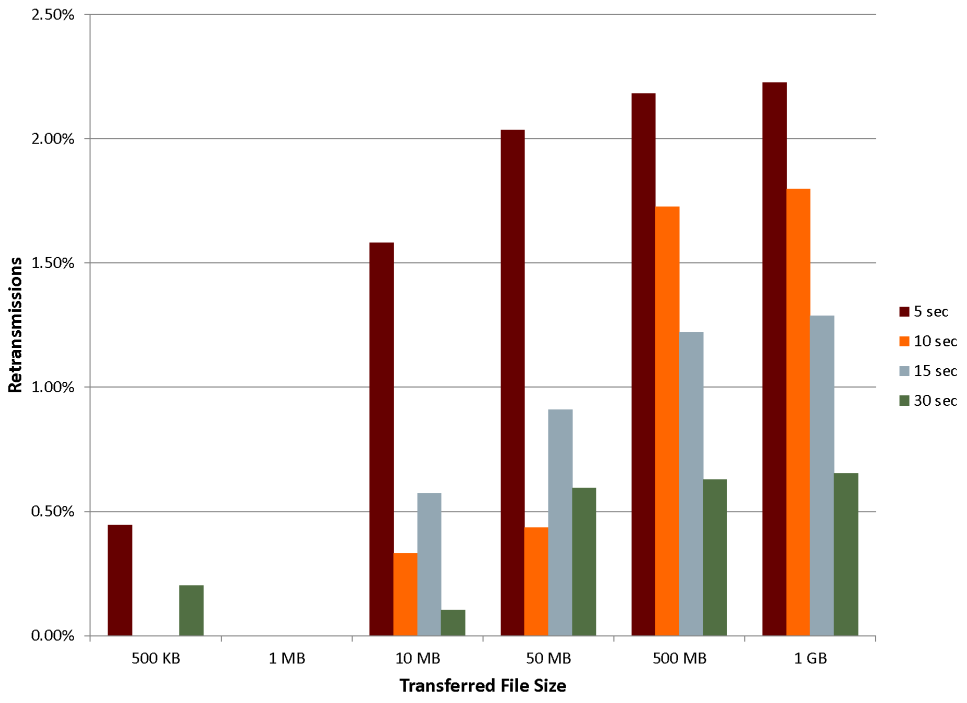


(b) Average packet loss

Figure 7: Average speed and packet loss of connectionless traffic using different rotation intervals

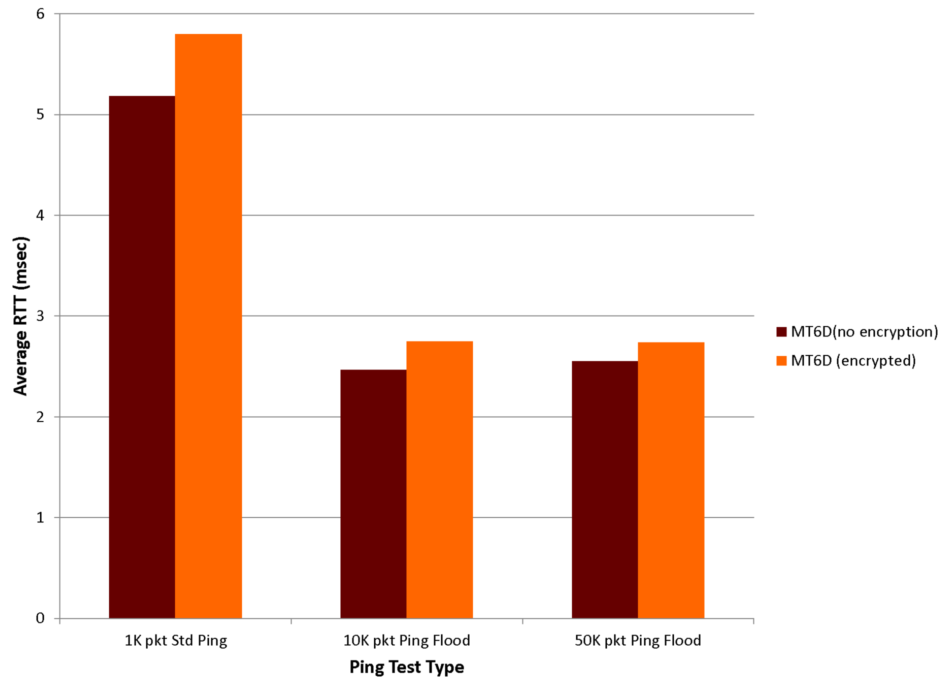


(a) Average transfer speed

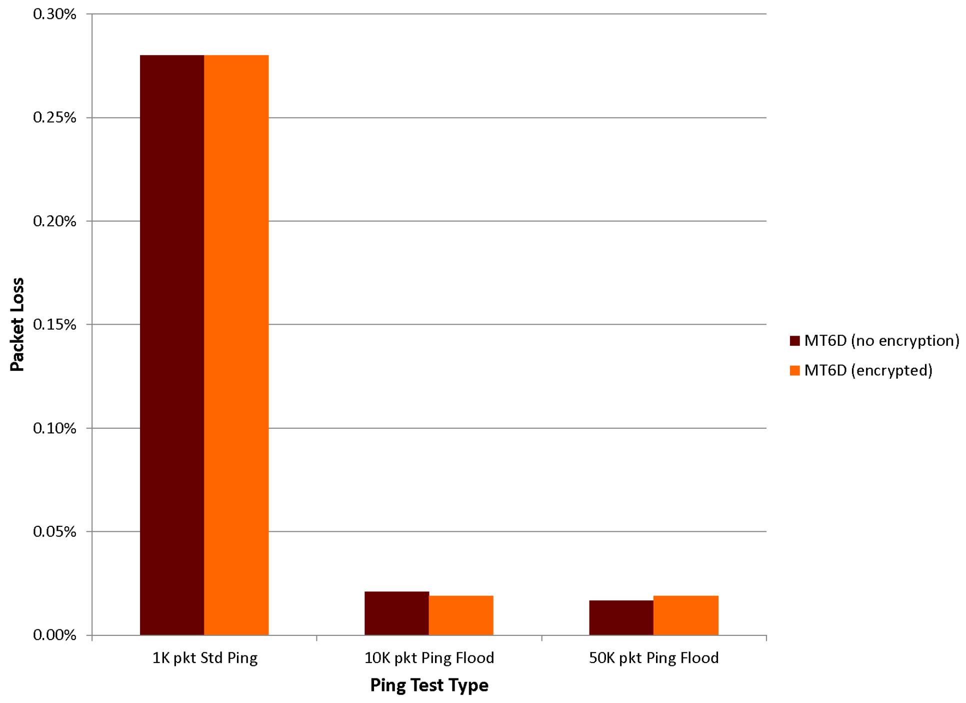


(b) Average percentage of retransmitted packets

Figure 8: Average transfer speed and packet loss of connection-oriented traffic using different rotation intervals

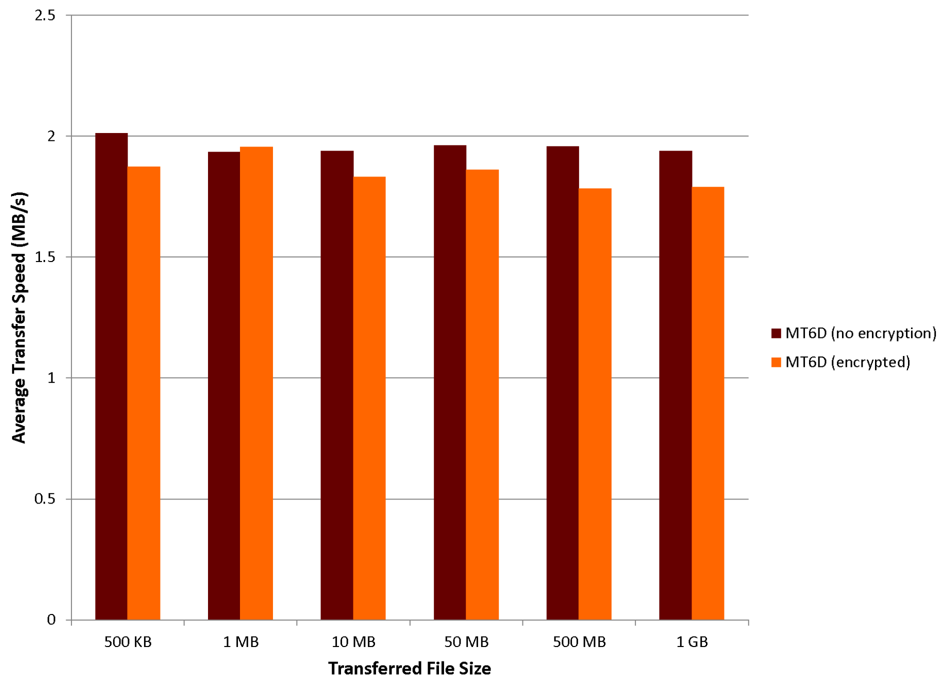


(a) Average RTT

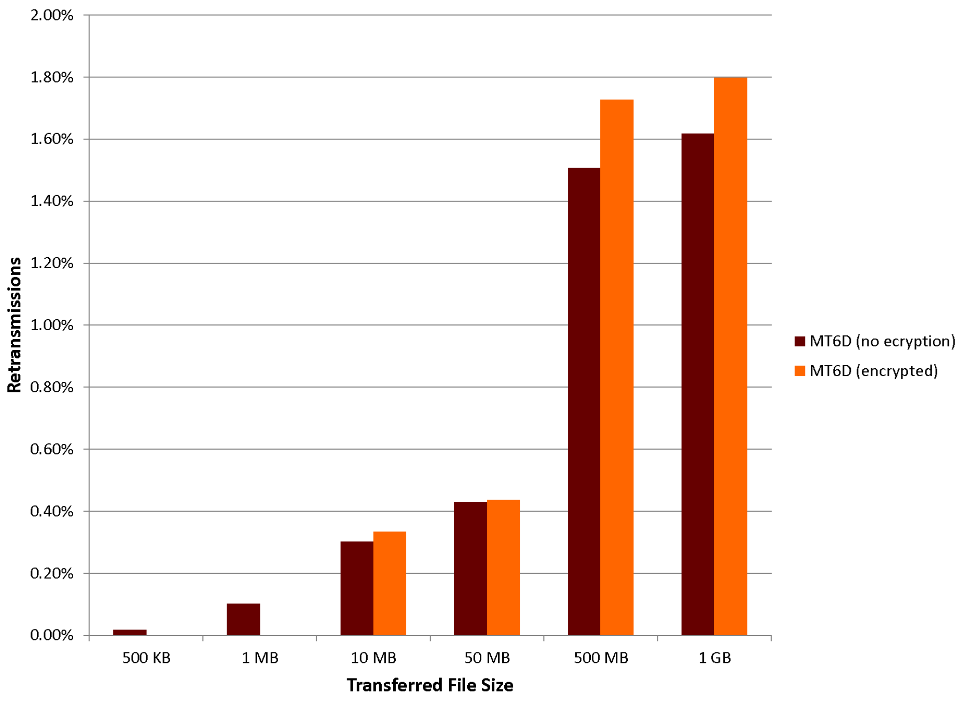


(b) Average packet loss

Figure 9: Average speed and packet loss of connectionless traffic using encrypted tunnel mode verses unencrypted tunnel mode

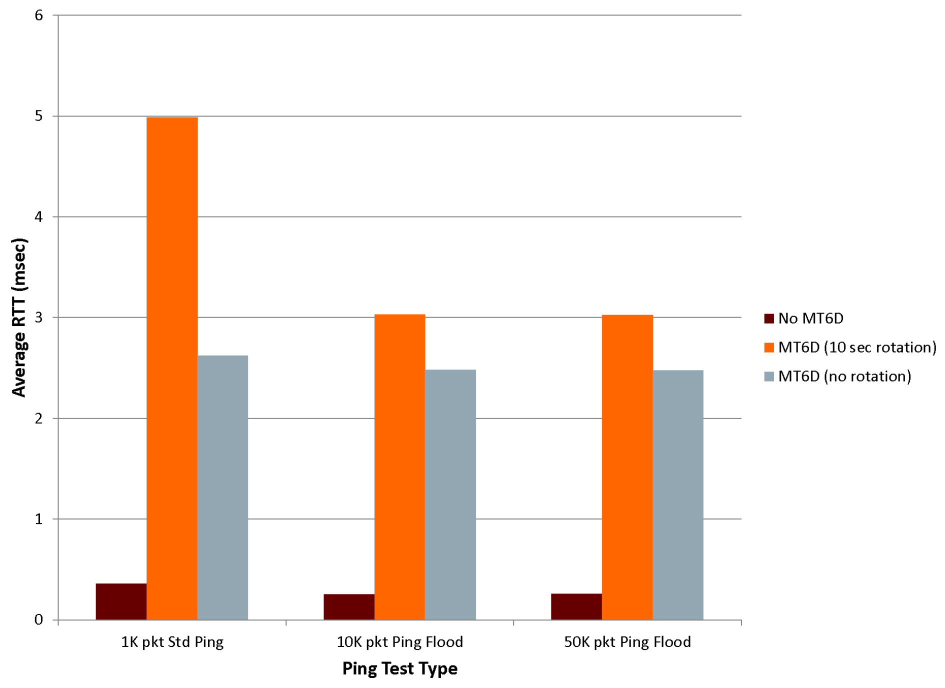


(a) Average transfer speed

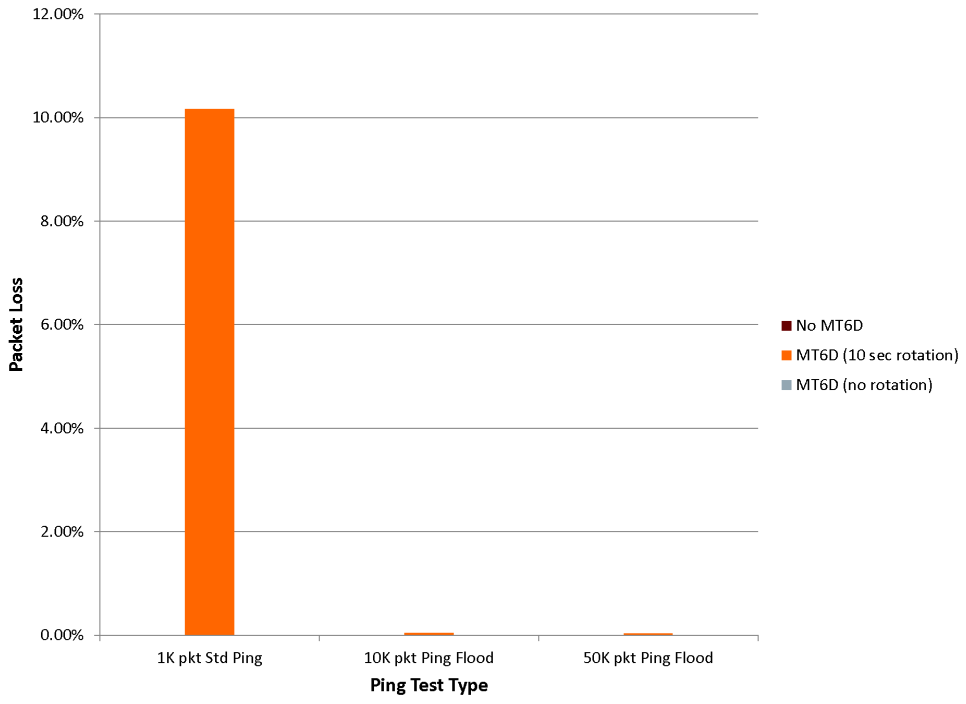


(b) Average percentage of retransmitted packets

Figure 10: Average transfer speed and packet loss of connection-oriented traffic using encrypted tunnel mode verses unencrypted tunnel mode

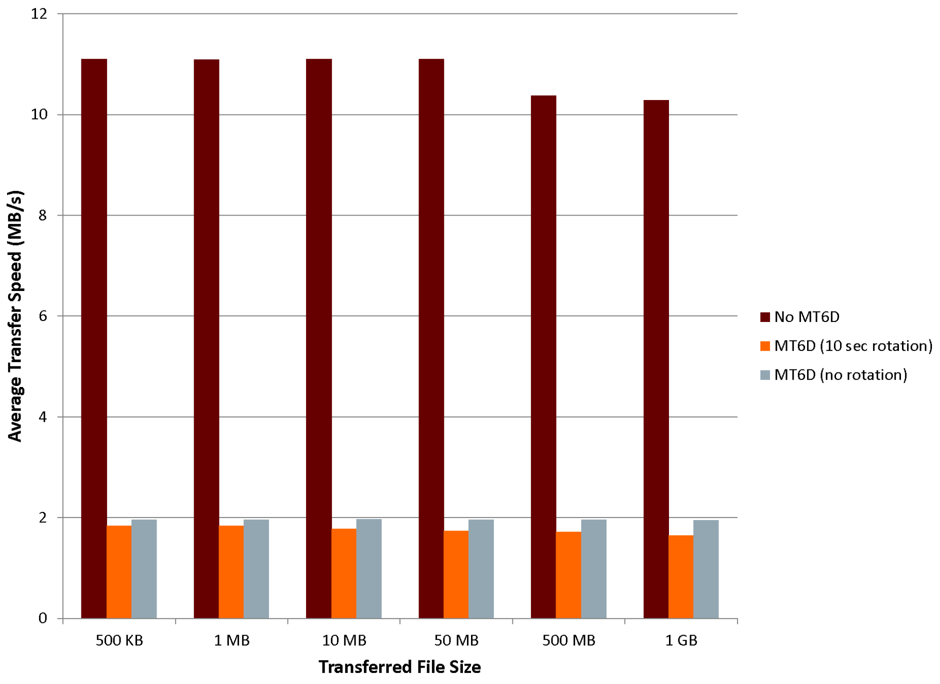


(a) Average RTT

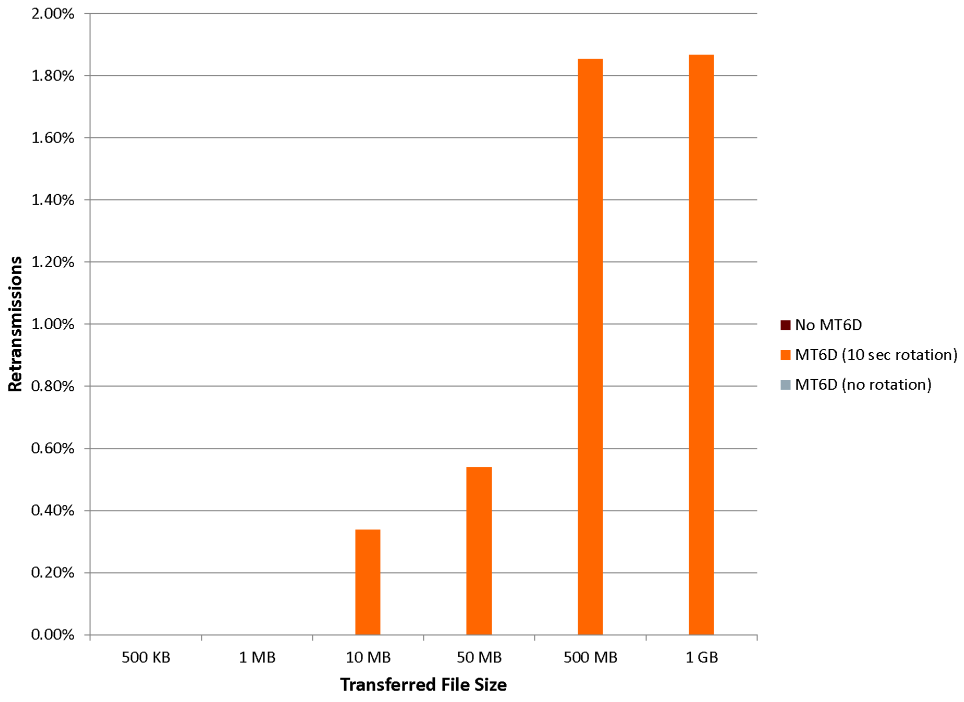


(b) Average packet loss

Figure 11: Average speed and packet loss of connectionless traffic for MT6D running between two subnets. A comparison was done without MT6D, with MT6D using the default test settings, and with MT6D using a fixed address.



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 12: Average transfer speed and packet loss of connection-oriented traffic for MT6D running between two subnets. A comparison was done without MT6D, with MT6D using the default test settings, and with MT6D using a fixed address.