Achieving Security and Privacy in the Internet Protocol Version 6 Through the Use of Dynamically Obscured Addresses

Matthew William Dunlop

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Joseph G. Tront, Chair
C. Patrick Koelling
Randolph C. Marchany
Scott F. Midkiff
Patrick R. Schaumont

March 15, 2012
Blacksburg, Virginia

Keywords: IPv6, Security, Privacy, Moving Target Defense, Dynamic Addressing

Achieving Security and Privacy in the Internet Protocol Version 6 Through the Use of Dynamically Obscured Addresses

Matthew W. Dunlop

ABSTRACT

Society's increased use of network applications, such as email, social networking, and web browsing, creates a massive amount of information floating around in cyber space. An attacker can collect this information to build a profile of where people go, what their interests are, and even what they are saying to each other. For certain government and corporate entities, the exposure of this information could risk national security or loss of capital. This work identifies vulnerabilities in the way the Internet Protocol version 6 (IPv6) forms addresses. These vulnerabilities provide attackers with the ability to track a node's physical location, correlate network traffic with specific users, and even launch attacks against users' systems. A Moving Target IPv6 Defense (MT6D) that rotates through dynamically obscured network addresses while maintaining existing connections was developed to prevent these addressing vulnerabilities.

MT6D is resistant to the IPv6 addressing vulnerabilities since addresses are not tied to host identities and continuously change. MT6D leverages the immense address space of IPv6 to provide an environment that is infeasible to search efficiently. Address obscuration in MT6D occurs throughout ongoing sessions to provide continued anonymity, confidentiality, and security to communicating hosts. Rotating addresses mid-session prevents an attacker from determining that the same two hosts are communicating. The dynamic addresses also force an attacker to repeatedly reacquire the target node before he or she can launch a successful attack. A proof of concept was developed that demonstrates the feasibility of MT6D and its ability to seamlessly bind new IPv6 addresses. Also demonstrated is MT6D's ability to rotate addresses mid-session without dropping or renegotiating sessions.

This work makes three contributions to the state-of-the-art IPv6 research. First, it fully explores the security vulnerabilities associated with IPv6 address formation and demonstrates them on a production IPv6 network. Second, it provides a method for dynamically rotating network addresses that defeats these vulnerabilities. Finally, a functioning prototype is presented that proves how network addresses can be dynamically rotated without losing established network connections. If IPv6 is to be globally deployed, it must not provide additional attack vectors that expose user information.

Dedication

With utmost sincerity, this work is dedicated to my wife, Tricia, and our four children. Without their strength and encouragement, none of this would have been possible. Thank you for believing in me and for your loving support during this endeavor. A special thanks goes out to my wife for shouldering the burden of maintaining the family throughout this entire process (and many others).

Acknowledgments

Though only my name appears on this dissertation, there have been a great many people who have helped contribute to my success. To those people, I give my sincerest appreciation.

My deepest gratitude goes out to my advisor, Dr. Joseph Tront. I have been amazingly fortunate to have an advisor who gives me the latitude to explore my own research ideas, while at the same supporting me through obstacles. Dr. Tront is what every graduate student hopes for in an advisor. Throughout the process, he was patient, friendly, and accessible. I always felt like he had time for me. He also made sure to provide the direction needed to keep me on track.

I also express my heartfelt thanks to Professor Randy Marchany. From the first time I met Randy (a year before coming to Virginia Tech), I knew I wanted to work for him. Randy is an expert in his craft and always excited about new research ideas. Throughout this process, Randy was an excellent sounding board. I cannot count the number of times I sat in his office and discussed ideas with him. I really appreciate him taking me on as a graduate student and offering me the opportunity to work in his lab.

To the rest of my committee, Dr. Pat Koelling, Dr. Scott Midkiff, and Dr. Patrick Schaumont, I am profoundly grateful they agreed to shepherd me through my Ph.D. journey. I know each of them is very busy, yet they were always there to listen and offer council whenever I asked. I also appreciate the ideas they gave me to make this research better.

I am truly grateful for the research collaboration I received from both Stephen Groat and William Urbanski. Their efforts and ideas helped take MT6D from a simple concept to a realized design. Many published works and even a filed patent would not have been possible without their assistance. I only hope that one day I can return the support they provided me.

I wish to thank a number of other undergraduate and graduate students who helped make this work possible. To Jamie Schwinabart and Matt Pierson, I offer special thanks. They are two of the brightest undergraduate students I have met. I was truly fortunate to have had their help coding the MT6D prototype and working through the numerous modifications and "enhancements" that occurred along the way. Thanks to Brittany Clore for her help getting an IPv6 simulation environment up and working. She diligently worked through issue after issue to achieve an unparalleled IPv6 simulation environment. I appreciate her dedication during this final push to finish up. To Brian Waldon, thanks for his early help in getting the first version of MT6D off the ground. I know it was challenging for him being an undergraduate, working full-time, and still having time to help me out.

Thanks to Philip Kobezak, Nicholas Pachis, and Rich Sparrow of the Virginia Tech Information Technology Security Office. Their willingness to drop what they were doing to get me what I needed for my research was invaluable. A special thanks goes out to Rich for his friendship, sense of humor, and willingness to listen to my frustrations.

I would also like to thank my parents, Jerome and Patricia Dunlop. The work ethic they instilled in me growing up led to my success in this and many other endeavors.

Finally, I thank the United States Army and the Electrical Engineering and Computer Science Department at West Point for planting the seed that led me to pursue a doctorate degree and affording me the opportunity to attend Virginia Tech.

Contents

List of Figures

List of Tables

List of Abbreviations

AES . . . . . .Advanced Encryption Standard

AH . . . . . .Authentication Header

ARP . . . . . .Address Resolution Protocol

ASD . . . . . .Adaptive Self-Synchronized Dynamic Address Translation

AUP . . . . . .Acceptable Use Policy

CBC . . . . . .cipher-block chaining

CGA . . . . . .Cryptographically Generated Address

DAD . . . . . .Duplicate Address Detection

DHCP . . . . . .Dynamic Host Configuration Protocol

DHCPv6 . . . . . .Dynamic Host Configuration Protocol for IPv6

DKG . . . . . .dynamic key generation

DNS . . . . . .Domain Name System

DoJ . . . . . .Department of Justice

DoS . . . . . .Denial-of-Service

DSO . . . . . .Dynamic Subnet Obscuration

DUID . . . . . .DHCP Unique Identifier

DYNAT . . . . . .Dynamic Network Address Translation

ESP . . . . . .Encapsulating Security Payload

EUI-64 . . . . . .64-bit Extended Unique Identifier

FCC . . . . . .Federal Communication Commission

GIL . . . . . .global interpreter lock

GPS . . . . . .Global Positioning System

GUI . . . . . .Graphical User Interface

HTTP . . . . . .Hypertext Transfer Protocol

i-DKG . . . . . .“in-band” DKG

IANA . . . . . .Internet Assigned Numbers Authority

ICI . . . . . .Interface Control Information structure

ICMP . . . . . .Internet Control Message Protocol

ICMPv6 . . . . . .Internet Control Message Protocol version 6

IID . . . . . .interface identifier

IP . . . . . .Internet Protocol

IPsec . . . . . .Internet Protocol Security

IPv4 . . . . . .Internet Protocol version 4

IPv6 . . . . . .Internet Protocol version 6

LAN . . . . . .local area network

LLMNR . . . . . .Link-local Multicast Name Resolution

MAC . . . . . .Media Access Control

mDNS . . . . . .Multicast DNS

MITM . . . . . .man-in-the-middle

MT6D . . . . . .Moving Target IPv6 Defense

MTU . . . . . .maximum transmission unit

NAT . . . . . .Network Address Translation

NDP . . . . . .Neighbor Discovery Protocol

NIC . . . . . .Network Interface Controller

NTP . . . . . .Network Time Protocol

o-DKG . . . . . .“out-of-band” DKG

OS . . . . . .operating system

OSPF . . . . . .Open Shortest Path First

OUI . . . . . .Organizational Unique Identifier

PII . . . . . .personally identifiable information

PRN . . . . . .pseudo random number

QoS . . . . . .Quality of Service

RIP . . . . . .Routing Information Protocol

RTT . . . . . .round trip time

SEND . . . . . .SEcure Neighbor Discovery

SHA . . . . . .Secure Hash Algorithm

SITL . . . . . .System-in-the-Loop

SLA . . . . . .service-level agreement

SLAAC . . . . . .StateLess Address AutoConfiguration

SRT . . . . . .Shared Routing Table

TARP . . . . . .Tunneled Agile Routing Protocol

TCP . . . . . .Transmission Control Protocol

TLS . . . . . .Transport Layer Security

TTL . . . . . .time to live

UDP . . . . . .Unreliable Datagram Protocol

VoIPv6 . . . . . .Voice over IPv6

VPN . . . . . .Virtual Private Network

Wi-Fi . . . . . .Wireless Fidelity

Chapter 1

Introduction

32 bits ought to be enough address space
- Vint Cerf, 1977

The current Internet Protocol version 4 (IPv4), is rapidly running out of address space [50, 67, 82]. The Internet Protocol version 6 (IPv6) solves the address space issue by providing nearly $8 \cdot 10^{28}$ times more addresses. Unfortunately, the cost, administrative overhead, and potential security issues have delayed IPv6 deployment. Researchers have even created some stop-gap techniques such as Network Address Translation (NAT), to prolong the life of IPv4. Despite these efforts, however, the majority of the Internet community will be forced to make the transition to IPv6 soon [46]. As with any new protocol, there will initially be undiscovered vulnerabilities. Now is the time, before IPv6 is fully deployed globally, to discover these vulnerabilities.

One vulnerability that exists is a direct result of how IPv6 forms addresses. In order to reduce administrative burden, IPv6 designers implemented a technique for nodes to generate their own addresses. This technique is called StateLess Address AutoConfiguration (SLAAC). Unfortunately, this method exposes a host's Media Access Control (MAC) address globally. Additionally, the portion of the address formed using the MAC remains static regardless of the network to which the host connects. A third party could use this static portion of the IPv6 address to track users from virtually anywhere in the world. The static address even facilitates targeted monitoring of network traffic. The reduced administrative burden is not worth the sacrifice in privacy.

To add security to IPv6 addressing, this research proposes a moving target defense that dynamically obscures IPv6 addresses. This chapter provides an overview of the research

effort. Section 1.1 outlines the problem statement under investigation. Some background and the motivation for this research are presented in Section 1.2. Section 1.3 details the objectives for this research while the questions answered by this research are presented in Section 1.4. The approach to conducting this research is briefly discussed in Section 1.5. The final section of this chapter, Section 1.6, outlines the structure for the remainder of this document.

1.1 Problem Statement

The purpose of this research effort is to design, implement, and test a technique for achieving a network layer moving target defense for IPv6. This technique, called the Moving Target IPv6 Defense (MT6D), protects users' security, privacy, and anonymity by dynamically rotating network and transport layer addresses. This research addresses the security flaws in the way IPv6 forms addresses. Specifically, this research solves the static nature of SLAAC and the global exposure of the MAC address. SLAAC forms the interface identifier (IID) portion of the IPv6 address, typically using the MAC address. Employing a static IID that remains the same regardless of subnet, exposes hosts to targeted attacks, address tracking, and network traffic correlation. Utilizing the MAC in the IID provides attackers with information that can be used to trace IIDs back to actual users. This research demonstrates these claims. This research also proves the validity of rotating addresses mid-session without impacting existing connection states. Further, it describes the overhead incurred with the new scheme and suggests that, in general, the benefits produced by the new scheme far outweighs the penalty incurred.

1.2 Background and Motivation

Migration to IPv6 is imminent. Based on predicted Internet growth, the number of network-connected devices will soon far surpass the size of the IPv4 address space [37, 48]. As a precursor to this eventual exhaustion, the Internet Assigned Numbers Authority (IANA) assigned its remaining five IPv4 address blocks on 3 February 2011 [47, 67, 82]. The address space of IPv6 is large enough to support Internet growth for the foreseeable future due to an address size of 2^{128} addresses (or $3.4 \cdot 10^{38}$ addresses). Manually managing this large address space will be complex and time consuming.

One solution being used to solve the problem of subnet management in IPv6 is SLAAC.

SLAAC allows an administrator to configure the network portion of the address, while each device automatically configures the host portion, or IID, of the address. The IID is often formed by extending the 48-bit MAC address to a 64-bit number, spanning half of the IPv6 address.

Using a node's MAC address in the IID has serious unintended consequences to a user's privacy. The issue is not only that the MAC address is used as the IID, but also that the IID remains static. As a result, no matter what network the node accesses, the IID remains the same. Consequently, simple network tools such as ping and traceroute permit tracking a node's geographic location from anywhere in the world. All attackers need to know is the location of the subnet. 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. Prevention of these vulnerabilities is the reason that MT6D was designed. A secondary goal of MT6D is to protect against certain types of network attacks.

MT6D preserves hosts' privacy primarily through dynamic obscuration of sender and receiver addresses. This 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, it is meant 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.

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 Denial-of-Service (DoS), replay, session hijacking, and man-in-the-middle (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. The only way for the attacker

to accomplish these address-based attacks is by compromising the symmetric key shared by the communicating hosts. Even a key compromise has a limited window of effectiveness in MT6D due to periodically changing symmetric keys. Application-layer attacks, however, are not prevented since MT6D operates at the network layer.

1.3 Research Objectives

One objective of this research is to analyze the vulnerabilities associated with IPv6 addressing techniques and develop a methodology for mitigating these vulnerabilities. The major risks associated with IPv6 addressing are that addresses are static and that they contain information that can expose users' identities. The second, and main objective of this research, is to design a dynamic addressing methodology. This methodology must continuously rotate addresses often enough that an attacker cannot achieve significant gain from launching physical or privacy attacks against systems. The methodology must also obscure addresses in such a way that no identifiable information about a host is exposed. By achieving these objectives, hosts will be protected against address tracking, traffic correlation, and targeted network attacks.

1.4 Research Questions

This research addresses the following questions:

1. What are the vulnerabilities in IPv6 addressing schemes?
   (a) How can attacks against these vulnerabilities be conducted?
   (b) What are the impacts to users as a result of these vulnerabilities?
   (c) Why do these vulnerabilities exist?
2. How can addressing in IPv6 be made more secure?
3. What are the benefits of dynamic addressing in IPv6?
   (a) How does dynamic addressing reduce the probability of an attacker targeting a host?
   (b) How does dynamic addressing limit the damage caused by a targeted attack?

   (c) What impact does the size of the subnet have on the statistical likelihood of target detection?

4. Is dynamic addressing feasible?

   (a) Will the default addressing protocol in IPv6 allow hosts to dynamically bind addresses?

   (b) Can addresses change mid-session without causing existing connections to fail?

   (c) Can both the sender and receiver maintain synchronized obscured addresses?

   (d) Can the sender and receiver periodically generate new synchronized keys?

   (e) What are the limitations of dynamic addressing?

5. Can dynamic addressing obscure both the host and subnet portions of the address?

   (a) Does rotating subnets improve the anonymity of communicating hosts?

   (b) How does obscuring subnets change the trust model?

   (c) What are the ramifications of a violation in the trust model?

6. What is the cost of dynamic addressing?

   (a) What is the overhead of dynamic addressing?

   (b) How much latency and packet loss does dynamic addressing incur on a live network?

   (c) How much latency and packet loss does dynamic addressing incur in a simulated network?

   (d) What are some methods that can increase dynamic addressing performance?

1.5 Methodology Overview

This research was conducted in three phases. The first phase involves analysis of the vulnerabilities of IPv6 addressing schemes. These vulnerabilities were proven by exploiting them on a live production IPv6 network. In the second phase, a method for dynamically obscuring IPv6 addresses was designed. This design leverages the default IPv6 stateless addressing scheme. In the final phase, a proof of concept prototype was developed that

proves the validity of the dynamic addressing methodology. This prototype is evaluated both in simulation and on a live production IPv6 network to gain an understanding of the performance issues and how it may be optimized in future research.

1.6 Organization of Dissertation

The remainder of this document is organized as follows. Chapter 2 provides an overview of relevant network protocols. Chapter 3 analyzes the vulnerabilities associated with IPv6 addressing techniques. Proof of the vulnerabilities is also provided. Some existing and proposed solutions to the IPv6 addressing vulnerabilities are discussed in Chapter 4. Chapter 5 examines the benefits of dynamic addressing over static addressing. Chapter 6 provides a brief overview of the dynamic address obscuration technique, called MT6D. The threats that MT6D is designed to protect against are discussed in Chapter 7. Chapter 8 introduces the design of MT6D. The configuration and initial results of a functioning prototype MT6D on a live network are presented in Chapter 9. Simulation testing of MT6D was performed to test under different network conditions. These results are presented in Chapter 10. Finally, future research and concluding thoughts are offered in Chapter 11.

Chapter 2

Overview of Network Protocols

This chapter describes the main protocols on which this research is based. The primary protocol is the next generation Internet protocol, IPv6. As of 2012, IPv6 is not yet globally deployed. The depletion of IPv4 address availability will eventually force adoption of IPv6.

Since this research focuses on IPv6 addressing, the remaining sections in the chapter describe the two IPv6 addressing approaches. The first is stateless addressing provided through SLAAC. The second is stateful addressing provided through the Dynamic Host Configuration Protocol for IPv6 (DHCPv6).

2.1 Internet Protocol version 6

IPv6 is the next generation network layer protocol. It was initially defined in 1998 by RFC 2460 [55]. IPv6 was designed as a successor to IPv4 and includes a number of enhancements. This section describes these enhancements as well as the specific format of the IPv6 address.

2.1.1 IPv6 Enhancements

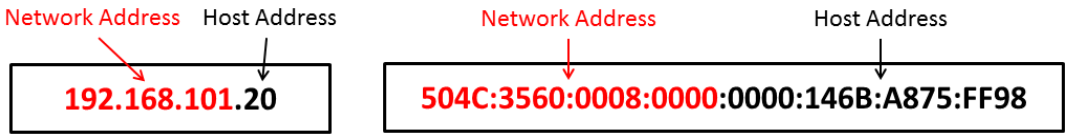
IPv6 claims five major improvements over IPv4 [55]. The biggest improvement is the increase in address size. Second, the header format was simplified. Another improvement was removing some of the restrictions on the option field as well as adding the capability to define new options. The addition of flow labels was also incorporated into IPv6. The fifth major improvement to IPv6 is the incorporation of Internet Protocol Security (IPsec) [43].

Increased Address Size

IPv6 was primarily developed to support more address space in the Internet [51]. An IPv4 address, consisting of 32 bits, provides approximately 4.2 billion possible address combinations. This address space is not sufficient to support the emerging myriad of Internet capable devices. Therefore, the IPv6 address was expanded to 128 bits. This new address size allows for 2^{128} possible addresses, approximately $5 \cdot 10^{28}$ addresses for every one of the 6.8 billion people [84] in the world. The difference between IPv4 and IPv6 address formats is depicted in Figure 2.1.

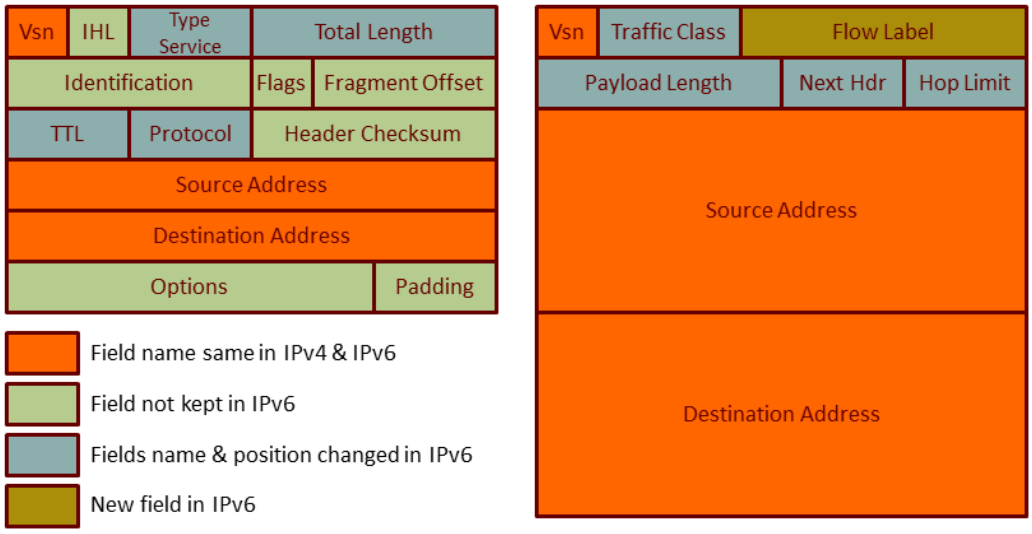
Simplified Header Format

The next major improvement in IPv6 is a simplified header. In IPv4, each packet header was of variable length from 20-60 bytes. The variability of each packet header was caused by the options field. The options field has to be checked by each router along the path, regardless if the packet contains any options. The effect is less efficient processing of IPv4 packets at each hop. Additionally, the header contains unnecessary and infrequently used fields. IPv6 makes the header much more efficient by removing these extraneous fields. The header length in IPv6 is fixed at 40 bytes. Additionally, the options are moved completely out of the header. The options in IPv6 are found at the start of the payload, which immediately follows the header. This removes the need to process any options unless there actually are some. The presence of options is indicated by the next header field. Each router need only check the 40-byte header and can, therefore, easily predict the start of the payload. This is much more efficient and especially helpful for real-time applications [18]. A comparison of the IPv4 and IPv6 headers are provided in Figure 2.2.



(a) IPv4 address format (b) IPv6 address format

Figure 2.1: IPv4 versus IPv6 address format. The IPv4 address in (a) contains 2^{32} bits and is represented by dotted decimal notation. The IPv6 address in (b) contains 2^{128} bits and is represented by colon-hexadecimal notation.



(a) IPv4 header format (b) IPv6 header format

Figure 2.2: IPv4 versus IPv6 header comparison. (a) reflects the IPv4 header format. The IPv6 header format is illustrated in (b). The different colors highlight the differences in the two headers.

Flexible and Expandable Options

In IPv6 the options have been moved outside of the header. By moving the options out of the header and into the payload, the number of options that can be applied to a packet is much more flexible. Instead of being limited to 40-bytes, like in IPv4, options can span as much of the payload field as needed. Each option has a next header field used to indicate if there is another option following. The location of the options and the way they are referenced allows for future extension of the protocol by removing restrictions on what gets defined for use and when it gets defined. Figure 2.3 illustrates how options in IPv6 fit into a packet. The header and each option contain a next header field that indicates the type of option that follows. If no options follow, the next header field contains a value of 59 [14].

Flow Labeling

Flow labeling allows for classification of packets belonging to particular flows. A client can assign a specific label to a group of packets that require special handling [14]. In a sense, flow label in IPv6 can be considered a Quality of Service (QoS) mechanism. Normally, any type of QoS is done end-to-end, above the network layer. By including flow labeling at the network layer, the higher layers do not have to be checked to determine to which flow a

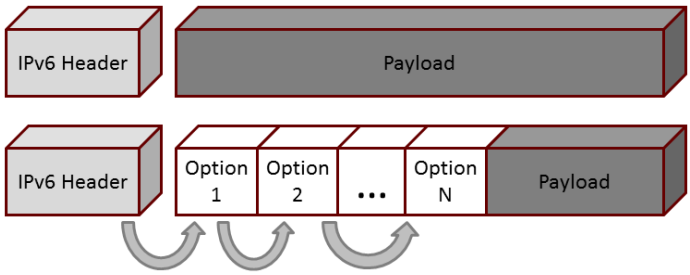


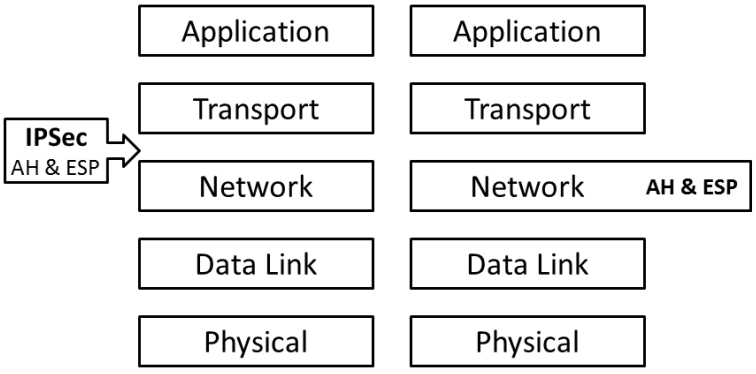
Figure 2.3: Options in IPv6 are external to the packet header. This provides more flexibility in the number and length of options.

packet belongs. This is useful because higher layers do not sort packets until they reach their destination. By including the flow label in the IPv6 header, each router can determine what flow a packet belongs to and handle it accordingly. At the time of this research, however, standardization of the flow label field has not been defined [14].

Integrated Security

The final major improvement in IPv6 is the integration of authentication and encryption into the protocol. When IPv4 was designed and defined, security was not a major focus. As Internet commerce and secure communication have become a staple of Internet activity, IPsec [43] was developed to address the security concerns when utilizing IPv4. IPsec served the purpose of adding security without having to redefine IPv4. Without redefining IPv4, IPsec became a layer 3.5 protocol and did not integrate well into the protocol stack (see Figure 2.4). This implementation causes problems when encryption or authentication is performed in tunnel mode. Because a packet has to be completely formed before IPsec can be applied and a new IPsec header added, the datagram has to travel down to layer 3, then back up to layer 3.5, and then back down to layer 3. Integrating these functions within IPv6 means that the entire operation can be performed at layer 3 as illustrated in Figure 2.4. This adds efficiency by increasing throughput and eliminating the need to store the packet during processing. Also, with IPsec in IPv4, security was usually applied only at the gateways due to the additional software required for packet processing. Including authentication and encryption in the IPv6 packet brings security to the end host.

As mentioned, IPsec provides both authentication and encryption to IPv6. Authentication is provided through the use of an Authentication Header (AH). Encryption is provided through the use of an Encapsulating Security Payload (ESP). IPsec can also be applied in



(a) IPsec in IPv4 (b) IPsec in IPv6

Figure 2.4: IPsec integration in IPv6. (a) represents how IPsec fit into the network stack in IPv4. (b) represents how IPsec is integrated into the network stack in IPv6.

two modes. The first mode is transport mode. In transport mode, only the payload is authenticated or encrypted. The other mode is tunnel mode. Tunnel mode encrypts or authenticates the entire packet.

2.1.2 IPv6 Address Format

As mentioned in Section 2.1.1, an IPv6 address contains 128 bits. Addresses in IPv6 are represented using colon-hexadecimal notation as illustrated in Figure 2.5. Since repeatedly referring to 32 hexadecimal numbers can be tedious, IPv6 addresses can be abbreviated as demonstrated in Figure 2.6. Consecutive hexadecimal quads of zeros can be represented by a double colon as illustrated in Figure 2.6. To be able to predict the the number of omitted quads, the double colon can only be used once in an IPv6 address [19].

Addresses in IPv6 contain both a subnet and a host portion of the address. The host

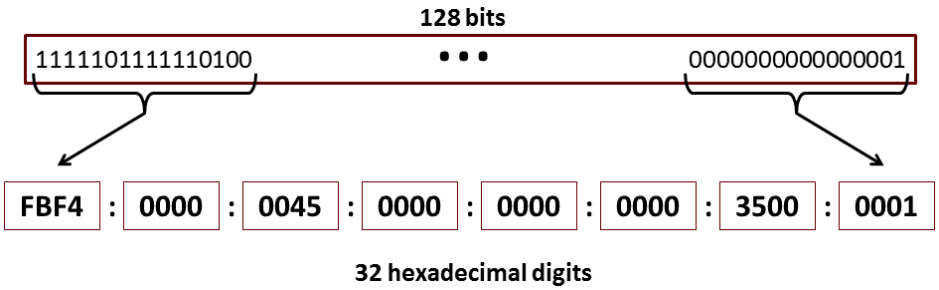


Figure 2.5: IPv6 colon-hexadecimal notation

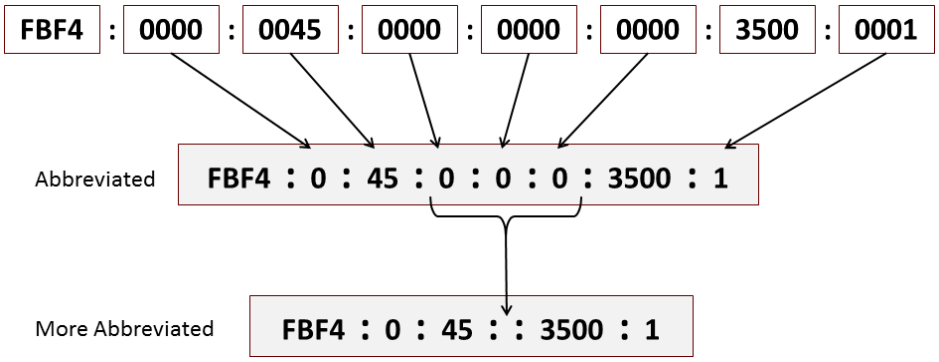


Figure 2.6: Abbreviation of an IPv6 address

portion of the address is referred to as the IID. Both portions can be of variable length, but are typically 64 bits based on the 64-bit Extended Unique Identifier (EUI-64) format [38] (see Figure 2.7).

There are three different types of IPv6 addresses – unicast, multicast, and anycast. Unicast addresses can be further broken down into global unicast and link-local [38]. Global addresses are publicly routable. Link-local, on the other hand, are private addresses, local to the subnet. They are typically used for global address configuration and neighbor discovery. Multicast addresses serve the same purpose in IPv6 as they do in IPv4. Anycast addresses [38] are new in IPv6. Anycast is similar to multicast in that a group of interfaces is defined and all group members receive packets addressed to that group. The difference is that a packet sent to an anycast address is routed to "nearest" interface according to routing metrics used. The interface that receives the packet forwards it to all other group members. Packet delivery examples for each of the three different message types are illustrated in Figure 2.8. Broadcast addresses no longer exist in IPv6.

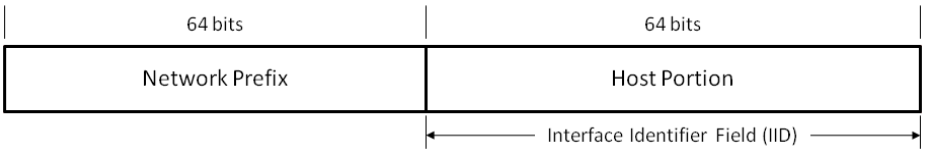
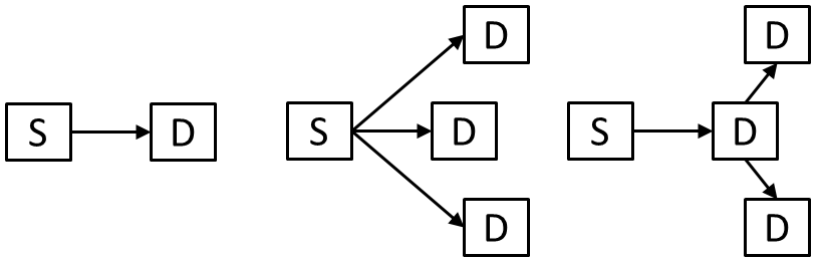


Figure 2.7: IPv6 address components



(a) Unicast (b) Multicast (c) Anycast

Figure 2.8: Packet delivery examples for the three different IPv6 address types

2.2 Stateless Address Autoconfiguration (SLAAC)

The large size of the IPv6 address space requires a new network address configuration architecture to simplify network administration. For this reason, IPv6 contains a means for hosts to self-determine their Internet Protocol (IP) addresses, called SLAAC. SLAAC functions by generating the IID component of the IPv6 address at the host. The host portion of the address is supplied by the Neighbor Discovery Protocol (NDP).

NDP was designed as a replacement for the Address Resolution Protocol (ARP). NDP [58] assists with self-determination of addresses by performing three main tasks. First, NDP keeps track of all nodes occupying a subnet. This is important when determining neighbor reachability and if duplicate addresses exist on the subnet. The second task is to supply hosts with subnet information required to form the first 64 bits of the address. The third task is to purge invalid links. NDP performs a number of other tasks as well that are crucial to the proper functioning of IPv6.

The primary means SLAAC employs to generate an IID is through the use of the EUI-64 format. The EUI-64 format uses the host's MAC address to form the IID. First, the host's 48-bit MAC address is split into two 24-bit halves. The 16-bit hex value `0xFFFE` is inserted between the two halves to form a 64-bit address. Also, the universal/local flag, located at bit seven of the 64-bit host portion, is set to 1. Figure 2.9 illustrates this process. The final step combines the 64-bit network address with the 64-bit host address to form a complete 128-bit IPv6 address as demonstrated in Figure 2.7. The 64-bit network address is supplied by the gateway device by way of a NDP router advertisement.

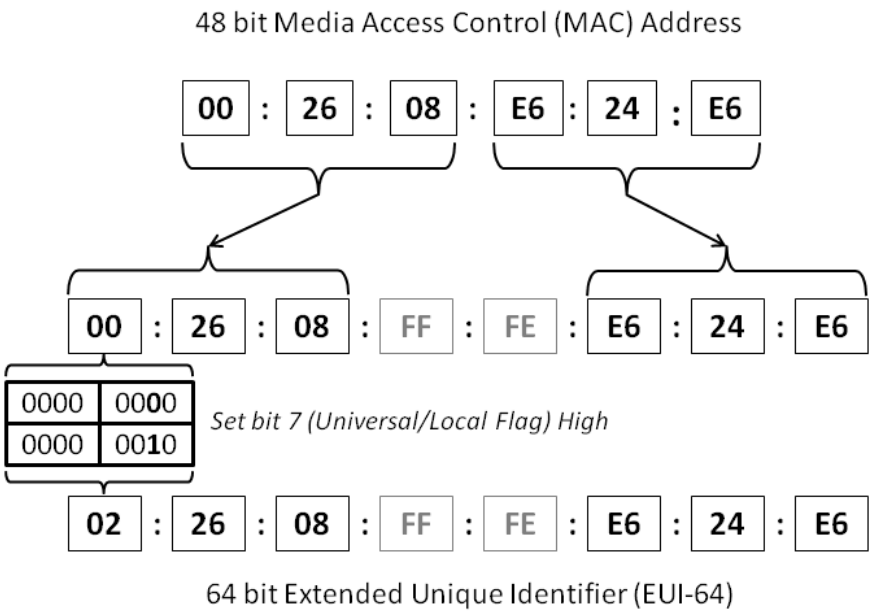


Figure 2.9: 64-bit Extended Unique Identifier (EUI-64) format

2.3 Dynamic Host Configuration Protocol for IPv6

DHCPv6 [17] provides a stateful, managed alternative to SLAAC. Stateful address assignment with DHCPv6 functions in much the same way as Dynamic Host Configuration Protocol (DHCP) in IPv4. Hosts connected to the network are issued addresses by a DHCP server. In a heavily populated IPv6 subnet, this could place an excessive administrative burden on network managers. For this reason, SLAAC was chosen as the default address configuration scheme for IPv6. There are advantages, however, to using DHCPv6. For example, DHCPv6 can restrict network access to only authenticated hosts. An administrator can also divide and restrict network access through the use of DHCP realms.

When DHCPv6 is used, a node exchanges a series of four messages with the DHCP server to obtain an IPv6 address. This message exchange sequence is illustrated in Figure 2.10. The client begins by sending a SOLICIT message using its link layer address as the source and a special DHCP server-specific multicast address as the destination. This SOLICIT message is received by all servers and relays on the subnet. Another component of the SOLICIT message is the client identifier, which contains the client's unique DHCP Unique Identifier (DUID). Any server capable of servicing the client's request, responds to the client's link local address with an ADVERTISE message. The ADVERTISE messages contains

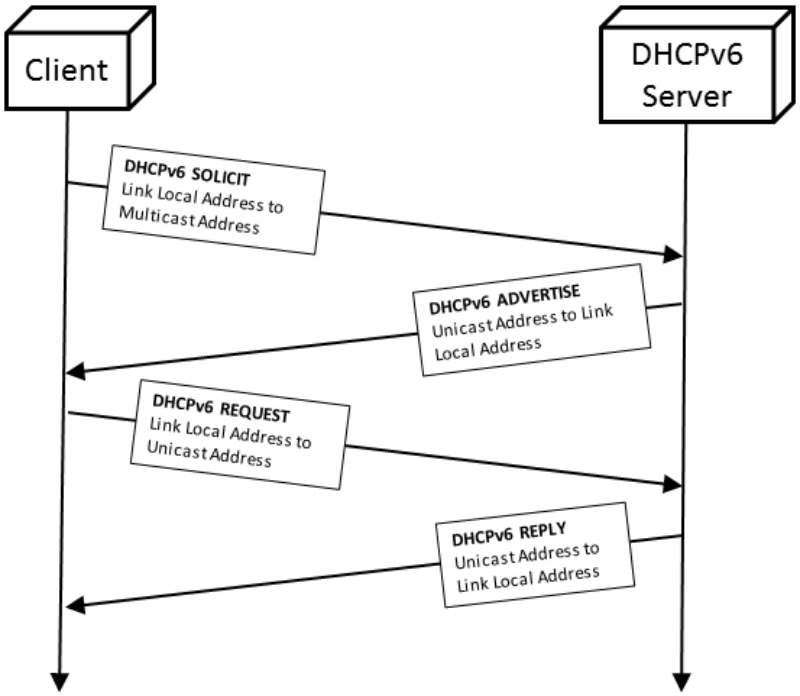


Figure 2.10: DHCPv6 message exchange

the leased address and any other configuration parameters necessary for the network. The ADVERTISE message contains both the client's and server's unique DUIDs. The client then chooses a server from the received ADVERTISE messages and sends a REQUEST message. The REQUEST message asks the server to confirm the advertised address as well as any additional configuration options. The server sends a REPLY message to the client confirming the address and configuration options. Regardless of whether DHCPv6 or SLAAC is used to issue addresses, servers and clients also use NDP [58] to exchange link-local addresses and learn of connected gateway devices.

2.4 Summary

This chapter presented the protocols that this research relies on. Since the core of this research is IPv6, an overview of the protocol was provided. Specifically, this research focuses on improving the security of IPv6 addressing. To that end, specific emphasis was placed on describing the format of IPv6 addresses. Both IPv6 address assignment approaches were also described.

Chapter 3

Address Tracking and Correlation

IPv6 offers two methods for forming addresses. The default method is a stateless addressing scheme, called SLAAC. This method allows hosts to form the last 64 bits of the address on their own. The second method uses DHCPv6 and is usually referred to as a stateful scheme since the state of the address is determined and maintained by the DHCPv6 server that assigns the address. DHCPv6 operates in much the same way as DHCP in IPv4.

There are flaws associated with both addressing schemes in IPv6. Both methods give away the identity of the hosts and permit targeted network attacks, address tracking, and traffic correlation. This chapter describes the flaws associated with both addressing schemes.

3.1 Stateless Address Tracking in IPv6

In order to reduce the administrative burden of managing such a large address space, IPv6 designers implemented a technique for nodes to generate their own addresses. This technique, called SLAAC, exposes a host's MAC address globally. Additionally, the portion of the address formed using the MAC remains static regardless of the network the host connects to. A third party could use this static portion of the IPv6 address to track users from virtually anywhere in the world. The static address even facilitates targeted monitoring of network traffic. The reduced administrative burden is not worth the sacrifice in privacy.

This sort of tracking was not possible in IPv4. In IPv4, a node's MAC address is restricted to the local subnet. Additionally, the MAC address is not associated with the IPv4 address. In fact, DHCP usually issues IPv4 addresses to hosts based on address availability. Furthermore, NAT also provides the unintentional benefit of protecting a host's identity by placing it within a private address space, not globally addressable.

3.1.1 Deterministic IID

Due to the current accepted definition of SLAAC on most operating systems, the IID of a node's IPv6 address is deterministic across networks. For the last 64 bits, the node automatically configures an address based upon the MAC address of its network interface device. By extending the 48-bit MAC address to 64 bits through the EUI-64 format [38] (see Figure 2.9), the IID of an IPv6 address is created.

While different operating systems configure IPv6 addresses differently, no current operating system (OS) implementations of IPv6 stateless addressing dynamically obscure the IID of all IPv6 addresses on the system. OS X and common Linux distributions, such as CentOS and Ubuntu, follow the EUI-64 format. The MAC address appears virtually unaltered in the IPv6 address. The Windows operating system obscures the host portion of an IPv6 address according to RFC 4941 and sets a temporary address [56, 83]. However, Windows operating systems also maintain another IPv6 address used for neighbor solicitation. This other IPv6 address contains an IID that is obscured but never changes, regardless of the subnet to which the node connects. Not dynamically obscuring a user's host portion for all of the IPv6 addresses associated with a system threatens a user's privacy. The static IID currently implemented in major operating systems can be identified with a particular node, even as the node changes networks.

Many mobile devices, such as Android and iPhone, support IPv6 in Wireless Fidelity (Wi-Fi). Their implementations follow the EUI-64 format providing these mobile devices with static IIDs that are easily tracked on their Wi-Fi connections. Since most users frequently carry their mobile devices and leave them on and connected, the ability to track a user is increased dramatically. While the need to address the privacy concerns in Mobile IPv6 has been identified, it does little good until the privacy concerns due to IID tracking are addressed. Since Mobile IPv6 would only be applied to the cellular connections and the majority of these wireless devices also deploy Wi-Fi, users can still be tracked through their wireless devices as they move between different Wi-Fi networks. Therefore, address privacy must be addressed in all connections of a mobile device to assure complete privacy.

3.1.2 Privacy Implications

The static IID created by the EUI-64 format and the Windows operating systems compromises a user's privacy. Creating a static IID from a MAC address allows nodes to be logically and geographically tracked as they travel to different networks. Since the EUI-64

format results in a deterministic IID, users can be tracked on a network by scanning different subnets and searching for the MAC-generated IIDs. Using simple commands such as ping and traceroute, the location of a user can be determined within reasonable geographic accuracy. Even the Windows obscuration of the IID within the IPv6 address does not protect a user. By locally capturing a user's traffic once, a specific user can be paired with the deterministically obscured IID and tracked with the same technique of searching subnets as used for unobscured host addresses. Since the obscuration occurs independent of the network, a Windows host carries the same obscured IID between networks.

By monitoring the traffic on a network over an extended period of time, a single user's traffic can be identified and analyzed. Armed with this data, a third party (whether malicious or not) can potentially tie a device to its actual user. As the user crosses different subnets, traffic can be collected and correlated by examining the static IID. This vulnerability to tracking does not typically apply when using IPv4. Most medium to large IPv4 networks implement DHCP, which changes user addresses randomly. As a result, DHCP logs are needed to tie traffic sniffed from a network with a particular user. Due to the deterministic IID in IPv6 address autoconfiguration, simple filters could be created to filter the traffic of a single user on any subnet. This would allow an interested party to identify and monitor a user's on-line activity through traffic analysis. In a dual-stack implementation where a node uses a mixture of IPv4 and IPv6, special Internet Control Message Protocol version 6 (ICMPv6) Neighbor Solicitation messages provide an interested party the IPv4 address linked with an IPv6 address. This correlation allows for traffic collection to extend to IPv4 for a single session.

Tracking a user or monitoring his/her on-line activity is not the only concern. If it was known that location or traffic monitoring was occurring, a malicious host could spoof the IID of an innocent node. The malicious node could then masquerade as the innocent node and create false traffic or locations using the innocent node's IID.

3.1.3 Testing

In order to prove geotemporal tracking and traffic analysis through IID analysis, testing was performed using IPv6 nodes on a live IPv6 network. Geotemporal tracking and traffic analysis were performed using an Android mobile device on the Virginia Tech wireless network with the cellular connection turned off. Since the Android operating system deploys an EUI-64 IID in stateless address autoconfiguration on Wi-Fi IPv6, both geotemporal tracking

and traffic analysis were possible.

Testing was conducted on the Virginia Tech production IPv6 network. Virginia Tech has a fully functional IPv6 network, providing globally unique addresses through stateless address autoconfiguration to every wireless and wired node on the network. The network contains six core routers which serve distinct geographic areas on campus. Subnets correspond with the core routers and, therefore, with distinct geographic locations on the Blacksburg campus. Packet capture for traffic analysis was performed at the border to assure that all traffic sent from different subnets could be captured and analyzed. Through subnet analysis, network sniffing, and IID analysis, geographic tracking and traffic analysis were performed on the Virginia Tech IPv6 network.

3.1.4 IID Tracking

Reconnaissance of the Virginia Tech campus located six unique wireless subnets. A simple script was created to continuously ping a specific IID on these six subnets and to record the date and time when the node successfully responded. Using a wireless node which was set to automatically associate with the Virginia Tech wireless network, the script was run while the node moved around the campus and associated with different access points. The results of this testing can be seen in Figure 3.1.

Geographically tracking users with static IIDs is possible on any stateless autoconfiguration network. While this example only demonstrates tracking node movement on the

Figure 3.1: Geotemporal plot of a wireless node's movement within the Virginia Tech network

Blacksburg campus of Virginia Tech, tracking could easily be expanded to cover other geographical areas that support IPv6. Currently, expansion is not possible due to the lack of significant-sized production IPv6 networks outside of the campus. By predetermining the network portion of the IPv6 addresses within an area of interest (e.g., a metropolitan area), an attacker can remotely scan for a user on any network and accurately determine the user's location.

IID Traffic Analysis

To perform traffic analysis on IPv6 IIDs, data was sent using an Android OS mobile device from multiple subnets at different times. The same wireless subnets were used for traffic analysis as were used to track user locations in Section 3.1.4. At the network border, a sensor was placed to monitor, sniff, and record all IPv6 traffic traveling over the network. The primary traffic collected was Google search queries. This is due to Google having a AAAA Domain Name System (DNS) record on the Virginia Tech network which returns an IPv6 address. Other traffic collected included YouTube search queries, Jabber client transmissions, and Gmail data.

The use of Transport Layer Security (TLS) inhibits IID traffic analysis since the data is encrypted for transmission. Some types of traffic, such as webmail traffic, bank traffic, and chat protocols, are often encrypted by default to prevent personally identifiable information (PII) from being intercepted by an attacker. Other traffic, such as search queries, social media posts, and daily browsing habits, have historically been unencrypted due to the extra bandwidth and processing required by TLS. Transmitting this traffic unencrypted allows for attackers to intercept data. Collected PII from this unencrypted data can be used to build a profile of users and determine the users' identities. For example, it was possible to monitor the user name and tweets sent by the test node over Twitter. Since many Twitter users tweet about their activity and location, it would be trivial for an attacker to monitor a user through Twitter. As protecting PII becomes more important due to increases in identity theft and related crimes, industry will likely respond by implementing encryption in everyday browsing activities. The introduction of more security focused websites, such as Google search through TLS, will make identifying a user through search traffic analysis increasingly more difficult. Until such a time, the large volume of unencrypted PII makes it relatively easy for attackers to exploit users through traffic analysis.

IID traffic analysis of users utilizing Windows operating systems is extremely difficult due to the use of privacy extensions [56]. Since the privacy extensions are configured to

automatically change the IID of a node at specific time intervals and as a node changes networks, it is impossible to use only the IID of a temporary address to analyze traffic. It is possible, however, to collect network traffic that most likely contains the target node's traffic. This is accomplished by analyzing the time to live (TTL) values of ping and traceroute packets sent to a node's permanent address and scanning for similar TTL values in packets sniffed on the network. This technique will, however, also contain many other nodes since TTL values may not vary for large portions of a network.

OUI Analysis

A MAC address is composed of two parts. The first half is called the Organizational Unique Identifier (OUI) and identifies the vendor that issues the Network Interface Controller (NIC). The second half is assigned by the vendor and is designed to act like a serial number to make the MAC unique [13]. Usually, agencies will contract a vendor to produce all of a specific product. This means that, with high probability, an entire product line shares the same OUI.

Analysis was conducted on the OUIs of traffic captured on the Virginia Tech network to determine the types of computers and operating systems communicating using IPv6. Of the 72,377 IPv6 addresses collected in 24 hours, 12,356 were expanded with the EUI-64 expansion format. Since Windows obscures the IPv6 address using privacy extensions, it is reasonable to conclude that the other 60,021 addresses are Windows systems. It is worth noting that the 12,356 EUI-64 addresses could contain a small margin of error since it is possible that privacy extensions could produce IIDs that mimic valid EUI-64 expanded addresses. It can be concluded, therefore, that approximately 83% of the network at Virginia Tech is comprised of Windows systems while the remaining 17% is made up of systems running some other operating system.

For the 17% of systems utilizing the EUI-64 expanded addresses, the OUI of each IID was analyzed and a list was compiled of the top five manufacturers as seen in Table 3.1. The large majority of these systems had wireless NICs registered to Apple, Inc. Since no mobile Apple operating system deployed IPv6 at the time this data was collected, all of the IPv6 traffic containing Apple OUIs comes from Apple computers. The remaining devices in the OUI analysis are registered to network interface manufactures. These OUIs most likely come from Linux and Unix systems using the default EUI-64 expansion format.

OUI analysis on collected traffic allows attackers to determine the most effective types of attacks to run on a specific network. To effectively use resources to gain entry into a

Table 3.1: Top five NIC manufacturers accessing Virginia Tech's network from EUI-64 systems

NIC Manufacturer	EUI-64 Traffic
Apple, Inc.	86.33%
Broadcom Corporation	5.23%
Intel	2.47%
3 Com Corporation	1.58%
Dell	0.42%

network, attacks should be run against the most common operating systems on a network. For example, the OUI analysis on the network at Virginia Tech shows that the majority of computers run Windows. To effectively launch attacks against a Windows machine in IPv6, attackers must obtain the permanent address of the machine. Therefore, a local device must be connected to the network that listens for the Neighbor Solicitation messages and any other multicast messages which use the permanent addresses of the Windows systems. While tools such as Nmap and Metasploit offer OS fingerprinting, these tools would waste resources scanning the large IPv6 address space and would return invalid, temporary addresses. Analysis of OUIs from captured traffic gives attackers a new tool to effectively collect statistics on system types connecting to a network.

An attacker could also use OUI analysis to locate all of a specific type of asset. This provides an attacker with the locations and numbers of specific types of systems. This may not seem that powerful. Imagine, however, that an attacker is able to identify a vulnerability specific to a particular brand of device. The attacker can target an exploit against those devices specifically. This type of attack may provide an attacker with another vector into critical or sensitive systems.

IID Spoofing

As previously mentioned, a malicious host can attempt to slander an innocent host by spoofing the innocent host's IID if IID analysis is in use. By spoofing a host's IID, the attacker can make the victim appear in potentially incriminating locations. Additionally, the attacker can produce slanderous traffic that would be associated with the victim. Of course, these types of attacks are only detrimental in environments where hosts are being monitored. In the subsequent experiments, four different operating systems were tested: Ubuntu Linux, OS X, Windows Vista SP2, and Windows 7.

Spoofing an IID using Ubuntu Linux and OS X is trivial. Since neither operating system uses any IID obscuration techniques, all the attacker needs to do is modify the MAC address and let the stateless address autoconfiguration reconfigure the address with the new IID. The MAC address was modified using the command: `ifconfig [interface] hw ether [hwaddr]`. The interface was brought down and then back up to produce a new IID. The same attack can be performed using a virtual machine with a manually configured MAC address. Using the MAC replacement technique requires the attacker to be on the same network segment as the victim. If the attacker is located elsewhere, he/she can simply spoof the entire IPv6 address.

Spoofing a Windows IID is more difficult due to default privacy protection enabled in the operating system. In Windows operating systems implementing privacy extensions [56], typical communications use a temporary, non-deterministic IID. Spoofing the temporary IID is not useful since it changes often and is not tied to a specific host. Therefore, the permanent IID is required. The only way to get the permanent Windows IID is by monitoring the neighbor solicitation messages, Link-local Multicast Name Resolution (LLMNR) messages, or Multicast DNS (mDNS) messages. This requires the attacker to be on the same network segment as the victim at some point since these messages are sent using the link-local or multicast addresses. Since even the permanent IID is obscured, the MAC replacement technique does not work. This was tested using the same NIC on multiple Windows Vista and Windows 7 machines. A different permanent IID was received each time. On a specific machine, however, the permanent address does not change as a node changes networks or over time. Therefore, the permanent address can be used for IPv6 address spoofing without issue.

3.1.5 Potential Consequences of IID Tracking

The ability to track users based off their stateless IPv6 addresses results in numerous consequences. Whether tracking is seen as helpful or malicious is a matter of perspective and application. First, the positive aspects of IID tracking are presented as a demonstration that not all applications are necessarily malicious. Next, some negative applications of IID tracking are presented. Regardless of the application, however, the positive aspects of IID tracking do not outweigh an individual's right to privacy.

The Good

There are many applications that network administrators, advertising executives, and law enforcement officials would argue provide positive results of having the capability to track users. One such application is referred to as converged security. This term refers to the ability to combine physical security with cyber security. By tracking a user's physical location, administrators are afforded with an additional layer of security. An application that advertising executives might find valuable is the capability to perform targeted marketing. Meanwhile, the numerous forensic possibilities provided by IID tracking would appear attractive to law enforcement officials.

Converged Security As threats against organizations and individuals evolve in scope and complexity, physical and logical security systems must converge to create a holistic approach to security [9]. By leveraging all pieces of an enterprise's infrastructure in creating a security solution, security policies can be effectively enforced and valuable intelligence can be collected to ensure the safety of the workers and the enterprise. Individuals can also deploy the same technologies on a smaller scale to protect their families and homes. Converged security expands the resources available to a security system, allowing for logical systems to improve physical security.

IID analysis could help an organization enforce current physical and logical security policies through monitoring of network resources. Using IID analysis could supplement current control measures and provide an extra layer of security to protect valuable resources. Physically, access control measures could be supplemented by IID tracking. If an attacker followed a valid user through a physical control access point to "tailgate" off of their access, physical access control would rely on either the user or security guards to recognize and catch the tailgater. Using IID analysis, a wireless access control point could be created which checks the IID of a user's wireless device. If not in the database, the user would be flagged and security would be alerted. Logically, file access could be logged and traced through IID analysis, providing a system administrator an extra safeguard to protect information. For example, if sensitive data was only supposed to be accessed by specific systems, IID analysis could be used by a logical control system to assure that only authorized systems access resources. Incorporating IID analysis into logical control systems adds another layer of defense for administrators.

In emergency situations, detailed tracking of resources and IIDs allows for an individual's location and building's human density to be established, allowing for a targeted response.

Determining who is in the building, where they are located, and where the population density is greatest in a disaster situation allows for first response teams to react with the proper resources to save the most lives possible. In a building fire, firefighters put themselves at risk searching for victims. Often, firefighters are exposed to unnecessary danger by searching unoccupied rooms. Crucial time is also wasted by this methodical search technique. By using IID analysis, the largest groups of victims could be identified to save the most possible lives. After the large groups have been evacuated, individuals could be quickly located through IID tracking. IID analysis can also be used to pinpoint those victims with special needs that might require special extraction equipment.

Similar to disaster scenarios, IID tracking could assist law enforcement during hostage situations. Law enforcement could use the IIDs of wireless devices within a building to determine the number, identities and approximate locations of hostages. The approximate number of captors and their locations might even be determined by looking at unknown IIDs within the structure. Armed with this information, SWAT teams can develop rescue plans that will minimize casualties. An example of this type of scenario is depicted in Figure 3.2(a), where white pins indicate the locations of hostages and gray pins indicate captor locations.

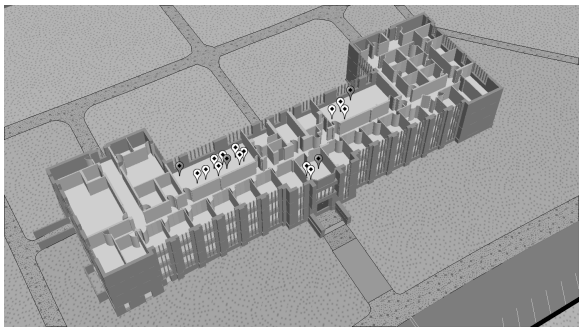
Personal and family security could also be enhanced through IPv6 tracking capabilities. Currently, many cellular phone service providers offer a locater services, such as a Global Positioning System (GPS), in phones to report the location of a unit. Since most users always carry their cell phones and have them powered up, this is a viable way of tracking people. While simple to implement and deploy, using a cellular phone as the only tracking device creates a single point of failure. If the phone is not in tower range, runs out of battery, or is turned off, the user cannot be tracked. As more devices become interconnected, the number of IPv6 addresses and IIDs associated with an individual will also increase. By monitoring all of the devices that have static or deterministic IIDs, tracking is moved from a single device to all of the Internet-capable devices that a user deploys daily. Since IID tracking is not reliant on GPS and can be used on any Internet-capable device, more of the devices that people carry can serve as tracking sources.

Targeted Marketing To find and attract customers, businesses rely on targeted marketing to focus resources on potential customers whose wants and needs align with the business's products or services. Targeting marketing requires large amounts of data about current and potential customers to render accurate decisions on where marketing resources should be deployed. Since advertising budgets are limited, focusing on customers who have a higher

The Good

In a hostage situation, the physical locations and identities of hostages can be ascertained to determine how many hostages there are and physically where they are located.

(a)



The Bad

Terrorists could track the number of distinct IIDs within a target structure before initiating an attack.

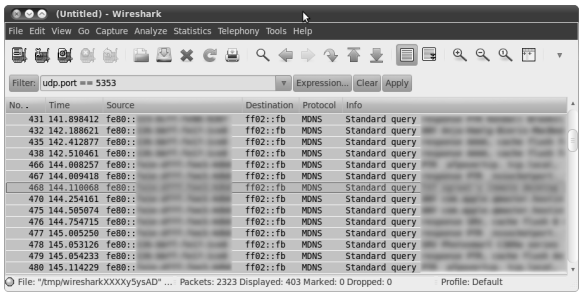
(b)

Advertisers could monitor where potential customers shop to develop targeted marketing.

A cyberstalker can track a target's daily activities and predict a movement pattern.

(c)

A forensics team could use a packet capture to identify all the users that accessed a crime scene subnet.



A foreign agent could use a packet capture to identify and target all nodes of a certain OUI.

Figure 3.2: Sample scenarios where IPv6 IID tracking can apply. Captions on the left describe positive applications of each depicted scenario, while captions on the right describe negative applications.

statistical chance of purchasing products offers a greater potential return on investment for a business. To generate the data needed for targeted marketing, information about a customer is often gathered through the use of "free" services. These services simultaneously offer the unsuspecting customer some benefit while helping build a more complete customer

profile tied to an identity. The profiles developed provide the business more intelligence on how to target specific customers. The more wary customers shy away from these types of "free" services. The tracking capabilities inherent to IPv6 SLAAC provide businesses with another vehicle to build customer profiles and learn general customer habits. As marketers are able to develop new and different methods to target users with applicable advertisements, businesses will increase revenue and lower overhead.

Deploying free Wi-Fi hotspots in businesses with Acceptable Use Policies (AUPs) that allow for traffic monitoring will help businesses to build detailed user profiles through traffic analysis. Any traffic that does not contain PII could be collected and analyzed to create customer profiles, showing an advertiser what types of customers are shopping in their stores. This traffic analysis model is currently primarily applicable in coffee shops and other businesses where laptops are used. Soon, mobile users may look for other free ways to transmit web traffic as cellular providers change their billing structure. Since phone data plans are moving towards megabyte restrictions instead of unlimited usage, the majority of wireless data transmission may begin to transition from cellular networks to Wi-Fi networks, when available, to save cellular data use [16]. Offering free Wi-Fi hotspots may allow businesses to glean legal, detailed customer information at a minimal cost. Businesses may even purchase Wi-Fi logs from other businesses to determine more complete profiles of potential customers' interests as illustrated in Figure 3.2(b).

Even without collecting potentially sensitive browsing data, valuable customer data can be gleaned from tracking wireless broadcast information. For customers not purchasing items, stores often rely on door counters and others crude methods to track store access. While these methods can track access, no connections can be made to an identity. Repeat customers cannot be identified or distinguished. Tracking IID broadcast information changes what before was the anonymous tick of a door counter into an identity. With an identity, businesses can record customer access and other statistics. Since the IID data does not contain any PII, companies can collect and store the data without the security or privacy concerns that come with PII data. For businesses wanting to avoid collecting sensitive information about their customers, using IID location analysis creates non-sensitive customer profiles.

Targeted marketing through IID analysis benefits both the vendor and consumer by offering a better user experience and improving marketing effectiveness. The current Internet model serves many irrelevant advertisements to users as they browse free websites. Advertisements are needed to pay for the cost of the webpage, but the organization cannot

effectively differentiate between different users to serve relevant ads. IID analysis is a simple and effective way for individual websites and marketing companies to build user profiles based off of IIDs. These profiles help companies better serve their customers and consumers. Specific advertising data can easily be associated with specific IIDs. Additionally, the OUI that is part of the MAC can help build profiles based off of the brand of computer being used to access the website. For example, Apple computer accessory advertisements should only be served to those with Apple OUIs in their IIDs. Either building profiles off of specific IIDs or using the OUI portion of the IID allows for direct and effective marketing.

Forensics While some disciplines are faced with excessive amounts of information, forensic investigations often suffer from a lack of data when trying to solve a crime or identify a criminal. Using IID analysis to identify or exonerate possible suspects provides investigators with another information source. Gleaning publicly available information, such as geotemporal locations from IID analysis, may help investigators determine a suspect's locations at the time of a crime. Also, geotemporal IID tracking could allow for the tracking and the recovery of stolen networked assets, such as cars or laptops. Finally, when trying to reconstruct the scene of a crime, IID analysis could tie a suspect to the crime scene.

Using IID analysis to determine a person of interest's location at the time of a crime could provide law enforcement with additional evidence to convict a suspect. Currently, many criminal cases rely on witness testimony to place an individual at a crime scene. Witness testimony is often inaccurate and unreliable and, therefore, discredited. Yet, scientifically supported evidence, such as fingerprints or DNA, often convinces juries and proves innocence or guilt. By logging wireless network access information, a wireless device could be used to implicate its user. Supplementing testimony with an IID gleaned from saved data, such as that in Figure 3.2(c), provides scientifically supportable data for evidence.

To track and recover stolen property, including cars and laptops, IID analysis could be used to geographically track networked assets. While asset tracking systems are currently deployed, the proprietary systems require specialized hardware and are dependent on network implementation. For example, the current LoJack® system used to track stolen vehicles requires a specialized transmitter and uses a proprietary radio system. Additionally, it could be subject to limited range in certain environments. As cars become networked, tracking systems integrated into the car's design could be embedded in such a way as to increase reliability and make removal difficult. This would prevent attackers from removing the transmitter and expand the coverage and reliability of the network.

Forensic examination of access logs and network traffic could also be supplemented by IID analysis. When logs collect information about an address that accessed the system, static IIDs provide information about the connecting system. Failed login attempts could be correlated with specific users, discouraging unauthorized access attempts. Monitoring of network traffic by the government may also be expanded in the near future. In 2003, a petition for expedited rulemaking was made to the Federal Communication Commission (FCC) on behalf of the Department of Justice (DoJ) and its investigatory agencies to allow all wireless header information to be sniffed without warrants [27]. Assuming the use of IPv6 SLAAC, legally sniffing all header information would allow for the DoJ to track any user's location and traffic. This information would facilitate the cost-effective and manpower-efficient tracking of people of interest.

The Bad

It is not difficult to imagine that the capability to track users can be exploited by malicious nodes for nefarious means. One of the more obvious malicious applications is cyberstalking. On a larger scale, terrorists can take advantage of the ability to track people and assets to further their goals without alerting authorities to their intentions. Foreign agents intent on espionage can use static IIDs to track and/or monitor targets of interest.

Cyberstalking There are many different definitions for cyberstalking. Most definitions differ by the means used to stalk a victim. Some stalkers use email while others eavesdrop on their victim's communications. Regardless of how the stalking is done, cyberstalking refers to the repeated unwanted attention of a stalker using some sort of electronic means [2].

Cyberstalking executed through common methods involves an active effort on the part of the stalker. Additionally, there is a level of expertise required. The average cyberstalker is not able to hack into a victim's machine or even place and run a packet sniffer. Assuming that the cyberstalker possesses the skills to accomplish these tasks, there is the risk of the intrusion being traced back to the cyberstalker. There is also the possibility that the victim has a secured machine, making penetration difficult. A mobile victim makes the cyberstalker's task more difficult. As a victim moves between subnets, the DHCP address provided is not logically connected to the victim. If the cyberstalker does not have a physical presence on the victim's machine, tracking the victim becomes much more difficult.

SLAAC in IPv6 alleviates many of the challenges and risks for a cyberstalker. Since the IID is static, the cyberstalker always knows half of a victim's IP address. The other

half, the subnet portion, is easily discovered by conducting a thorough reconnaissance of the geographic areas frequented by the victim. Armed with the IID and subnets, the cyberstalker can continually ping the likely subnets for the victim's IID. A successful ping reply indicates that the victim is at that specific location. This form of attack will not alert the victim, yet will provide the cyberstalker with the victim's daily movements. The attacker may even be able to establish a movement pattern for the victim as illustrated in Figure 3.2(b). This pattern could be used to plan a physical assault, burglary, or other crime. The proliferation of handheld network-capable devices aids a cyberstalker's ability to keep constant tabs on victims.

Monitoring a victim's network traffic is also made easier by IPv6 SLAAC. A cyberstalker running a packet sniffer on a congested network does not need to perform tedious traffic analysis to filter out the victim's traffic. Since the IPv6 IID is static, the cyberstalker can easily set up a filter rule to exclude all traffic not matching the victim's IID. This provides the cyberstalker with only traffic belonging to the victim. DHCP in IPv4, on the other hand, issues IP addresses to victims that change based on lease duration. These DHCP addresses are unassociated with the hosts' identities. Traffic monitoring in IPv4 is not nearly as straightforward as it is when using IPv6 autoconfigured addresses.

Tracking victims or monitoring their network traffic may not even be the goal of the cyberstalker. It may be that the cyberstalker wants to cause pain and humiliation to the victim. SLAAC in IPv6 provides the cyberstalker with this capability. Since the IID is static and linked to a specific device, a cyberstalker can pretend to be the victim. This form of attack works when the victim is being monitored by a third-party, such as an employer. By spoofing the victim's IID, a cyberstalker can visit locations of ill-repute or conduct illegal Internet activities. Although the victim can likely prove his/her innocence through computer logs, the initial accusation followed by the subsequent discomfort felt by the victim accomplishes the cyberstalker's goals.

Terrorism A potentially disastrous application of IPv6 SLAAC is toward forwarding the goals of terrorists. The Code of Federal Regulations defines terrorism as "the unlawful use of force and violence against persons or property to intimidate or coerce a government, the civilian population, or any segment thereof, in furtherance of political or social objectives" [59]. The best way to cause fear is through surprise and shock value. Terrorists can take advantage of the tracking capabilities provided by static IIDs to plan and organize attacks without attracting attention.

Since users can be easily identified through their IIDs, terrorists can use the IIDs to target individuals. A terrorist can use traffic analysis as described previously to pair a target with his/her IPv6 IID. The terrorist can then use cyberstalking techniques to develop a movement pattern for the target. Once a pattern is discerned, the terrorist can choose the most suitable location to trap the target and wait. If the terrorist's intention is assassination, the chosen location can be set up to remotely detonate an explosive device when the victim's IID is detected. If the goal is fear, a similar trap can be sprung remotely. In either case, the terrorist can be miles away and evade capture. This type of attack would be difficult to detect and has the potential to cause widespread panic.

A terrorist can also use IID tracking in an attempt to undermine an organization. By monitoring the locations and network traffic of individuals within an organization, a terrorist can apply social network analysis principles to determine the key people within an organization. There are certain people, not necessarily the leadership, within an organization that are critical to the functionality of the organization [49]. Once these people are identified, a terrorist can target each person individually. If these people are removed, the organization will falter and possibly fail.

Another way terrorists can take advantage of IID tracking is in the planning of attacks against soft targets. Soft targets are typically non-military targets that are unarmored and/or undefended. Terrorist attacks are usually planned against soft targets since terrorists are often ill-equipped to confront military targets. Additionally, explosives targeted against soft targets generate higher casualties and, therefore, more media attention [78]. Attacks against soft targets usually require a great deal of planning. During the planning phase, terrorists can use packet sniffers to collect all the distinct IIDs that access the network of the target. If the number of victims is the terrorists' goal, then who owns the IIDs is unimportant. Armed with a list of IIDs, the terrorists can run a script to ping each IID and maintain a counter. Once the counter reaches a predetermined threshold (see Figure 3.2(a)), the attack can be launched. The terrorists need not be anywhere within the vicinity of the target. This type of tracking is not possible in IPv4 due primarily to the use of network address translation (NAT), which hides the number of nodes on a subnet. DHCP in IPv4 also makes this type of attack difficult. Since IPv4 addresses are issued nondeterministically, a terrorist would have to physically scan the entire subnet to determine the number of active hosts. Depending on the size of the network, this could take quite a long time.

Terrorists can also track locations of law enforcement to plan the initiation of an attack or hostage situation. Law enforcement vehicles will each have unique identifiable IPv6 IIDs.

Terrorists could take advantage of this by collecting the IIDs tied to all law enforcement vehicles operating within a particular region. Since law enforcement vehicles communicate wirelessly, collecting these IIDs is trivial if a thorough reconnaissance is conducted. Armed with the IIDs of area law enforcement, terrorists can initiate an attack when there are no law enforcement in the proximity of the attack site. Additionally, terrorists can anticipate the response time required for law enforcement to arrive.

Terrorists can also take advantage of IPv6 IIDs to track military troop movements and size. Since many military units are becoming digitized, units and possibly even individual soldiers may communicate over IPv6 wireless networks. Terrorists could use the IIDs to identify specific units. Through traffic analysis they could identify which units are communicating with one another. They could possibly even predict future operations. If individual soldiers are communicating over an IPv6 network, terrorists can pinpoint and track key leaders as well as unit strength.

Using a device's MAC address to compose the IID is another vulnerability that terrorists can exploit. The MAC address is composed of two parts. The first half is called the OUI and identifies the vendor that issues the NIC. The second half is assigned by the vendor and is designed to act like a serial number to make the MAC unique [13]. Usually, agencies will contract a vendor to produce all of a specific product. This means that, with high probability, an entire product line shares the same OUI. Terrorists could use this to locate all of a specific type of asset. Minimally, a terrorist could learn how many assets there are and possibly their locations. More importantly, a terrorist could identify a vulnerability specific to a particular brand of asset. The terrorist could then use the OUI to limit the search for those assets within an organization and launch an exploit against the vulnerability. For example, the Smart Grid is used to deliver electricity to consumers [81]. If Smart Grid devices were vulnerable to a particular exploit, terrorists could target the exploit against all systems sharing the same OUI. Terrorists in control of the Smart Grid could cause a host of problems, such as shutting down electrical power to a geographic area.

Espionage The monitoring and tracking capabilities resulting from SLAAC in IPv6 can be used for espionage. Espionage can take on several different meanings. In terms of National Security, espionage refers to the exposure of sensitive data to a foreign intelligence or security service [15]. Industrial espionage refers to the stealing of business secrets from competitors [25]. Regardless of the definition, an agent intent on espionage can use IPv6 IIDs to gather intelligence on people, track targets of interest, or gather information through

traffic monitoring.

A foreign agent could use static IIDs to determine the number and identities of personnel that work within a classified facility. The agent gains nothing from the facility itself since the network is shielded. The parking lot, however, is fair game. The majority of employees own network-capable handheld devices. The addresses of these devices could be collected as employees arrive for work or depart at the end of the day. An agent with a directional antenna and the appropriate collection equipment could potentially sniff handheld device traffic from miles away [31] as shown in Figure 3.2(c). Collection of this sort would likely escape attention since the antenna could be pointed at the parking lot and be positioned well outside of a suspicious distance. Traffic analysis could then be conducted to determine the identity of each collected IID as described in Section 3.1.4.

In addition to determining employee identities, targets of interest could be easily spied on using IID tracking. Knowing the IID of a target employee, an agent could use the cyber-stalking techniques described previously to track the target's daily routine. This information could be used to blackmail the target if the target was tracked to incriminating locations. It could also be used to plan the best time or place to attempt to recruit the target. A foreign agent may also track the target back to his/her residence with the intent of collecting the IIDs of family members. The foreign agent could then establish daily routines for family members and use the information gathered to threaten the target. What makes these types of attacks so serious are that they can be executed from anywhere in the world using tools like ping.

Once the foreign agent knows the locations that a target likes to frequent, he/she can set up collection stations at those locations. Using the known IID, the foreign agent can easily collect all of a target's traffic. Although most employees are situationally aware enough not to communicate sensitive information in public locations, there is a chance that with enough monitoring someone will.

3.2 Stateful Address Tracking in IPv6

The inherent privacy and security flaws in SLAAC will lead network administrators to consider DHCPv6 as a stateful addressing alternative. DHCPv6 allows administrators more control over address distribution than SLAAC. It provides many of the same features as the DHCP implemented in IPv4, including logging and auditing capabilities. To provide more precise auditing and configuration, DHCPv6 deploys a DUID. Similar to the way the MAC

address is used to identify hosts at the data link layer, the DUID is used to identify unique hosts to the DHCPv6 server. While DHCPv6 is a stateful protocol which can be configured to change nodes' addresses frequently, the static DUID allows nodes to be tracked through a static value in many common DHCPv6 messages.

The DUID allows for correlation of users' addresses over multiple sessions, creating similar privacy and monitoring concerns as SLAAC. While RFC 3315 states that the DUID is optionally static [17], current OS client implementations use a permanent DUID. Since the DUID for a client persists between sessions and networks, users can be geotemporally tracked and have their traffic correlated. Typically, adversaries must be on the link-local network to receive the SOLICIT and ADVERTISE messages containing DUIDs. They can also remotely monitor and track clients by planting modified relays at targeted sites to forward multicast DHCPv6 messages for analysis. Once the DUID contained in the DHCPv6 message is obtained, adversaries can sniff the network for the DHCPv6 response or query the DHCPv6 server for the hosts' addresses. Since stateful addressing has similar privacy concerns as SLAAC, IPv6 address assignment must be improved before large scale deployment weakens privacy for unsuspecting users.

3.2.1 Stateful Address Configuration Vulnerability

While the insider threats of a DoS or MITM attacks have already been recognized in DHCPv6 [17], the privacy implications of using a static DUID have not been evaluated. The static identifier, created to be publicly broadcast and globally unique, provides a simple vector for an adversary to identify a unique node. As mentioned in Chapter 2, Section 2.3, the SOLICIT and ADVERTISE messages contain users' DUIDs. This inclusion allows adversaries to identify nodes through their static DUIDs and monitor traffic for the session through the leased address. If adversaries miss a targeted node's ADVERTISE messages, but already know the node's DUID, they can send unauthenticated client-initiated INFORMATION-REQUEST messages to the DHCPv6 server requesting information on a leased address associated with a particular DUID.

3.2.2 Vulnerability Exploitation

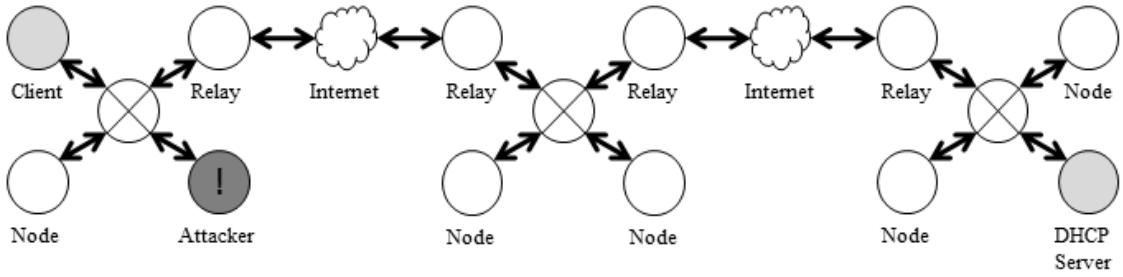
The vulnerability associated with static DUIDs in DHCPv6 was exploited on the live Virginia Tech IPv6 network using a Dibbler [52] DHCPv6 server and client running Ubuntu 10.04. Scenarios were set up with a mock adversary positioned both on and off the local area net-

work (LAN) running DHCPv6. In each scenario, the mock adversary attempted to capture DHCPv6 messages sent between a target host and a DHCPv6 server. Once collected, the DUID in each message was analyzed. DUIDs and the associated DHCPv6 leased addresses, available either from packet captures of network traffic or by querying the DHCPv6 server, were correlated. To correlate DHCPv6 addresses on the LAN, messages were sniffed directly, between the DHCPv6 server and client, and indirectly, through DHCPv6 relays programmed to pass the DHCPv6 message between different segments of the LAN. Remote address correlation was accomplished through compromising DHCPv6 relays, set to forward DHCPv6 messages to a remote third party.

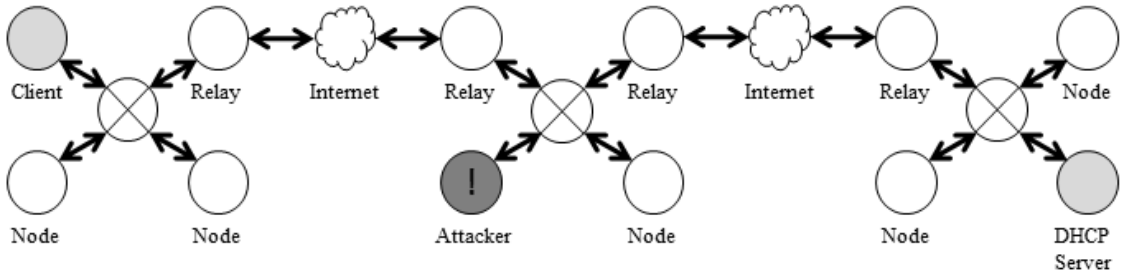
Local Monitoring

DHCPv6 addresses were correlated locally by sniffing and spoofing DHCPv6 messages inside the LAN. Since the LAN allows for link-local and multicast messages to pass freely, the mock adversary was able to read the messages sent to and from a DHCPv6 relay. The adversary was also able to spoof the identity of a DHCPv6 client to query the DHCPv6 server for more information, including the addresses leased to a specific DUID. In large DHCPv6 addressed networks, addresses can be correlated at three locations. The first location is on the same router or switch as the client. An adversary can easily sniff the network traffic and correlate DUID and client address information without querying the DHCPv6 server. The second location is between relays. The adversary is on neither the same switch as the DHCPv6 server nor the client. The adversary relies on sniffing messages passed between the relays to sniff DUIDs and then querying the server to gain client address information. Finally, when on the same switch as the DHCPv6 server, an adversary can sniff traffic for DUID and address information. He or she can also masquerade as the client. Being on the same subnet as the server, it is trivial for the adversary to intercept address leases and traffic and send false responses to the server before the actual client is able to. By doing so, the adversary is capable of denying the user network access.

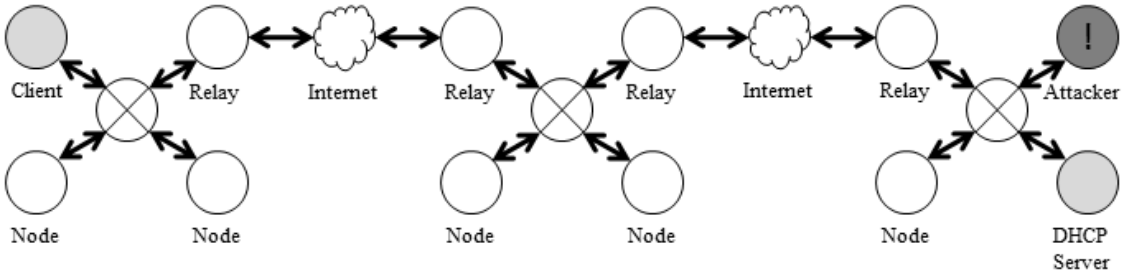
In Figure 3.3(a), the adversary is on the same router as the targeted client. When the client initially connects to the network, a SOLICIT message is sent to the multicast address of the DHCPv6 server. This message contains the DUID and link-local address of the client. By sniffing this message, the adversary can capture the client's identity. The server responds to the client's link-local address with the ADVERTISE message, containing the leased address and any other configuration parameters. The adversary captures this response sent to the link-local address and matches it with the link-local address in the



(a) Attacker on subnet of client



(b) Attacker on subnet of DHCPv6 relay



(c) Attacker on subnet of DHCPv6 server

Figure 3.3: Three different scenarios of DHCPv6 message sniffing inside a LAN

SOLICIT message containing the client's DUID. With this information, the adversary is able to compromise the client's address and identity for the session.

The adversary can also move to a subnet that contains neither the DHCPv6 server nor the client, but does contain a DHCPv6 relay. This scenario is shown in Figure 3.3(b). On a relay subnet, a client's address can still be compromised using an approach similar to the approach in Figure 3.3(a). The SOLICIT multicast message sent to the DHCPv6 server from the client is forwarded by the relay. Since the message is multicast, the adversary can register itself as a receiver of the DHCPv6 multicast addresses with the router through NDP. This NDP exploit allows the adversary to sniff the server's traffic. When the server sends the response using the link-local address, the adversary can either register itself as

a relay, receiving the message, or query the DHCPv6 server through an INFORMATION-REQUEST for the address leased to the sniffed DUID. Since the server does not perform address validation on the source of the message, the message will be returned to the adversary with the client's address, again compromising the session.

Figure 3.3(c) shows an adversary sniffing DHCPv6 messages on the same router or switch as the DHCPv6 server. This configuration gives the adversary direct access, since the router can be configured through NDP to send the traffic of the client being attacked directly to the adversary. The adversary can again use the same methods of attack as for the scenario in Figure 3.3(a). When the client initially sends its SOLICIT message to the server, the adversary can sniff the message and analyze the DUID and link-local address. When the server responds with the DHCPv6 configured address to the client, the adversary can signal the router to register the leased addresses to itself. The adversary then acts as a man-in-the-middle, passing packets to the client only after he/she has intercepted and forwarded them.

Each of the three vulnerable scenarios described in this section were successfully executed on the test Dibbler DHCPv6 network. Nodes deploying the Dibbler DHCPv6 client received stateful configured addresses from a Dibbler DHCPv6 server. Different LANs were provided with DHCPv6 access through the use of Dibbler DHCPv6 relays. The DUIDs and link-local addresses of these nodes were recorded and traffic was sniffed at the prescribed locations. All SOLICIT and ADVERTISE messages were successfully captured in the trials.

Remote Monitoring through DHCPv6 Relays

To monitor the addresses of DHCPv6 nodes remotely, a compromised DHCPv6 relay can sniff DHCPv6 SOLICIT messages and send them to a remote host for analysis as illustrated in Figure 3.4. Since there is no mandatory authentication or authorization of DHCPv6 relays running on a network, a modified DHCPv6 relay could be enabled on a network. Receiving all of the same messages as a legitimate DHCPv6 relay, the modified relay can forward the messages to a remote adversary and allow for remote correlation of leased addresses and any sniffed traffic. In the trials conducted using the live DHCPv6 Dibbler configuration, INFORMATION-REQUEST messages were successfully sent and received to learn a node's leased address.

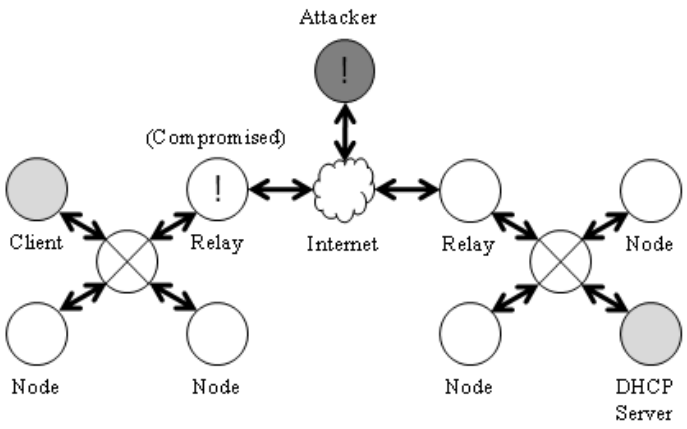


Figure 3.4: A compromised DHCPv6 relay passing DHCPv6 messages from a LAN to an adversary

3.2.3 Dynamic DUID

Dynamic DUIDs are one method to try to mitigate the vulnerability associated with static DUIDs. Tront et al. [80] discuss two primary means of implementing dynamic DUIDs. The first method is based off an IPv6 IID obscuration technique called Cryptographically Generated Addresses (CGAs). CGAs (discussed in more detail in Chapter 4) require the use of public keys and a series of hash calculations to generate addresses. The second technique, which was developed and tested by Tront et al., was developed from another IPv6 IID obscuration technique called privacy extensions; also discussed in Chapter 4. This second technique was implemented due to its reduced computational expense.

The privacy extensions-based dynamic DUID technique uses the MD5 hash of a pseudo random number (PRN) to generate the dynamic DUID. New dynamic DUIDs are generated based off three possible triggers. The first trigger is time. A new dynamic DUID is generated after a set amount of time passes. The second trigger is based on the subnet a host is connected to. Each time a host connects to a different subnet, a new DUID is generated. This prevents hosts from having persistent DUIDs that follow them from subnet to subnet. The third trigger is based on system state. A new DUID is generated when host machine is rebooted, put into standby mode, or similar. As noted by Tront et al., dynamic DUIDs have no effect on other DHCPv6 extensions.

While dynamic DUIDs seem like an acceptable solution to static DUIDs, there are performance-related disadvantages that prevent deployment. The first disadvantage is the

time it takes a client to lease a DHCPv6 address from the server. For each new DUID, six DHCPv6 messages are exchanged between the client and the server. Tront et al. demonstrated this process to take an average of 1.95 seconds. While this latency may be acceptable when connecting to a network, it may not be acceptable to incur this added latency after every set time interval. Another issue related to dynamic DUIDs is server performance. DHCPv6 servers maintain a cache of DUIDs until a timeout period is reached. Tront et al. demonstrated that as the number of cached DUIDs increases, the time it takes the DHCPv6 server to lease an address also increases. Before dynamic DUIDs become a viable solution to the IPv6 stateful addressing flaw, the performance issues will need to be addressed.

3.3 Summary

The ability to track users' stateless addresses with static IIDs combined with the ability to track DHCPv6 issued addresses compromises privacy and security in all known addressing methods in IPv6. The potential for IPv6 addressing schemes to be used for cyberstalking or even terrorism is proof enough that there are issues that need to be addressed. Before IPv6 is extensively deployed, research must be conducted to ensure privacy and security concerns are protected.

Stateless addressing in IPv6 uses an IID that is globally static in most implementations. Though simple methods of obscuring IIDs have been identified, no current method obscures addresses often enough to prevent network attacks or privacy violations. Of the current deployment-ready solutions, the privacy extensions described by RFC 4941 [56] provide the best overall solution. Privacy extensions, discussed in detail in Chapter 4, obscure the IID with minimal computational overhead.

Although offered as a potential alternative, stateful addressing is not a viable solution for the stateless addressing vulnerabilities. The static DUID used by DHCPv6 allows a host to be targeted in much the same way as the static IID used in SLAAC. While DUID correlation and analysis is limited in scope, the possibility of relay networks or compromised relays extends an adversary's reach beyond the local networks. If stateful addressing becomes an accepted solution to IPv6 addressing issues, it will require significant redesign.

Chapter 4

Overview of Address Protection Techniques

There are three primary classifications for protecting a user's IID from tracking and monitoring. The most straightforward classification is static obscuration of the IID. The second classification, which (before MT6D) exists only in theory, is dynamic obscuration. The final classification is third party address assignment and tunneling.

4.1 Static Address Obscuration Techniques

There are two methods proposed for obscuring the IID when using stateless generation of IPv6 addresses. The first method uses Cryptographically Generated Addresses (CGAs) [3]. The second method uses privacy extensions for static IIDs [56].

4.1.1 Cryptographically Generated Addresses

CGAs are IPv6 IIDs that are generated by hashing a sender's public key and other parameters [3]. Although CGAs can be used with multiple applications, they were originally designed to work with the SEcure Neighbor Discovery (SEND) protocol [1] to prevent denial of service attacks. Since CGAs obscure a sender's IID, they could prevent tracking a user through their autoconfigured IPv6 address. There are, however, a number of disadvantages to using CGAs for dynamically obscuring an IPv6 IID.

The main disadvantage to using CGAs is the cryptographic cost of CGA generation. In order to generate a CGA, the sender must apply the SHA-1 hash algorithm to the CGA

parameters a minimum of two times. The CGA parameters consist of the concatenation of a pseudo-random number, the sender's public key, nine zero octets, and optional fields. The first hash calculation generates Hash2. The ease of achieving an acceptable Hash2 value depends on the strength of the security parameter (Sec). The Sec is a three-bit value used to determine how many leading zeros Hash2 must contain. Hash2 must have $16 \cdot Sec$ leading zeros. If it does not, a pseudo-random number is incremented and Hash2 is recalculated. Hash2 continues to be recalculated until this condition is met. This can result in a large number of hash calculations. In fact, RFC 3972 states that on average it takes $O(2^{16 \cdot Sec})$ iterations to generate a CGA [3].

After Hash2 is successfully calculated, Hash1 is calculated using the CGA parameters with the final pseudo-random number used to generate Hash2. The left-most 64 bits of the SHA-1 output are used as the IID with the exception of bits 0-2, 6 and 7 used for other purposes. Duplicate address detection is conducted as required [1]. If a collision occurs, a collision count field is incremented and Hash1 is recalculated. This is repeated at most three times after which, CGA calculation stops [3]. It is possible that after all of these hash calculations no CGA will result and the process will need to start anew.

These repeated hash calculations require minimal overhead for the average personal computer, but are likely infeasible for most handheld devices due to limited power and computational capability. To help mitigate this cost, RFC 3972 discusses that CGAs can be precomputed or offloaded to more powerful computers [3]. This solution, however, does not help a user connecting to a network for the first time, which is not uncommon when roaming with a handheld device. Additionally, the CGA generation cost makes it prohibitive for users to generate new CGAs. Thus, CGAs for subnets will be deterministic.

There are two possible implementation alternatives to reduce the cost of CGA generation. The first involves setting $Sec = 0$. By setting $Sec = 0$, Hash2 is not required to have any leading zeros. This means that the calculation of Hash2 is not required [3]. The problem with this alternative is that the CGA becomes susceptible to a brute-force attack. If the CGA is used only to prevent IPv6 address tracking, this may not be a concern. However, if the intent is to also protect the IID from denial of service or address spoofing, this alternative is not acceptable.

The second implementation alternative is to omit the subnet prefix from CGA generation [3]. By omitting the subnet prefix, senders need only calculate the CGA once. The same CGA can be used for multiple subnets without having to go through the expensive CGA generation for every subnet. The downside is that the IID is again deterministic across

multiple subnets. This alternative maintains the benefit of protection against brute-force attack through the security parameter, but leaves senders vulnerable to address tracking.

Even if users do change their CGA every time they connect to a subnet, they can still be tracked. Senders using CGAs send their CGA parameters along with their IPv6 address for verification purposes. The CGA parameters contain the users' public key. Unless senders also generate new public keys for every connection, they can be tracked through their CGA parameters.

An additional disadvantage of using CGAs is that an attacker can easily impersonate or slander a user by forming a CGA claiming to be the target. Asymmetric key pairs are self-generated to eliminate the key management infrastructure. As a result, anyone can generate a key pair and claim to be someone else. RFC 3972 claims that CGAs protect against address spoofing [3]. This is only true in the case where an attacker attempts to hijack an existing session between two nodes. An attacker is free to initiate new sessions claiming to be someone else. By doing this, an attacker can make it appear as if a target is behaving in a certain way to others who are monitoring the target on the network. Spoofing is common in IPv4, but CGAs in IPv6 make a spoofed address appear more authentic.

Requiring the use of public keys to form CGAs is another drawback. One of the strengths of stateless autoconfiguration of IPv6 addresses is that it is transparent to users. Requiring users to generate asynchronous key pairs removes this transparency. Many users do not have asynchronous key pairs or even understand the concept. As a result, there will be users that will not implement CGAs to obscure their addresses.

The remaining vulnerabilities do not affect tracking of users, but how an attacker can take advantage of flaws in CGAs to achieve malicious results. For instance, a man-in-the-middle can cause all data checks to fail by modifying any of the CGA parameters sent with the IPv6 address. This can be accomplished by changing the collision count to a number greater than two, because the collision count is only valid for values of 0-2. An attacker also can hijack a connection by finding a CGA hash collision. Only 59 bits (64 bits minus bits 0-2, 6, and 7) of the SHA-1 hash are used for the CGA. Using the birthday attack [77], a collision can be found in $2^{59/2}$ guesses on average.

Despite the numerous disadvantages of CGAs, there are several advantages. The primary advantage is, of course, that CGAs obscure a sender's IID. Even if senders do not go through the costly procedure of computing a new CGA every time they connect to a network, they will at least have a different deterministic CGA for every subnet (assuming they don't exclude the subnet prefix from the CGA computation). This will make an attacker's attempt at

tracking a target much more difficult. In order for an attacker to track a target, the attacker will need to know the sender's CGA on every subnet. Another benefit of using CGAs is that they require no key management infrastructure [3]. Since CGAs can use user-generated asymmetric key pairs, there is no need for a certification authority, making CGAs scalable.

4.1.2 Privacy Extensions

The privacy extensions proposed in RFC 4941 provide another way to obscure the IPv6 IID. Where the goal of CGAs is primarily to provide security of IPv6 addresses for the SEND protocol as discussed in Section 4.1.1, the goal of implementing privacy extensions is to prevent an attacker from correlating network traffic with a particular user. Privacy extensions produce a random IID by hashing the concatenation of a user's IID and a 64-bit "history value." This "history value" is initially generated from the rightmost 64 bits of a hashed random number. Subsequently, the "history value" becomes the previously calculated 64-bit hash result. The random IID is formed using the leftmost 64 bits of the resultant hash calculation with the "u" bit (bit 6) set to zero for local scope. Duplicate address detection is then performed on the random IID to detect duplicate IIDs on the local network. If a duplicate IID is detected, a new "history value" is formed and the process is repeated [56].

Using privacy extensions to obscure a user's IID is much more appealing than using CGAs for several reasons. First, the cryptographic cost is much lower. Assuming no duplicate address is detected, privacy extensions require only one hash calculation by the sender at IID generation and none by the receiver. With CGAs, the sender requires on average $O(2^{16 \cdot Sec})$ hash calculations to generate a CGA, and the receiver is required to complete two hashes to verify the CGA. The cryptographic cost of using privacy extensions is more feasible, allowing a node to change their IID often.

The use of privacy extensions does not require the use of a public key [56]. As a result, implementation of privacy extensions are more transparent to the user. Additionally, there are no required accompanying parameters that can be used to link a sender to the random IID.

Privacy extensions are more effective at obscuring IIDs than CGAs because of the lifetime of a generated IID. Having users generate new IIDs when they connect to new subnets effectively masks users' activities on the Internet, but it does nothing for users that never migrate from a particular subnet. A CGA user that never moves between subnets (e.g., a desktop computer) would not necessarily ever generate a new random IID. The privacy

extensions specification uses a TEMP_VALID_LIFETIME parameter to set the maximum amount of time a random IID can be used before needing to be regenerated [56]. This feature improves on DHCP addresses that may last months or longer.

Implementing random IIDs through privacy extensions does have disadvantages. Just as with CGAs, IID collisions are possible [56]. Since only 64 bits of the hash calculation are used, the chance of hash collisions increases. Similar to CGAs, the generation process terminates after a set number of IID collisions.

As mentioned previously, privacy extensions specify several different parameters to limit the time a random IID is valid. The default values of these parameters are set too long. The TEMP_PREFERRED_LIFETIME parameter is set to one day and specifies the preferred life of a random IID. The default value of TEMP_VALID_LIFETIME is one week. An application can force an address to use TEMP_VALID_LIFETIME rather than TEMP_PREFERRED_LIFETIME. Additionally, RFC 4941 states that a new random IID *should* be computed when connecting to a new subnet [56]. It does not state that this *must* be done. Users choosing not to change IIDs and accepting the default expiration times could be monitored for up to a week at a time before an attacker needs to reestablish an identity pairing. Fortunately, RFC 4941 allows users to modify these defaults.

An unfortunate side-effect of privacy extensions is that IIDs are obscured from everyone, including network administrators. Combined with frequently changing and obscured IIDs, privacy extensions make fault isolation and debugging difficult [56]. With CGAs, public keys are tied to addresses. When DHCP is used, system administrators can track changes in addresses, while still protecting a user's identity.

CGAs are more robust than privacy extensions in that they can protect a sender's address from spoofing once a session has been initiated. Since there is no verification process implemented by privacy extensions, an attacker can easily inject new traffic claiming to be the sender. However, preventing this type of malicious activity is not one of the design goals of privacy extensions. This type of attack, however, is mitigated by frequently changing a user's IID.

Despite the fact that CGAs and privacy extensions both help obscure users' IIDs, they still use static addresses. Static addresses leave hosts vulnerable to numerous types of attack. Additionally, they open hosts up to address tracking and traffic correlation. Dynamic address obscuration techniques protect hosts from targeted attacks as well as privacy-related attacks.

4.2 Dynamic Address Obscuration Techniques

MT6D involves dynamic address obscuration. There have been other proposals that also dynamically obscure network addresses. One obscures network addresses for the purpose of privacy and the other obscures addresses to prevent certain classes of network attacks. The most relevant proposals are presented along with a discussion of how they differ from MT6D.

4.2.1 Method and Communications and Communication Network Intrusion Protection Methods and Intrusion Attempt Detection System

A technique by Sheymov [75] was designed with the goal of dynamic obscuration. Instead of focusing strictly on addresses, Sheymov defines cyber coordinates. Cyber coordinates can represent any piece of information in cyber space, to include network addresses. Sheymov's objective behind dynamic obscuration is to provide intrusion protection from certain classes of network attacks. MT6D also prevents these same types of network attacks. The additional benefit MT6D provides is anonymity. Anonymity is not a goal of Sheymov and his design does not provide it. Sheymov uses DNS to assign permanent names to devices. These names are attached to dynamic cyber coordinates, or addresses. An attacker will have little problem correlating traffic from and to hosts using their particular DNS names. Another advantage MT6D has over Sheymov's method is in how dynamic addresses are determined. In Sheymov's method, dynamic addresses are created and distributed from a management unit. The management unit presents an additional attack vector. In MT6D, addresses are calculated on demand by the MT6D hosts. There is no need for a management unit. Any host in a communicating pair, can calculate both the obscured source and destination addresses for every packet sent.

For enhanced security and anonymity, MT6D is able to change addresses regardless of the state of any ongoing connections. Sheymov cannot achieve this without breaking the existing session and having to repeat the connection establishment process. MT6D does not disturb preexisting sessions during address change.

Another weakness of Sheymov's method is that it does not prevent traffic correlation. It concentrates on obscuring fields in the packet header, but not the payload. Packet payloads can be pieced together to discern information about the communicating hosts. In the default embodiment of MT6D, the packet payload is encrypted along with the header information

to prevent both tracking and correlation.

Sheymov's method also has no method of masking the subnet of the communicating hosts. A third party sniffing a sparsely populated network can, with reasonable accuracy, pinpoint the hosts implementing the obscuration technique. MT6D provides a solution for this with Dynamic Subnet Obscuration (DSO).

4.2.2 Dynamic Approaches to Thwart Adversary Intelligence Gathering

Kewley et al. [44] also proposed a technique for dynamically obscuring host addresses called Dynamic Network Address Translation (DYNAT). Like MT6D, DYNAT obscures both network and transport layer addresses. MT6D improves upon DYNAT in a few ways. First, MT6D obscures both the source and destination network and transport layer addresses. DYNAT only obscures the destination address, while the source address remains unobscured. An attacker witnessing DYNAT communications can target the source for attack or tracking. Second, MT6D rotates addresses regardless of the state of ongoing sessions. With DYNAT, address obscuration is only possible between sessions. Another advantage is that MT6D hosts purge stale addresses. DYNAT allows previously used addresses to resolve. Previously used addresses can be used by adversaries to attack target systems.

DYNAT claims to prevent traffic correlation, but attackers can use fields from the Transmission Control Protocol (TCP) header, such as sequence number, for traffic correlation. MT6D can prevent traffic correlation by encrypting the packet payload along with the original header information. DYNAT also has no method of masking the subnet of the communicating hosts. A third party sniffing a sparsely populated network can, with reasonable accuracy, pinpoint the hosts implementing the obscuration technique. As mentioned, MT6D provides a solution for this with DSO.

4.2.3 Method and Apparatus for Providing Adaptive Self-synchronized Dynamic Address Translation

Fink et al. [28] proposed a technique that likely grew out of DYNAT. Their technique, called Adaptive Self-Synchronized Dynamic Address Translation (ASD), rotates addresses using symmetric keys. Symmetric keys are established through a handshake process between a trusted sender and receiver enclave. When a symmetric key expires, the enclaves again

enter into a key exchange handshake operation. Obscured addresses are selected from those available to the ASD enclave. The idea of having an enclave of available addresses was also used with DYNAT.

MT6D improves upon ASD by basing address rotation on a changing nonce (e.g., a timestamp) instead of a rotating session key. This allows MT6D hosts to communicate without the need to authenticate. An issue with hosts having to authenticate is that these communications minimally give away the identity of the trusted enclaves communicating. Additionally, each time hosts authenticate in ASD they have to enter into a handshake process. This adds additional overhead and latency. Since MT6D hosts do not need to authenticate to rotate addresses, this additional overhead and latency is avoided.

Like Sheymov's method, ASD does not obscure the packet payload, thus allowing traffic correlation. As mentioned, packet payloads can be pieced together to discern information about the communicating hosts. MT6D can prevent traffic correlation by encrypting the packet payload along with the original header information. Like DYNAT and Sheymov's technique, ASD also has no method of masking the subnet of the communicating hosts. A third party sniffing a sparsely populated network can, with reasonable accuracy, pinpoint the hosts implementing the obscuration technique. As mentioned, MT6D provides a solution for this with DSO.

4.2.4 Agile Network Protocol for Secure Communication with Assured System Availability

Munger et al. proposed a technique referred to as the Tunneled Agile Routing Protocol (TARP) [54] that obscures addresses by tunneling encrypted packets through a series of TARP routers. Although addresses do not change mid-session, each packet may be tunneled through a different series of TARP routers. Each packet is encrypted beneath multiple encryption layers and undergoes a minimum number of hops to prevent traffic correlation. Any TARP router is capable of determining the final packet destination.

MT6D differs from TARP in a number of ways. First, address obscuration in MT6D can occur mid-session without the use of encryption. This reduces the overhead and latency required to operate MT6D. Even when MT6D uses encryption to protect the packet header and payload, only one layer of encryption is required. Another advantage of MT6D is that there is no requirement for a global trust model. Keys exist only between communicating end hosts. With TARP, all TARP routers must trust each other. A compromised TARP

router is able to track the destination address of all packets passing through it.

TARP routers are able to change their network addresses, but must do so by notifying all other TARP routers. This notification puts the identity of TARP routers at risk, particularly if there are compromised TARP routers in the network. To allow this communication, all TARP routers also maintain an unchangeable address, which further puts the TARP network at risk. MT6D does not require the use of any static addresses. Additionally, hosts are able to predict each other's addresses without the need for any exchange of packets.

MT6D also differs in that it can be implemented on a per host basis. This minimizes the risk of packets being compromised before encapsulation. TARP is installed on routers, which assumes no comprised hosts within the trusted network. A compromised host inside the TARP trusted network would be able to observe communications prior to any encryption being applied. Another difference is that TARP routers can be targeted for attack. By launching a denial of service against a TARP router, an attacker can effectively deny service to all hosts on the subnet. MT6D devices use only the obscured address, which frequently rotates. If an attacker locates and attacks a MT6D address, the attack can last for at most the length of time between address changes.

4.3 Address Assignment and Tunneling Techniques

The two remaining techniques for protecting host addresses were designed to fulfill different functions in IPv6. The third party address assignment technique is designed to provide stateful addressing for IPv6 as described in Chapter 2, Section 2.3. The second technique is designed to provide security and authenticity of network traffic as explained in Chapter 2, Section 2.1.1.

4.3.1 Third Party Address Assignment

DHCPv6 provides third party address assignment for IPv6 [17]. Instead of allowing a client to configure his/her own address, a DHCP addressed network requires a DHCP server in order for a client to get an address on a network. DHCP addresses are leased to clients when they connect to the network. If the network is not overloaded with clients, a client may receive the same DHCP address each time he/she connects. For heavily populated networks, clients may receive different addresses each time they connect.

There are several advantages to using DHCPv6. First, DHCPv6 provides IIDs that are

not necessarily tied to a user. A user may receive a different DHCPv6 address each time he/she connects. Whereas collisions were possible with the address obscuration techniques discussed in Section 4.1, a properly configured DHCPv6 server should not issue duplicate addresses. Although DHCPv6 can operate without any cryptographic cost for address generation or verification, it is worth noting that DHCPv6 can be configured to use optional authentication [17]. DHCPv6 authentication uses shared keys which, although more cryptographically efficient, do not scale well.

Despite these advantages, DHCPv6 does not necessarily protect a user's identity well because the IPv6 address space is sparsely populated. As a result, there will likely be little competition for addresses when connecting to even the most populated subnets. Therefore, users should expect to get the same address each time they connect to the network. This may change over time as more devices connect to the network, and the address space becomes more densely populated. The DHCPv6 specification, however, promotes static addresses. Unless specifically requested by the client using the Identity Association for Temporary Addresses (IA_TA) option, the client will be issued a non-temporary address [17]. A static address, whether generated by the client itself or issued by a DHCP server, exposes the user to monitoring. If IA_TA is implemented, it would be provided through the use of privacy extensions discussed in Section 4.1.2.

Even if temporary addresses are requested, an attacker may still be able to monitor a user. For example, the server controls how often a client's address changes, not the client. The client does have the option to specify a preferred-lifetime and a valid-lifetime, but the timing of an address change is ultimately determined by the server. Additionally, communications between client and server use DUIDs as discussed in Chapter 2, Section 2.3. The DUID is a globally unique value that should never change [17]. The DUID is used in many of the DHCP exchange messages and could be used by an attacker to track a target's presence on a network. Once the attacker locates the user, the attacker can monitor the DHCP exchanges to harvest the IPv6 address. The scope of this attack is limited to the subnet of the user, the DHCP server, or any relays.

A final downside to using DHCPv6 versus stateless address autoconfiguration is that it must be managed. This could add a tremendous burden on network administrators as more and more devices connect to the network. DHCPv6 also increases the chances of incorrect configuration, which may lead to other vulnerabilities.

4.3.2 Address Tunneling

Address tunneling can be achieved through the use of IPsec. IPsec is not a very comprehensive method to use for masking IPv6 addresses, but it can provide excellent obscuration from external attackers. IPsec provides authentication and/or encryption to network layer packets. For the purposes of obscuring IPv6 addresses, only the encryption aspect provided by the ESP illustrated in Figure 4.1 is referred to. Specifically, ESP used in tunnel mode provides address obscuration. When used in tunnel mode, ESP encrypts the entire IPv6 packet and provides a new IPv6 header, complete with new source and destination addresses [42]. The new source and destination addresses are those of the endpoints of the tunnel. It should not be possible for devices external to the tunnel to learn the true identity of the sender or receiver. The tunnel endpoints are typically network gateway devices. Although it is possible for the sender and receiver to act as tunnel endpoints in IPv6, this technique would gain very little privacy. A host acting as its own tunnel endpoint would be easy to link as the actual target host. The ability to link to the target host can be prevented by using one of the obscuration techniques described in Section 4.1 after applying ESP. Using IPsec for this purpose, however, would then be pointless.

There are three main benefits to using IPsec in tunnel mode to obscure IPv6 addresses. As mentioned previously, the sender's address is hidden from those external to the tunnel. Second, the cryptographic burden of encryption and decryption is offloaded to the gateway devices. This makes address obscuration feasible for devices limited by battery or computational power, such as handheld devices and sensors. Also, since the address used is that of the gateway devices, there are no address collisions as there were in the address obscuration techniques discussed in Section 4.1.

Unfortunately, IPsec in tunnel mode does not provide any address protection from an attacker inside the sender's or receiver's subnets. Since address obscuration does not occur until the packet reaches the gateway, an attacker monitoring the subnet will have no trouble monitoring users through their IIDs. This does, however, limit the scope of the attack to the two subnets mentioned.

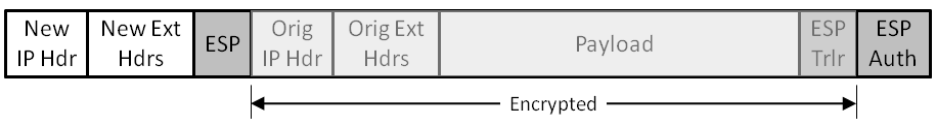


Figure 4.1: IPv6 packet encrypted using IPsec in tunnel mode

Perhaps an even bigger issue is the requirement for a key management infrastructure. Wide-scale deployment of IPsec requires a global trust model in place as well as a management infrastructure [7]. In today's infrastructure, only those networks with security requirements utilize IPsec. It is not reasonable to assume that networks would implement IPsec for address obscuration purposes.

4.4 Summary

A number of different methods can be used to obscure a user's IID from monitoring. Each method comes with its associated benefits and shortcomings. Of the different obscuration techniques, dynamic address obscuration provides the best security for the host. Static address obscuration can hide a host's actual identity from a third party, however, it does not prevent targeted network attacks or monitoring. The benefits of dynamic addressing are discussed in more detail in Chapter 5.

Chapter 5

Benefits of Using Dynamic Addresses

As computers and networks become embedded in critical services throughout society, the privacy and security implications of non-changing network addresses expose users to tracking and attack. At the link layer, MAC addresses associated with a network interface are susceptible to flooding and spoofing attacks. At the network layer, IP addresses are susceptible to spoofing, tracking, and targeting. Both the MAC and IP addresses of servers and other host machines are usually static to allow for clients to successfully communicate. This often leaves servers open to attack because these fixed addresses are easy targets to locate, which if compromised, can create a DoS attack for all attached clients. Another concern is mobile hosts whose non-changing network addresses can be geotemporally tracked. This is a clear violation of a user's privacy.

Research has shown that an adversary spends the majority of his/her time preparing for an attack [45]. Specifically, 95% of an adversaries effort is spent preparing for an attack. Only 5% of an adversary's effort is actually spent on attack execution. This lopsided expenditure of effort may be worthwhile if a target address is unlikely to change. Dynamic addresses, on the other hand, make it feasible for a target address to change before an attacker completes the attack preparation phase.

The variables that impact how effectively dynamic addressing protects a host and the impact these variables have on each other are explored. One variable is the number of dynamic bits in the address, or bits available to change. The fewer dynamic bits, the more likely an adversary can use brute force techniques to correlate addresses. Another variable is the frequency of the address change. An address with fewer dynamic bits needs to change more often to avoid identification. No temporary address can remain static for too long without risking data correlation. A third variable to consider is the population density

of the subnet. A sparsely populated subnet would make address identification easier for the adversary since fewer addresses are in use. Alternatively, a densely populated subnet would make address identification considerably more challenging due to the additional hosts creating traffic on the network. Although it is easy to simply maximize all the variables, computational expense prevents this. Minimizing computational expense is particularly important for power-constrained devices.

To combat the security and privacy concerns of non-changing addressing, analysis was conducted on how dynamic network addressing would increase security, privacy, and reliability. Dynamic addressing refers to addresses in which some or all of the address non-deterministically changes, possibly even mid-session. Dynamic addressing prevents would-be adversaries from tracking users over time and as they move through different networks. This is because the changing addresses cannot be correlated to a single user. Dynamic addressing also protects against traffic correlation by network sniffing attacks due to the difficulty of associating a user with a changing addresses. Dynamic addressing provides additional security by creating a moving target defense that prevents adversaries from targeting specific machines. The increased security offered by dynamic network addressing protects privacy and data for network users.

5.1 Issues with Using Static Addresses

Static addresses are necessary to allow users to repeatedly find resources. Without providing some sort of notification of an address change, users must have a single, static identifier to locate resources. For example, IP addresses, whether static or dynamic, are often connected with DNS names. DNS names are updated with the current IP address to facilitate location of resources on the Internet by remembering a single value. Without a static value connected to networked resources, whether DNS names or IP addresses, users would not be able to find the resources. Even DHCP leased addresses, which are widely assumed to be dynamic, rarely change.

While static addressing is critical to assist users in finding resources, static addresses allow malicious users to easily locate targets for attack. For example, DNS names and IP addresses are publicly available pieces of information. These vectors allow adversaries to easily conduct scans to locate target hosts. Once a target is located, the adversary can focus on the target found and reasonably assume that the target's static identifier will not change. An adversary is able to make this assumption since identifier changes would interrupt service

for valid users. To ensure the reliability and security of service, critical services must deploy some sort of moving target defense that changes static identifiers while allowing continuity of service for trusted users.

5.2 Analysis of Dynamic Address Factors

There are three factors that contribute to an adversary's ability to detect a target host on a subnet. The first factor is the number of dynamic bits in the address, which affects the size of the subnet. In a small address space, it is trivial for an adversary to check each address. The second factor is how often a target host's address changes. If the address remains static, an adversary has as much time as necessary to locate the host. The third factor is the density of the address space, or the number of other hosts on an IP subnet. If an adversary does not know the target host's address on a subnet, multiple other addresses will make identifying the target more difficult.

For purpose of analysis, an adversary actively scanning an IP subnet with unicast addresses to identify a target host is investigated. There are other methods an adversary can use to detect target hosts on a network. One such technique is a broadcast ping, allowed by IPv4. Many gateway devices block broadcast pings. Another method is to passively scan a subnet with a packet sniffer. This method has scope limitations as the adversary must have a presence on the same subnet as the target host. A unicast scan is more likely since there are multiple ways for an adversary to accomplish it that avoid common security measures implemented on networks.

5.2.1 Size of Address

The larger the address space, the more time it takes an adversary, on average, to locate the target address on an IP subnet. Table 5.1 illustrates this by comparing subnets of various sizes. In the table, the three most common IPv4 classful address blocks are used as examples. The typical subnet size used in IPv6 is also compared. Scanning an entire class C address space is trivial and can be accomplished in less than a minute while scanning an entire IPv6 subnet is currently infeasible. In IPv4, an adversary will be able to locate any target host within a reasonable amount of time, unless the host is defending against active scanning.

Table 5.1: Scan time comparison of subnets of various sizes. The scan time is based on a sequential scan with a 150 millisecond average round trip time for a single packet [30]

Address Type	Address Size (bits)	Address Size (hosts)	Scan Time
IPv4 Class C Subnet	8	256	38 sec
IPv4 Class B Subnet	16	65,536	3 hrs
IPv4 Class A Subnet	24	16,777,216	29 days
IPv6 Subnet	64	$1.845 \cdot 10^{19}$	$8.77 \cdot 10^{10}$ yrs

5.2.2 Frequency of Address Change

The more frequently an address changes, the more difficult it is, on average, for an adversary to successfully locate and target a specific address. This is particularly true if the address changes more rapidly than an adversary can scan the subnet. As mentioned in Section 5.2.1, a larger address space takes longer to scan. It follows that addresses on a larger subnet need to change less frequently.

To understand the relationship between changing and non-changing addresses, the number of attempts it takes an adversary to locate a static address on a subnet is analyzed. Since the address is static, the adversary's odds of guessing the address increase with each subsequent guess. This probability follows a hypergeometric distribution. In the case of locating specific hosts on a subnet, the probability can be written as:

$$P_d = 1 - \frac{\binom{N-h}{r}}{\binom{N}{r}} \tag{5.1}$$

where N represents the total possible addresses in the subnet, h represents the number of target hosts, and r represents the number of guesses an adversary takes in an attempt to find the target address(es).

The best case for the target host is if its address changes at the same rate that an adversary scans a single address or faster. To provide the fairest assessment, a scenario where the adversary is aware of the target host changing his/her address is assumed. As a result, the adversary randomizes his/her address guesses, allowing repetition of addresses. This is in contrary to the normal approach where an adversary exhaustively scans a subnet without repetition. The probability of detecting the target host using an exhaustive search is slightly lower due to the possibility of a host address changing to a previously guessed

address. In the adversary-aware scenario, the probability of detecting the target host remains the same with each subsequent guess and follows a cumulative binomial distribution as shown in Equation 5.2.

$$P(R \leq r) = \sum_{x=1}^{r} \frac{1}{N} \left(1 - \frac{1}{N}\right)^{x-1} \tag{5.2}$$

where N again represents the total possible addresses in the subnet and r represents the attempt during which detection occurs. Figure 5.1 depicts the difference between the probabilities of a static address versus a changing address that follows a binomial distribution. A subnet of size 256 hosts is used as an example for this figure.

It is unlikely, however, that a target address will change at the same rate an adversary scans a subnet. A target host can decrease the probability of detection by changing its

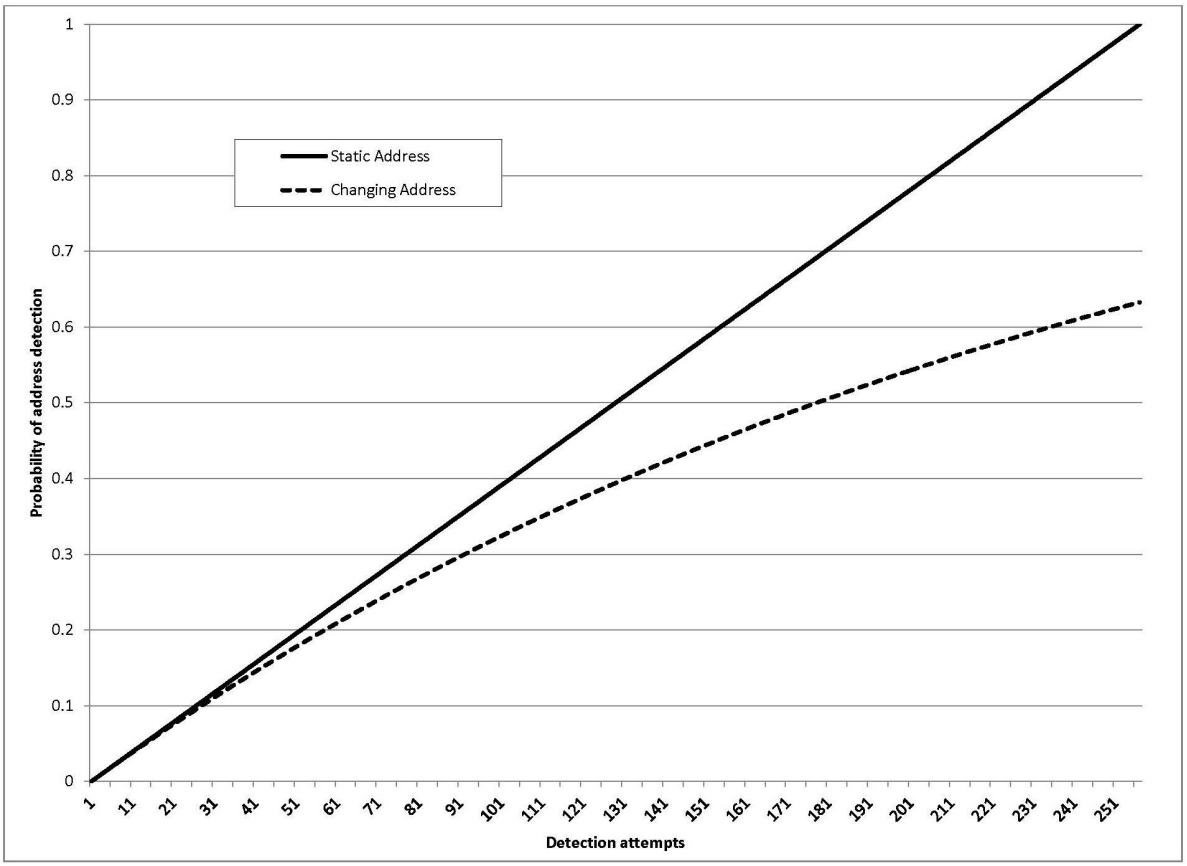


Figure 5.1: Probability of an adversary detecting a target address within r attempts. The solid line represents the probability given a static address while the dotted line represents the probability if the address is changed at the same rate it is scanned.

address more frequently than the time it takes an adversary to scan the entire subnet. In this scenario, it is assumed that the adversary knows the frequency of the address changes. This assumption is made to provide the adversary with the highest probability of target detection, and thus demonstrate the worst-case scenario for the target host. In this scenario, the probability of detecting a target address follows Equation 5.1 until the address changes. After the address changes, Equation 5.1 resets to $r = 1$. If each address change is classified as a round, the probability of detection on round z can be written as:

$$P(Z \leq z) = \sum_{x=1}^{z} P_d(1 - P_d)^{x-1} \tag{5.3}$$

Figure 5.2 illustrates the difference between a static address and addresses that changes after an adversary scans r addresses. The address that changes every round ($r = 1$) follows a binomial distribution. The figure demonstrates that, as the frequency of change approaches the time it takes an adversary to scan a single address, Equation 5.3 converges to a binomial distribution. Alternatively, as the adversary is able to scan more of the address space between address changes, Equation 5.3 converges to Equation 5.1.

5.2.3 Density of Address Space

The more sparsely populated the address space is, the more difficult it is for an adversary to pinpoint the target host. The reason this is true is that the adversary does not know the address of the target host. If the adversary did know the address, he/she would not need to scan the subnet. Assuming the adversary has no additional information pertaining to the identity of a host (e.g., operating system), a successful scan reply provides no indication of success.

The probability of detecting a host increases with the number of hosts there are on a subnet. The probability of detecting a host can be calculated using Equation 5.1. In Figure 5.1, $h = 1$ to represent a single target host. In this case, h is equal to the total number of active hosts on the subnet. As already mentioned, this does not indicate that a host detected by an adversary is the target host.

This factor degrades an adversary's capability of detecting a target host. In the single host scenario discussed in Section 5.2.2, locating a target takes time. Once the target is located, though, the adversary knows he/she has identified the target host because there are no other hosts on the subnet. In the much more likely scenario of multiple hosts on the

subnet, an adversary will get false positives. The term false positive is used to mean that the adversary receives indication of success when the located host is not the target. The false positive rate increases with the number of non-target hosts on the subnet. Unlike a password attack where success provides an adversary access to a machine, a successful scan reply tells the adversary little about whether the discovered host is the target host. Even in the case of multiple discovered hosts, the adversary does not know which host is the target. Of course with additional information, such as operating system or protocol, the adversary can filter out hosts not matching a certain profile.

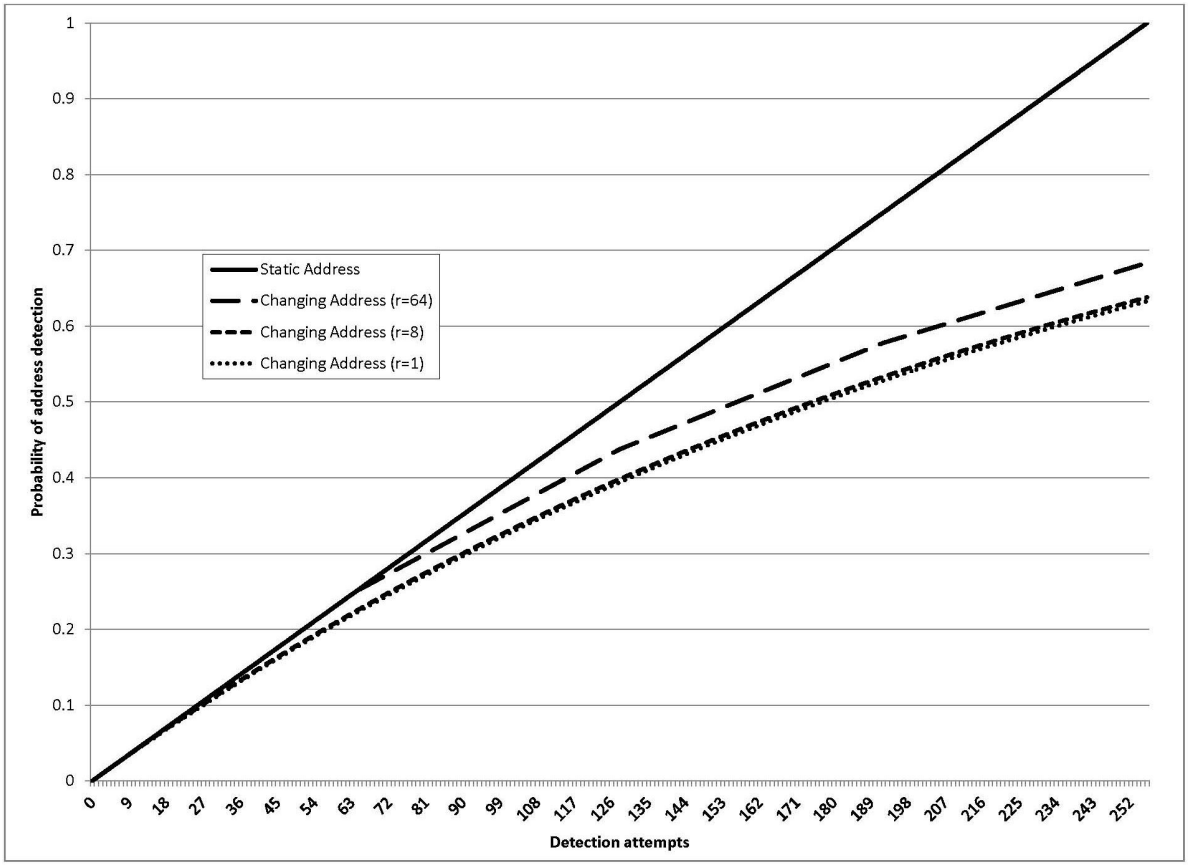


Figure 5.2: Comparison of the detection probability for various address change rates. The comparison is based on the probability an adversary has of detecting a static target address within 256 attempts, on a subnet of 256 addresses, versus the probability of detecting an address that changes after an adversary scans r addresses over z rounds.

5.3 Interaction of Dynamic Address Factors

The three factors described in Section 5.2 are not independent. As certain factors increase, other factors can decrease while still maintaining the same probabilities. For example, there is a relationship between address size and frequency of address change. There is also a relationship between subnet density and frequency of address change.

Increasing the size of the address allows for the frequency of the address change to decrease without degrading security. As the size of the address increases linearly, the size of the address space increases exponentially. The increased address space requires more time and resources from an adversary to exhaustively scan. Beyond a certain address size, an adversary cannot exhaustively scan the exponentially growing network quickly. Therefore, it is possible for the host to decrease the frequency of the address change without increasing the probability it will be detected. Since each address change requires computation on the part of the host, decreasing the frequency of address change is desirable. A larger address space can result in less computational requirement with the same probability of detection as that of a smaller address space with more frequent address changes.

A comparison can be done with different sized subnets to illustrate the relationship between address size and rotation interval. Keeping a static address rotation interval and set number of attempts, the probability of an adversary detecting an address decreases exponentially as the address size increases. Table 5.2 illustrates how significant the impact of address size is on the probability of detecting a host. For this comparison, the adversary is assumed to guess 256 addresses before giving up. The target host's rotation interval matches the time it takes for the adversary to scan 64 addresses. Another way to look at this is that the target host rotates through four addresses in the span of the adversary's 256 attempts. In an IPv4 class C subnet this is equivalent to one quarter of the address space. The table demonstrates that more than half the time an adversary can expect to locate the target host on an IPv4 class C subnet. On an IPv6 subnet, the probability of locating a target host is

Table 5.2: Probability of successfully locating a target address after 256 attempts when the address rotates every 64 attempts ($r = 64$)

	Subnet Type			
	IPv4 Class C	*IPv4 Class B*	*IPv4 Class A*	*IPv6*
Probability of Success	0.684	0.00390	$1.53 \cdot 10^{-5}$	$1.39 \cdot 10^{-17}$

Table 5.3: Probability of detecting a target host in simulation on an 8-bit subnet within 64, 128, 192, and 256 guesses. Each listed probability is the average over 100,000 iterations.

	Probability of detection within:			
	64 guesses	*128 guesses*	*192 guesses*	*256 guesses*
Static Address	0.249	0.503	0.748	1
Changing Address (r=64)	0.248	0.435	0.578	0.682
Changing Address (r=8)	0.225	0.398	0.533	0.637
Changing Address (r=1)	0.222	0.393	0.528	0.632

near zero.

Density of address space also affects frequency of address change. As the density of the addresses space increases, the probability of correlating an address with a specific host decreases. The increased density occurs because more hosts populate the subnet. As previously mentioned, a dense subnet results in a higher probability of an adversary detecting a host that is a false positive. Therefore, a targeted host can use the dense network to lower its probability of being detected. Density in the address space also has a negative effect on increasing the possibility of address collisions. In this context, address collision means that a host changes its address to a pre-existing address on the subnet. Since each host must have a globally unique address to ensure connectivity, address collisions must be avoided on the subnet. Repeated address collisions could prevent a host from sending or receiving network traffic, thus decreasing throughput and QoS. While increased density in the address space provides a host with a lower probability of detection, network density must be balanced with the probability of address collisions to ensure network connectivity.

Address size can help with the issue of address collisions. It is desirable to have a densely populated subnet to increase the probability of an adversary finding a false positive. By increasing the address size, the address space increases. A larger address space allows for more hosts on the subnet without overpopulating the subnet. This means that a larger subnet can be less densely populated. The result is that a detected host still has the same probability of being a false positive while a host changing its address has a lower probability of address collision.

5.4 Simulation Results

To validate the analysis of changing addresses in Section 5.2.2, four different address change rates are simulated. The rates simulated were a static address (never changes) and addresses that changed after an adversary scanned 64 addresses ($r = 64$), eight addresses ($r = 8$), and one address ($r = 1$). The simulation results are listed in Table 5.3. The table highlights four search intervals. The four intervals are 64, 128, 192, and 256 guesses. For each interval, a simulated adversary attempted to locate a target host within the specified interval. Each interval was simulated for 100,000 iterations. The probability displayed is the average over the 100,000 iterations. The probabilities produced, match the calculated probabilities at each interval as depicted in Figure 5.2.

5.5 Security through Dynamic Addressing

Establishing a moving target defense is an effective way of protecting users' privacy and data. Changing hosts' addresses, referred to as dynamic addressing, accomplishes this protection. If target addresses continually change, an adversary loses the expectation of narrowing the search space with successive guesses. If the adversary is able to locate a targeted host, a dynamically changing host address limits the time an adversary has access to the host. Since the discovered address changes, the adversary no longer knows the host's location on the network. Additionally, the nature of dynamic addressing prevents other types of targeted attacks, which rely on static addressing.

Changing the addresses of hosts allows them to logically move within a subnet. As illustrated in Figure 5.2, the more often an address changes, the more difficult it is to locate and target the host. A changing address, combined with other factors such as address size and subnet density, help achieve a moving target defense. A large address space supporting many hosts but sparsely populated makes it difficult to pinpoint a specific target host. Other network hosts result in false positive for an adversary while unoccupied address spaces reduces the possibility of address collisions. The incorporation of dynamic addressing considerably reduces the probability of detecting a target host while still maintaining connectivity.

Dynamic addressing also protects against certain classes of network attacks. For example, an adversary attempting a targeted DoS attack first has to find the target host on the subnet. Even if the adversary finds the host, the attack can last at most as long as the interval between address changes. Other targeted network attacks, such as session hijacking and MITM, are

constrained by the same limitations as DoS attacks. To attack dynamically addressed hosts, an adversary must be able to either quickly find the host after an address change or predict the address change. If a sufficiently randomized dynamic address obscuration algorithm is utilized, targeting hosts in a large address space should not be possible.

5.6 Summary

As users exchange more personally identifiable information over the Internet, it is increasingly important to protect users' security and privacy. One of the best ways to accomplish this is through the use of a moving target defense. At the network layer, this can be achieved by dynamically changing host IP addresses. Frequently changing addresses are probabilistically more difficult to detect than static addresses. Dynamic addresses also provide an additional layer of security for hosts that are detected by an adversary. An adversary is unable to compromise hosts for a significant period of time since the hosts' network address changes. Dynamically changing IP addresses provide security and privacy by creating a moving target solution implementable as low as the network layer of the protocol stack.

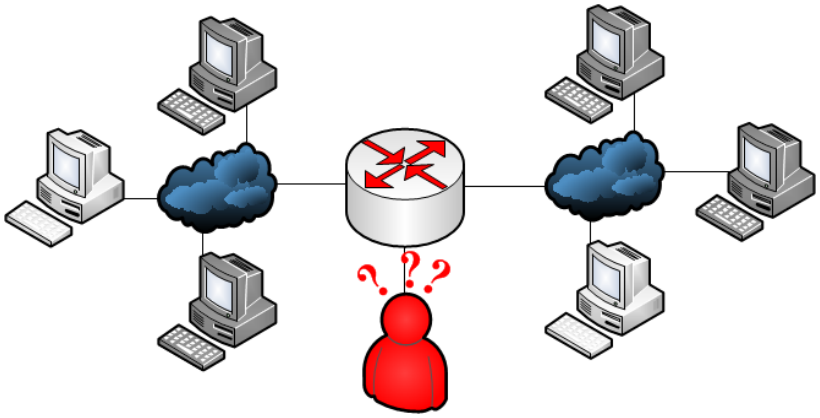
Chapter 6

MT6D Overview

MT6D provides a means for hosts to communicate with each other over the public Internet while maintaining anonymity from targeting, tracking, and traffic correlation. The system does this through tunneling IPv6 packets inside "MT6D packets." Both sender and receiver network and transport layer addresses are dynamically obscured in the new header created by MT6D. MT6D can also encrypt the entire tunneled packet to prevent an attacker from analyzing payloads or header fields.

MT6D rotates addresses based off a set of parameters known only to the two communicating hosts. There is no requirement for a trust model that extends beyond the communicating pair of nodes. MT6D also has no need for communicating hosts to exchange parameters prior to each communication session, meaning that dynamic addresses cannot be linked to host identities. A unique feature of MT6D is that the dynamically obscured addresses can change in the middle of ongoing sessions without breaking the connection or requiring a new handshake. This feature allows addresses to change as often as the security posture dictates rather than being constrained by ongoing network sessions. Figure 6.1 provides a simplified example of what two hosts using MT6D over a network might look like to an attacker verses what is actually occurring.

Dynamic addresses used in MT6D are not linked to hosts' identities, thus requiring attackers to scan subnets for targets. In IPv4, scanning is feasible taking from approximately 30 seconds for a class C subnet to the worst case of 29 days for a class A subnet. MT6D leverages the immense address space of IPv6 to provide an environment so large that an efficient search is infeasible. The time it would take an attacker to scan a single IPv6 subnet is approximately $8.77 \cdot 10^{10}$ years. In the unlikely event that an attacker locates a target, the damage he/she can inflict is limited to the interval between address rotations; reacquiring the



(a) Attackers view of two hosts using MT6D; the communicating hosts appear and disappear on seemingly random addresses

(b) Actual network configuration

Figure 6.1: Depiction of two hosts using MT6D on a network.

target is statistically infeasible. Even if an attacker captures all network traffic, it appears as if multiple hosts are communicating.

The architecture of an MT6D device mimics a network bridge. Outbound packets are sent to an encapsulator that constructs a MT6D packet. The MT6D packet contains the entire original packet excluding original addresses. Encryption can also be applied to the entire original packet during encapsulation to prevent an attacker from analyzing the payload or header fields of packets collected through the use of a packet sniffer. When an MT6D packet arrives at its destination, the packet enters a decapsulator which restores the packet to its original form. The design of MT6D facilitates implementation either integrated into a system or as a separate stand-alone device. An integrated implementation has the advantage of maximum portability, while the stand-alone implementation is platform and operating system independent.

Chapter 7

Threat Protection Offered by MT6D

MT6D is designed primarily to protect against attacks targeted at specific hosts. Although MT6D was not designed to protect against unspecified attacks, some measure of protection is afforded by virtue of its design. MT6D makes no distinction between adversaries internal to the trusted network and those external to it since either type of adversary is dealt with in the same way.

7.1 Targeted Attacks

Targeted attacked are those aimed at a particular host or group of hosts. These attacks can be classified as passive or active.

7.1.1 Passive Attacks

Adversaries passively targeting hosts are interested in discovering host identities and activities. Activities can extend to where hosts are and what they are doing there. Discovering a host's identity is an attack against a host's anonymity. Adversaries are able to determine the identities of hosts through observing authenticated network traffic, observing SLAAC addresses, or correlating network traffic over time.

Discovering a host's activities is an attack on the user's privacy. Host locations can be determined assuming an adversary knows the IPv6 addresses of the targeted hosts. Static addresses support the extended tracking of hosts since they remain constant over time. Addresses that use MAC-based IIDs are particularly susceptible to location tracking. MAC-based IIDs also make hosts more susceptible to traffic correlation since IIDs remains constant

regardless of subnet. Even IIDs that do not remain constant over multiple subnets are typically still static for each particular subnet.

MT6D protects against targeted passive attacks in a couple different ways. First, host addresses are obscured and frequently rotated, which achieves anonymity. Privacy is achieved since adversaries are unable to link addresses with specific hosts at specific locations. Single sessions can be spread over multiple sender and receiver addresses, thus preventing an adversary from easily correlating multiple packets as belonging to the same host or even the same session. Second, entire packets may be encrypted, preventing both traffic correlation and observation of authenticated traffic.

7.1.2 Active Attacks

Adversaries interested in disrupting, intercepting, or modifying the network communications of targeted hosts will launch active attacks. These attacks can come on the form of DoS, with the goal of disrupting network communications, or MITM, with the goal of intercepting and modifying network communications, for example. Hosts with static network addresses are especially susceptible to active attacks since a successful attack against a specific network address persists. MT6D provides dynamic addressing which serves two purposes. First, adversaries cannot locate the hosts they are trying to target due to the use of obscured addresses. Second, an adversary who happens to locate a targeted host can only attack the host for, at most, the length of time between address rotations. After such time, the adversary is forced to relocate the target host.

7.2 Unspecified Attacks

Adversaries may not be concerned with which specific hosts they attack. Unspecified attacks may be aimed at passively collecting network traffic to determine the number of hosts on a network or the nature of network communications. Actively, adversaries may attempt to block any discovered hosts from using network resources. Since this type of adversary is unconcerned with the specific identity of his/her victims, MT6D cannot prevent the adversary from attacking them. MT6D does, however, limit the damage inflicted by unspecified attacks. For example, the network density cannot be accurately determined due to the number of new addresses an adversary observes. It is even difficult for an adversary to determine the nature of network communications because a single session may be spread

over multiple source and destination addresses. Unspecified active attacks are also limited. Since host addresses constantly rotate, active attacks against observed hosts are limited to the amount of time between address rotations. Subsequent unspecified active attacks against the same host are probabilistically more difficult the more active addresses there are on a subnet.

Chapter 8

MT6D Design

MT6D provides a means for hosts to communicate with each other over the public Internet while maintaining security, privacy, and anonymity. The system does this through dynamically obscuring both sender and receiver network and transport layer addresses. Dynamically rotating addresses make MT6D hosts much more difficult to locate on a subnet than static hosts. This claim is supported in Chapter 5 and by using Shannon's explanation of entropy [74]. The lower the entropy, the more certain an outcome becomes. 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.

MT6D increases the number of active addresses on a subnet. Let H represent the total possible addresses within an IPv6 subnet. The number of active hosts on a subnet can be represented by h. In a network using static addresses, the number of active addresses on the subnet equals h. Hosts using MT6D generate multiple addresses over time. If all hosts on a subnet are using MT6D, the number of active addresses on the subnet would far exceed h. The total number of addresses observed can be represented by:

$$h' = h\left(\frac{T}{\Delta t}\right)$$

where h' is total number active MT6D addresses, 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. This relationship can be annotated as follows:

$$h < h' \leq H$$

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. This chapter describes how MT6D is designed to achieve dynamic obscuration of addresses.

8.1 Overview

MT6D secures two communicating hosts by intercepting and encapsulating all packets before they are sent over an untrusted network. Figure 8.1 provides a conceptual example of two hosts communicating using MT6D. When host A sends a packet to host B, it is intercepted by a MT6D device either integrated within or directly connected to host A. The MT6D device associated with host A then encapsulates the packet within a special MT6D packet. Encapsulation by a MT6D device entails encrypting the entire original packet and then attaching a MT6D-specific header. This specialized header uses addresses known only to the sending and receiving MT6D devices. Once the packet is encapsulated, it is forwarded over the MT6D-secured communication zone to host B's MT6D device. Host B's MT6D device decapsulates the MT6D packet by removing the MT6D-specific header and decrypting the original packet. The original packet is then forwarded on to host B. The encapsulation/decapsulation process is transparent to both hosts A and B. When host B sends a reply back to host A the process is reversed. Host B's MT6D device encapsulates the packet, while host A decapsulates it.

One of the key features of MT6D is that each packet sent between hosts A and B could feasible use a different set of addresses within the MT6D-secured communication zone. Addresses can change regardless of the state of any existing connections. The frequency of address changes is determined by agreement of the two communicating hosts.

8.2 Dynamic Addressing

MT6D uses dynamically rotating addresses to obscure the identities and packet contents of communicating hosts. Dynamic addressing modifies the network-layer and transport-layer addresses of the sender and receiver nondeterministically. Network-layer addresses are modified by obscuring the IIDs of communicating hosts. Obscured IIDs are computed using three components obscured by a function. The first component is a value specific to an individual host (e.g., a MAC address). The second component is a secret shared by the

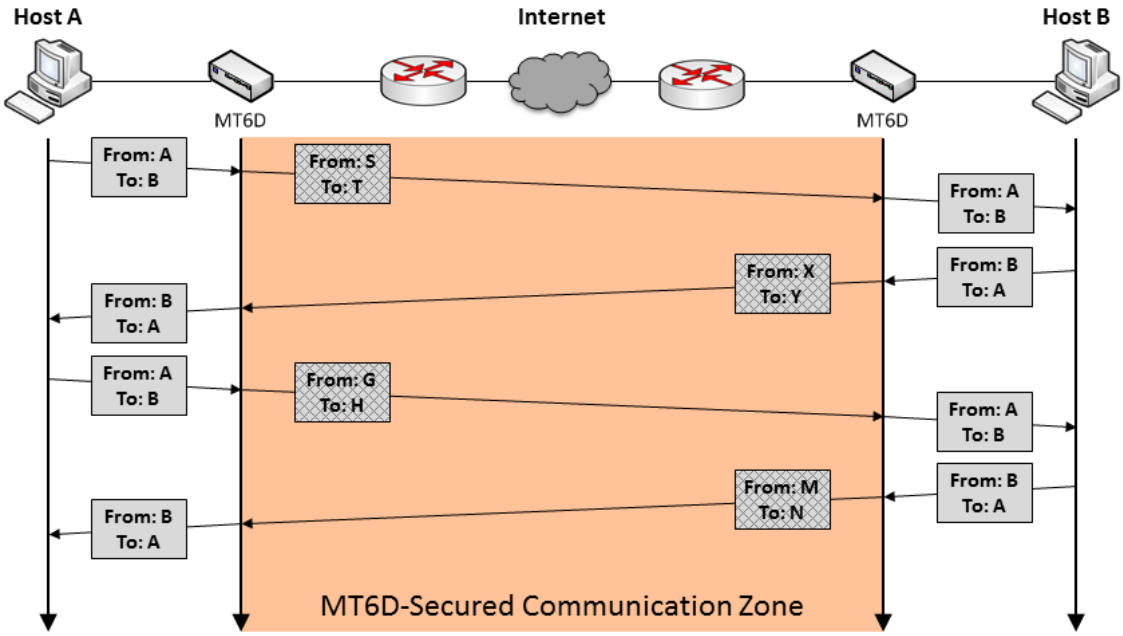


Figure 8.1: Conceptual flow of two hosts communicating using MT6D

sender and receiver (e.g., symmetric key). The third component is a changing value known by both parties (e.g., time). The only one of these three values that must be kept secret is the shared secret. The function results in a 64-bit output used as the MT6D IID and has the form:

$$IID'_{x(i)} = f\{IV_x * S * CV_i\}_{64} \tag{8.1}$$

where $IID'_{x(i)}$ represents the obscured IID for host x at a particular instance i, IV_x represents a value specific to the individual host x, S represents the shared secret, and CV_i represents the changing value at instance i. The three components are combined using an operation denoted by $*$. The 64-bit function result is denoted by $f\{\cdot\}_{64}$.

One possible implementation is computed using a host's EUI-64 IID [38], a shared symmetric key, and a timestamp. The EUI-64 IID is a good parameter choice to represent IV_x for now because it is a piece of information that already exists and identifies the host. While it is possible for duplicate IIDs to exist, an EUI-64 IID contains 2^{47} unique values. As networks become more populated, it may be necessary to use a different value for IV_x.

A shared symmetric key is chosen as the shared secret mainly for speed. Symmetric algorithms are up to 1,000 times faster than asymmetric algorithms [12]. It is also possible for hosts to generate PRNs from symmetric or asymmetric keys [4] to use as shared secrets.

Though this is not currently implemented, it may be a good future alternative to prolong key life and avoid generating new keys.

Time represents the changing value for simplicity. Time is something both hosts know without requiring additional coordination. The main limitation of using time as the changing value is clock drift. If the clocks are not exactly synchronized, it is possible that two hosts will calculate different values for $IID'_{x(i)}$. Clock drift is less of a concern in MT6D because the frequency with which communicating hosts rotate addresses will likely be long enough to allow for loosely synchronized clocks.

The host's EUI-64 IID, shared symmetric key, and timestamp are concatenated and hashed using the Secure Hash Algorithm (SHA), SHA-256 [60]. The obscured IID is constructed from the leftmost 64 bits of the hash (bits 0-63) and has the form:

$$IID'_{x(i)} = H[IID_x||K_S||t_i]_{0\to 63} \tag{8.2}$$

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 formed by concatenating the host's subnet with $IID'_{x(i)}$ as follows:

$$IP'_{x(i)} = Subnet_x||IID'_{x(i)}$$

In addition to dynamically obscuring IIDs, ports must also be obscured. If port numbers are left unobscured, a third party can use the port numbers of collected packets to correlate the packets with one another. MT6D includes two techniques to dynamically obscure the source and destination ports. The first technique allows the host to specify a port address range that MT6D should use. The benefit of specifying a port range is that MT6D can be configured to use common ports that more closely mimic normal network traffic. Another benefit is that a port range can be specified that conforms to firewall rules. The second technique obscures port numbers using a similar technique as that of the IID obscuration in Equation 8.2. For example, obscured ports could leverage the unused bits of the hash calculation in Equation 8.2 as follows:

$$Src\_Port_i = H[IID_{Src}||K_S||t_i]_{64\to 79}$$
$$Dest\_Port_i = H[IID_{Dest}||K_S||t_i]_{64\to 79}$$

The source port, annotated by Src_Port_i, utilizes the next 16 bits of the source IID hash (bits 64-79) while the destination port, annotated by $Dest\_Port_i$, utilizes bits 64-79 of the destination IID hash. The obscured port numbers are used as the port numbers in the MT6D header. Since the current implementation of MT6D encapsulates all packets using the Unreliable Datagram Protocol (UDP), no other transport-layer header fields need to be obscured. Figure 8.2 illustrates the MT6D UDP header.

8.3 MT6D Tunneling

Rather than rewrite each original packet using the obscured IPv6 addresses of the sender and receiver, the original packet is encapsulated in a MT6D tunnel. Tunneling the original packet allows any established end-to-end connections between the source and destination to be retained as well as any flags specific to that session. Not only does the application of MT6D become transparent to the host, but also the MT6D connection can have different configuration settings.

A MT6D packet is formed by first removing the source and destination addresses from the original packet. The IPv6 addresses are removed to reduce overhead on the packet. Each tunnel endpoint is able to replace these addresses before they are delivered to the receiving host. The Ethernet frame is also removed to anonymize the MAC addresses. This also does not present an issue since it will be reconstructed at the destination MT6D device. The entire packet is then prepended with a MT6D header as illustrated in Figure 8.3. The MT6D header is formed using the dynamically obscured source and destination addresses discussed in Section 8.2. The MT6D packet is also configured to use a transport layer protocol that supports connectionless communication.

In one implementation, each packet is encapsulated in UDP to prevent 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

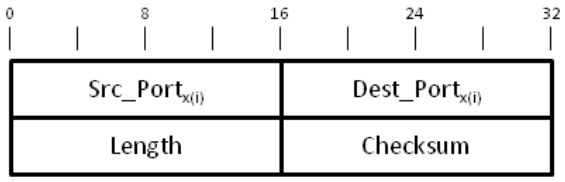


Figure 8.2: MT6D UDP header format

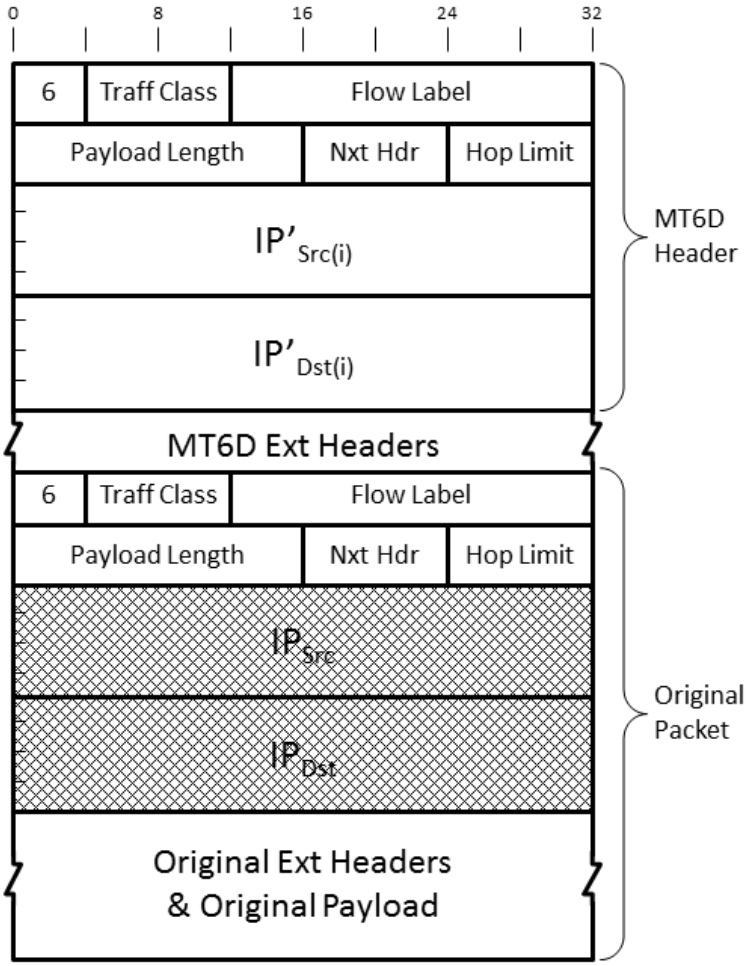


Figure 8.3: Original packet tunneled within a MT6D packet. Note that the original source and destination addresses are removed to prevent address tracking and reduce overhead.

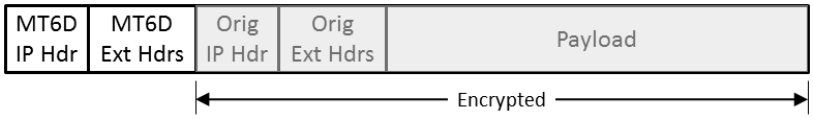
original packet. Since transport layer protocols are end-to-end, decapsulation will occur before the host processes the original packet. A session using TCP will still exchange all required TCP-related information. This information will simply be wrapped in a MT6D UDP packet. Additionally, any lost packets that were originally TCP will be retransmitted by the end host after a retransmission timeout occurs. Fragmentation is also not an issue since fragmentation occurs at the source in IPv6.

8.3.1 Encrypted Tunnel

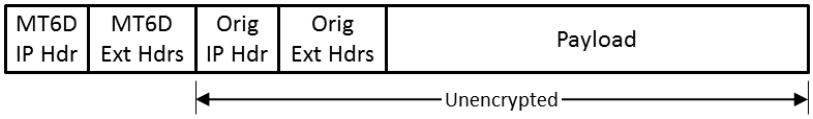
By default, MT6D encrypts each original packet before appending it with the MT6D header. An example of a MT6D encrypted packet is shown in Figure 8.4(a). By encrypting the original packet, a third party is unable to glean any useful information from the packet. For example, if the original packet is 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 content of any captured network traffic is kept private through encryption.

Another benefit of encrypted tunnels is that hosts are able to authenticate traffic to each other while still maintaining their anonymity to any third party. Since the original authenticated packets get encrypted using a symmetric key, a third party will be unable to detect that the packets are authenticated as illustrated in Figure 8.5. Additionally, a third party observer will glean no identifiable information about the communicating hosts from captured packets.

Encryption in MT6D is implemented through the use of a destination options header [14]. The destination options header is a type of IPv6 extension header that is only evaluated by the destination node. Since destination options are not evaluated by intermediate nodes, they are not required to be publicly defined and accepted. Since destination options are a type of extension header, they must be a multiple of 8 bytes. For that reason, the encryption destination option is 8 bytes long and is formatted according to Figure 8.6. The *Next Header* field, as explained in Chapter 2, Section 2.1.1, is used to notify IPv6 of any additional extension headers included in the packet. The *Hdr Ext Len* field indicates the length of the extension header. For encryption, it is set to zero since the extension header is 8 bytes long and the length value does not include the first 8 bytes. The *Option Type* field is used to

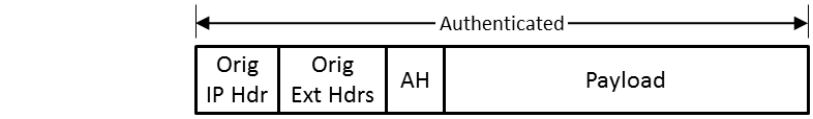


(a) Encrypted tunnel

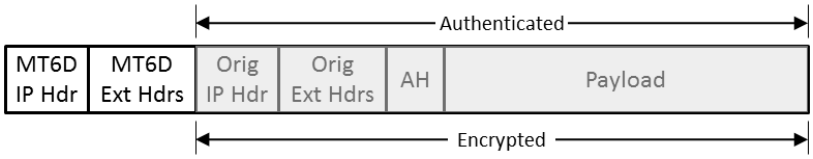


(b) Unencrypted tunnel

Figure 8.4: Original IPv6 packet encapsulation within a MT6D tunnel.



(a) Original authenticated IPv6 packet



(b) Authenticated IPv6 packet encapsulated by MT6D

Figure 8.5: Example authentication implementation. The authenticated packet in (a), once wrapped in MT6D encapsulation as illustrated in (b), no longer gives away the identity of the authenticating host.

indicate the type of destination option included – in this case, encryption. The type value for encryption, defined by MT6D, is 219. The *Option Data Len* is set to one indicating 1 byte of data for the encryption destination option. The *Data* field uses the first 4 bits to define the type of encryption used while the last 4 bits specify any required padding. Any type of encryption can be used to encrypt data, but currently Advanced Encryption Standard (AES) encryption in cipher-block chaining (CBC) mode is used. In this mode, blocks must be 16 bits long. Padding is used to specify the number of bits to add to the original packet to make it a multiple of 16 bits. The remaining 3 bytes of the destination option extension header are unused and padded. An adversary could potentially use the destination options extension header as an oracle to confirm that a host is using MT6D since it contains a header field specific to MT6D. While this may help an adversary identify a MT6D user, it does not provide any information about the actual user's identity or that multiple captured packets are associated with the same communicating host pair. This is particularly true when multiple host pairs on a subnet are using MT6D.

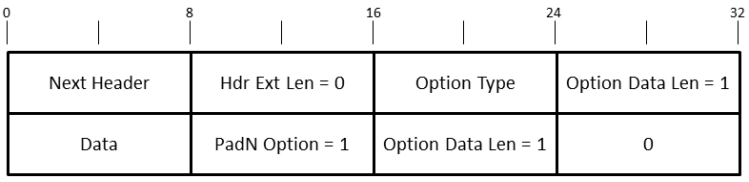


Figure 8.6: Encryption destination option header format

Modes of operation that encrypt each block independently (e.g., electronic codebook) should be avoided when selecting the encryption mode to use. Packets often contain patterns of repeating block, which when encrypted independently will result in ciphertext patterns. Additionally, blocks of plaintext can be removed, inserted, or reordered without detection [73]. For efficiency, modes of operation should be avoided that prevent parallelization, such as output-feedback mode. Although modes that do not allow parallelization are not necessarily less secure, they are less efficient and can impact performance.

Counter mode has distinct advantages over other block cipher techniques. For example, both encryption and decryption can be parallelized. Counter mode also does not require padding where most other modes of operation do. The use of padding is susceptible to a padding oracle attack [68] where an adversary may be able to recover plaintext corresponding to any block of ciphertext. Counter mode also requires much less code to implement since only the encryption function needs to be implemented (versus encryption and decryption). The main disadvantage of using counter mode is if a nonce is reused, information about the entire message can be leaked rather than just the initial block [85]. Another disadvantage that impacts MT6D users is synchronized counters are required between each communicating host pair.

Stream ciphers are another possible choice for implementing encryption in MT6D. Stream ciphers typically operate on one bit at a time. In a hardware implementation, a stream cipher may be more efficient [73]. Since the prototype MT6D is implemented in software, a block cipher was easier to implement; particularly since there are cryptographic libraries that can be easily integrated into the MT6D prototype. Another consideration with stream ciphers is that keys are typically required to be longer. Longer keys may make generating new keys (discussed in Section 8.6) more challenging.

8.3.2 Unencrypted Tunnel

MT6D includes the option to tunnel the original packets unencrypted as illustrated in Figure 8.4(b). Since the source and destination addresses are stripped from the original packet header, address tracking is not feasible. Additionally, a third party is unable to determine which two hosts are communicating. Unencrypted tunnels do not, however, prevent traffic correlation since the remainder of the original headers and payloads stay intact. Traffic correlation does require deep packet inspection because any relevant header fields are embedded within the MT6D packet payload. Unencrypted tunnels may be preferred in

environments where minimizing computational expense is more important than preserving privacy. Whether using encrypted or unencrypted tunnels, protection from targeted network attacks is still preserved.

8.4 Varying Address Rotation Times

It is necessary to vary address rotation times for each communicating address pair. Since hosts cannot set their own address rotation times without agreement from the other hosts they communicate with, a method needs to be in place to ensure that hosts using MT6D do not all have to rotate addresses at the same time. A set rotation time is not desirable from a security and privacy perspective, nor from a network management perspective. Therefore, a means to vary the rotation interval was developed that can occur without interaction from other MT6D hosts.

There are two main factors to consider for implementing a variable rotation time that does not require host interaction. First, communicating hosts must share some value that can be used to determine the rotation time. Second, any host should be able to start MT6D at any time and still determine the correct time to rotate its address. To meet both of these requirements the address rotation time (RT) is computed as follows:

$$RT = \left\lfloor \frac{\text{UTC}}{(H(K_S) \bmod c_1) + c_2} \right\rfloor \tag{8.3}$$

where UTC represents the coordinated universal time, $H(K_S)$ is the hash of the symmetric key shared between two hosts, and c_1 and c_2 are two arbitrary constants set in the MT6D system parameters to determine the range of the interval and the minimum rotation time, respectively. The shared symmetric key is used in the varying rotation time equation since it is a value already known by both communicating hosts and it allows for each communicating pair to have a different rotation time. To facilitate hosts starting MT6D at any point in time with the proper rotation time, the current time is divided by the rotation interval in Equation 8.3. If the rotation interval were added to the current time, the rotation time would be dependent on the start-up time of each MT6D device. The method used in Equation 8.3 provides a means for hosts to know the proper rotation time without prior coordination.

Rotating addresses unpredictably increases the difficulty a third party has of identifying which hosts are using MT6D. Coron and Kizhvatov [10] discuss how predictable execution can lead to side-channel attacks. The authors go on to prove that adding random delays

increases the attack complexity for the adversary. Although their discussion is aimed at differential power analysis, the same principles hold with address rotation times. If all hosts using MT6D change addresses at the same time, it is feasible for an attacker to determine the number of hosts using MT6D by observing address changes at rotation times. Similarly, a single host using MT6D is easily identified by observing a changing address at set intervals. Conversely, hosts rotating addresses at varying times are more difficult to pinpoint because address rotations can occur at any time and frequency. Figure 8.7 illustrates how address rotation may occur over time for a single host. In the diagram, host A may be connected to N different hosts. The address A_{1x} that host A shares with host B is different than the address A_{2x} that host A shares with host C and so on. Since host A has N addresses bound at any given time and they all rotate at different intervals, an adversary will have a difficult time isolating packets to a specific host. MT6D host detection is made more difficult when hosts are not continuously transmitting. In the event that a host is not continuously transmitting it is possible that address rotations will occur without packets being transmitted. When this happens, any attempt to tie address rotations to a particular host is made more difficult due to a break in any perceived rotation pattern. It is worth pointing out that knowing a host is using MT6D does not provide an attacker with knowledge of the host's identity.

From a more practical perspective, varying address rotation times lessens the computa-

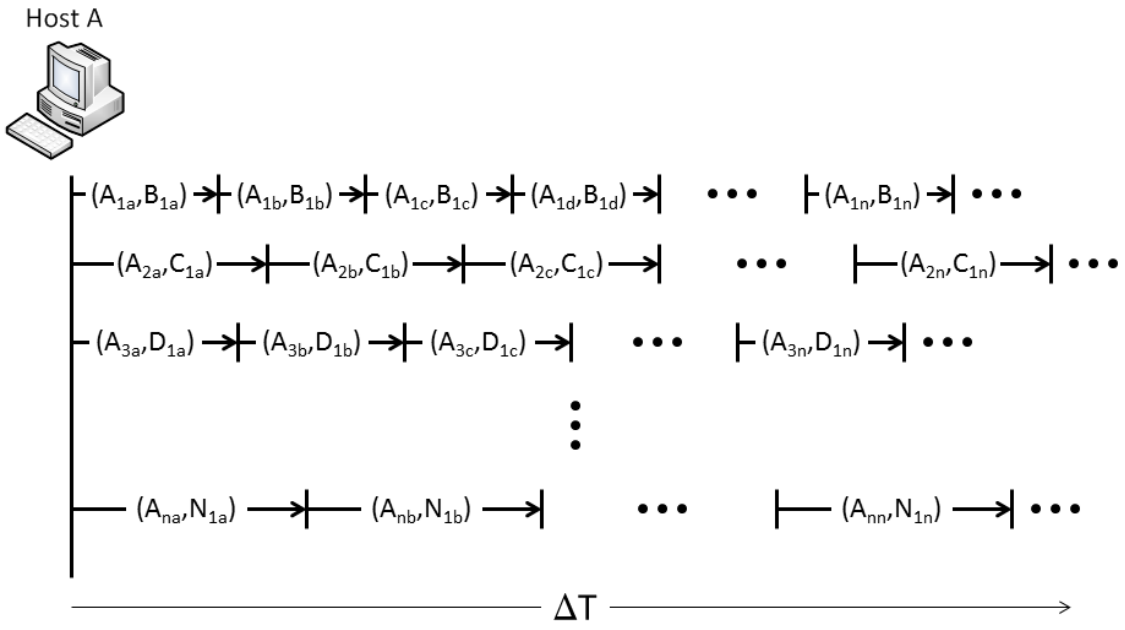


Figure 8.7: Visualization of address rotations for a single host using MT6D

tional burden on networking equipment. A large number of MT6D users on a subnet, all changing addresses at the same time, has the potential to place a burden on a router by forcing it to simultaneously bind every new address. Varying the rotation times of hosts using MT6D distributes the binding of new addresses.

8.5 Symmetric Key Usage

In its preferred implementation, MT6D uses symmetric keys shared between each communicating host pair to generate IIDs. How the symmetric keys are incorporated is described in Section 8.2. Each symmetric key is known only to the two communicating hosts. This design decision was made to eliminate the need for tying a host's identity to the key in use while simultaneously reducing the scope of a key compromise to the specific host pair using the key.

Symmetric keys in MT6D are either preloaded, exchanged out-of-band, or exchanged in-band. Of the three techniques, in-band key exchange is the least preferred because it is the most susceptible to eavesdropping. Eavesdropping on an in-band key exchange exposes the real IPv6 addresses of the communicating hosts. This research does not attempt to solve key exchange issues, it only points out the possible options for establishing keys. It is worth noting that even if a malicious host learns a host's real IPv6 address, he or she cannot match observed MT6D packets to that particular host.

8.6 Dynamic Key Generation

Symmetric keys should change periodically to prevent the possibility of key compromise. MT6D includes a means to periodically generate new symmetric keys called dynamic key generation (DKG). Two different methods of dynamically generating symmetric keys exist in MT6D. These two methods are referred to as "out-of-band" DKG (o-DKG) and "in-band" DKG (i-DKG).

8.6.1 "Out-of-Band" DKG

There are some implementations of MT6D which may pre-distribute current and future symmetric keys. This type of distribution scheme can be referred to as generating keys "out-of-band." Agencies such as the military will typically distribute keys "out-of-band."

The main benefit to using o-DKG is that there is no requirement for hosts to exchange keys; the keys are all pre-loaded. This also means less chance for error due to missed packets or bit errors. The main disadvantage is scalability. Pre-distribution of keys requires each device to be physically loaded with keying material. This takes time, resources, and a pre-knowledge of how many keys are needed before another pre-distribution is needed.

8.6.2 "In-Band" DKG

"In-band" DKG was developed to provide a means for hosts to periodically generate new symmetric keys without giving away the identity of the two communicating hosts, nor generating additional network traffic. In the i-DKG process, one of the two communicating hosts' MT6D devices initiates the key exchange. This host becomes the lead in the exchange and controls key generation and the key adoption time. Since there is no assigned hierarchy among MT6D hosts, allowing the initiator of i-DKG to take the lead in the process avoids unnecessary interactions between communicating pairs.

Key Generation

Once a MT6D device of a communicating host initiates i-DKG, it generates a new symmetric key using a random number generator. The new symmetric key is then encrypted. The preferred method of encryption is with the receiver's public key. If the receiver's public key is unknown, the new symmetric key can be encrypted with the current symmetric key. It is preferred to use the receiver's public key over the current symmetric key in the event that the current symmetric key becomes compromised. Encrypting the new symmetric key with the receiver's public key can potentially recover the session, while encrypting the new symmetric key with a compromised symmetric key will likely result in the new symmetric key also being compromised.

Packet Format

Since MT6D is completely transparent to communicating hosts, no MT6D-specific messages are sent between hosts. To do so would require additional overhead and potentially provide attackers with information they could use to target a host. Therefore, i-DKG information is included in the main MT6D packet header or as a destination option [14] in the MT6D header used to tunnel packets.

The format for the i-DKG destination option is illustrated in Figure 8.8. As discussed in Section 8.3.1, destination options in IPv6 are a special type of IPv6 extension header [14]. To use destination options, the next header value of the IPv6 base packet header or of the previous extension header is set to 60 [14]. Within the destination option extension header, the header extension length (*Hdr Ext Len*) signifies the overall length of the extension header. For a 128-bit symmetric key, *Hdr Ext Len* = 3, indicating an extension header length of 32 bytes. The option type is set to 204 to signify i-DKG. The option data length (Opt Data Len) is the number of bytes of data contained in the destination option, not including the option type or length fields. For a 128-bit key, *Opt Data Len* = 21. The first octet of data following the option data length is reserved for the key encryption type (*Key Encrypt*). This is the encryption algorithm used to encrypt the key prior to embedding the key into the destination option. The time between key changes (CT) follows and is 4 bytes long. The next group of bytes is the new encrypted symmetric key. The remainder of the extension header is padded to be a multiple of 8-octets.

Key Exchange

As mentioned, i-DKG is initiated by one of the two communicating hosts. Each host establishes a baseline key change interval in the MT6D settings. A small random time window is added to the baseline interval to minimize the likelihood of both hosts initiating i-DKG at the same time. The baseline interval plus the small random time window comprises

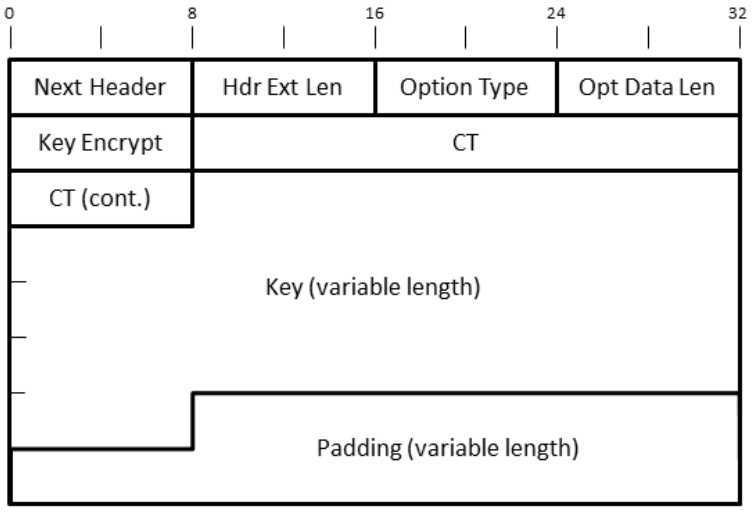


Figure 8.8: Destination option format for i-DKG

CT. The host with the lower CT is the initiator. The initiator begins the i-DKG process after half of the key change time window ($CT/2$) has elapsed by notifying the other host of the impending key change. Since both hosts may not be actively communicating, the remaining $CT/2$ is used as a timeout value to ensure that keys do not change without both hosts being in agreement. Once a host initiates i-DKG, it treats every $CT/8$ of elapsed time as a new i-DKG session until it receives an acknowledgment from the other host. This means that after every $CT/8$ with no acknowledgment, the initiator sends a new symmetric key which will take effect after another $CT/2$ of elapsed time.

Since it is critical that both hosts are aware of an upcoming key change, i-DKG uses a handshaking process to ensure agreement before adopting new symmetric keys. A one-bit flag is used to accomplish the handshake. Rather than incurring additional overhead by adding another destination option, i-DKG makes use of the first bit of the source port belonging to each communicating host. The source port was chosen because it typically corresponds to the ephemeral port, which is usually not blocked by firewalls. Modifying the port number has no impact on the transmitted packet since port numbers are randomly assigned by MT6D. The choice of where in the packet to put the one-bit flag is arbitrary and can easily be located elsewhere. Prior to one of the two communicating hosts initiating i-DKG, both hosts will have the i-DKG flag set to "0". Under normal conditions where two hosts are exchanging frequent packets, i-DKG will occur as illustrated in Figure 8.9 and as described below.

1. Host A, with a lower CT initiates i-DKG after $CT_A/2$. In the next packet from host A to host B, host A sets the i-DKG flag to "1" and includes the i-DKG destination option according to the format shown in Figure 8.8.

2. Host B receives the packet containing the new symmetric key from host A. Host B acknowledges successful receipt and decryption of the new symmetric key by setting its i-DKG flag to "1" on the next packet destined for host A.

3. Host A receives the acknowledgment from host B and sets its i-DKG flag back to "0". It is not until this acknowledgment packet is received, that host A reverts its flag back to "0".

4. Host B receives a packet from host A with the i-DKG flag set to "0" and sets its own i-DKG flag back to "0". It is not until host B sees a packet from host A with the i-DKG flag set to "0" that host B reverts its flag back to "0".

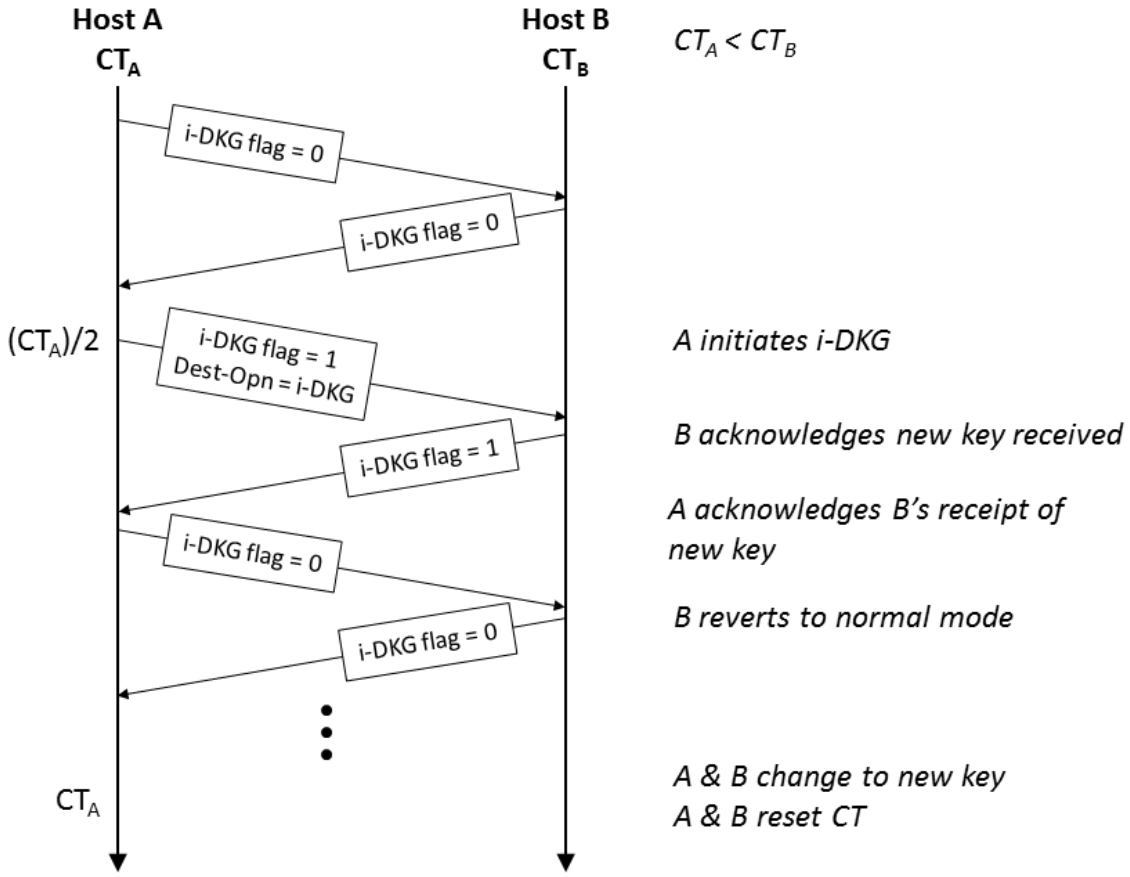


Figure 8.9: Normal packet flow for i-DKG

5. If and only if both hosts have their i-DKG flag set to "0" and after a total of CT_A elapsed time, both hosts adopt the new symmetric key.

There are cases where i-DKG does not execute successfully. This most often refers to the case where host A does not receive an acknowledgment that host B received the new symmetric key. There are a number of reasons host A may not receive an acknowledgment. One reason is that either host may be finished with the current communications session and have no more packets to send. Since i-DKG messages piggyback on normal traffic, some or all of the i-DKG messages will not be sent. Another reason is simply a dropped packet containing new key information. Even though the intended receiver may request the dropped packet be retransmitted, neither host is aware the packet contained i-DKG information. This information is only shared between MT6D devices. Regardless of why the i-DKG process does not execute successfully, host A will attempt retransmissions of the new symmetric key

every $CT/8$ as described below.

1. Host A, with a lower CT initiates i-DKG after $CT_A/2$. In the next packet from host A to host B, host A sets the i-DKG flag to "1" and includes the i-DKG destination option according to the format shown in Figure 8.8.

2. After $CT/8$ passes since the new symmetric key was sent, host A either receives no packets from host B or host A receives packets with the i-DKG flag still set to "0".

3. Host A begins the i-DKG sequence again.

4. Steps 2 and 3 continue until host A receives either an acknowledgment from host B or host B initiates its own i-DKG session.

5. The remainder of the i-DKG process proceeds as explained under normal conditions.

It is possible that host B will initiate i-DKG after host A has initiated i-DKG. This can occur when either host B turns on MT6D after host A sent the first i-DKG packet or the packet from host A was dropped. When this occurs, host B becomes the initiator. This prevents confusion due to contending keys and avoids possible loss of a valid symmetric key. The obvious downside of this feature is that host B can force host A to always follow its key generation schedule. Since host B is assumed to be trusted, this type of malicious activity is not expected nor would host B gain a significant benefit from it.

There are two possible scenarios that could cause one host to complete the key generation while the other host does not. The result is that hosts A and B will have a key mismatch preventing them from communicating. In order to restore communications, hosts A and B will have to establish a new key using one of the aforementioned techniques. One of the two possible scenarios is malicious while the other is not.

The malicious scenario occurs when an adversary attempts to manipulate i-DKG flags in such a way that one host believes the i-DKG process terminated successfully while the other hosts believes it failed. This type of attack can occur if an adversary is able to intercept packets destined for one of the hosts and modify its i-DKG flag. Although this attack is possible, it is difficult to achieve. For this attack to succeed, a number of events have to occur:

1. The adversary has to be on the communication path between the sender and receiver. This usually means that the adversary has a presence on either the sender's or receiver's subnet. While this is entirely possible, it requires proximity to the target host.

2. The adversary has to be able to identify which observed packets are using MT6D, are in the midst of a i-DKG exchange, and belong to the same pair of hosts. The latter condition is difficult to achieve in a subnet with multiple MT6D users.

3. The adversary has to be able to intercept the appropriate MT6D packet, block the target host from receiving the packet, manipulate the i-DKG flag, and forward the packet to the target host. Although this is nothing more than a MITM attack, MT6D and the i-DKG process make it more difficult to achieve. Since all subsequent MT6D packets contain the same i-DKG flag until the next acknowledgment is received, the adversary would need to intercept and manipulate all subsequent packets. This is difficult under normal circumstances. When MT6D rotates addresses, the adversary can no longer be sure which packets belong to the target host or even if they are in the midst of an i-DKG exchange.

Manipulating an i-DKG exchange is also more difficult since either host can initiate the process and the initiation time is randomized based on a PRN. Additionally, the host that thinks the exchange failed will initiate a new i-DKG exchange which will override the previous one.

The second scenario is not malicious and occurs when the final acknowledgment from host B is dropped. In this case, host B will think that the i-DKG process executed successfully, while host A will think that it failed. If no more packets are exchanged between hosts A and B before CT_A, then host B will adopt the new key and host A will not. Although this flaw is possible, it is unlikely. If host B sends any more packets to host A before CT_A, they will all contain the appropriate i-DKG flag. As long as host A receives any one of the packets, host A will know i-DKG concluded successfully. If host A does not receive any more packets from host B before CT_A, host A will assume i-DKG failed. As long as host A sends a packet to host B, host A will reattempt the i-DKG exchange. This will cause host B to discard the previous key and begin the process again.

8.7 Dynamic Subnet Obscuration

MT6D, as described thus far, dynamically obscures the IID of the IPv6 address, but does nothing to obscure the subnet a host is communicating from. It should be noted that an adversary gains little from knowing a host's subnet. The host's identity and packet contents are still protected by IID obscuration and encryption. Through subnet analysis an adversary

might be able to discern that a host from a particular subnet is communicating with a host from another particular subnet. Even though both hosts' IIDs are constantly changing, an adversary can reason that all packets containing the same two subnets are part of the same conversation; this is particularly true in sparsely populated subnets. For this type of subnet correlation to occur, the adversary needs to have a presence on the subnet of either the sender or receiver.

The DSO concept was devised to prevent subnet correlation. MT6D can support DSO using encrypted tunnels between federated MT6D nodes. This is achieved by nondeterministically routing packets through trusted MT6D hosts. Each intermediate MT6D host that handles the packet will compute a new DSO header for the packet. Tracing the original packet from the source subnet to the destination subnet will be difficult since packets are routed through other intermediate MT6D nodes. Subnet correlation is made more difficult since responses and subsequent packets do not necessarily pass through the same intermediate MT6D nodes. Packets between two communicating hosts will be perceived by an adversary as arriving from and departing to different subnets. DSO will incur additional latency due to extra hops and processing imposed on each packet. For extremely sensitive communications, however, it may be desirable use DSO to prevent any type of subnet correlation.

A downside to implementing a concept like DSO is that it changes the trust model from direct trust to transitive-based trust [70]. Both the sender and receiver have to trust every intermediate MT6D node that handles each packet; some of whom they may not have a direct trust relationship with. DSO was conceptualized to provide intermediate nodes with only enough information to successfully pass a packet and thus minimize the level of trust communicating pairs are required to place in intermediate nodes. Layered encryption is used to conceal information from intermediate nodes. The original packet is encapsulated by the sender in a MT6D-formatted packet using a key shared with the receiver. The packet is encapsulated again inside another MT6D-formatted packet using a key shared by the sender and the first intermediate node. This effectively creates a "double tunnel" for each packet using DSO. Outside the inner tunnel, but inside the outer tunnel is a destination options header that includes the actual source and destination address of each packet. This is the only information available to intermediate nodes. Each MT6D packet using DSO can feasibly take a different path through the network as illustrated in Figure 8.10. As a result, intermediate nodes may only handle a small portion of DSO-forwarded packets; further limiting the information available to an intermediate node. The threat of uncooperative intermediate nodes is mitigated by varying the paths packets travel through the network.

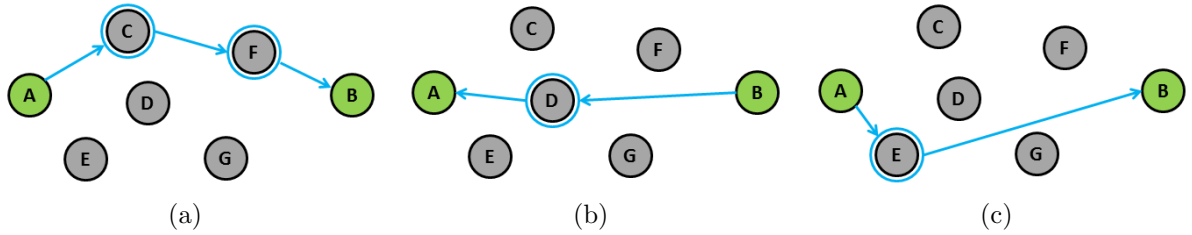


Figure 8.10: Conceptual view of DSO packet forwarding. Nodes A & B are the two communicating hosts. Nodes C - G are other hosts on the network using MT6D. The three different views illustrate that each packet may traverse a different path between the two communicating hosts.

An uncooperative intermediate node may intentionally not forward DSO-forwarded packets. Dropped packets resulting from an uncooperative intermediate node do not prevent successful communication between the sender and receiver; they only increase packet loss, which is handled according to the transport layer protocol.

The following steps describe how a packet is handled between two MT6D hosts using DSO:

1. The sender, host A, encapsulates a packet destined for host B in a MT6D-formatted and encrypted packet. Using MT6D encrypted tunnels on the original packet prevents any intermediate MT6D nodes from reading the packet contents. This packet also includes a destination option indicating that DSO will be used. The destination option includes:

   (a) An option type which acts as a flag to indicate that DSO is being used.

   (b) A TTL to prevent loops.

   (c) The actual IPv6 addresses of hosts A and B. Including the IPv6 addresses of hosts A and B in the destination option allows intermediate MT6D nodes to know the final destination and the final destination to know the sender's address. This is necessary since IPv6 addresses are removed from the original header and since the original packet is encrypted.

2. Host A chooses an intermediate MT6D node from the list of MT6D hosts that it trusts.

3. Host A encapsulates the already encapsulated MT6D-formated packet inside a second MT6D-formated packet using host A as the source and the trusted intermediate node as the destination. The second MT6D tunnel is also encrypted to protect the information

contained in the destination options header of the inner tunneled packet. The outer tunnel does not include a DSO destination options header. This serves two purposes. First, it masks the use of DSO to any third party. Second, and more importantly, it masks hosts not using DSO.

4. The intermediate MT6D node, upon receipt of a packet:
   (a) Decapsulates it
   (b) Checks for the presence of DSO destination options header (if none exists, the packet is not using DSO and the host proceeds with normal decapsulation)
   (c) Checks to see if it is the final destination for the packet
      i. If so, proceed to Step 5.
      ii. Else, continue to next step.
   (d) Checks if it has a trust relationship with the packet destination.
      i. If so, the packet destination will be the next trusted MT6D node.
      ii. Else, another intermediate node is chosen as the next trusted MT6D node.
   (e) Decrements the DSO TTL field. The intermediate node discards the packet if the DSO TTL is zero and the packet destination is not the next trusted MT6D node.
   (f) Encapsulates the already encapsulated MT6D-formated packet inside a second MT6D-formated packet using its address as the source and the next trusted MT6D node as the destination.
   (g) Sends packet to next trusted MT6D node.
   (h) Repeat Step 4.
5. Host B, upon receipt of the packet from the intermediate MT6D node:
   (a) Decapsulates the outer MT6D tunnel
   (b) Retrieves the original packet source and destination addresses from the DSO destination options header.
   (c) Retrieves the appropriate symmetric key required to decapsulate the inner tunnel using the original packet addresses.
   (d) Decapsulates the original packet and delivers it to the protected host.

Note in Step 4(d)i. that each intermediate node tries to deliver the packet to the final destination, if possible. There is little benefit from having more than one intermediate node handle each DSO packet. An adversary capturing packets on either the source or destination subnet will see packets sent from and arriving to different destinations, regardless of the number of intermediate nodes. Since subsequent packets are routed through different intermediate nodes, an adversary on the same subnet as an intermediate node may only capture one packet sent between hosts A and B. In fact, the more intermediate nodes that handle each packet, the more nodes with transient trust that will see the source and destination addresses of hosts A and B.

The use of DSO does have some performance-associated limitations. First, latency is added to each packet as it is routed through one or more intermediate nodes. Next, the additional overhead incurred by adding a second MT6D header will reduce the maximum transmission unit (MTU) of packets sent from the protected host. This also incurs additional latency because more packets will need to be sent to account for the smaller MTU. Higher packet loss will result from packets not reaching the intended destination before the TTL expires. The probability of packet loss from an expiring TTL can be reduced by adding a table of whom trusted hosts trust ("friends of friends"). A "friends of friends" table can be used to make better intermediate node selections. A MT6D host can first look to see if any of its trusted nodes also trust the packet destination. The sender can then choose one of these trusted nodes to be the intermediate node. Packet loss will also increase from a higher probability of packets arriving at the intended destination after the next address rotation. Packet loss due to address rotation while a packet is in transit can be reduced by using the next address rotation for DSO packets sent close to the end of an address rotation interval. As previously discussed, packet loss can also occur as a result of uncooperative intermediate MT6D nodes dropping packets rather than participating in the DSO process.

8.8 Architecture

The architecture for a single MT6D host is illustrated in Figure 8.11. The architecture at the other end of the MT6D tunnel mirrors Figure 8.11. MT6D is designed to be virtually transparent to the user. Each packet is encapsulated by the sender and sent to the internal MT6D NIC. This NIC can be a physical NIC, when MT6D is implemented in a separate device, or a virtual NIC, when implemented in software on the host. The internal NIC directs all incoming packets to the MT6D encapsulator.

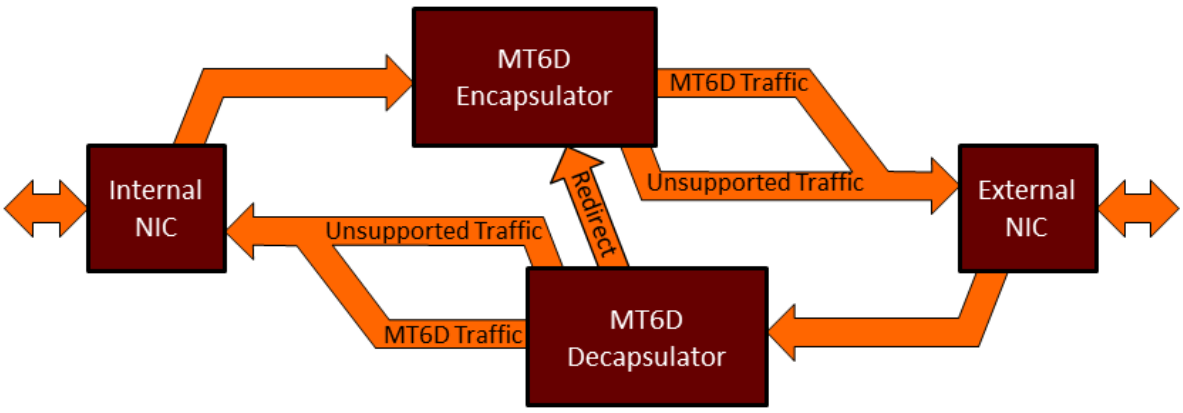


Figure 8.11: MT6D host architecture. The unsupported traffic path is optional and exists to facilitate communication with hosts that do not use MT6D (e.g., some web servers).

The MT6D encapsulator, shown in Figure 8.12, 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. An optional unsupported traffic path exists to facilitate communication with hosts that do not use MT6D (e.g., some web servers). Depending on the desired level of security, unsupported traffic is either blocked or immediately forwarded to the nearest gateway device. Packets that match profiles have the source and destination network addresses removed from the packet header. Packets are then checked to see if encryption is desired. Encryption is optional but is enabled by default. Encryption protects against traffic correlation and hides authenticated traffic as described in Section 8.3.1. Whether the packet is encrypted or not, the next step is to check if the packet is using DSO. If it is, then the packet is tunneled using a DSO header. Otherwise, the packet is placed inside a MT6D tunnel depicted in Figure 8.4. The final step is to pass the packet to the external NIC and transmit it.

The MT6D decapsulator, shown in Figure 8.13, receives all inbound packets. Each packet is received by the external NIC and checked for a MT6D profile. Those packets that do not match profiles are considered unsupported and optionally delivered immediately to the host. Packets that match MT6D profiles have the tunnel header stripped off. If the header is a DSO header, the packet is redirected to the encapsulator where it can be forwarded to

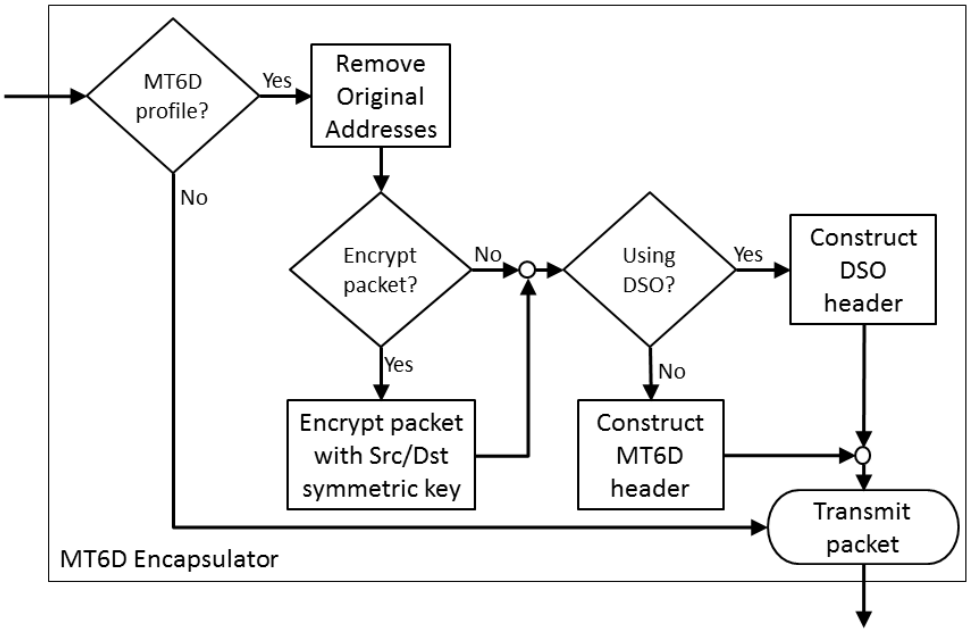


Figure 8.12: Encapsulator flow diagram

the next DSO host or its final destination. If the packet is at its destination host, it is 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 internal NIC.

8.9 Flow

Figure 8.14 illustrates the flow of a single MT6D host. Each MT6D host runs two main 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 threadsafe shared memory resource, which is referred to as the Shared Routing Table (SRT). When MT6D is initialized, it synchronizes its clock with a universal time server using the Network Time Protocol (NTP). NTP also synchronizes clocks at randomized poll intervals to minimize packet loss due to clock drift.

The process that generates and binds IPv6 addresses is referred to as the address obscurer. This is the process on the left side of Figure 8.14. 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

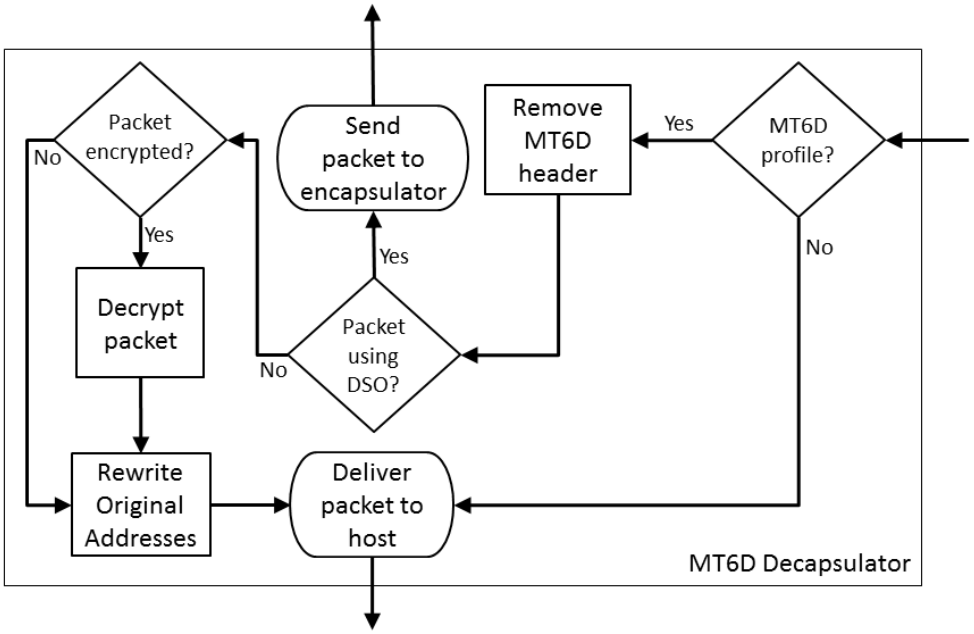


Figure 8.13: Decapsulator flow diagram

time increments for all sender/receiver profiles. The MT6D host will also bind the MT6D source addresses for each profile entry to its external NIC. After the SRT is initialized, the address obscurer will wait for the next time increment. When t_i increments, the current MT6D addresses are overwritten by the future MT6D addresses and new future MT6D addresses are calculated and stored in the SRT. The new future MT6D source addresses are again bound to the external NIC.

The listener process continually listens for packets. Each time a packet is detected, a subprocess, called a packet handler, is spawned. The packet handler first determines whether the packet is outbound or inbound. Outbound packets are sent to the encapsulator (see Figure 8.12) 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 sent to the decapsulator (see Figure 8.13) for processing and transmission. If the packet is using DSO, the packet is repackaged, redirected to the encapsulator, and retransmitted. Otherwise, the decapsulator fetches the original addresses and symmetric key from the SRT and delivers the packet to the host.

The SRT is a threadsafe memory resource, meaning that it includes safeguards to reduce the risk of shared access conflicts. It is shared by both the address obscurer and the listener. Its purpose is to provide a central storage location for IPv6 addresses and symmetric keys.

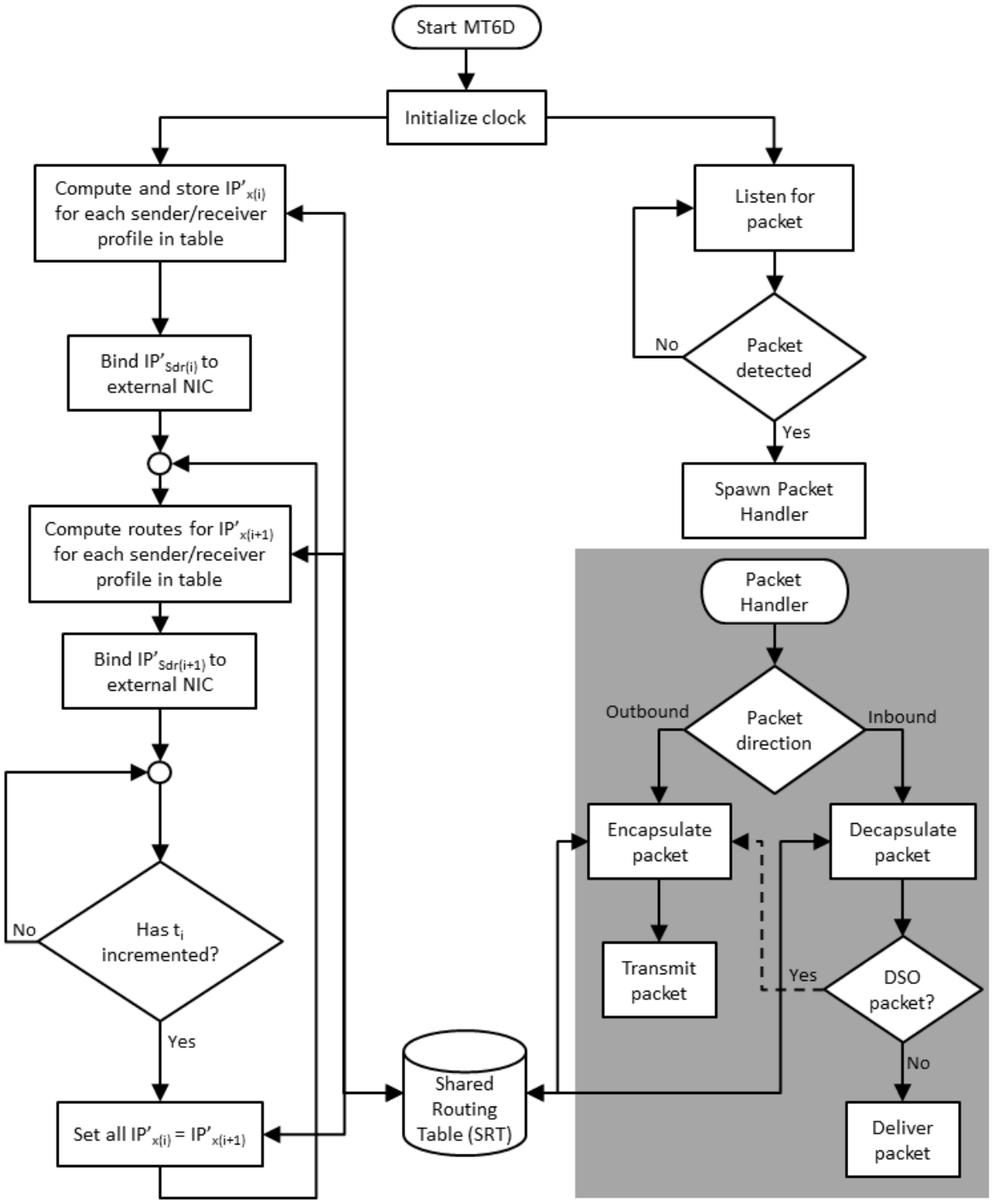


Figure 8.14: MT6D flow diagram

At all times, the SRT stores addresses for both t_i and t_{i+1}. Expired addresses are purged from the SRT.

8.10 Overhead

Encapsulating packets in MT6D incurs some additional bits of overhead. The overhead is a result of adding additional headers and removing fields from the original packet. In most cases, the cumulative overhead added to each packet is 16 bytes, as explained in the remainder of this paragraph. The new IPv6 header added by MT6D incurs 40 additional bytes. All MT6D traffic is sent via UDP in the current implementation. The UDP header adds 8 bytes for a total of 48 bytes. The Ethernet frame header adds the final 14 bytes for a total of 62 additional bytes of overhead. This 62 bytes of overhead is then reduced by removing unnecessary fields from the original IPv6 packet. The source and destination addresses in the original IPv6 packet header are removed completely during encapsulation and then added back in during decapsulation. This offsets the packet size by 32 bytes reducing the total overhead to 30 bytes. Another 14 bytes is eliminated by removing the Ethernet frame header from the original packet. A new Ethernet header is formed by MT6D after decapsulation. The final added overhead is 16 bytes.

Different options in MT6D will incur additional overhead. Encrypted tunnels in IPv6 incur eight additional bytes due to the destination options header described in Section 8.3.1. With encrypted packets, the total overhead increases to 24 bytes. The use of i-DKG also incurs additional overhead due to its destination option header. The amount of overhead incurred by i-DKG depends on the length of the symmetric key. For a 128-bit symmetric key, the added overhead is 32 bytes as illustrated in Figure 8.8. Regardless of whether encryption is used, the total overhead for an i-DKG packet exchanging a 128-bit symmetric key is 48 bytes. Since all destination options are combined in a single extension header, the encryption destination option replaces padding in the i-DKG destination option. Overhead incurred by i-DKG is only applied to the actual packet initiating i-DKG.

MT6D must advertise reduced MTUs that account for overhead to avoid ICMPv6 "Packet Too Big" messages. The maximum MTU allowed by Ethernet is 1500 bytes, excluding jumbo frames. When encryption is in use, MT6D advertises a maximum MTU of 1476 to the protected host. When i-DKG is in use, the advertised maximum MTU is 1452. In the event that an ICMPv6 "Packet Too Big" message is received from an intermediate node, the MT6D device decrements the MTU by the appropriate overhead value before forwarding it

to the protected host.

8.11 Control Message Handling

Control messages produced by ICMPv6 require special handling by MT6D. As described in Section 8.3, MT6D creates a tunnel between communicating hosts. The majority of control messages are generated by intermediate nodes (outside of the tunnel), such as routers. This means that control messages sent back to the sender will be sent from the IPv6 address of the router. The sender's MT6D device will not have a valid profile (described in Section 8.8) in the SRT that corresponds to the address pair of the control message packet. It is, therefore, necessary for the sender's MT6D device to reform control message packets for delivery to the sender. Since error messages have a different format than NDP messages, they are handled differently.

8.11.1 Neighbor Discovery Messages

Neighbor discovery messages are typically received from unprotected devices such as routers, other hosts on the network, or MT6D devices themselves. These messages are usually sent to multicast groups. Since multicast groups are not host-specific, they are able to be passed straight through to the protected host. The protected host itself will send out messages that do require manipulation. For example, router solicitations are sent to determine the nearest gateway, the subnet prefix to use with SLAAC, and the correct MTU that the gateway can handle. For router solicitations originating from the protected host, the MT6D device spoofs the source address of the packet with the current obscured address and forwards it. The router advertisements replying to the solicitations will arrive as multicast packets and are passed through.

The protected host will also send out neighbor solicitations to add link-layer addresses to its neighbor cache or for Duplicate Address Detection (DAD). Neighbor solicitations will be dropped at the MT6D device. The MT6D device will then generate a neighbor advertisement using its own link-layer address. Neighbor advertisements generated by the MT6D device ignore the possibility of DAD. This does not negatively impact network operations since the likelihood of an address collision is very small. The probability of collision on an IPv6 subnet can be written as:

$$P_c = \frac{h}{2^{64}} \tag{8.4}$$

where P_c represents the probability of a collision and h represents the number of other active addresses on the subnet. Even with one million active addresses on the subnet, the probability of collision is only $5.4 \cdot 10^{-14}$. If address collisions became a concern, it is possible to redesign MT6D to pass neighbor solicitations and reply with specially crafted advertisements that indicate whether or not a duplicate address exists.

There are two types of neighbor advertisements; solicited and unsolicited. Solicited neighbor advertisements occur as a result of a neighbor solicitation. These should not be generated by the protected node since MT6D will not pass external solicitations to the protected host. The opposite is true as well. No external neighbor should reply to a neighbor solicitation from the protected host since these solicitations are dropped by the MT6D device. Unsolicited neighbor advertisements occur when a host is aware that its link-layer address has changed. Unsolicited neighbor advertisements from the protected host get dropped since external hosts should not know the protected host's real link-layer address. Unsolicited neighbor advertisements from external hosts also do not get passed because the protected host should not have entries in its neighbor cache that contain link-layer addresses other than the link-layer address of the MT6D device.

8.11.2 Error Messages

Error messages received by the MT6D device require different handling than NDP messages. It is important to distinguish error messages received by the MT6D device from other error messages. Error messages generated by the receiver require no special handling because they will be encapsulated by MT6D the same as any other packet type. Error messages received by the MT6D device had to have been generated by an intermediate device. In IPv6, there are a number of error messages that intermediate nodes generate. A common example specific to IPv6 is the "Packet Too Big" message. The "Packet Too Big" message is used in IPv6 to notify the sender that the packet exceeds the MTU of a node somewhere along the physical link.

The format of an ICMPv6 error message is similar to that of Internet Control Message Protocol (ICMP) error message in IPv4. The IPv6 header includes the destination address of the sender of the packet that generated the error and the source address of the node at which the error occurred. The payload of the packet uses the first 32 or more bits for the ICMPv6 header. The remaining bits are filled with as much of the packet that caused the error as will fit without the entire packet exceeding the minimum IPv6 MTU of 1280 bytes.

When an ICMPv6 error message is received by the MT6D device it requires restructuring by the MT6D device. The first step decapsulates the portion of the packet, contained in the payload, that caused the error. Decapsulation involves removing the MT6D header, decrypting the portion of the original packet contained in the error message, and injecting the original source and destination addresses back into the original header. The next step replaces the payload of the ICMPv6 error message with the remaining portion of the original packet. The destination address of the ICMPv6 packet must also be replaced with the source address of the actual sender. This is done since the destination address of the ICMPv6 error message is the sender's obscured address. The checksum in the ICMPv6 header is then recalculated, and the packet is delivered to the protected host. For a "Packet Too Big" error, the MTU field is also decremented by the MT6D overhead, described in Section 8.10, to ensure that the protected host sends subsequent packets with a MTU that accounts for MT6D overhead.

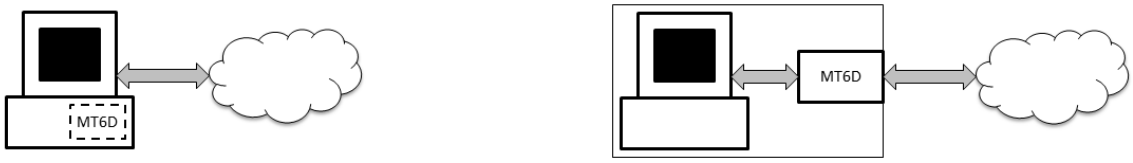
8.12 Configuration

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

8.12.1 Integrated Software

One possible configuration of MT6D is to integrate it onto the host device as illustrated in Figure 8.15(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 without the need for additional appliances. 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 settings is also easier. There is no need to transfer keys or preferences to a separate device.

The software design of MT6D also differs for each configuration. An understanding of how the Python global interpreter lock (GIL) operates is necessary to understand how the different configurations impact performance. In Python, parallel execution within the same process is not permitted. The GIL exists to ensure that only one thread at a time runs in the



(a) MT6D configured on the host device in software

(b) MT6D configured as a stand-alone device

Figure 8.15: MT6D configuration examples

interpreter [5]. When MT6D packets arrive at an actual NIC, the `libpcap` tool [79] is used to capture packets from the wire. `Libpcap` is seen by Python as a separate thread. Whenever `libpcap` captures a packet, the GIL locks other Python threads. Since each packet arriving at an actual NIC requires intervention by `libpcap`, there will be a significant increase in latency when MT6D is running. Other functions in MT6D are written as separate processes to avoid locking by the GIL.

When MT6D is integrated into the host, the internal NIC (depicted in Figure 8.11) is implemented virtually on the host. This is beneficial because it eliminates overhead caused by the Python GIL. Instead of requiring `libpcap` to capture outbound packets, a standard Linux TUN/TAP interface is used to create the virtual internal MT6D interface. The TUN/TAP interface acts as a network tap that receives every outbound packet and forwards it to the MT6D encapsulator described in Section 8.8. The TUN/TAP interface allows packet capture to operate in user space verses kernel space. With an actual NIC, a kernel-level packet capture is required. The internal NIC being virtual allows the use of a TUN/TAP interface. The TUN/TAP interface reduces latency because it can be run as a separate process eliminating thread locking by the GIL for outbound traffic. Inbound traffic destined for the external NIC still requires the use of `libpcap`. An optimized version of MT6D, written in a more efficient programming language, should not require the use of `libpcap`, as explained in Chapter 11, Section 11.4.

8.12.2 Stand-alone Device

As another option, MT6D can be configured on a stand-alone device, as shown in Figure 8.15(b). A stand-alone MT6D device is transparent to the user. All a user has to do to use the stand-alone MT6D is plug it in and connect network cables. This is especially useful if the user has devices running different operating systems. The stand-alone device configuration is also platform independent. Probably the biggest advantage of configuring MT6D

as a stand-alone device is that the computational complexity of using MT6D is offloaded to the MT6D device. This is particularly beneficial when MT6D is used to provide protection for computationally constrained devices, such as sensors.

The stand-alone configuration requires a different software design than when MT6D is integrated into the host. Both the internal and external NICs are hardware NICs since a stand-alone implementation requires a separate device that connects to both the host and the network. This means that both NICs have to be set in promiscuous mode to capture packets off the wire. As a result, `libpcap` is required on both the internal and external NICs. The use of `libpcap` on both interfaces, will result in degraded performance as compared to MT6D integrated into the host machine. It should be mentioned that the `libpcap` requirement is a result of the current software design. It is entirely possible for an optimized version of MT6D to eliminate the use of `libpcap` entirely. This enhancement would require a kernel-level implementation either in software or hardware.

MT6D can also be used as a border device in a trusted environment in this configuration. Configuring MT6D in this fashion mimics a Virtual Private Network (VPN). The main difference, which is an improvement over traditional VPNs, is that addresses sent over the external network are constantly rotating. There are two benefits to configuring MT6D in this fashion over the already stated benefits of a stand-alone configuration. 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 configuration.

8.13 Summary

The design of MT6D was described in this chapter. First, the method used for defining dynamic addresses formation was presented . An explanation of the tunneling techniques used by MT6D followed. The next section described how varying address rotation times are achieved. The method MT6D uses to dynamically generate new symmetric keys was also discussed. MT6D primarily obscures IPv6 IIDs, but it does include a mechanism to dynamically obscure subnets as well; this mechanism was explained. The architecture and flow of MT6D ensued. The chapter included a discussion on overhead incurred by using MT6D as well as an explanation on how MT6D handles various control messages. The chapter ended with a description of the different MT6D configuration options.

Chapter 9

Implementation Results on Live IPv6 Networks

A proof of concept prototype software implementation of MT6D was developed to validate the MT6D design. The prototype operates in both integrated and stand-alone modes (see Chapter 8, Section 8.12). The MT6D 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. The first set of tests was conducted using MT6D as integrated software on the communicating devices. The second set of tests was conducted using MT6D as a stand-alone device between each communicating host and the network.

All live network testing was conducted on the Virginia Tech production IPv6 network. The live network, described in Chapter 3, Section 3.1.3, 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. While some performance data is provided by these tests, MT6D is not at all optimized in its current prototype version. Performance testing on an optimized system is left to be reported by future MT6D researchers. The performance measures collected here are all relative measures that can be used validate functionality and to guide the development of an optimized MT6D. In this functional testing, a static address rotation interval was established to maintain consistency. One thousand ping packets were sent from the client to the server at a rate of one packet per second to measure basic/minimal

functionality. 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 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 test settings were an address rotation interval of 10 seconds, AES encryption on all MT6D-tunneled traffic, and i-DKG turned off.

9.1 Integrated Software Testing

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.

Each packet gets encapsulated by the sending host before being transmitted. The speed at which this occurs is greatly dependent on the computing power of the sending host. It is likely that the performance results reported in this chapter will vary with MT6D installed on machines with different capabilities. Establishing functionality, however, not speed is the goal of the MT6D prototype. Performance results are provided to demonstrate, first, that MT6D can reasonably pass live network traffic and, second, how different features of MT6D impact throughput and packet loss.

MT6D requires two NICs as illustrated in Figure 8.11. 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 9.1. 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. All tests were started between 1700 hours and 1800 hours local time since the network load fluctuates throughout the day. From historical data, the network typically experiences a consistently high load over a multi-hour span during the chosen time period. It was not preferred to test MT6D with low network loads since the goal of live network testing was to determine how MT6D performs under live network conditions.

9.1.1 Same Subnet Testing

The test scenario described in Section 9.1 was conducted with two MT6D hosts connected to the same subnet. When two hosts are connected to the same subnet, direct delivery of packets can occur without router involvement. This is due to the way neighbor discovery functions. In IPv6 intra-subnet communications, a host wishing to communicate with another host first obtains prefix information from local router advertisements to determine if the destination host is on the same subnet. A neighbor solicitation message is sent to the destination if the sender does not have the destination host stored in its destination cache. The destination host replies with a neighbor advertisement which the source adds to its destination cache for future communications [58]. 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.

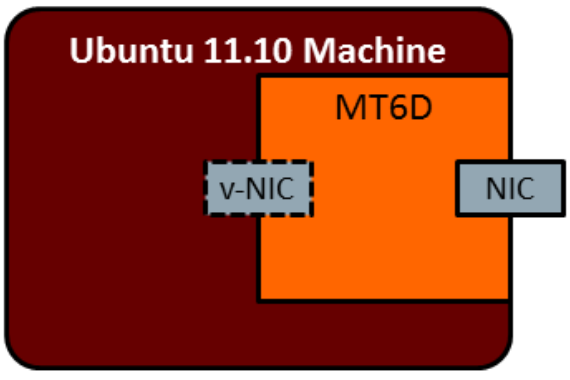


Figure 9.1: Diagram of MT6D integrated into the test machine

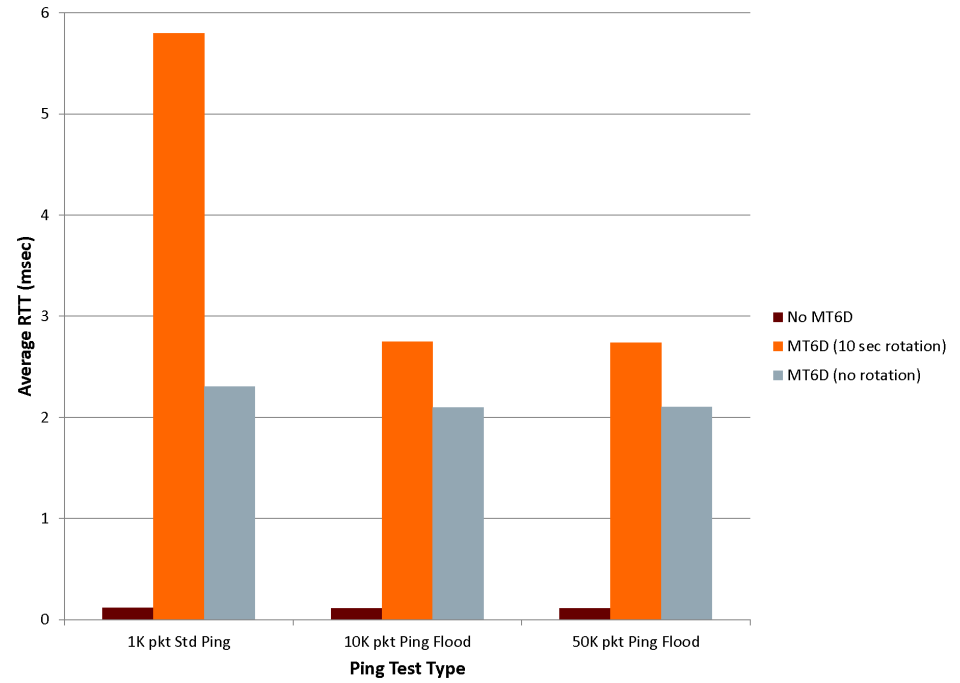
Tests on the same subnet were conducted to establish performance baselines independent of the effects of intermediate routers. Under these conditions, MT6D was tested with different settings to gain an understanding of how each setting impacts performance. The different settings were compared against the default settings established in Section 9.1.

Benchmarking Default Test Settings

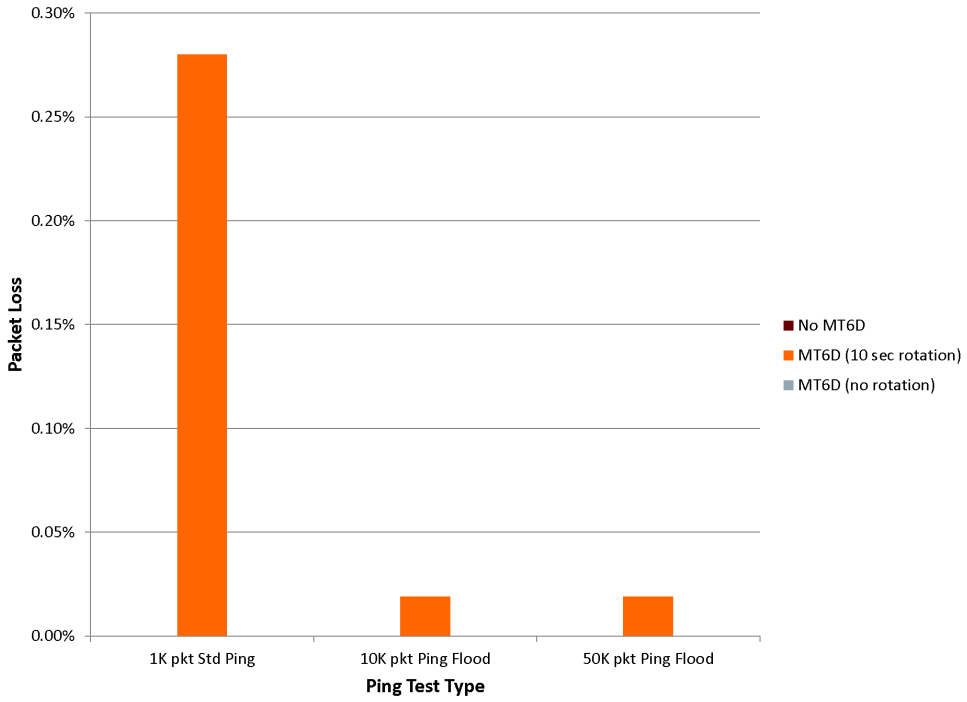
The first set of tests was aimed at determining the performance impact the MT6D prototype has on network traffic. The test scenario was run on the network without MT6D running to measure this. No routers were involved with the exchange of test traffic since both hosts were located on the same subnet. 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 9.2(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 9.2(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 9.2(a), the average RTT of a 1,000 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 do the standard ping tests. 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 [57]. 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



(a) Average RTT



(b) Average packet loss

Figure 9.2: 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

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 9.2(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 9.2(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 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 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 9.2(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 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:

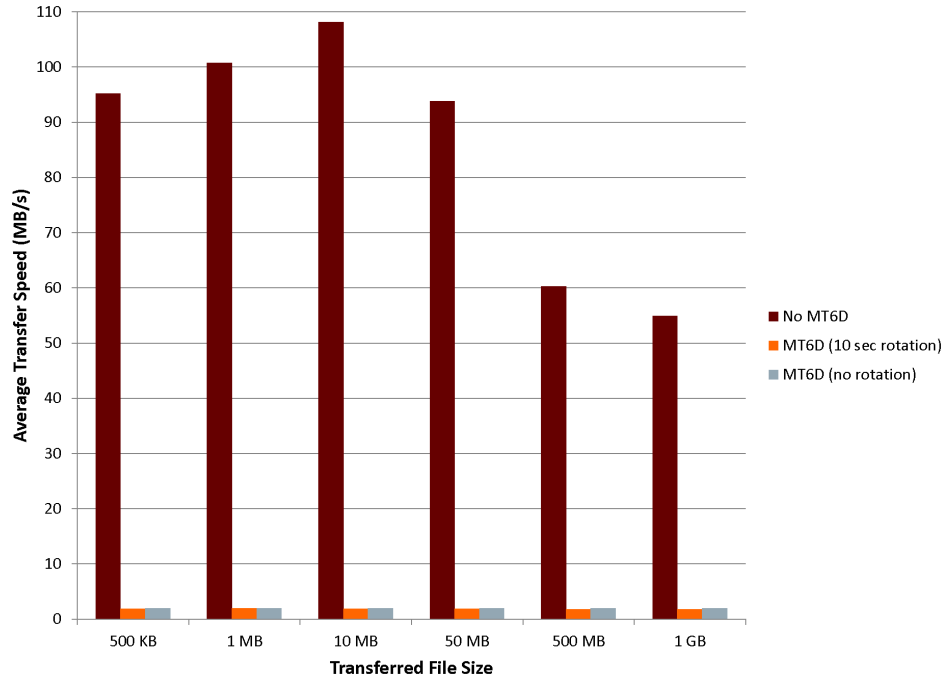
$$P_c = \frac{h}{2^{64}}$$

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. Even if the entire IPv4 address space worth of hosts (nearly 4.3 billion hosts) was located within a single IPv6 subnet, the probability of an address collision would only be $2.3 \cdot 10^{-10}$.

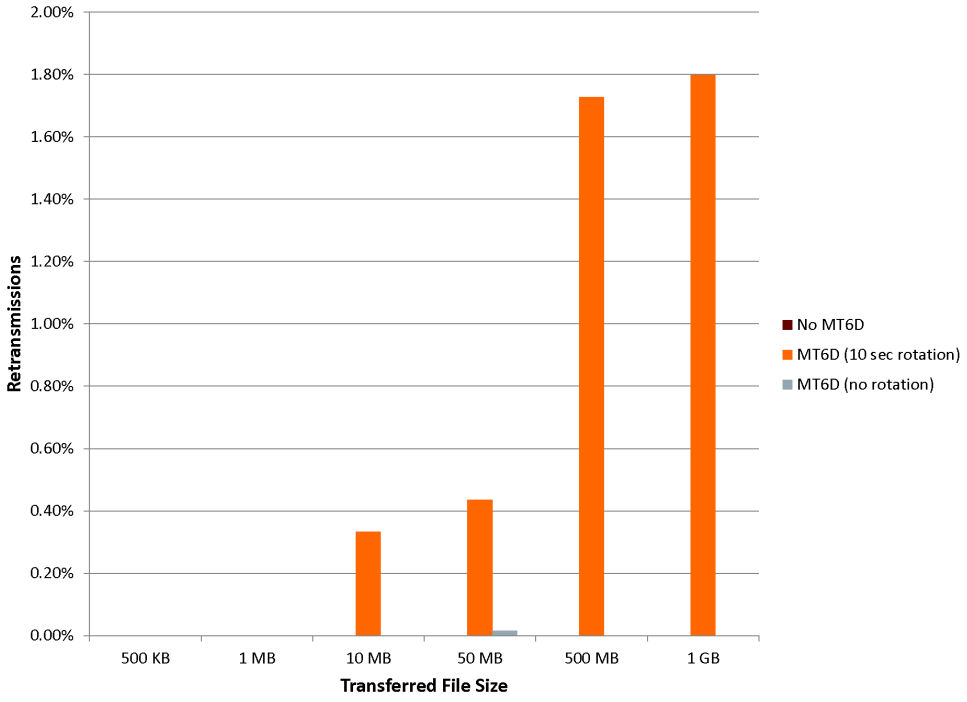
Connection-oriented tests were also conducted to verify MT6D's capability of maintaining negotiated sessions while changing addresses. Additionally, these tests validated the successful retransmission of dropped packets. Without MT6D running, the average transfer speed was between 54-108 MB/s as depicted in Figure 9.3(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 reinforced 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 9.3(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 9.4 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. Time, on the x-axis, is the number of seconds since the capture was started. The y-axis is the number of packets per second. Note that the number of retransmissions spikes every 10 seconds in conjunction with an address rotation. Between rotations, there are no retransmissions.

It was mentioned that packet processing in software limits the throughput of MT6D. Aside from `libpcap` (discussed in Chapter 8, Section 8.12.1), another significant impact on



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.3: 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

performance in the Python prototype is caused by regularly polling to determine when to rotate addresses. The code was modified to sleep between address rotations in an attempt to improve performance. This modification did improve performance, but only by approximately 100 KB/s. The modification was not adopted, but is mentioned here because it resulted in a important side-effect that significantly increased packet loss. This side-effect caused entire rotation intervals to be lost. Clocks can be loosely synchronized when MT6D systems are regularly polling for the proper rotation interval. One host may rotate addresses before the other host in the event of a small amount of clock drift. Although this will result in packet loss, the loss is minimal. When the MT6D systems sleep between intervals, they wake up only to rotate addresses and then go back to sleep. If the two systems' clocks are not tightly synchronized, it is possible for them to occasionally wake up on different clock ticks. The two systems will bind different addresses when this happens and be unable to communicate until they wake up for the next address rotation. Figure 9.5 shows the results of both the client and server simultaneous initiating ping floods with each other. The black line represents ICMPv6 echo requests while the red line represents ICMPv6 echo replies. The graph illustrates a dropped address rotation interval at approximately 565 seconds after the packet capture was started. The packet capture was taken from the client's v-NIC illustrated in Figure 9.1. During the loss intervals, the ICMPv6 echo request line dips. This reflects the outbound echo requests still leaving the client, but none received from the server. Inbound requests are not received by the external hardware NIC since the server computed a different address for the client. The ICMPv6 echo reply line drops to zero because neither

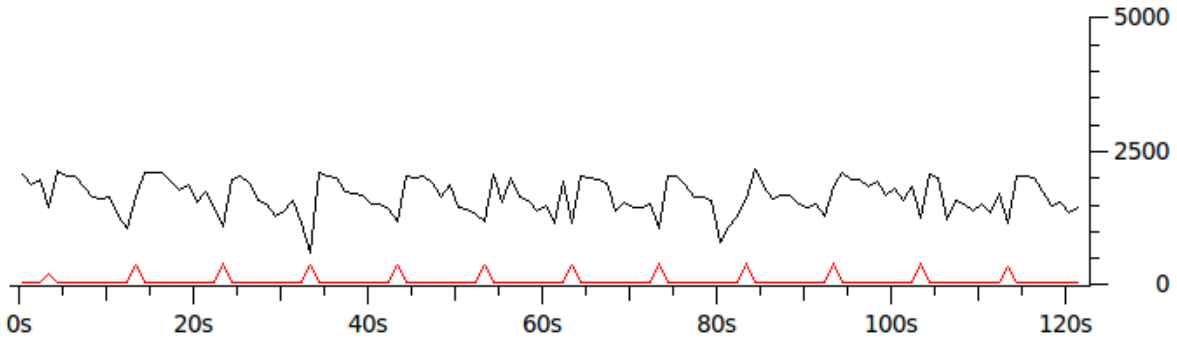


Figure 9.4: Graph of packet loss during address rotations. The x-axis counts the number of seconds since the packet capture was started while the y-axis counts the number of packets per second. The black line represents TCP traffic. The red line shows the number of TCP retransmissions.

host received the echo requests required to generate a reply. The other small dips and spikes in the graph represent the latency incurred during an address change followed by a surge of packets as the MT6D devices empty out their queues. These dips and spikes should be able to be minimized by optimizing MT6D.

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 9.6(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

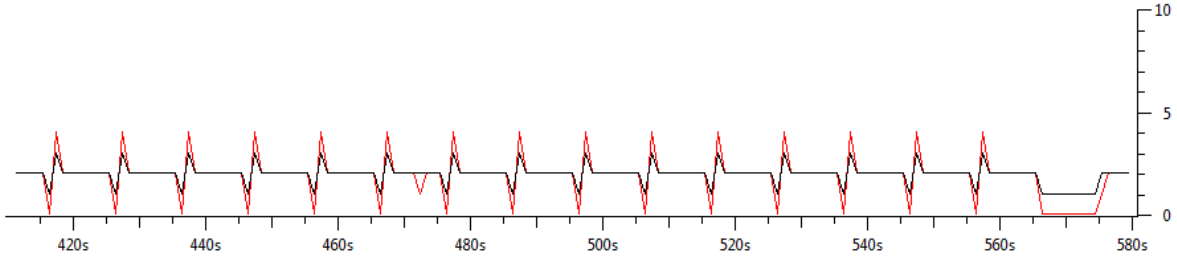


Figure 9.5: Graph of a lost rotation interval due to sleeping error. The x-axis counts the number of seconds since the packet capture was started while the y-axis counts the number of packets per second. The black line represents ICMPv6 echo requests. The red line represents ICMPv6 echo replies.

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 9.6(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.

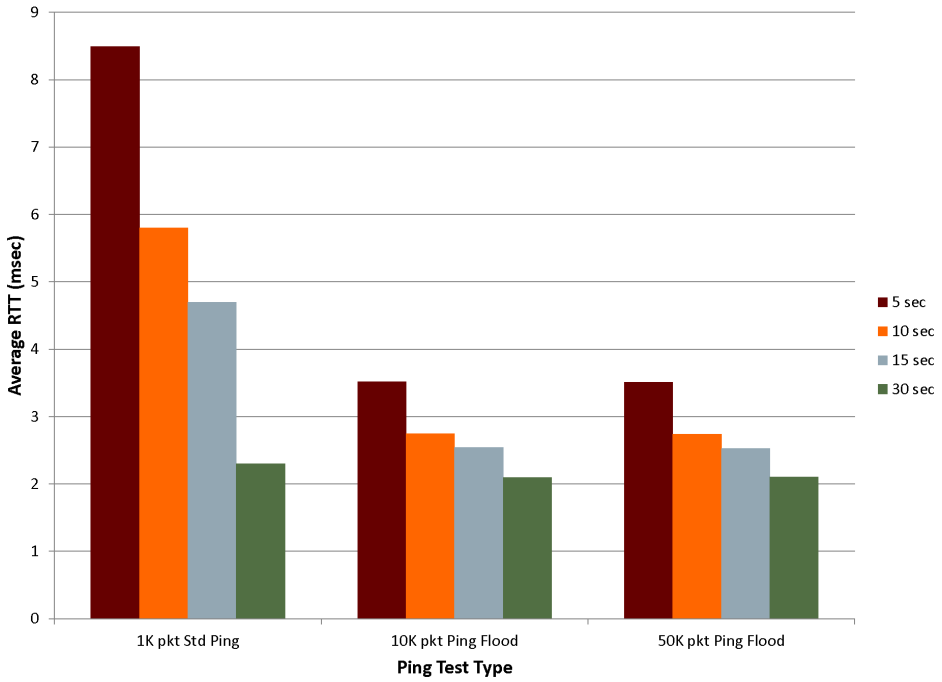
In contrast, throughput increased and packet loss decreased when address rotation intervals were greater than 10 seconds. Using the 30 second address rotation interval as an example, the average RTT decreased with similar magnitude between a 10 second rotation interval and a 30 second rotation interval 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 9.7(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.

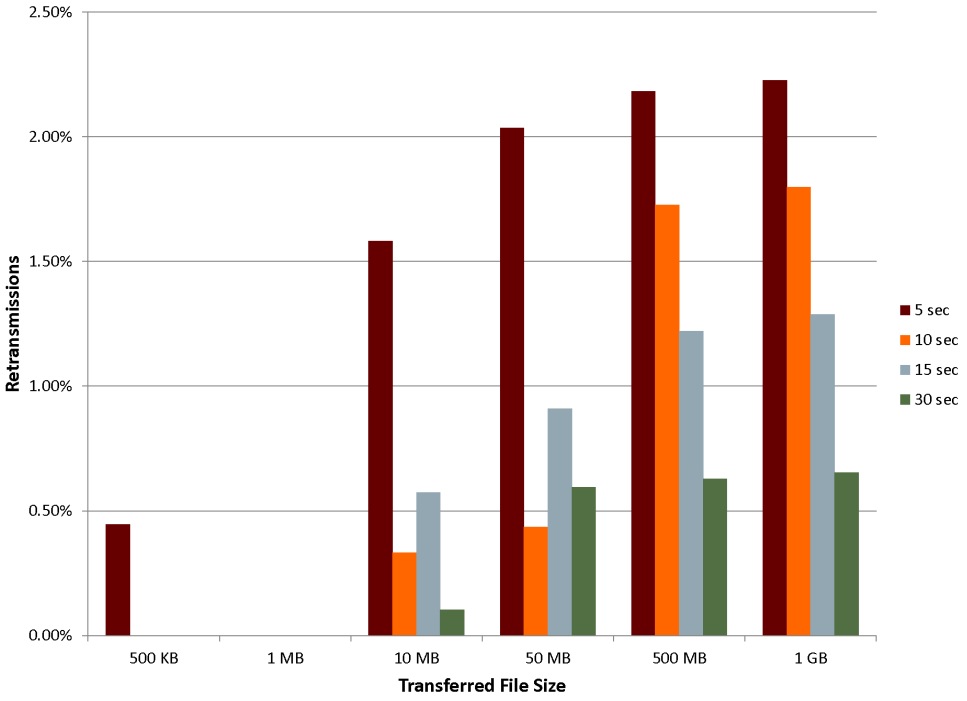
Connection-oriented traffic, as with connectionless traffic, had higher packet loss with smaller address rotation intervals. Figure 9.7(b) illustrates the relationship of packet loss to address rotation interval for the different sized file transfers. File transfers below 10 MB again experienced little 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.

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 as described in Chapter 8, Section 8.3. The goal of these tests was to determine how much of a performance impact is caused by

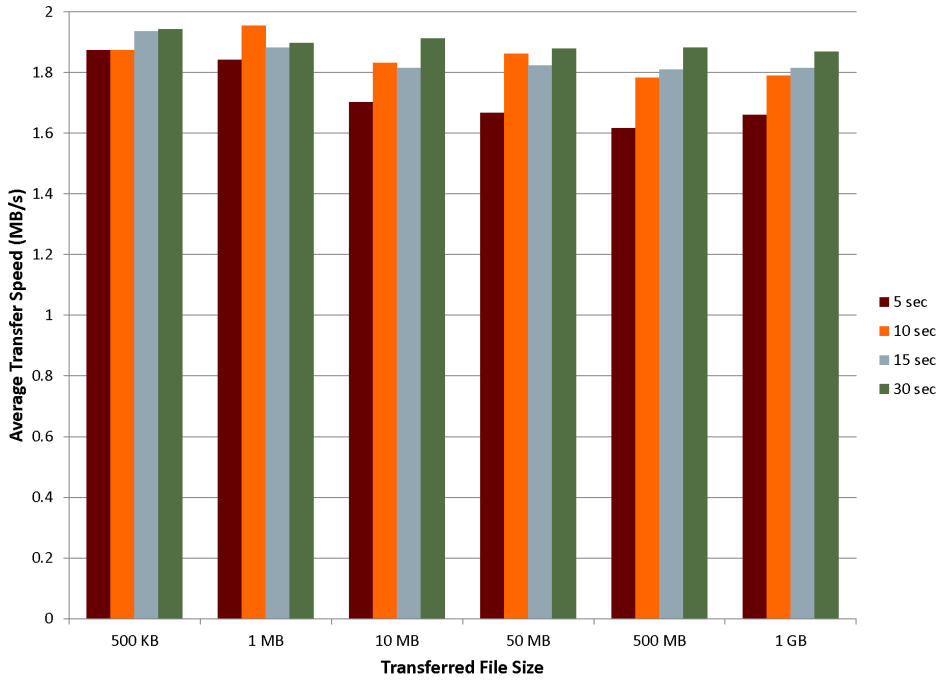


(a) Average RTT

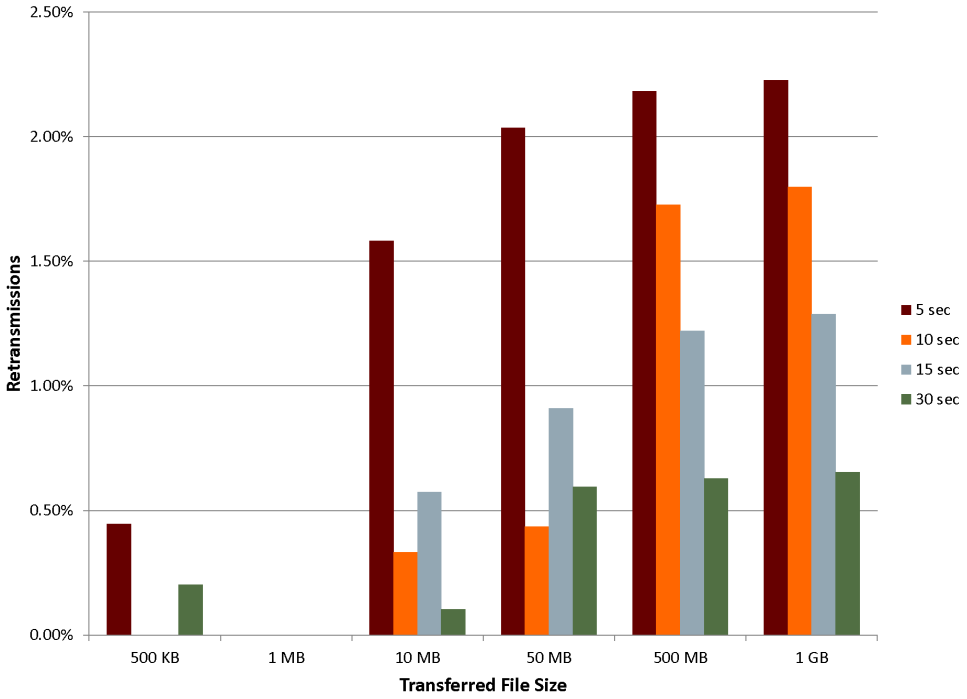


(b) Average packet loss

Figure 9.6: Average speed and packet loss of connectionless traffic using different rotation intervals



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.7: Average transfer speed and packet loss of connection-oriented traffic using different rotation intervals

encryption. Packets in encrypted tunnel mode are encrypted with 128-bit AES encryption using 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. Both modes also had i-DKG disabled.

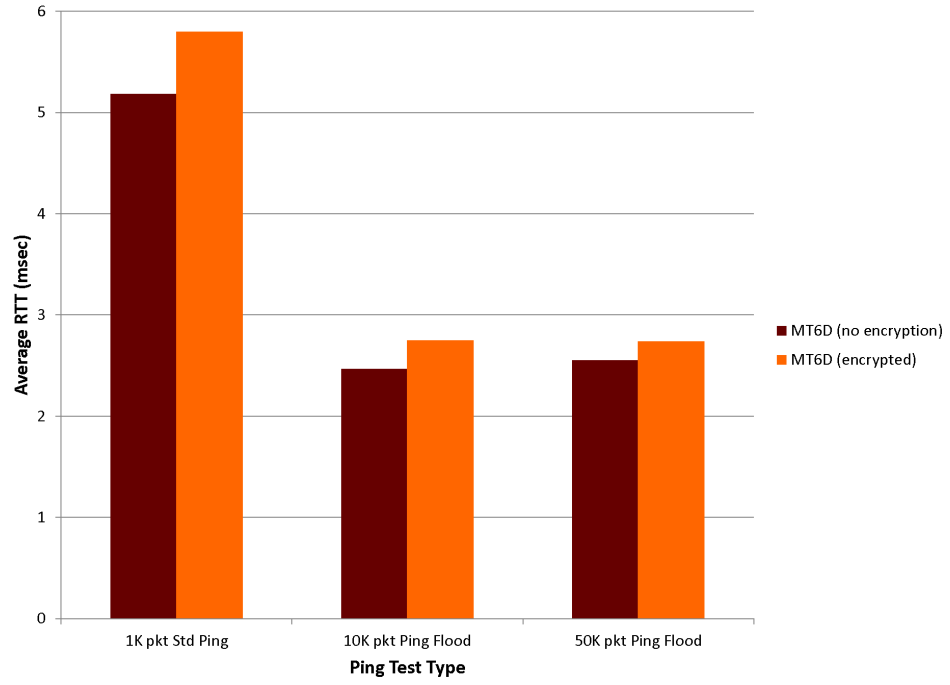
Connectionless traffic tests illustrate that there is not a significant performance impact because of the encryption of packets in MT6D (see Figure 9.8(a)). With the 1,000-packet standard ping test, encryption added 0.61 msec to the average RTT. The ping flood tests incurred approximately 0.2 msec RTT increase due to encryption. As demonstrated in Figure 9.8(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 9.9(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 an increase in packet loss for encrypted tunnel mode as shown in Figure 9.9(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.

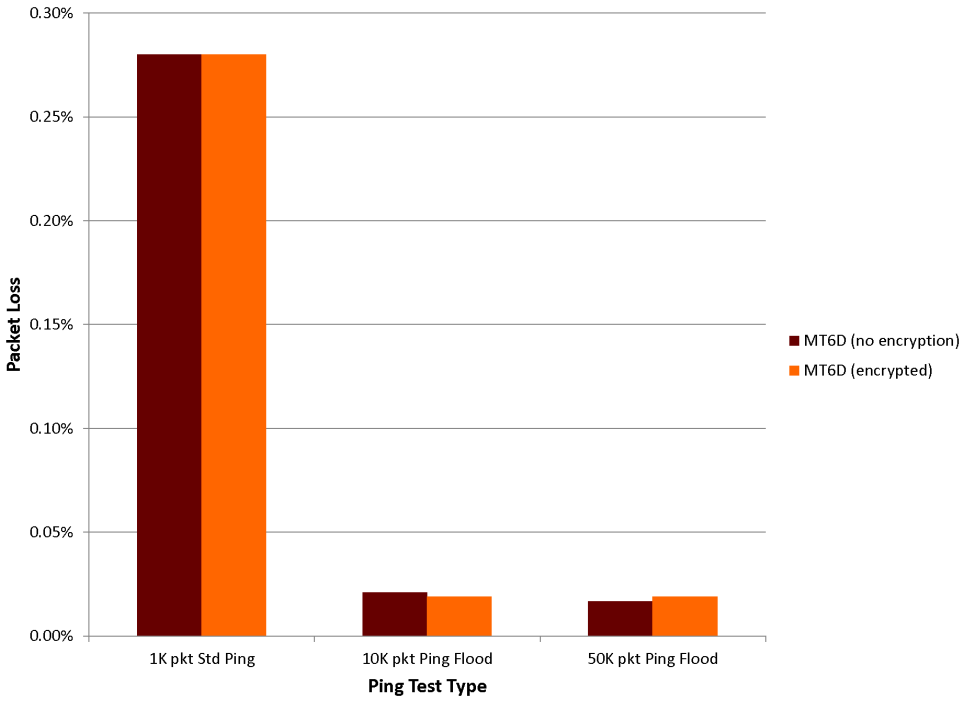
Evaluating the Impact of using i-DKG

A series of tests were conducted using i-DKG with a key generation rate of 500 seconds to evaluate the impact i-DKG has on performance. This frequent rate was chosen purely to force key regeneration during testing. It is unlikely that a user would want key generation to occur this frequently. In most cases, key generation would occur after a number of hours or even days. A key generation rate of 500 seconds also ensures that key generation occurs between one and two times on longer test runs. Since key generation will impact performance, forcing it to occur more often artificially degrades performance to the point that the results are no longer meaningful. A key generation rate of 500 seconds also allows many tests to complete without key generation occurring. These tests are useful in demonstrating any added latency or packet loss due to i-DKG running when no key generation is actually occurring.

The results of connectionless traffic tests using i-DKG are illustrated in Figure 9.10. Each

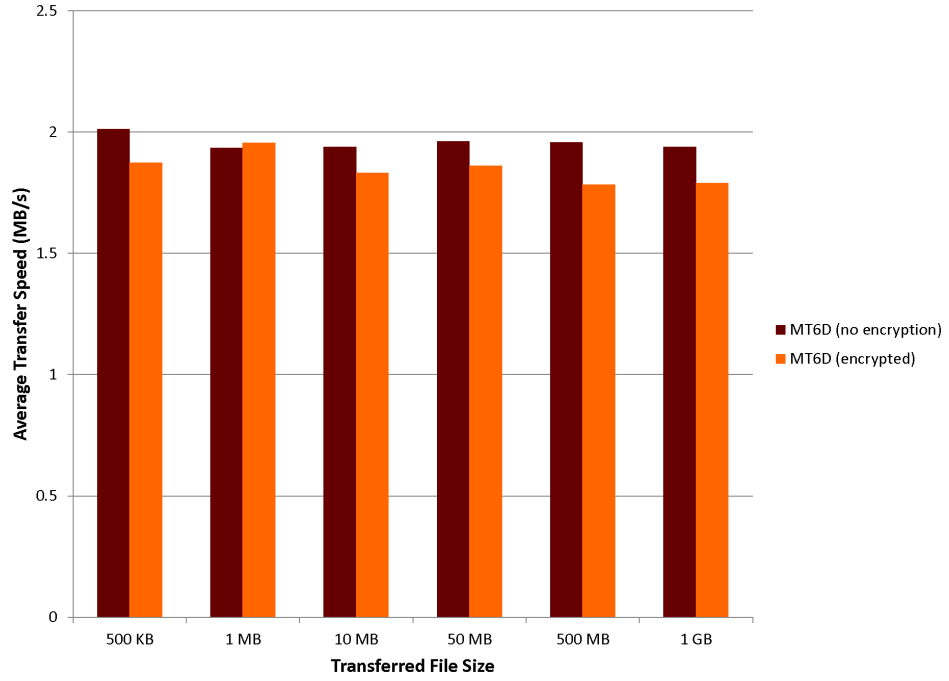


(a) Average RTT

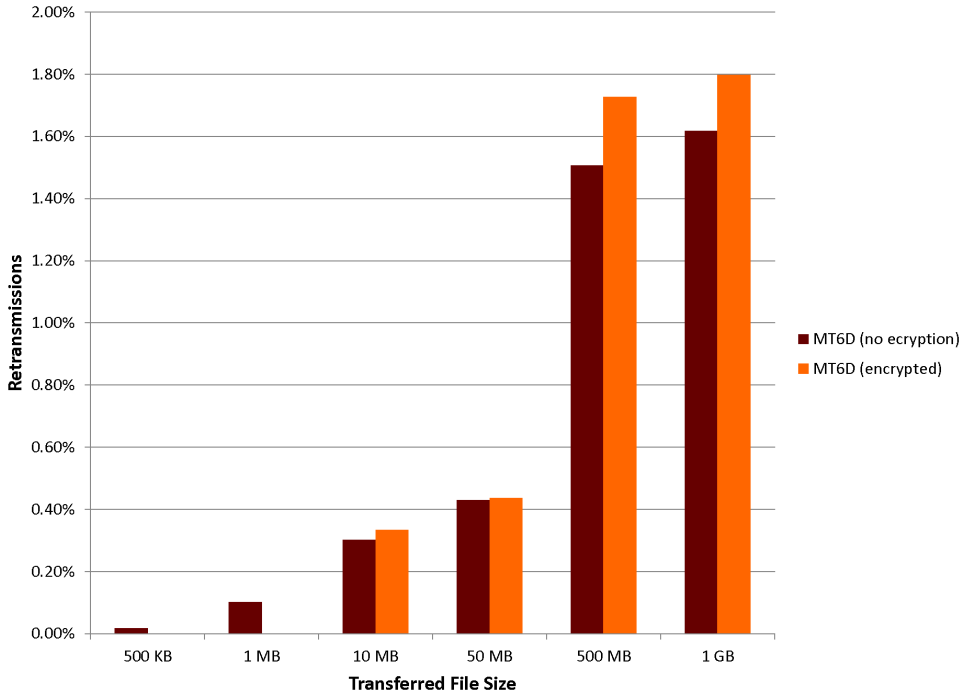


(b) Average packet loss

Figure 9.8: 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 9.9: Average transfer speed and packet loss of connection-oriented traffic using encrypted tunnel mode verses unencrypted tunnel mode

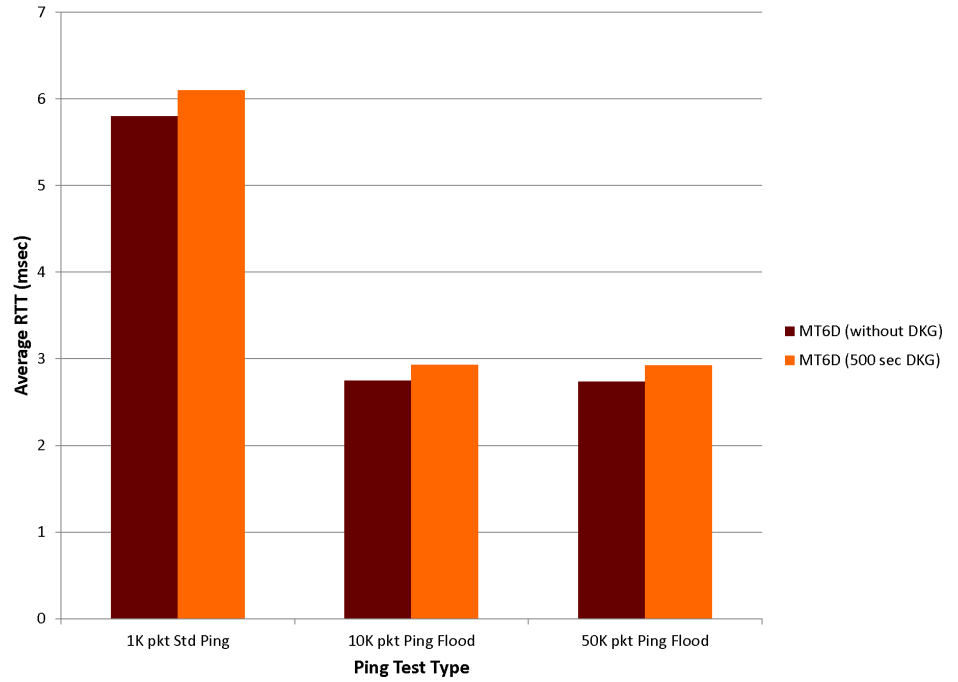
1,000-packet standard ping iteration ran for 1,000 seconds. A 500-second key generation rate ensured that new keys were generated on average two times per iteration. Figure 9.10(a) shows an average RTT increase of approximately 0.3 msec. This small amount of added latency is negligible and could be a result of additional processing while the sender adds the i-DKG header and the receiver processes it. These results make sense since the i-DKG header can fit inside the ICMPv6 echo request and reply packets without reducing the MTU of the original unencapsulated packets. Figure 9.10(b) shows an increase in packet loss of approximately 0.11% when i-DKG is active. This added loss accounts for approximately one packet on average. Since addresses rotate every 10 packets during the standard ping tests, this result demonstrates that key generation does not result is a temporary key mismatch. A key mismatch would cause at least a 10-packet loss. The added loss does demonstrate that i-DKG does not always cause packet loss. If it did, there would be on average at least an additional two packets lost per test.

The added latency of using i-DKG was almost the same for the ping flood tests.

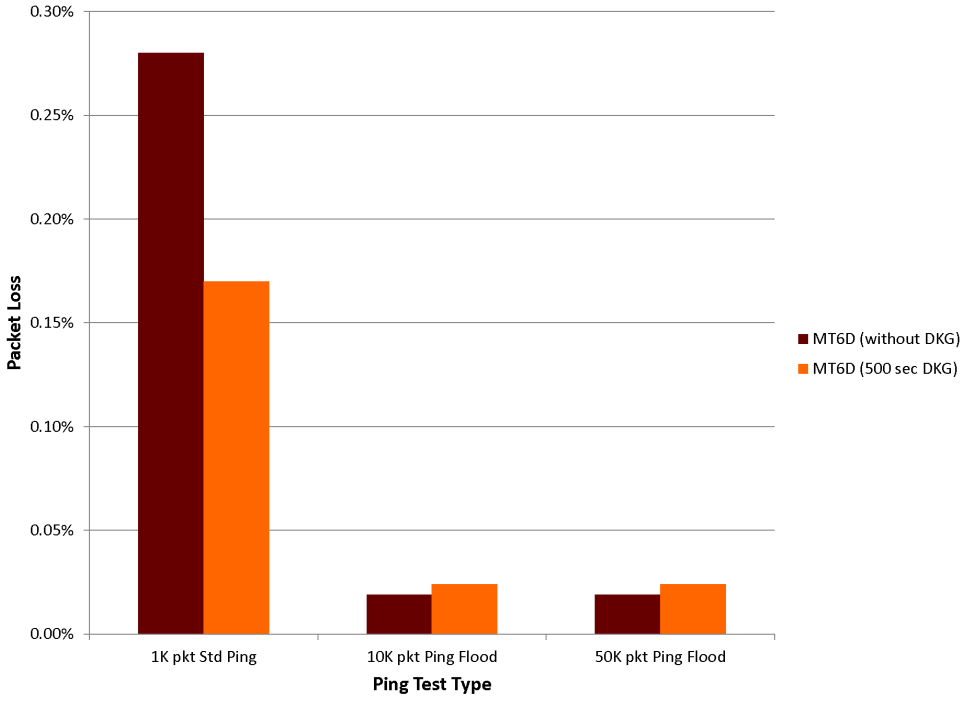
Ping flood tests completed fast enough that key generation did not occur. The difference between average RTTs, as illustrated in Figure 9.10(a), is negligible (less than 0.2 msec). If the difference in latency were caused by i-DKG, tests using i-DKG would experience an increase in packet loss as well. As demonstrated in Figure 9.10(b), packet loss when i-DKG is active is actually lower. This confirms that i-DKG does not cause additional latency when not actively changing keys.

Connection-oriented traffic further demonstrates that i-DKG does not impact performance except when actively generating new keys. File transfers less than 1 GB executed within the span of a single key. For all of these file transfers there was negligible performance difference between iterations without i-DKG and those with i-DKG as illustrated in Figure 9.11. File transfers of 1GB took, on average, 618 seconds, resulting in one key change. The decrease in throughput caused by a single key change during the 1GB file transfers was approximately 13.6 KB/sec (see Figure 9.11(a)). This accounts for a having to slightly reduce the packet payload size due to the inclusion of the i-DKG destination options header. The average packet loss, as shown in Figure 9.11(b), was only 0.02% higher. It was expected that the difference in packet loss would be negligible due to the high volume of packets transferred per second.

Another test was conducted to determine the robustness of i-DKG over an extended period. Initial testing led to the discovery of a flaw that occurs only when both hosts simultaneously send a key. Two keys sent simultaneously both get accepted and result in a

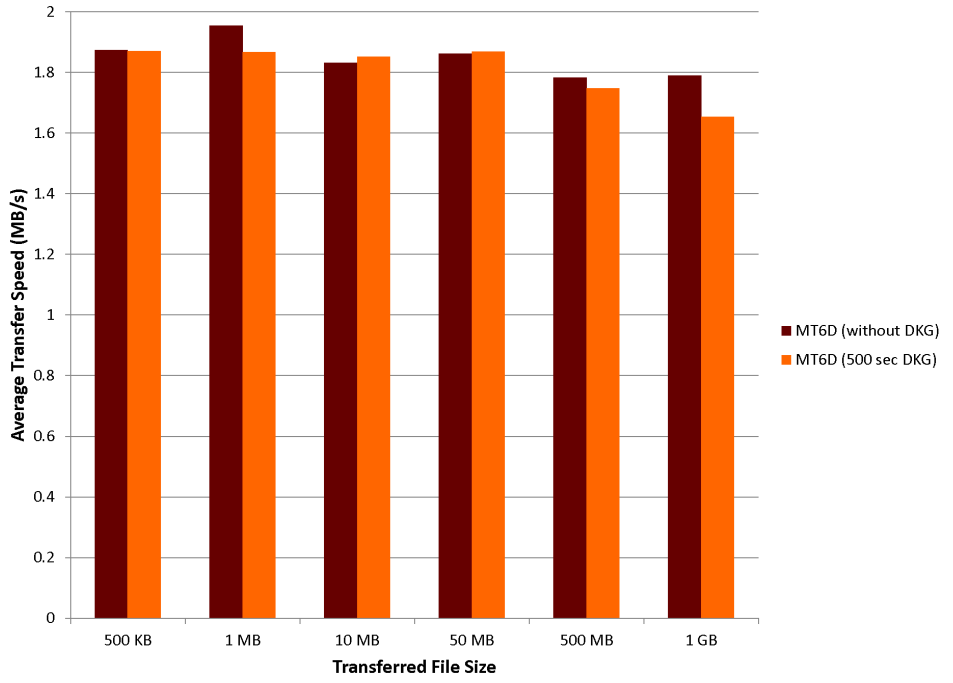


(a) Average RTT

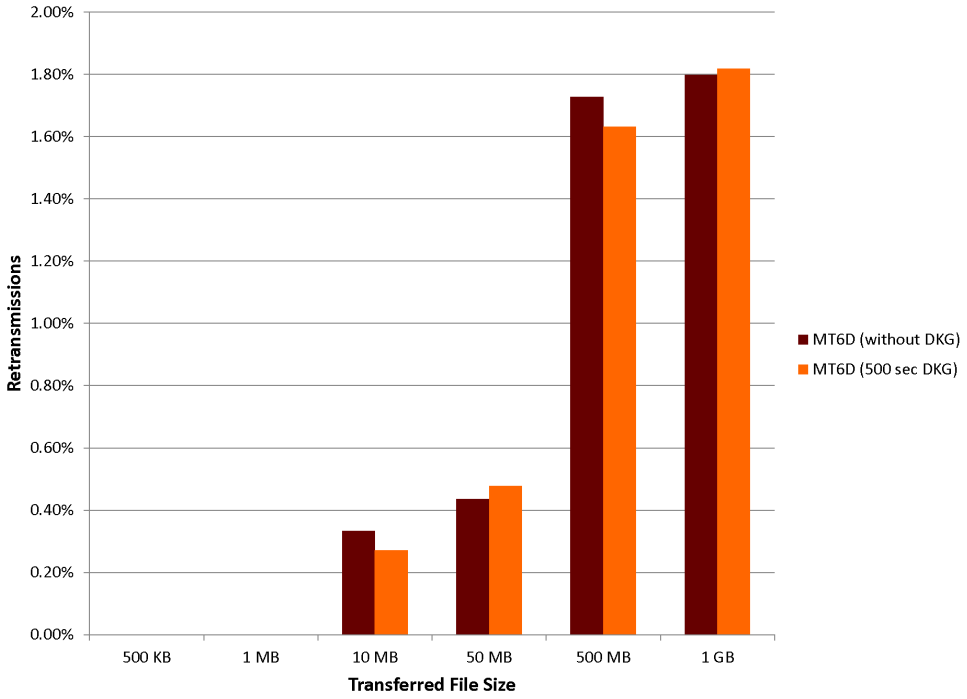


(b) Average packet loss

Figure 9.10: Average speed and packet loss of connectionless traffic for MT6D running i-DKG at a 500-second key rotation interval



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.11: Average transfer speed and packet loss of connection-oriented traffic for MT6D running i-DKG at a 500-second key rotation interval

key mismatch. This issue was eliminated by having both hosts adopt the key that is larger numerically. After addressing the flaw, MT6D using i-DKG was left running over a week span. MT6D was set to rotate addresses every 10 seconds during this test while i-DKG generated new keys every 500 seconds. A standard ping between the client and server was also running to generate traffic. Both the client and server were able to successfully generate new keys every 500 seconds during the test period without losing the MT6D connection. To quantify these results, each host rotated through over 60,000 addresses and changed keys over 1200 times.

9.1.2 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 configuration as those used in benchmarking default test settings in Section 9.1.1: testing across subnets 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 9.2 and 9.3 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 discussed in Section 9.1.1, 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 [58] 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 1,000 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 Section 9.1.1 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 9.2(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 9.14(b) shows a packet loss of 10.16% for two hosts located on different subnets. This 10% increase in packet loss corresponds with 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 9.12:

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.

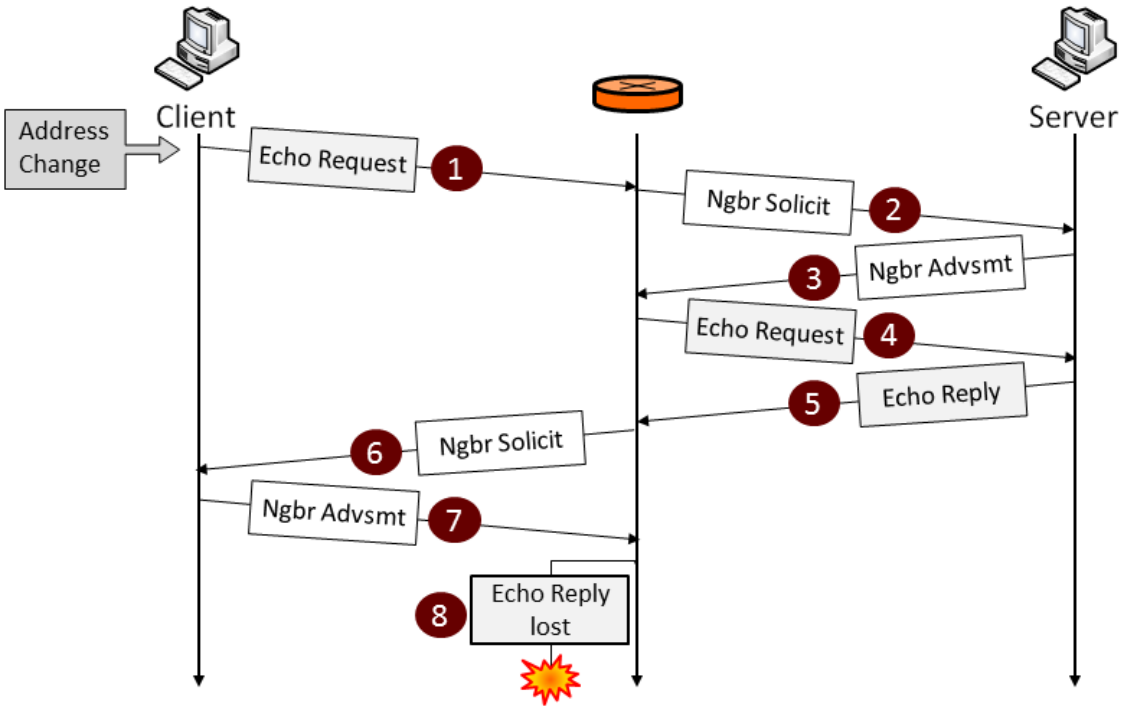


Figure 9.12: Ping flow after an address rotation. The first ping reply after an address rotation is erroneously dropped by the router.

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 9.12 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 9.13. 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 9.14(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

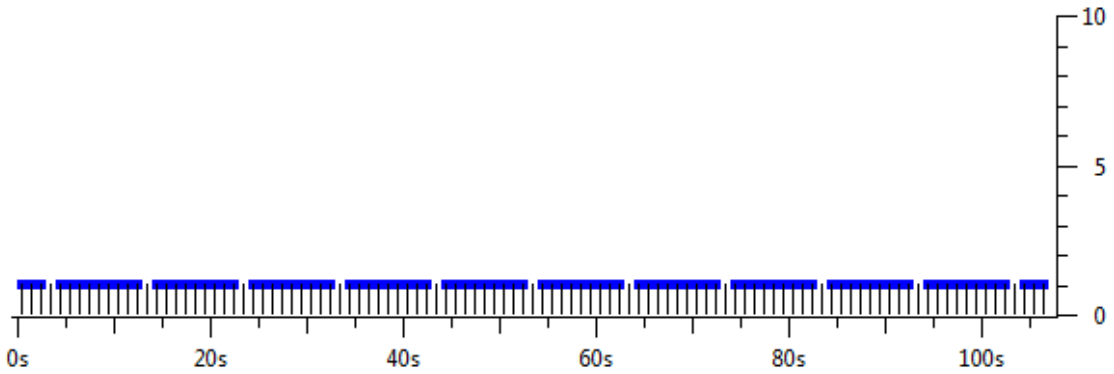
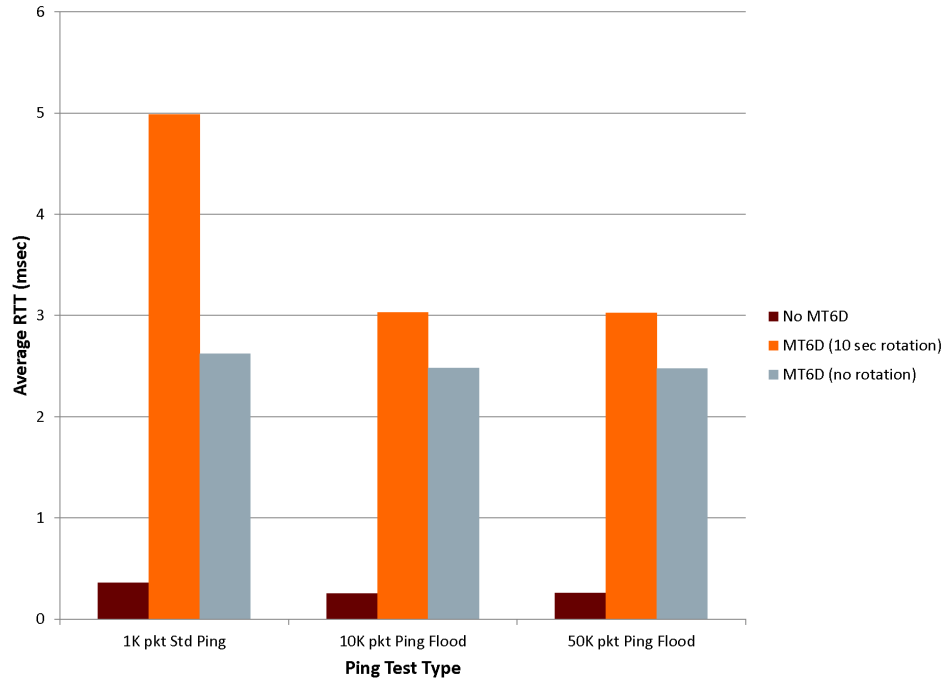
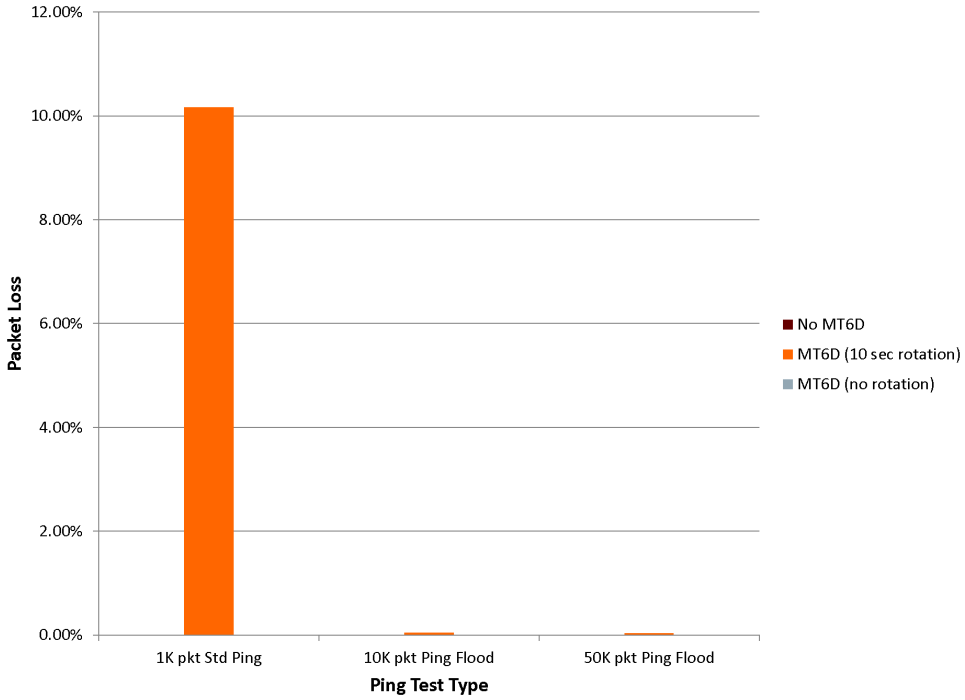


Figure 9.13: Dropped echo replies from the IPv6 router to the client. The black vertical lines each represent a sent echo request. The blue horizontal line represents echo replies received. The x-axis shows the time, in seconds, since the packet capture was started while the y-axis shows the number of packets sent per second.



(a) Average RTT



(b) Average packet loss

Figure 9.14: 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.

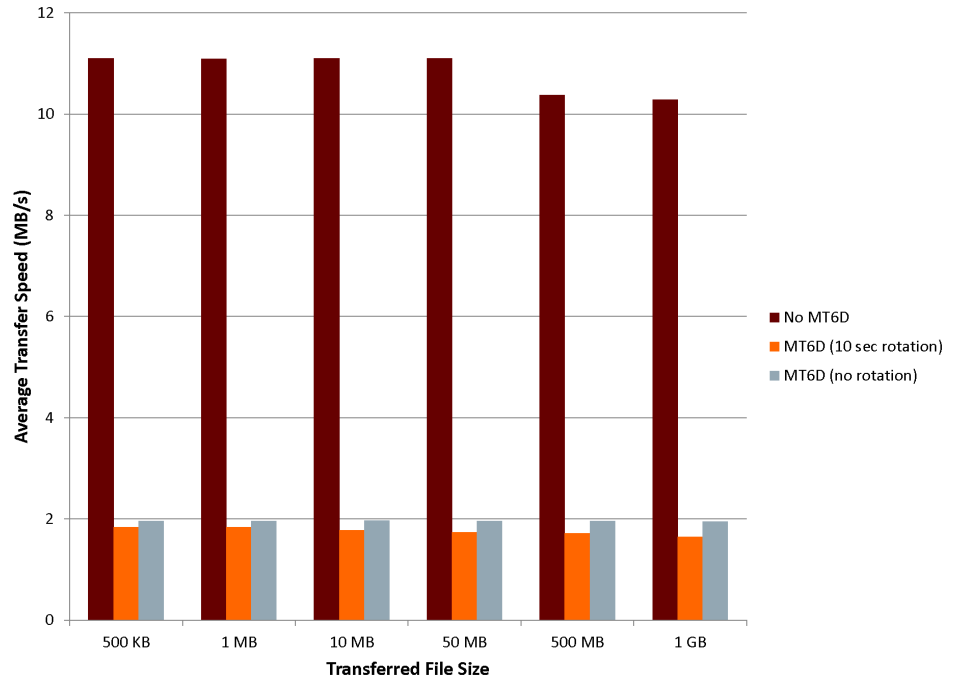
average standard deviation of 17.853 msec. As discussed in Section 9.1.1, the high standard deviation was a result of the hosts queuing packets after an address change while neighbor discovery occurred. Since the router is dropping the first ping after an address change, the average RTT is reduced to 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 9.2(b) to Figure 9.14(b) demonstrates a less than 0.02% difference. A comparison of average RTTs (see Figures 9.2(a) and 9.14(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.

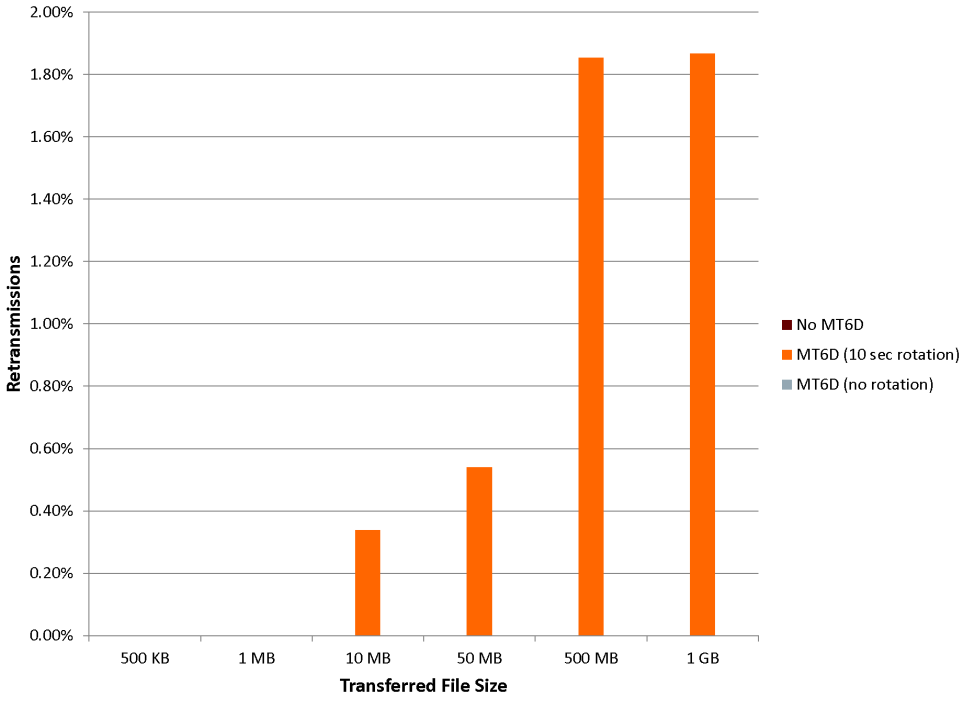
Connection-oriented test results provide further evidence that rotating addresses when crossing subnets does not significantly impact performance as compared to operating MT6D between hosts on the same subnet. As with the ping flood tests, the average transfer speed for each of the different file sizes was nearly the same (see Figures 9.3(a) and 9.15(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 9.3(b) and 9.15(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.

To determine if there was a performance impact from passing MT6D tunneled traffic over multiple routers, testing was conducted that involved two hops between the source and destination. The multiple subnet tests described so far in this section involve only a single hop between the source and destination. This means that the same router handled all queuing and neighbor discovery between the source and destination. A series of two-hop tests were conducted; one with MT6D running in its default test settings and one without MT6D running. The tests without MT6D showed negligible performance difference between one hop and two hops.

There was a slight difference between one-hop and two-hop results when MT6D was running. For the standard ping tests, illustrated in Figure 9.16(a), the average RTT was approximately 0.3 msec faster over two hops than one hop. The standard deviation of 13.933 msec was also approximately 1 msec lower than the one-hop tests. These lower times are a result of higher packet loss shown in Figure 9.16(b). The packets dropped were primarily those being queued by the router, as seen with the one-hop tests. The difference with two-hop tests was that the second hop router was sometimes dropping the first ping request sent



(a) Average transfer speed



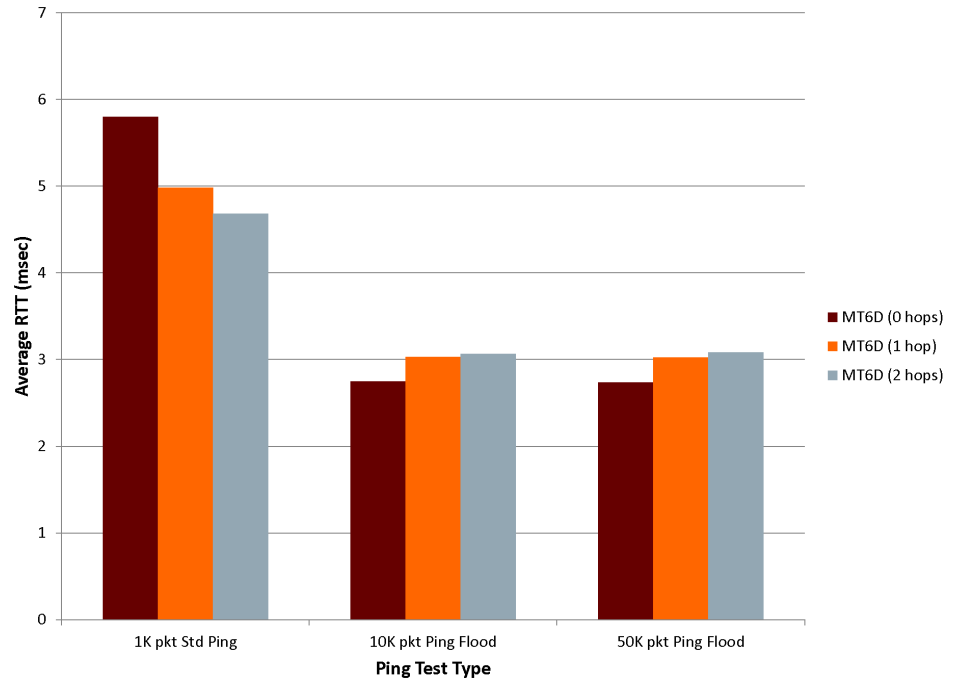
(b) Average percentage of retransmitted packets

Figure 9.15: 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.

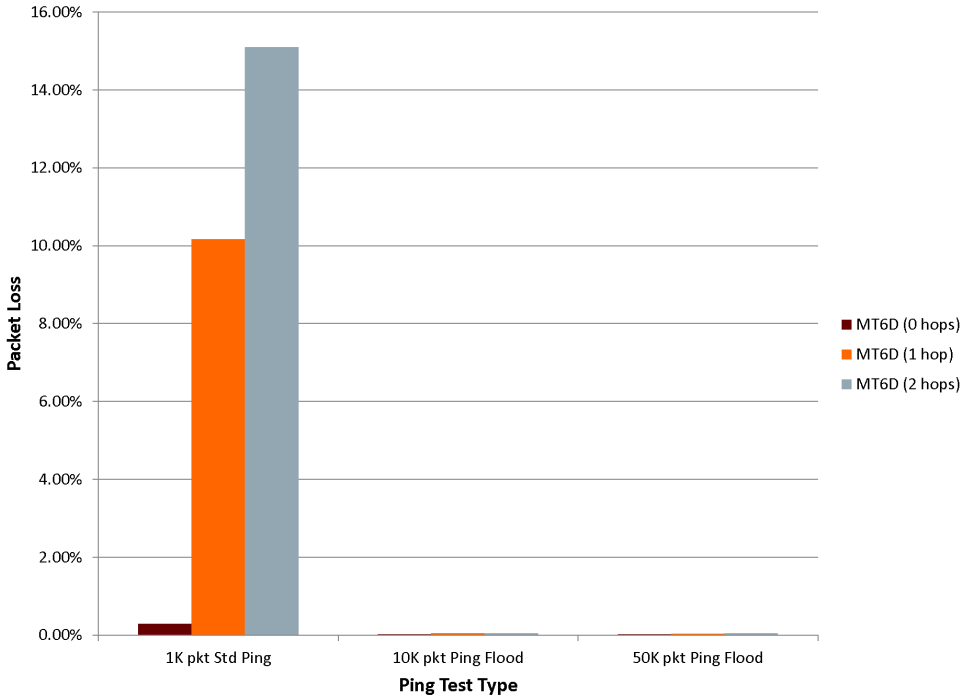
after an address rotation. Recall that single hop tests primarily dropped the first ping reply; this was also true with two hop tests. Packet captures showed that the dropped ping requests and replies were lost during the neighbor discovery process. Approximately 50% of address rotation intervals lost two sequential pings rather than just one (as seen in the single hop tests). The packet captures showed that the ping request would make it to the second hop router, the router would begin the neighbor discovery process, and then the packet would get dropped. Regardless of whether the ping request was passed successfully or dropped, the first ping reply in response to a successful ping request would be dropped by the router closest to the sender during the neighbor discovery process. The result was that half of the address rotations would experience one lost ping, while the other half would experience two lost pings. Although it is unclear why the second hop router only occasionally dropped the first ping request after an address rotation, it is likely due to neighbor discovery taking longer than the router was willing to queue the packets. This would account for the reduced average RTT and standard deviation. It is also possible that there is a flaw in how the routers queue incoming IPv6 packets. For the ping flood tests, the average RTT was slightly higher, which accounts for the packet having to traverse additional hops. The difference in packet loss between the one-hop and two-hops test was negligible.

Connection-oriented tests demonstrate that the number of hops has little impact on throughput. Figure 9.17(a) shows some files that on average transferred slower than during one hop testing, while others transferred faster. These fluctuations reflect variations in the live network due to background traffic. Packet loss, illustrated in Figure 9.17(b), was consistently higher for all file transfers that experienced one or more address rotations. Based on the observed standard ping packet loss, a higher packet loss for connection-oriented traffic is consistent with expectations. The additional packet loss is due to additional loss at the second hop router during neighbor discovery.

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 [58] 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

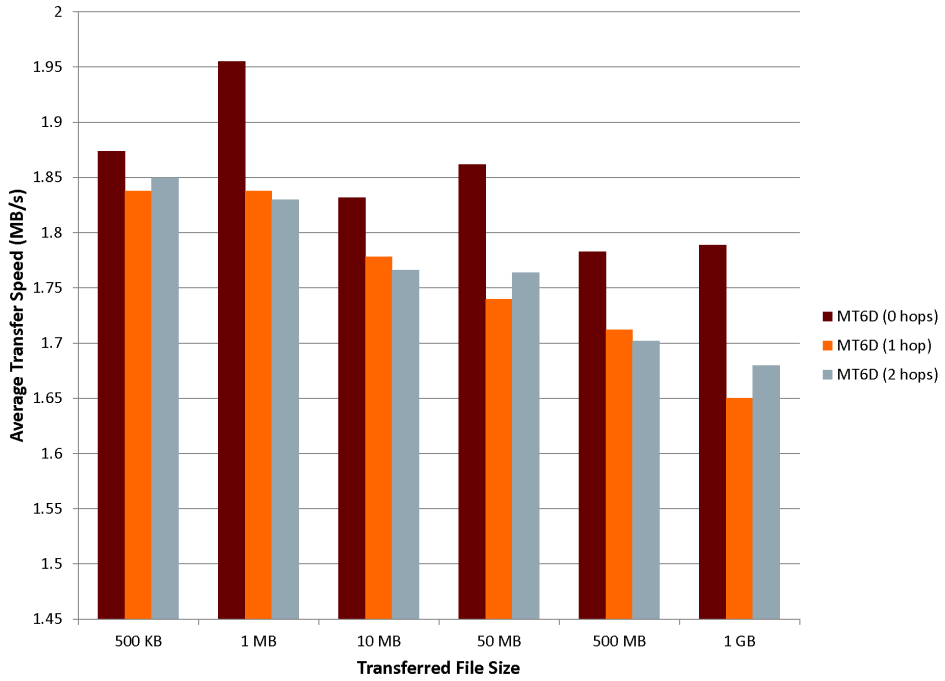


(a) Average RTT

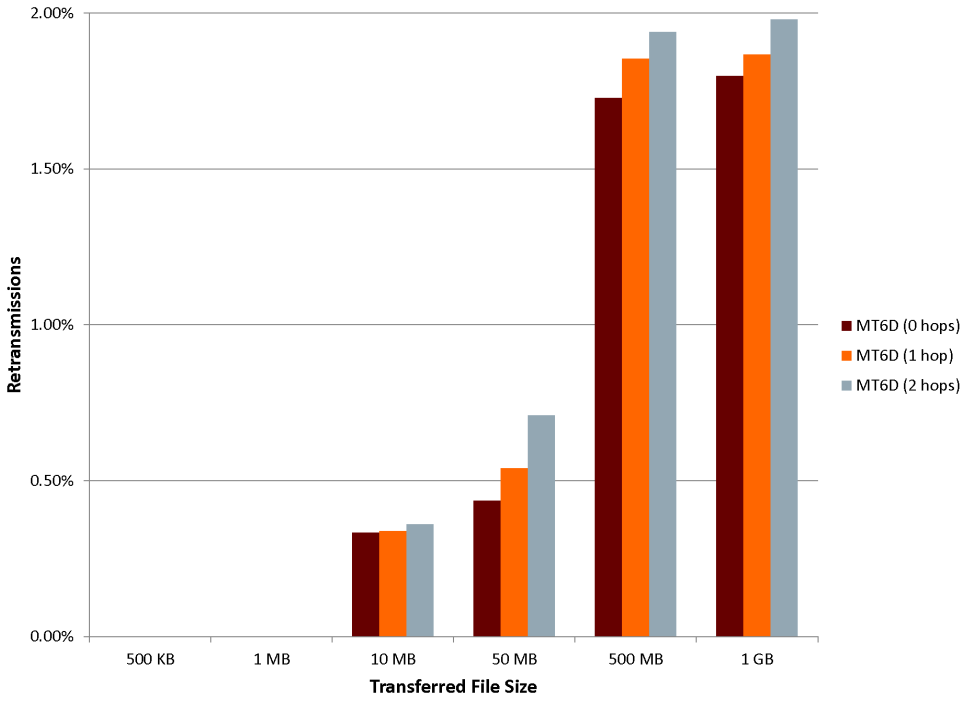


(b) Average packet loss

Figure 9.16: Average speed and packet loss of connectionless traffic for MT6D traversing a varying number of hops. The graph compares results from running MT6D using default test settings on a single subnet (0 hops), between two subnets crossing a single hop, and between two subnets traversing two hops.



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.17: Average transfer speed and packet loss of connection-oriented traffic for MT6D traversing a varying number of hops. The graph compares results from running MT6D using default test settings on a single subnet (0 hops), between two subnets crossing a single hop, and between two subnets traversing two hops.

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 9.3(a). The same two hosts communicating without MT6D across different subnets had an average throughput of approximately 11 MB/sec as shown in Figure 9.15(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.

9.1.3 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 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 Voice over IPv6 (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 [53]. Mumble was chosen due to its ease of setup and IPv6 support. Mumble runs a voice codec called speex [76] instead of the more standard G.711 [41]. 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 with i-DKG disabled. 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 QoS for acceptable latency according to G.114 [40] 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 [26, 29, 39]. 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" [6] 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 [66]. 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.

9.2 Stand-alone Device Testing

The next series of live tests were conducted to determine the performance impact when MT6D was operating as a stand-alone device (see Chapter 8, Section 8.12.2). In this set of tests, the Dell Optiplex machines described in Section 9.1 served as the two end hosts. These two end hosts are again referred to as the client and the server. Two Dell PowerEdge SC 1435 servers containing a 2.0 GHz AMD CPU with 2GB DDR2 667MHz RAM and running 32-bit Ubuntu Linux 11.10 were set up with MT6D as stand-alone devices. Each PowerEdge machine contains two Fast Ethernet NICs, which represent the internal and external NICs depicted in Figure 8.11. The network diagram for MT6D stand-alone device testing is depicted in Figure 9.18.

Testing in Section 9.1 demonstrated that the biggest impacts on performance when using MT6D were a result of changing the rotation interval. Tests that compared tunnel modes or key generation did not significantly impact performance. For this reason, only rotation intervals were varied for testing with MT6D operating on stand-alone devices. The first set of tests compared the default 10-second rotation interval with no address rotation interval to establish a baseline. The second set of tests varied the address rotation interval between 5, 10, 15, and 30 seconds. All tests used AES encrypted tunnels. None of the tests used i-DKG.

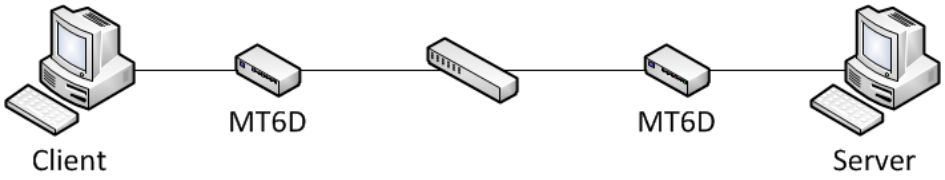


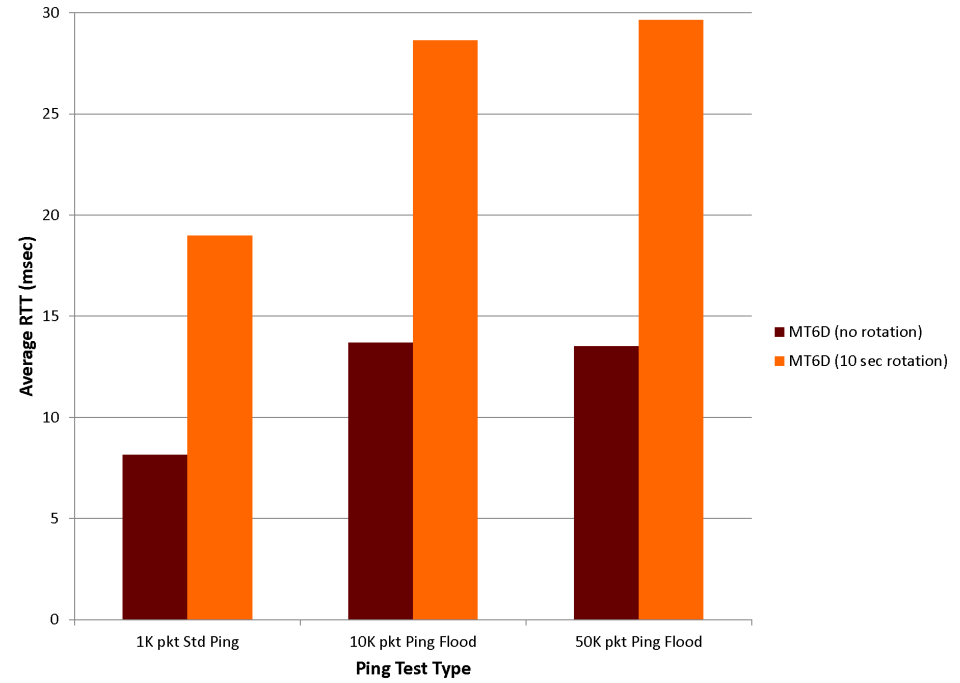
Figure 9.18: Network Diagram using the MT6D stand-alone test device configuration

Benchmarking Default Test Settings

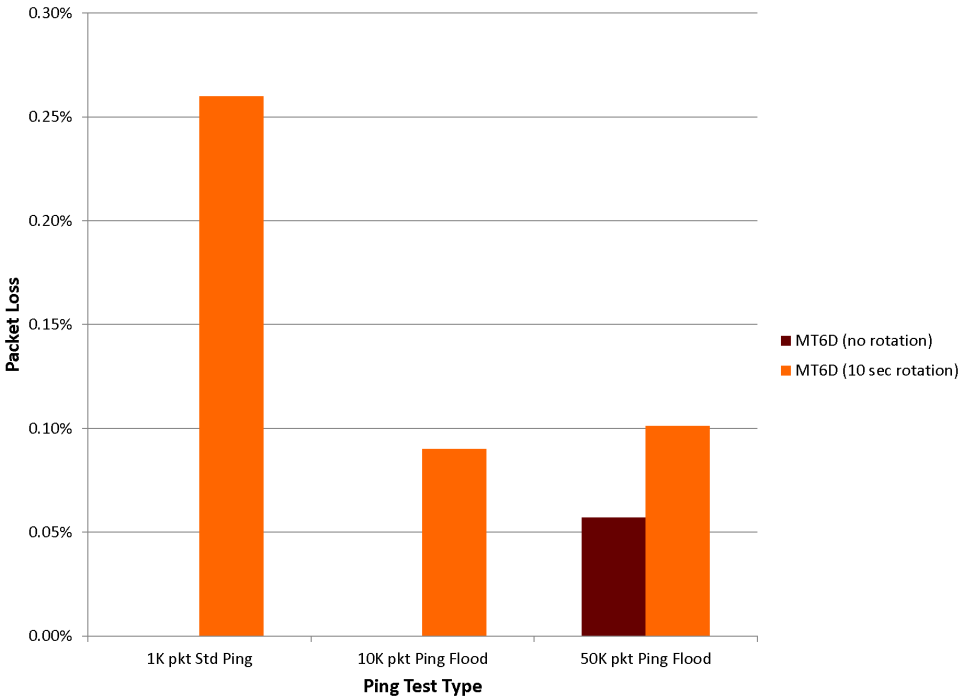
Benchmarking the performance of MT6D was necessary before comparing its performance using different rotation intervals. Benchmarking was completed by comparing MT6D operating under the default test settings described at the beginning of this chapter with MT6D operating without rotating addresses. Testing was again conducted with both hosts located on the same subnet. No routers were required in the exchange between the client and server as discussed in Section 9.1.1. Test results without MT6D were previously established in Section 9.1.

Connectionless traffic tests conducted with MT6D operating in its stand-alone configuration demonstrate the impact the Python prototype has on performance. Specifically, these tests demonstrate the performance degradation due to using `libpcap` with Python. Figure 9.19(a) illustrates an average RTT of 8.16 msec for the 1,000 packet standard ping test when MT6D is not rotating addresses. This is four times higher than the 2.30 msec observed for the same test when using the integrated software configuration. Since MT6D is not rotating addresses, this additional latency is primarily a result of code implementation differences between the two configurations. A discussion of how the two configurations differ in design was provided in Chapter 8, Section 8.12. The use of `libpcap` was limited to inbound packets in the integrated software configuration to minimize locking of other Python threads by the GIL. In the integrated software configuration, this was achieved through the use of the TUN/TAP interface to handle all outbound packets. Since the stand-alone configuration is a separate device, the TUN/TAP interface could not be used. Instead, the stand-alone configuration acted as a bridge that required `libpcap` to pick-up traffic from both the internal and external NICs (see Figure 8.11). The result is that all other Python threads are locked by the GIL whenever `libpcap` is manipulating a packet.

Since `libpcap` is operating on both inbound and outbound packets, `libpcap` dominates the code execution. Two treemaps were made from a profile of the listener process described in Chapter 8, Section 8.9. The profile was built using `cProfile` [11] and visualized into tree maps using RunSnakeRun [69]. Figure 9.20 illustrates the two treemaps. The first treemap, shown in Figure 9.20(a), visualizes the code profile of the entire listener process. `Libpcap` occupies so much of the code execution that it is the only component of the listener process visible in the treemap. The second treemap, shown in Figure 9.20(b), profiles the packet handler sub-process within the listener process. With `libpcap` remove from the profile, it is possible to to see the other portions of the code execution. Given that the listener process is dominated by `libpcap`, the significant decrease in performance is not surprising. Some



(a) Average RTT



(b) Average packet loss

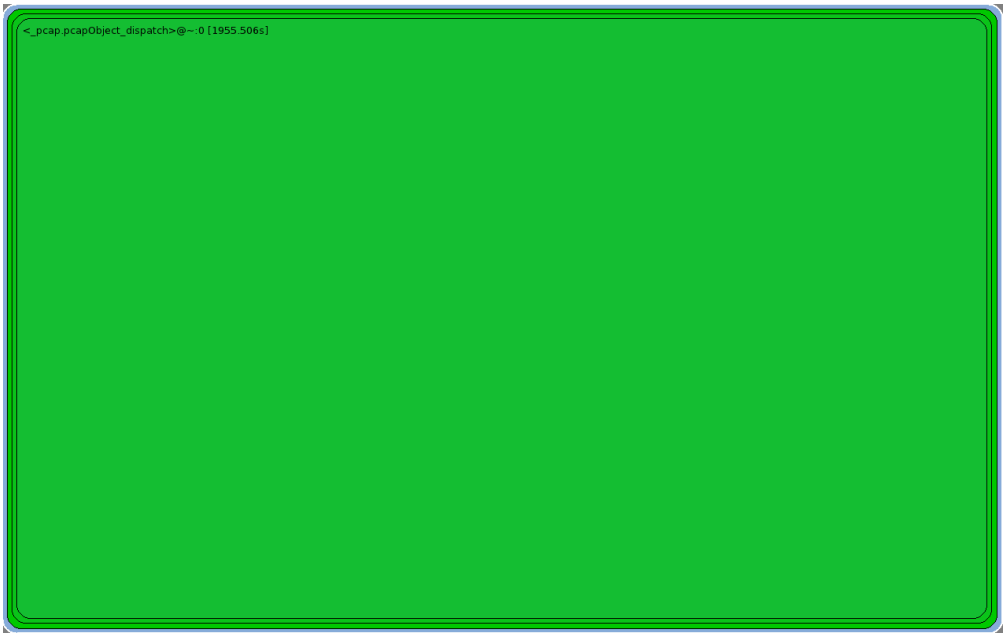
Figure 9.19: Average speed and packet loss of connectionless traffic using the stand-alone device configuration comparing no MT6D, MT6D using the default test settings, and MT6D using a fixed address

of the additional latency is also a result of the stand-alone device having considerably less processing power than the client and server. MT6D executes directly on the client and server when using the integrated software configuration.

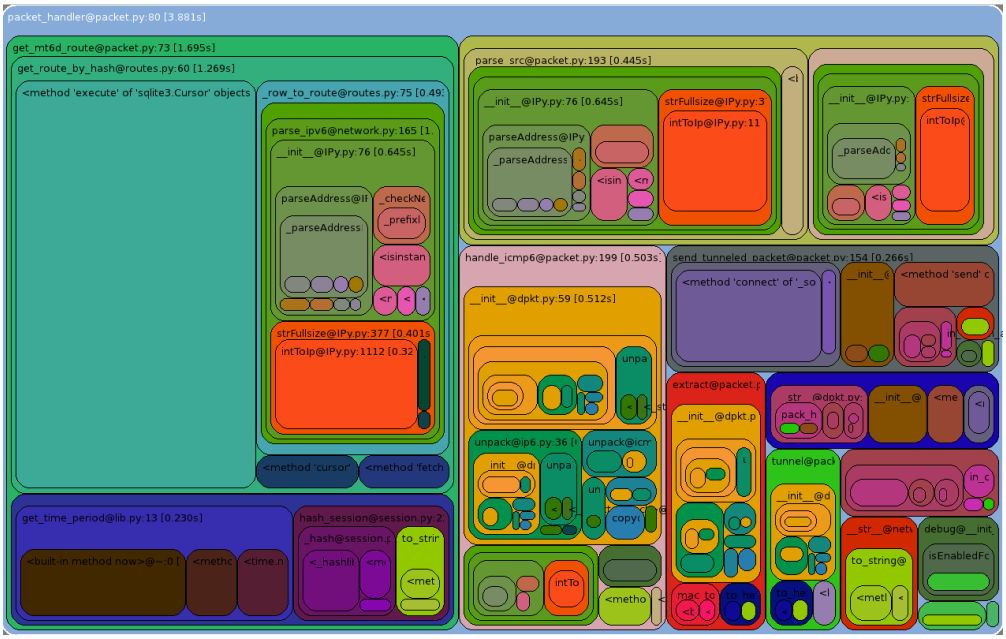
A similar decrease in the magnitude of performance was observed for the 1,000 packet standard ping test rotating addresses every 10 seconds. In the integrated software configuration the average RTT was 5.80 msec. For the stand-alone configuration shown in Figure 9.19(a), it was 18.98 msec; this is just under a fourfold latency increase. Packet loss percentages for the 1,000 packet standard ping tests were only slightly higher for rotating addresses in the stand-alone configuration than in the integrated software configuration as illustrated in Figures 9.2(b) and 9.19(b). This was expected since the packet rates were the same in both configurations (one packet per second).

Connectionless ping flood tests using the stand-alone configuration experienced a greater decrease in speed. The average RTT for all ping flood tests with fixed addresses was 13.60 msec using the stand-alone configuration compared to 2.10 msec using the integrated software configuration. This is a sixfold increase. When addresses were rotating every 10 seconds, the average RTT for all the ping flood tests using the stand-alone configuration was 29.14 msec (see Figure 9.19(a)). This is over 10 times higher than the average RTT of 2.74 msec for the integrated software configuration. The dramatic decrease in ping flood performance is due to the internal NIC sending packets as fast as possible while the external NIC is receiving packets at close to the same rate. This causes the NICs to compete with each other for access to `libpcap`. The `libpcap` race condition occurs on both the client and the server. There was virtually no packet loss for fixed address tests using MT6D. This matches other tests when addresses do not rotate. When addresses were rotating every 10 seconds, packet loss increased from an average of 0.02% shown in Figure 9.2(b) to an average of 0.10% shown in Figure 9.19(b). The higher packet loss is a byproduct of the added latency. Higher RTTs extended the average completion time past 100 seconds for the 10,000 ping flood tests and past 500 seconds for the 50,000 ping flood test. The extended time means that more address rotations occurred. Previous tests all demonstrated that packet loss occurs during address rotations. The additional packet loss accounts for approximately eight lost packets out of 10,000 or approximately 40 lost packets out of 50,000.

Connection-oriented tests also showed a considerable decrease in performance when using the MT6D stand-alone device configuration. When MT6D operated with a fixed address, the average throughput was approximately 1.9 MB/s using MT6D in integrated software configuration mode. The throughput decreased to approximately 600 KB/s when using the



(a) Profile of the MT6D listener process. The only process visible is `libpcap`.

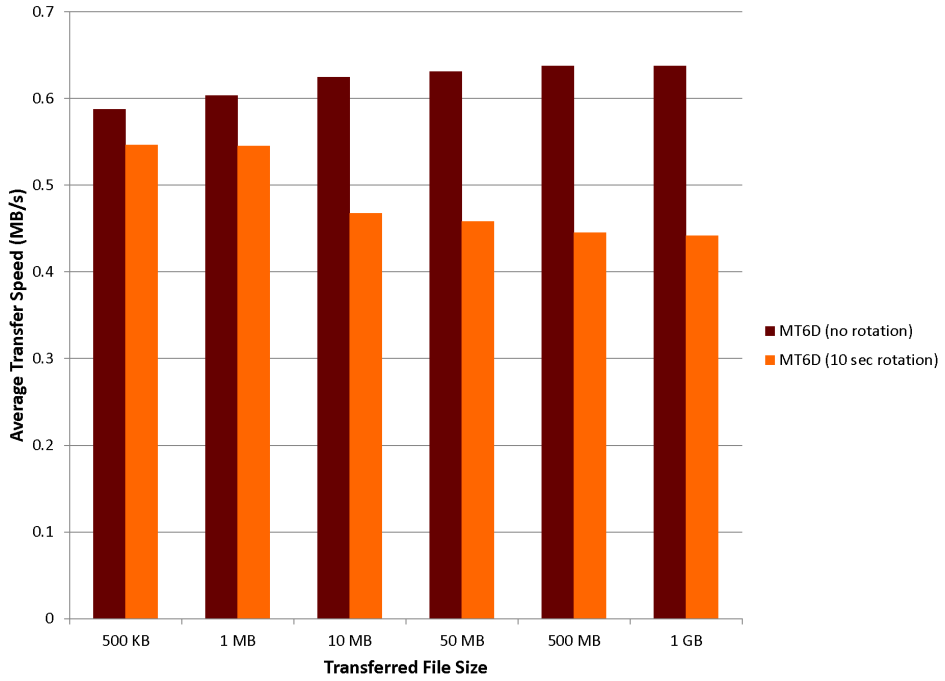


(b) Profile of the MT6D packet handler sub-process. Other components become visible when excluding `libpcap` from the code profile.

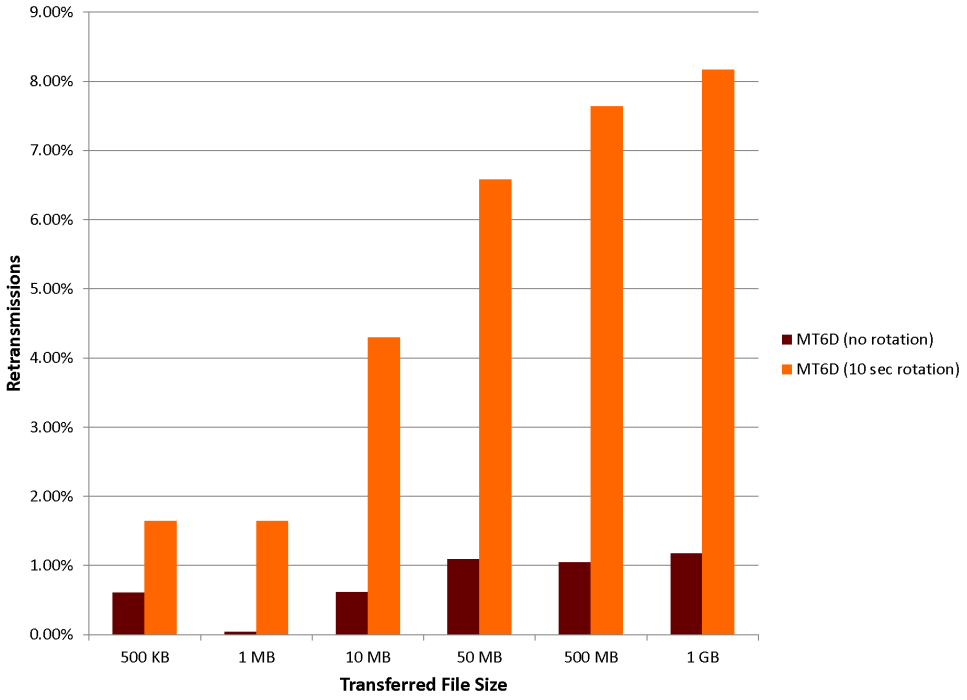
Figure 9.20: Treemap visualizations of the MT6D listener process

stand-alone device configuration as illustrated in Figure 9.21(a). This decrease in speed is comparable in magnitude to what was observed with the connectionless tests. Unlike previous tests with fixed addresses, the stand-alone device configuration had some packet loss occur (see Figure 9.21(b)). In addition to factors previously mentioned, loss is higher in the stand-alone device configuration as a result of TCP congestion control. The end hosts are not aware of any bottlenecks that occur due to processing on the MT6D devices when MT6D is operating in stand-alone device configuration. In normal network operations, intermediate network-layer and higher devices can notify the sender of congestion as soon as it occurs. Data-link layer devices, such as bridges, are not capable of notifying the sender of congestion. Each stand-alone MT6D device acts as a network bridge passing packets through both the sender's and the receiver's MT6D devices without congestion being detected. Packets arriving at the receiver arrive much slower than the sender sent them due to processing by both MT6D devices. From the receiver's perspective, no congestion is occurring. The sender, as a result, never gets notified that packets are being sent faster than the stand-alone MT6D device can handle. Thus, these packets are dropped. Each sender's stand-alone MT6D device would need to have the capability of adjusting the sender's congestion window to address the lack of congestion notification. This change would require each stand-alone device to be reconfigured to perform network-layer functions. Modifying the stand-alone device in this way would require more processing of each packet and likely result in additional latency. Congestion control may not be as big of an issue if the dependency on `libpcap` were removed.

The performance decrease for connection-oriented tests when MT6D was rotating addresses every 10 seconds in its stand-alone device configuration further illustrates the congestion control issue. File transfers less than 10 MB transferred within the span of a single address rotation. As a result, the file transfer speed was only slightly less than it was when using a fixed address. File transfers 10 MB and larger have at least one address rotation occur during the file transfer. Consistent with all other tests, transfer speed is less and packet loss is higher. While the packet loss of 8.17% for the 1 GB file transfers (illustrated in Figure 9.21(b)) is much higher than the 1.80% observed using the integrated software configuration, it is proportionate to the decrease in file transfer speed. Figure 9.22 verifies that the majority of packet loss occurred due to address rotations. The graph illustrates that the volume of TCP retransmissions (the red line in the graph) matches the volume of TCP retransmissions in Figure 9.5. What is interesting about Figure 9.22 is that the volume of TCP traffic (the black line in the graph) is approximately four times less than



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.21: Average transfer speed and packet loss of connection-oriented traffic using the stand-alone device configuration comparing no MT6D, MT6D using the default test settings, and MT6D using a fixed address

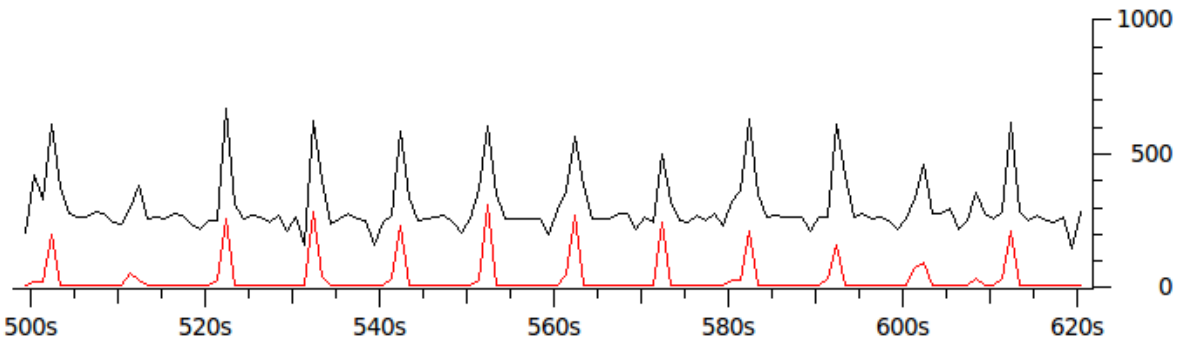


Figure 9.22: Graph of TCP retransmissions when using the MT6D stand-alone device configuration. The black line shows the total TCP traffic while the red line shows just the retransmitted packets. The x-axis represents time in seconds since the packet capture was started. The y-axis represents the number of packets sent per second.

the volume in Figure 9.5. With a fourfold decrease in performance and a similar volume of packet retransmissions, a fourfold increase in packet loss is expected.

Varying Address Rotation Intervals

The next set of tests conducted using the stand-alone device configuration varied the rotation interval between 5, 10, 15, and 30 seconds. These were the same intervals tested in Section 9.1.1 using the integrated software configuration. The previous rotation interval tests proved that as the rotation interval increases the performance improves. The same is true when MT6D operates in the stand-alone device configuration.

The connectionless tests, shown in Figure 9.23(a), demonstrate that the average RTT decreases as the rotation interval increases. These results match the performance progression observed in Figure 9.6(a) for MT6D using the integrated software configuration. The first obvious difference is speed. As previously explained, the stand-alone device configuration runs slower than the integrated software configuration. The second difference, which was seen in Figure 9.19(a), was that the average RTT for ping floods is considerably higher than for standard pings. The difference in RTT between each time interval matched expectations.

Packet loss for connectionless traffic also improves as the address rotation interval increases as shown in Figure 9.23(b). For the 1,000 packet standard ping tests, the packet loss was close to the same in both MT6D configurations. This was observed during benchmark testing of the stand-alone device configuration and is due to packets being sent as such low rate that both configurations are able process each packet before the next one arrives. The

packet loss for the ping flood tests using the stand-alone configuration is higher than when using the integrated software configuration. This is also expected and was discussed during benchmark testing for the MT6D stand-alone configuration.

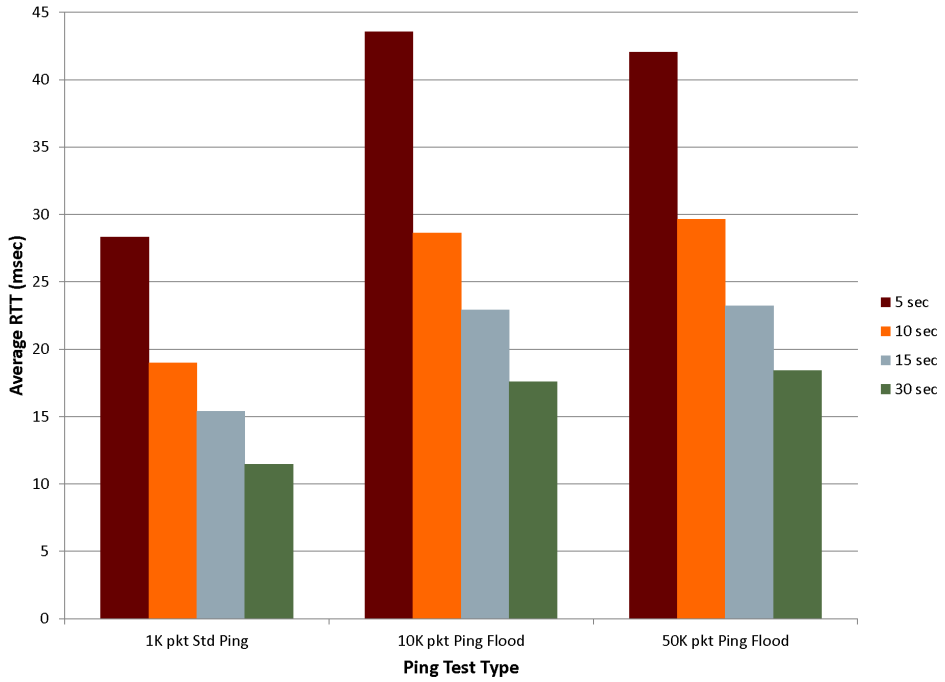
The performance for connection-oriented tests decreases as the rotation interval decreases as illustrated in Figure 9.24. This is consistent with results from MT6D integrated software configuration testing. As the rotation interval increases from 10 seconds to 30 seconds, the average transfer speed steadily approaches the transfer speed when not rotating addresses using the MT6D stand-alone device configuration. When the rotation interval was decreased to five seconds, however, the transfer speed drops sharply. For the 500 MB and 1 GB file transfers, the average transfer speed is barely above 200 KB/sec (see Figure 9.24(a)). Based on the current prototype's reliance on `libpcap`, these results indicate that rotation intervals below 10 seconds should not be used.

Packet loss, shown in Figure 9.24(b), increases as the file transfer size increases. Packet loss is comparatively low for file transfers below 10 MB. Since these file transfers complete within the span of a single address rotation, a lower packet loss is expected. For files 10 MB and larger, packet loss increases. This is consistent with results described during benchmark testing that illustrated how the same volume of TCP retransmissions, combined with a lower file transfer rate, result in higher packet loss. This relationship also explains why packet loss drops with longer address rotation intervals. A longer rotation interval means less address rotations during the file transfer.

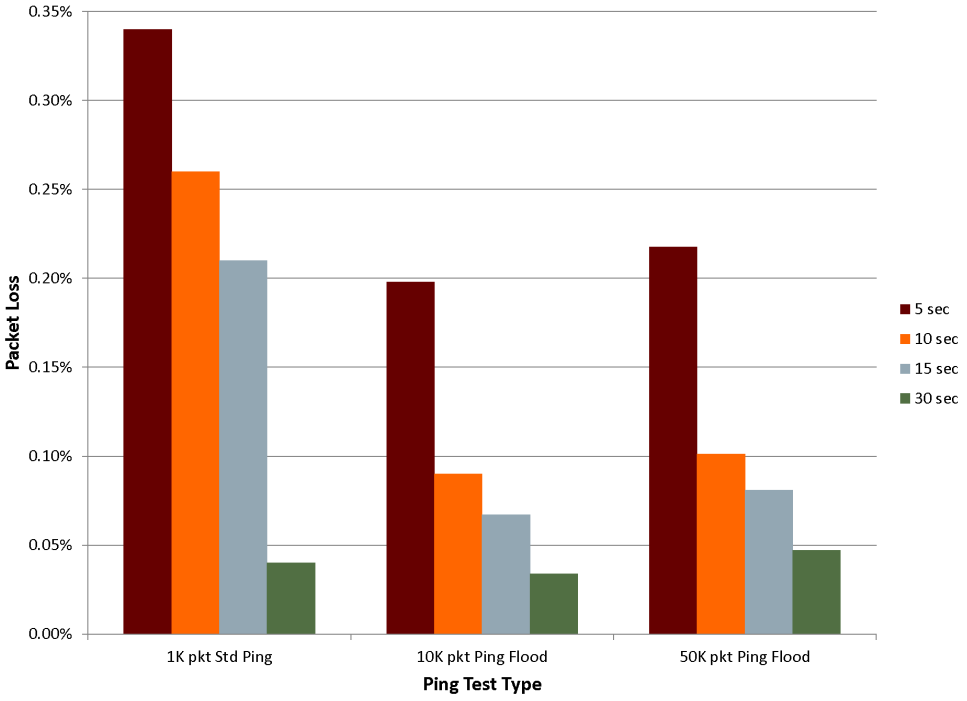
9.3 Summary

This chapter 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 of MT6D was used during all initial testing. The integrated software configuration testing first established a baseline set of results that all MT6D options (e.g., encryption, i-DKG, etc.) were compared against. These baseline results were gathered while MT6D rotated addresses every 10 seconds using encrypted tunnel mode, but not using i-DKG. These same baseline results formed the basis of all subsequent testing, to include the stand-alone device configuration.

Subsequent testing was done using the stand-alone device configuration. For both configurations, MT6D was tested using UDP and TCP traffic. Test results demonstrated that MT6D was able to operate seamlessly when handling either type of network traffic. This

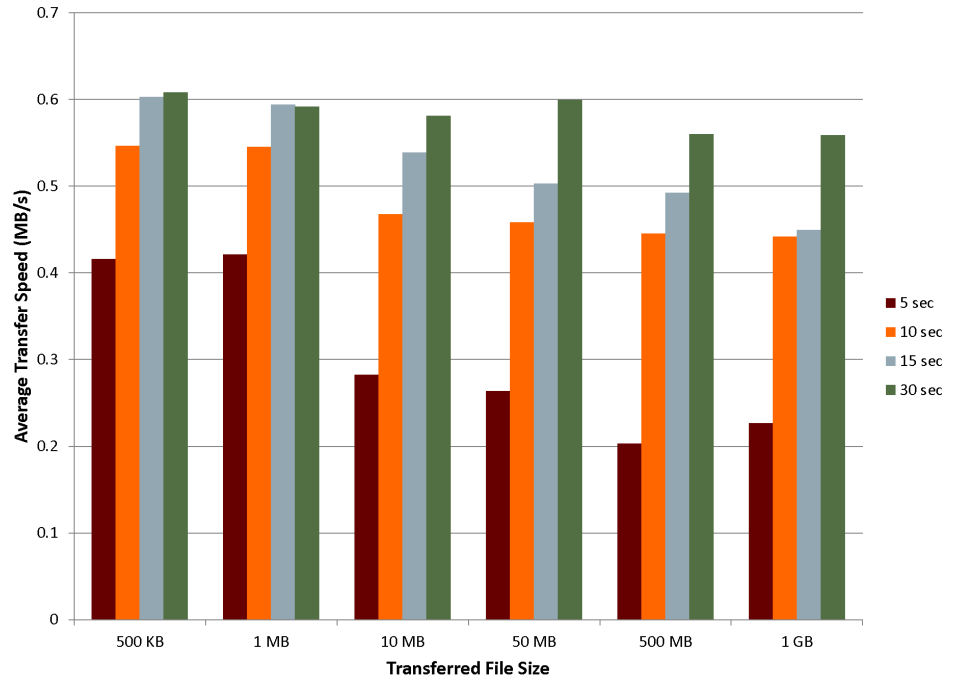


(a) Average RTT

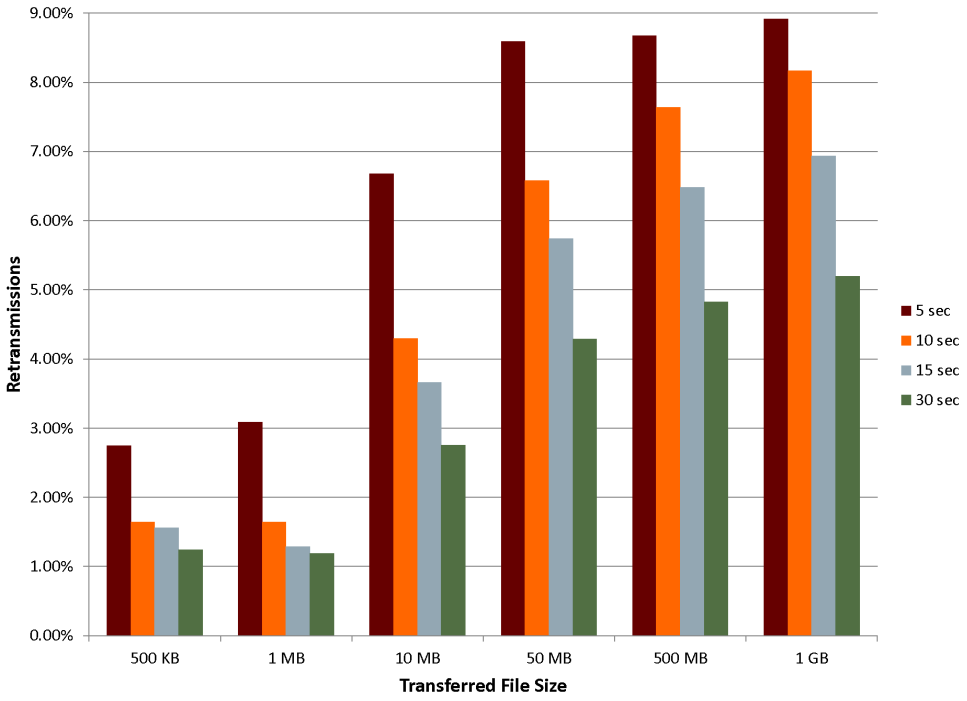


(b) Average packet loss

Figure 9.23: Average speed and packet loss of connectionless traffic using the stand-alone device configuration and different rotation intervals



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 9.24: Average transfer speed and packet loss of connection-oriented traffic using the stand-alone device configuration and different rotation intervals

was especially significant with TCP traffic, since ongoing sessions were not dropped nor did they require renegotiation. As expected, MT6D in its current prototype version incurred a significant amount of latency. Much of this was attributed to software implementation. MT6D also caused packet loss, primarily during address rotations.

A comparison of baseline test results for both the integrated software and the stand-alone device configurations showed that the integrated software configuration significantly outperformed the stand-alone device configuration. The stand-alone configuration is capable of an average throughput of 2 MB/sec with less than 2% packet loss. The stand-alone device configuration is only capable of approximately a 600 KB/sec throughput and experiences packet loss above 7%. This is primarily a result of how MT6D is implemented in Python. An optimized version of MT6D will likely result in much better performance and more consistent results between the different configuration options.

Chapter 10

Analysis of MT6D in Simulation

The testing conducted in Chapter 9 utilized the live Virginia Tech IPv6 network. While this testing produces real results that prove MT6D's capability to operate under live conditions, it is not clear how much the results are impacted by other traffic on the network. It is also not clear how much impact network configurations have on MT6D test results. The goal of this chapter was to use a network simulator to isolate MT6D testing to gain a better understanding of performance without the impact of unrelated network traffic. The problem is that not many network simulators are capable of supporting IPv6, much less custom IPv6 implementations. The main simulators that claim to be IPv6 capable are NS3 [61], OMNeT++ [62], and OPNET [63].

OPNET was chosen primarily because it includes a System-in-the-Loop (SITL) module [64] that allows live devices to connect to the simulation. In particular, OPNET includes a unique configuration with its SITL module called Live-Sim-Live that allows two live devices to connect with each other through the simulated network. This was an important feature for MT6D testing because, at the current stage of development, porting MT6D into the simulation was not feasible. Additionally, SITL facilitates rapid testing of different MT6D configurations and future prototype versions.

Despite the goal of using OPNET to get a better understanding of the performance impacts of using MT6D, testing revealed that OPNET has a number of limitations with simulating IPv6. A discussion of modeling IPv6 in OPNET is provided in Appendix A. Some of the main limitations are a lack of support for most extension header types, a lack of support for a number of ICMPv6 message types, improper formatting of supported ICMPv6 message types, and the inability to generate IPv6 background traffic. As a result, testing MT6D in OPNET was limited to a few different scenarios and configuration settings.

Although the ability of OPNET to simulate IPv6 did not provide as much flexibility as desired, testing did help validate the results from Chapter 9. Testing the limited options in simulation also provided interesting performance results.

10.1 Isolated MT6D Simulation Testing

A simulated network was built to test the performance of MT6D without the impact of other network traffic. The simulation network did not include any outside hosts other than the client and server described in Chapter 9, Section 9.1. A one-hop and a two-hop network configuration were built in simulation to mirror the testing over different subnets conducted in Chapter 9, Section 9.1.2. A zero hop network configuration was not tested in simulation because the OPNET translation of each packet would effectively behave similar to the one-hop configuration. None of the MT6D tests included encryption because of OPNET's lack of support for most extension headers as described in Appendix A. Recall from Chapter 8, Section 8.3.1, that encryption requires the destination options extension header. The inability to test MT6D in encrypted tunnel mode did not significantly impact testing since Figures 9.8 and 9.9 illustrate only a small difference in performance between the two tunnel modes.

The test scenario for simulation testing was kept as close to the live testing scenario described in Chapter 9 as possible. To that end, the same batch of tests were conducted to measure both connectionless and connection-oriented traffic. Basic functionality was tested using sets of 1,000 ping packets sent from the client to the server at a rate of one packet per second. High traffic volume of connectionless traffic was tested using 10,000-packet and 50,000-packet ping floods sent from the client to the server. Connection-oriented traffic was tested with the client using HTTP over TCP to download files ranging from 500KB to 1GB from the server. Ten iterations of each test were conducted. The MT6D default settings for simulation testing had to be modified to account for the lack of support for encryption. The default settings were an address rotation interval of 10 seconds, no encryption on MT6D-tunneled traffic, and i-DKG turned off.

The same client and server machines described in Chapter 9, Section 9.1 were used to maintain consistency between live and simulation testing. For all simulation testing, the integrated software configuration was used. The integrated software configuration was chosen to gain an understanding of the upper limits of MT6D performance without the impact of live network conditions.

10.1.1 One-Hop Testing

One-hop testing consisted of a simulated Cisco 7507 router connected to the client and server with two SITL links. The network diagram is illustrated in Figure 10.1. Each SITL link portrayed in the diagram is an actual connection to the real client and server. Each interface on the router had a different subnet address to mimic the one-hop live testing described in Chapter 9, Section 9.1.2. This configuration resulted in the router having to queue packets and conduct neighbor discovery whenever it received a packet with an IPv6 address not in its neighbor cache.

Since OPNET's IPv6 capability does not support destination options extension headers, encrypted tunnels (described in Chapter 8, Section 8.3.1) and i-DKG (described in Chapter 8, Section 8.6) could not be tested in simulation. These limitations narrowed the testing to comparisons with and without MT6D, and tests of different MT6D address rotation intervals. It was also not feasible to test how the network performs under a load of nodes using MT6D because the current MT6D prototype cannot be ported into the simulation without significant redesign and conversion to C.

Two sets of tests were conducted over the one-hop network. The first set of tests established a baseline for performance using the default MT6D settings. Baseline testing compared normal network performance (no MT6D) with the performance of MT6D using the default 10-second rotation interval and again with no address rotations. The second set of tests varied the address rotation interval between 5, 10, 15, and 30 seconds.

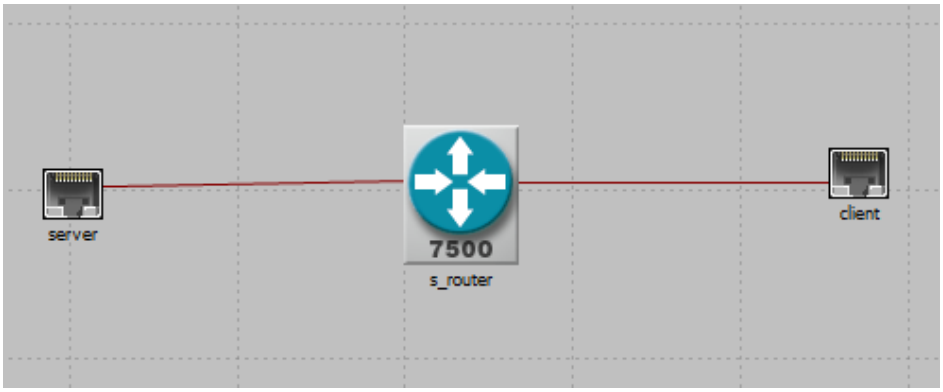


Figure 10.1: OPNET one-hop network configuration

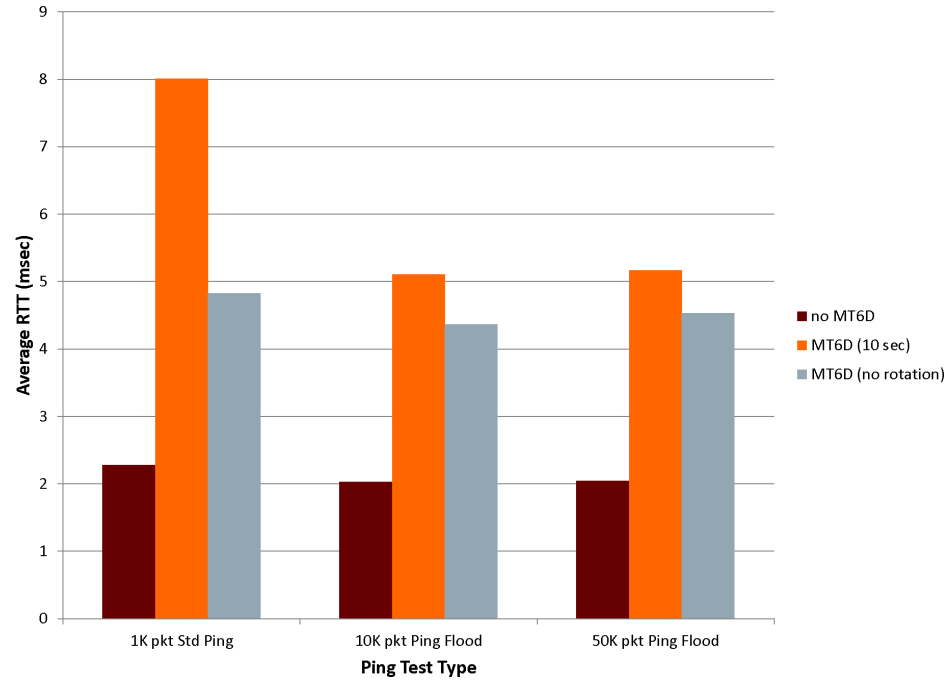
Benchmarking Default Test Settings

Benchmarking MT6D performance in simulation serves two purposes. First, it establishes performance expectations for other simulation test scenarios. Second, it produces results that can be compared with corresponding live test results. The test results discussed in this section can be compared with one-hop live tests conducted in Chapter 9, Section 9.1.2.

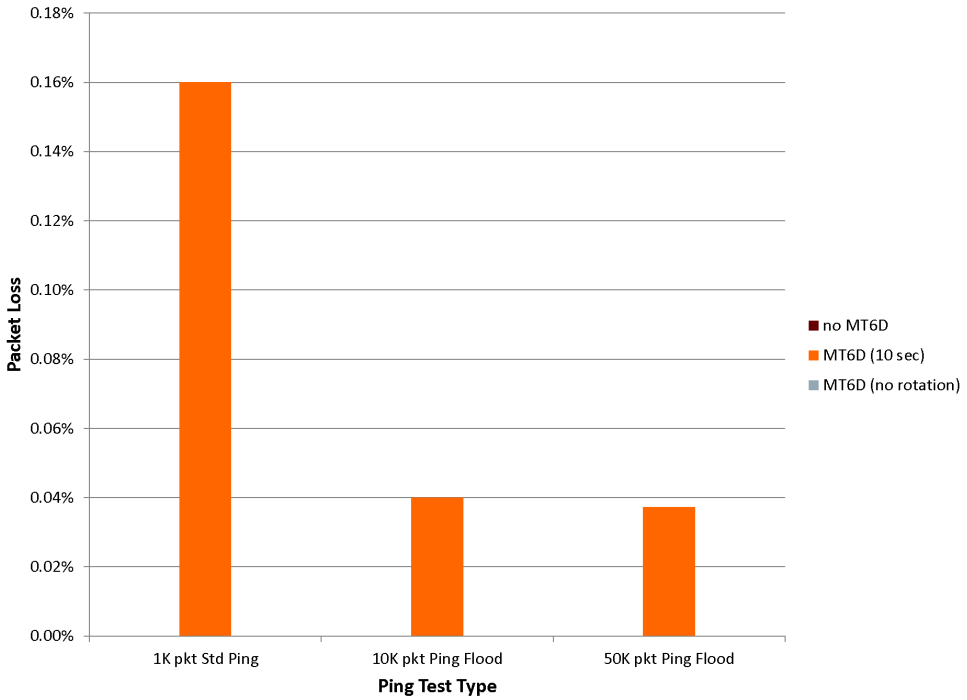
Connectionless tests produced results consistent with other benchmark tests in Chapter 9. The pings without MT6D had lower RTTs than all tests using MT6D. Likewise, the tests using MT6D with a fixed address performed better than when MT6D rotated addresses every 10 seconds. A comparison of the average RTTs for all connectionless tests shown in Figure 10.2(a) against the corresponding live network one-hop tests in Figure 9.14(a) shows an average 2 msec increase in RTT for simulation testing. This added latency is a result of queuing delay in the OPNET SITL module. The SITL module queues packets for translation from live to simulation and vise versa. Figure 10.3 illustrates that queuing delay in the SITL module is between 2-3 msec, which matches the average increase in RTT.

A comparison of packet loss between live and simulation testing produced interesting results. The packet loss for both sets of ping flood tests in simulation was on average 0.04% as shown in Figure 10.2(b). This matches exactly with what was observed on the live network in Figure 9.14(b). The loss for the 1,000 packet standard ping, however, was not the same. On the live network, MT6D experienced 10.16% packet loss. This was explained in Chapter 9, Section 9.1.2 as an error in the way the live router was queuing packets that causes every first ping reply after an address change to be dropped. Since packets are sent at a rate of one packet per second, the resultant packet loss increase should be 10%. Simulation testing confirmed this. The average packet loss in simulation was 0.16%; exactly 10% lower than live network testing.

Connection-oriented test results in simulation (see Figure 10.4(a)) show that tests without MT6D have higher throughput than tests using MT6D. Tests using MT6D with a fixed address have a higher throughput than when MT6D rotates addresses every 10 seconds. Simulation tests using MT6D had an average throughput that was approximately 200 KB/sec higher than live testing illustrated in Figure 9.15(a). A higher throughput was expected in simulation due to operating in an isolated environment. What is interesting about the throughput results is that the average throughput in simulation without MT6D is only slightly above 3 MB/sec. On the live network, the average throughput without MT6D was close to 11 MB/sec. These results indicate that available bandwidth has little impact on MT6D throughput; unless the bandwidth is below 2 MB/sec. This is promising for two



(a) Average RTT



(b) Average packet loss

Figure 10.2: Average speed and packet loss of connectionless traffic over a one-hop simulation network comparing no MT6D, MT6D using the default test settings, and MT6D using a fixed address

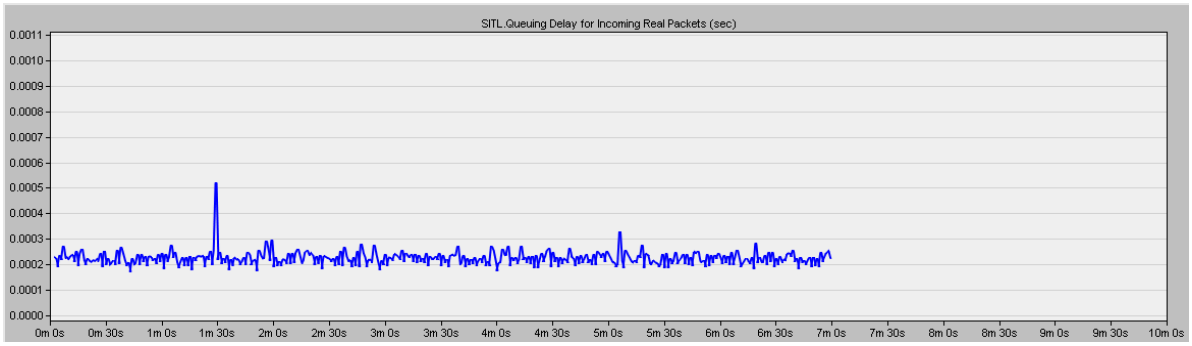
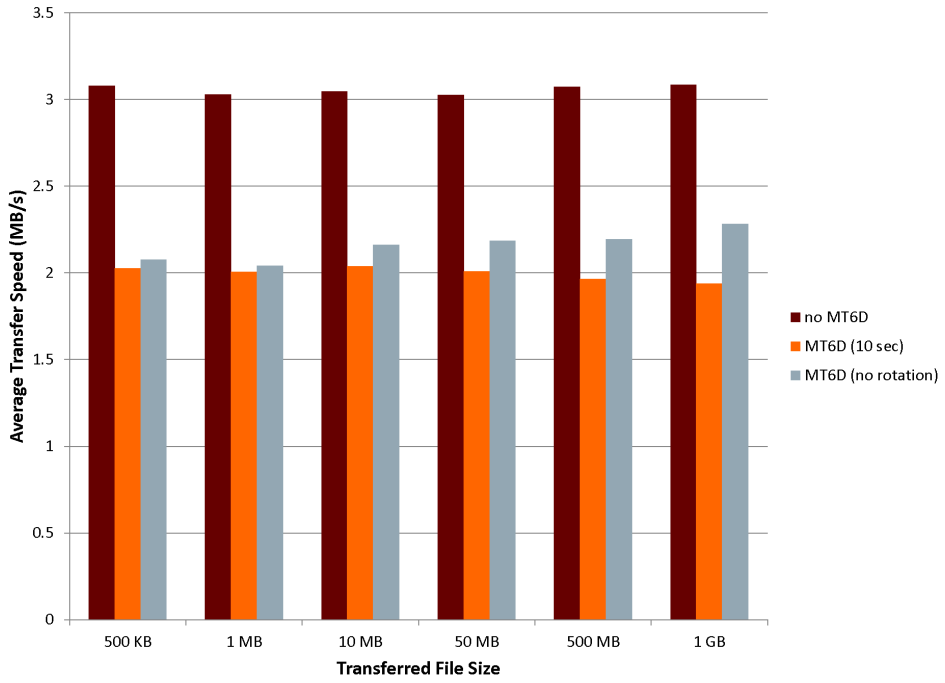


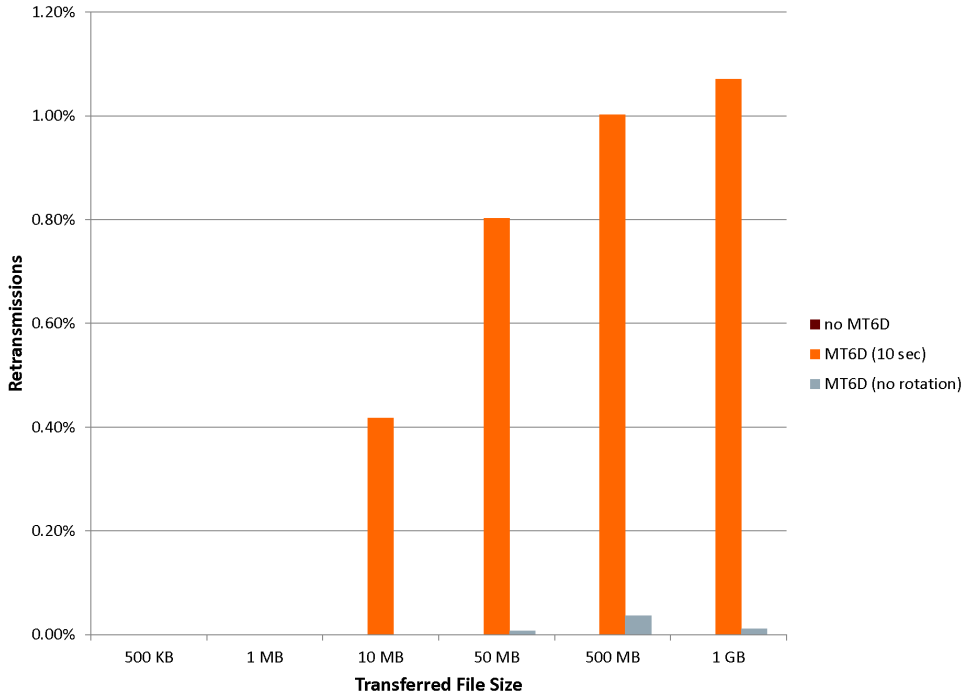
Figure 10.3: Queuing delay in the OPNET SITL module. The x-axis displays the number of seconds the simulation has been running while the y-axis displays the queuing delay in seconds.

reasons. First, it provides further evidence that the performance of MT6D is constrained by its implementation. An optimized version of MT6D should be able to achieve higher performance. Second, MT6D should be able to operate over bandwidth-constrained networks while still providing adequate performance.

Packet loss percentages for connection-oriented testing in simulation also matched expectations. Neither set of tests that used fixed addresses experienced anything other than negligible packet loss as illustrated in Figure 10.4(b). Packet loss was observed during simulation tests that used MT6D rotating addresses every 10 seconds. Figure 10.4(b) shows that packet loss increases as the transferred file size increases. This is consistent with all previous testing on the live network. Since packet loss results from address rotations, more address rotations produce higher packet loss rates. File transfers below 10 MB had no packet loss because addresses did not rotate during the file transfers. Packet loss for 10 MB file transfers was less than half that of larger file transfers. This is due to a single address rotation occurring in six of the iterations. File transfers greater than 50 MB provide a better indication of actual packet loss rates since every iteration experienced multiple address rotations. For the 500 MB and 1 GB file transfers, packet loss in simulation tests was around 1%. This is an 80% improvement over the one-hop live network packet loss for the same file sizes as seen in Figure 9.15(b). Lower packet loss is expected in a isolated network. These results also prove that the OPNET SITL module is capable of passing IPv6 packets without loss; this is despite any queuing that occurs in the SITL module.



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 10.4: Average transfer speed and packet loss of connection-oriented traffic over a one-hop simulation network comparing no MT6D, MT6D using the default test settings, and MT6D using a fixed address

Varying Address Rotation Intervals

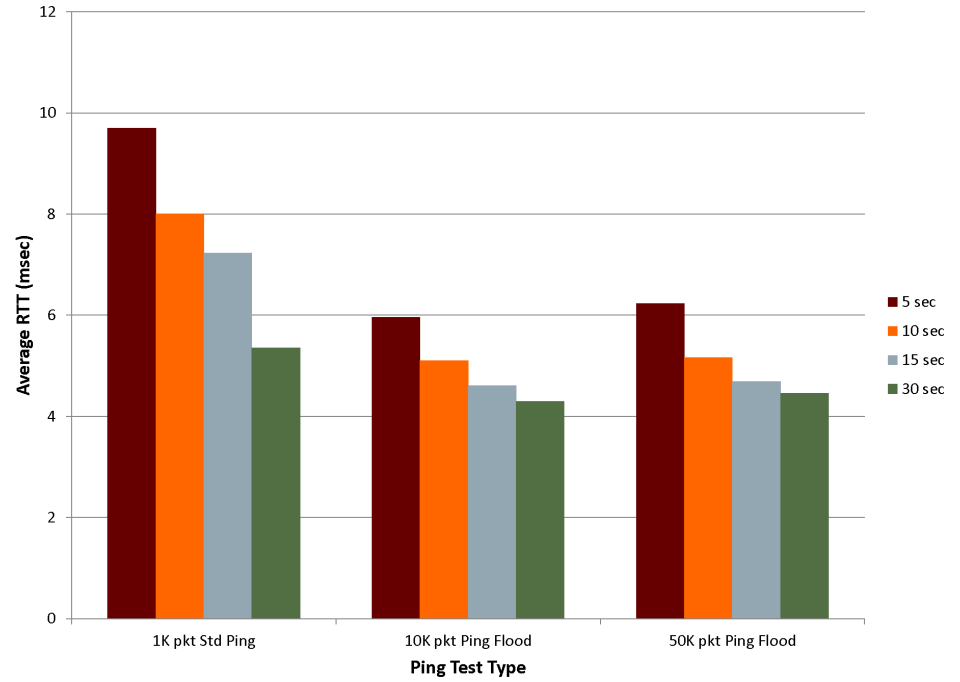
A second set of tests were conducted in simulation to determine if MT6D performance when varying the address rotation interval was comparable to similar live tests performed in Chapter 9, Section 9.1.1. The tests were not expected to mirror those in Chapter 9, Section 9.1.1 since they were performed over the same subnet while the simulation tests were performed using the one-hop network depicted in Figure 10.1. The rotation intervals used were 5, 10, 15, and 30 seconds as in previous tests where the rotation interval was varied.

Connectionless tests illustrated in Figure 10.5(a) demonstrate that as the rotation interval increases the average RTT decreases. These results are consistent with those in Figure 9.6(a). The simulation tests have a longer average RTT than the live tests by approximately 2-3 msec. This added latency is expected due to queuing delay by the OPNET SITL module as shown in Figure 10.3.

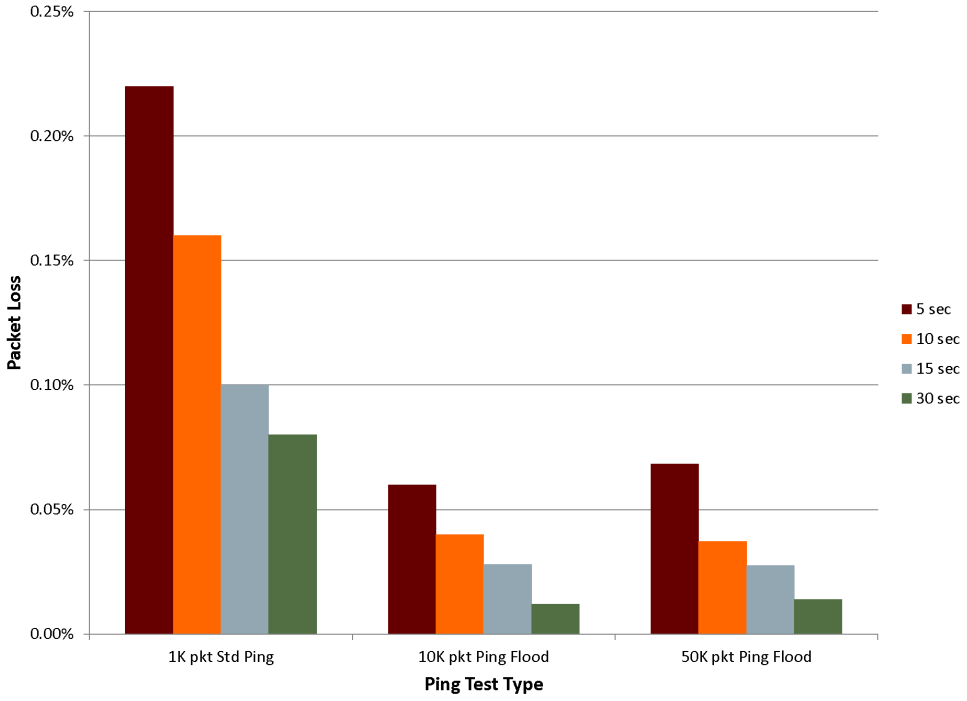
Packet loss for connectionless traffic shown in Figure 10.5(b) is lower in simulation in general than over the live network. The only exception is for the 30 second address rotations. In live testing there was no packet loss when addresses rotated every 30 seconds (see Figure 9.6(b)). Though packet loss is lower in simulation, the magnitude difference is similar to live testing. For the 1,000 packet standard ping test, live tests lost an average of one packet more than tests run in simulation. For ping floods, the difference is only 0.05% less packet loss in simulation. These additional packets could have been lost during live testing due to any number of different issues related to live network conditions. Some packet loss is still expected in simulation since packet loss occurs as a result of MT6D address rotations. Low packet loss in the one-hop simulation network with varying address rotation intervals further demonstrates that the high packet loss (above 10%) seen in Figure 9.14(b) is related to network equipment and is not indicative of MT6D.

Connection-oriented testing produces higher throughputs in simulation than during live testing as illustrated in Figures 10.6(a) and 9.7(a). Simulation throughput is on average between 1.8 MB/sec for the five-second rotation interval and 2 MB/sec for the 30-second rotation interval. Throughput results are less consistent for smaller file transfers as witnessed during live testing. This is not surprising since smaller sample sizes complete in a short amount of time. Any fluctuations in the client or server will be amplified in the test results. System fluctuations will average out for larger file transfers.

Packet loss during connection-oriented simulation testing (see Figure 10.6(b)) follows less of a consistent pattern with file sizes below 50 MB. This is also true of live testing as

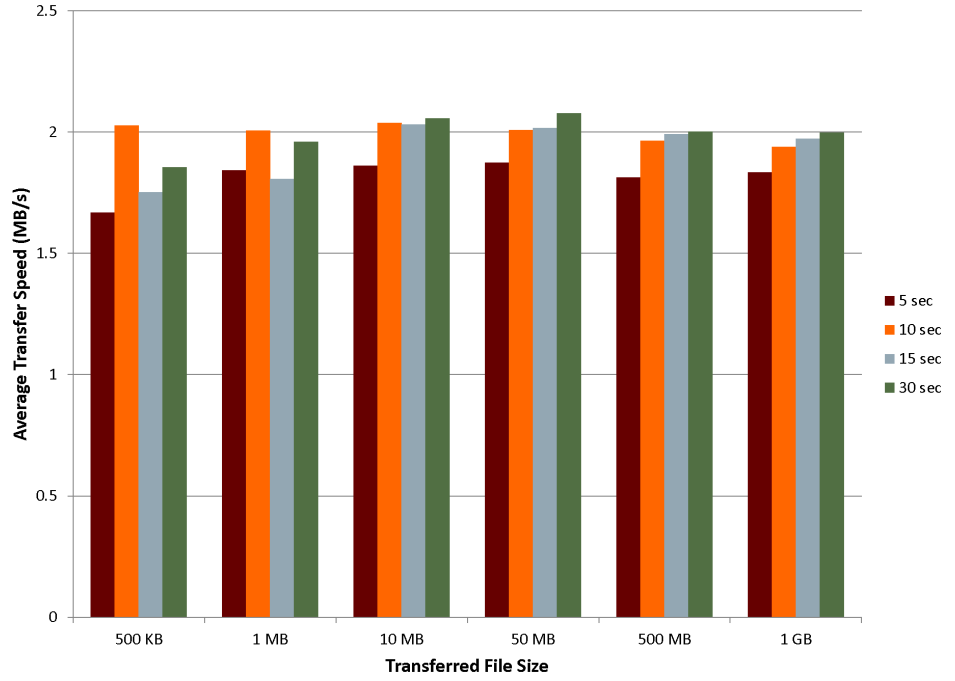


(a) Average RTT

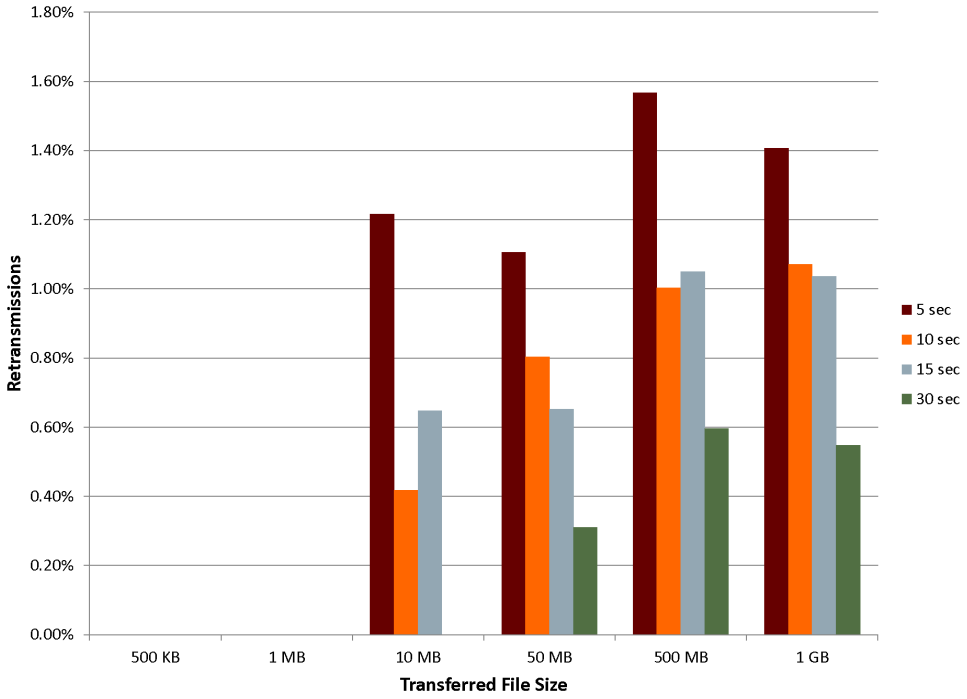


(b) Average packet loss

Figure 10.5: Average speed and packet loss of connectionless traffic over a one-hop simulation network comparing different rotation intervals



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 10.6: Average transfer speed and packet loss of connection-oriented traffic over a one-hop simulation network comparing different rotation intervals

illustrated in Figure 9.7(b). The variability for small file sizes occurs because of when address rotations may occur within each test iteration. Smaller file sizes can typically execute within the span of a single address rotation; depending on the length of the address rotation interval. Depending on when each iteration executes, some files will have an address rotation occur during transfer while others will not. The larger the file transfer size, the less impact an extra address rotation has on overall packet loss.

OPNET also includes capability to visualize network statistics. This capability was leveraged to validate the claim made in Chapter 9 that packet loss occurs only during address rotations. Figure 10.7 illustrates a 100,000 ping flood sent from the client to the server. The graph was generated by the SITL interface on the client side. The red line indicates packets sent by the client while the blue line indicates packets received. The dips in the graph occur every 10 seconds and represent decreased packet volume due to an address rotation. In most cases, the red line is not visible. This indicates received packet volume matching sent packet volume. When the red line is visible during a dip, this usually indicates a packet dropped. The points on the graph where packets are lost, occur only during dips in packet volume. This indicates that loss occurs only during address rotations. The ping flood test took 416 seconds to execute and lost 43 packets. Packet loss matches this claim, since addresses rotated either 41 or 42 times.

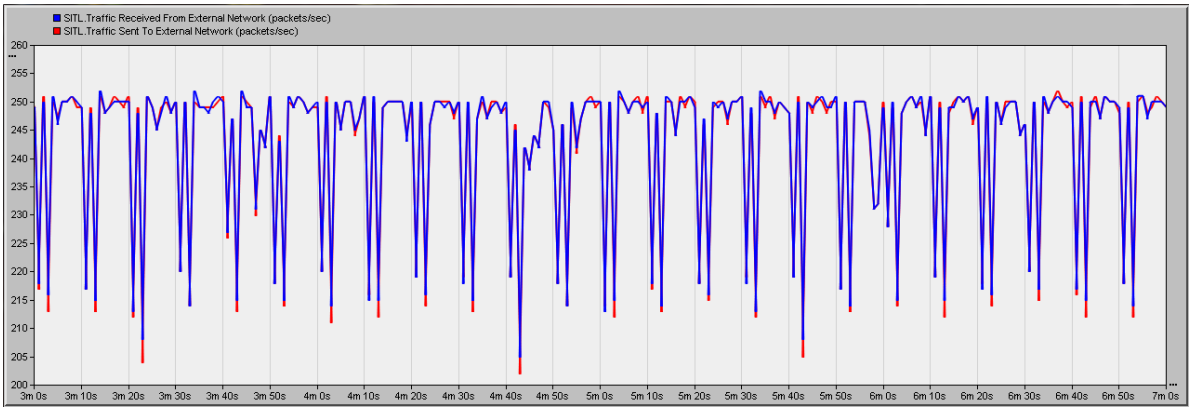


Figure 10.7: OPNET visualization of a 100,000 ping flood using MT6D. The x-axis shows the number of seconds since the simulation started while the y-axis shows volume of packets per second. The red line is the volume of packets sent by the client. The blue line is the volume of packets received by the client.

10.1.2 Two-Hop Testing

A series of tests were also conducted using a simulated two-hop network. These tests were designed to compare performance in simulation with live testing over a two-hop network performed in Chapter 9, Section 9.1.2. The tests in this section are similar to the benchmark tests. A series of tests were run without MT6D to test network performance. Another set of tests was then conducted with MT6D operating on a 10-second address rotation interval. Neither encryption nor i-DKG were enabled due to the lack of support of extension headers by OPNET described in Appendix A.

The two-hop network was built in simulation using two simulated Cisco 7507 routers connected to each other by a 10 gigabit Ethernet link. The client and server were each connected to one of the two routers by a SITL link. The network diagram is illustrated in Figure 10.8. Each SITL link represents a connection to an actual host machine. The same two host machines were used in two-hop simulation testing as were used in all other tests.

Connectionless traffic tests demonstrate that adding more hops does not significantly degrade the performance of MT6D. Figure 10.9(a) shows that MT6D rotating addresses every 10 seconds only incurs an additional 0.3 msec in average RTT over one-hop simulation testing. This extra time is due to the additional processing that occurs at the second hop. This is confirmed by comparing simulated network traffic not using MT6D. The same 0.3 msec increase is seen for normal network traffic without MT6D in Figure 10.9(a) as that of the one-hop simulation network in Figure 10.2(a).

Packet loss was consistent across tests as well. Comparing the loss for MT6D rotating addresses every 10 seconds in Figure 10.9(b) with that of Figure 10.2(b) shows negligible

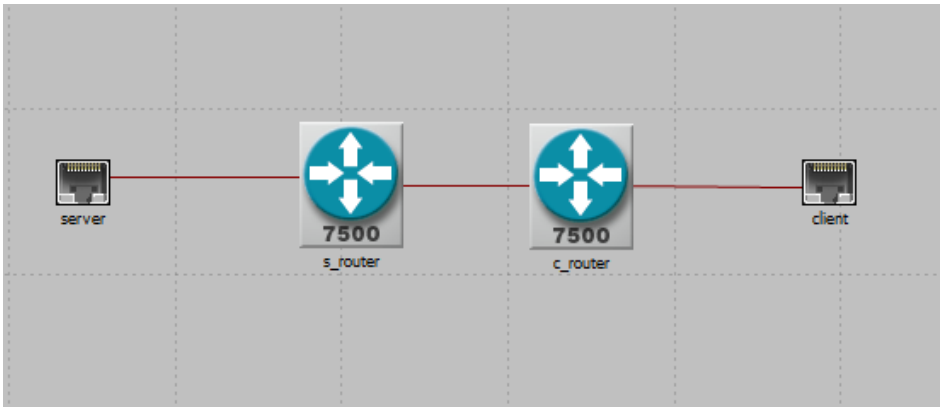
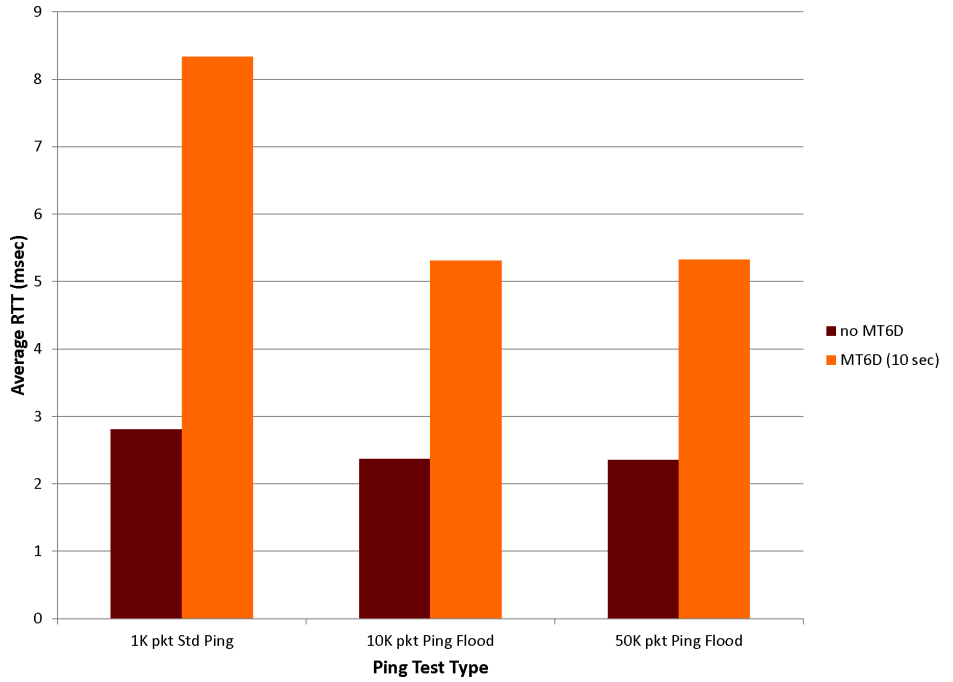
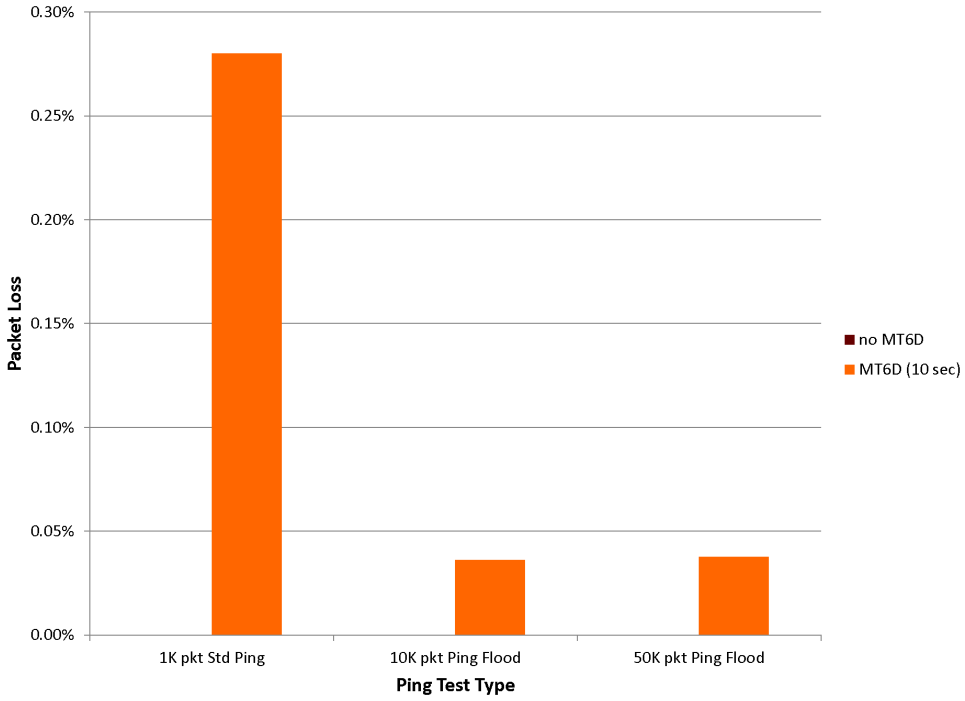


Figure 10.8: OPNET two-hop network configuration



(a) Average RTT



(b) Average packet loss

Figure 10.9: Average speed and packet loss of connectionless traffic over a two-hop simulation network comparing no MT6D with MT6D using the default test settings

difference. The ping flood loss between the one-hop simulation network and the two-hop simulation network was identical. The difference in loss between the two simulation network configurations for the 1,000 packet standard ping flood tests was 0.12%. This additional loss accounts for the two-hop simulation network dropping one additional packet.

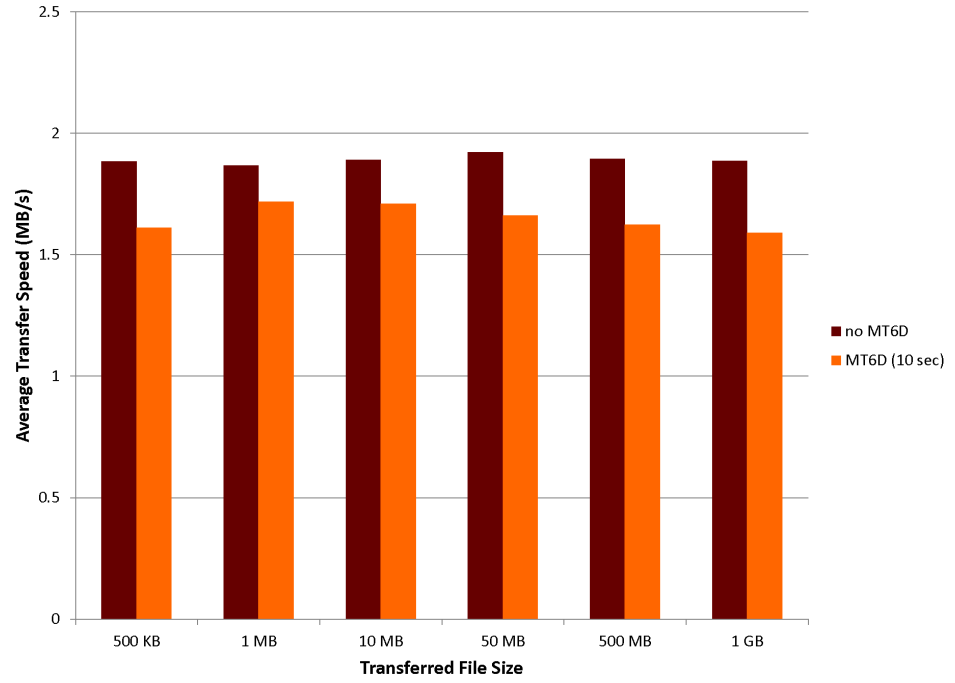
While the average RTTs for MT6D stayed consistent between one-hop and two-hop simulated networks, this was not the case with connection-oriented traffic. The average throughput for one-hop simulation testing using MT6D averaged 2 MB/sec. The throughput for 2-hop simulation testing, as illustrated in Figure 10.9(a), averages just over 1.6 MB/sec. The reason for the decrease in performance is a result of an overall decrease in network performance. The average throughput for normal network traffic not using MT6D averages 1.9 MB/sec as opposed to over 3 MB/sec for one-hop simulation testing. These results demonstrate that MT6D performs close to the maximum network throughput when operating in a environment that is bandwidth-constrained below MT6D's normal throughput. Packet loss for MT6D was consistent between the two simulation network configurations as illustrated in Figures 10.4(b) and 10.10(b).

The live two-hop network discussed in Chapter 9, Section 9.1.2 performed better in terms of throughput than the simulated two-hop network. This was expected due to additional latency imposed by the simulation environment. Queuing delay from the OPNET SITL module accounts for the decrease in simulation throughput for connectionless tests. Reduced throughput of normal network traffic in simulation forces a reduced average throughput for connection-oriented traffic handled by MT6D.

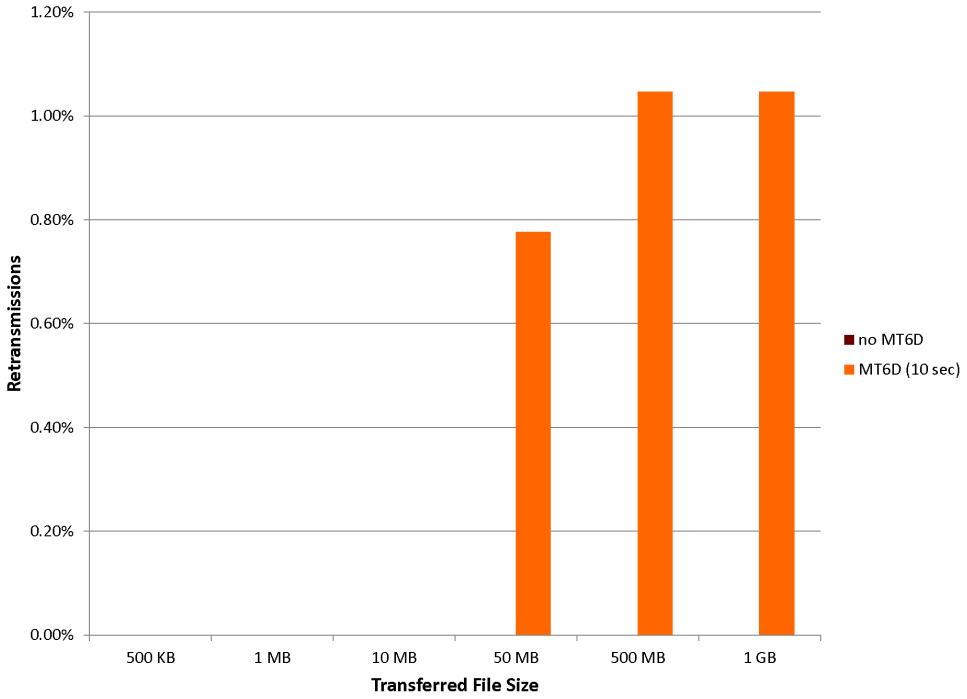
Connection-oriented packet loss for the live two-hop network is almost double the packet loss in simulation. This difference in packet loss demonstrates the extent to which live network conditions impact MT6D. The live 1,000 packet standard ping tests demonstrate this impact. Two-hop live testing results in a standard ping test packet loss over 15% when using MT6D with its default test settings. This loss is a result of routers dropping packets after address rotations, as described in Chapter 9, Section 9.1.2. The same tests in simulation result in an average packet loss of 0.28%. This is further evidence that the packet dropping anomaly is not inherent to MT6D.

10.2 Summary

This chapter tested MT6D performance in simulation. The goal was to determine the performance of OPNET isolated from outside network conditions. Although limitations in



(a) Average transfer speed



(b) Average percentage of retransmitted packets

Figure 10.10: Average transfer speed and packet loss of connection-oriented traffic over a two-hop simulation network comparing no MT6D with MT6D using the default test settings

OPNET's IPv6 model prevented extensive simulation testing, some important results were produced. First, OPNET testing reinforced the claim from Chapter 9 that MT6D performs at approximately 2 MB/sec as long as the network speed is greater than 2 MB/sec. This conclusion supports the claim that optimizing MT6D will improve its performance. OPNET testing also proved that the packet loss observed during live multi-hop network testing was due to the networking equipment, not MT6D. OPNET testing did not produce the excessive packet loss that live testing did. OPNET also provided information about how much SITL queuing delay occurs as packets traverse the simulation network. This information made it possible to isolate the performance impacts of the actual MT6D devices. OPNET's capability to visualize network statistics also validated the claims made in Chapter 9 that packet loss occurs during address rotations.

Chapter 11

Conclusion

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

- Winston Churchill, 1942

This chapter presents a summary of the research conducted in this dissertation, as well as the major contributions of the research. A discussion of some of the limitations of MT6D is also included. There are possible future research directions regarding dynamic address obscuration, which are also presented. Finally, the dissertation concludes with some closing thoughts.

11.1 Summary of Research

The primary goal of this research was to develop a method for protecting the security and privacy of networked hosts by dynamically obscuring their IPv6 addresses. Dynamic obscuration of addresses significantly complicates an adversary's ability to launch targeted network-layer attacks or to track a host's activities on the Internet. Without some sort of defense capability, these threats are very real in IPv6 due to demonstrated security flaws in the addressing schemes. The prototype developed during this research proves that the moving target concept is valid. Test results, both on a live and simulated network, demonstrate that MT6D can handle connectionless, connection-oriented, and even real-time traffic.

The chapters within this dissertation detail and document the research endeavors that establish the validity of a network layer moving target defense in IPv6 and set the groundwork for future research in this area. Chapter 1 introduced the problem under investigation. It

laid out the background that motivated the research as well as the objectives of the research. Key research questions were identified and the methodology to accomplish the research was described.

Chapter 2 provided some background on IPv6. The enhancements of IPv6 over IPv4 were discussed first. A discussion of the IPv6 address format as well as the different address types followed. IPv6 includes two methods for establishing addresses; stateless and stateful. Each method was described in Chapter 2.

Due to the way IPv6 forms addresses, hosts are vulnerable to address tracking and correlation. Chapter 3 discussed how this was possible. The chapter first explained the security and privacy implications of stateless addresses in IPv6. These vulnerabilities were demonstrated on the Virginia Tech campus-wide production IPv6 network. The chapter then explained security vulnerabilities related to stateful address assignment. Since stateless addressing is the default addressing scheme in IPv6, these flaws were demonstrated in a lab environment.

Some researchers have attempted to develop obscuration techniques to address the security issues related to IPv6 address formation. Chapter 4 provided an overview of these techniques. Each technique was divided into one of three categories: static address obscuration techniques, dynamic address obscuration techniques, and address assignment and tunneling techniques. The advantages and disadvantages of each technique were discussed.

Chapter 5 analyzed the benefits of using dynamic over static addressing. The chapter started by discussing the security issues that can result from using static addresses. Analysis into the different factors that improve dynamic addressing followed. There was also a discussion about how these factors interact with each other. The analysis of the benefits of dynamic addressing was backed up with simulation results. The chapter concluded with a discussion of how dynamic addressing improves security.

The next two chapters provided some initial background on MT6D. Chapter 6 gave a basic overview of MT6D. Chapter 7 discussed the threats that MT6D was designed to defend against. The threats were broken down into targeted and unspecified attacks. Targeted attacks were further broken down into passive and active attacks.

Chapter 8 was where the design of MT6D was presented. A discussion was provided on how MT6D forms obscured addresses and why the parameters for obscuring addresses were chosen. A description of the different MT6D tunneling techniques followed. After that, the timing of address rotations and key usage was explained. A technique for dynamically obscuring subnets was also presented. With the groundwork of the design established, the

architecture, flow, and overhead of MT6D was explained. A section followed that addressed how MT6D handles error and control messages. The chapter ended with a look at the different configuration options.

Live testing of the proof of concept prototype was provided in Chapter 9. The goal of testing was to determine the performance impact of MT6D on network traffic. Testing of both the integrated software and stand-alone configurations was conducted. Different MT6D parameters were also varied to establish the impact they had on performance. Additional tests were conducted with hosts a varying number of hops from each other to test the impact routers have on MT6D network traffic. Since it is likely that MT6D would be used with real-time traffic, tests of streaming media and VoIPv6 were conducted.

Simulation testing was conducted in Chapter 10 to test MT6D performance in a controlled and isolated environment. OPNET Modeler with the SITL module was used to build the simulation environment. A number of flaws in how the OPNET SITL module handles IPv6 were discovered while building the IPv6 simulation environment. These flaws are discussed in Appendix A. Some of the same live tests were repeated in simulation. The simulation test results verified the live results.

11.2 Contributions

This dissertation made three contributions to the state-of-the-art IPv6 research. First, it fully explored the security vulnerabilities associated with IPv6 address formation and demonstrated them on a production IPv6 network. Second, it provided a method for dynamically rotating network addresses that defeats these vulnerabilities. Finally, a functioning prototype was presented that proves how network addresses can be dynamically rotated without losing established network connections. Specific topics related to these contributions include the following:

IPv6 Addressing Vulnerabilities The research discussed the security flaws in both stateless and stateful addressing. Attacks that exploited these vulnerabilities were demonstrated on a live IPv6 network. Examples of how an adversary could use these vulnerabilities to his/her advantage were discussed. There was also some discussion on how the size of the IPv6 address space led to the current addressing scheme design.

Secure Addressing in IPv6 Different methods were discussed that could potentially improve the security of IPv6 addressing. Each of these methods was categorized and advantages and disadvantages of each were provided. Conclusions were drawn as to which category of address protection offered the best security improvements.

Dynamic Addressing Benefits A discussion of how dynamic addressing reduces the probability of an attacker targeting a host was provided. The ways in which dynamic addressing limits the damage caused by a targeted attack was also described. An analysis of how subnet size impacts the probability of target detection was used to demonstrate how dynamic addressing is more successful when implemented on an IPv6 network.

Dynamic Addressing Feasibility Proof that IPv6 will support dynamic binding of addresses was demonstrated through design and implementation of a MT6D prototype. Testing confirmed that addresses can rotate mid-session without disrupting any existing communications sessions. Testing also confirmed that both the sender and the receiver are able to maintain synchronized address rotations over prolonged periods of time and under various network traffic conditions. The ability of two communicating hosts to generate new keys "in-band" was demonstrated.

Obscuring the Entire IPv6 Address A discussion of a possible method for obscuring the IPv6 subnet address as well as the IID was described. The benefits and limitations of IPv6 subnet obscuration were provided. Specific benefits included increased anonymity while limitations included a different, broader trust model.

Dynamic Addressing Cost The overhead, latency, and packet loss incurred by MT6D were demonstrated and discussed. Measurements were collected on both a live and simulated network. Some techniques were described and tested that could potentially improve performance.

11.3 Limitations

Despite its strengths, MT6D does have some limitations. Since MT6D is a network-layer defense, it does not prevent attacks that occur at layers lower than the network layer. For example, MT6D cannot prevent adversaries from attempting to capture MAC addresses

since these type of attacks occur at the data link layer. This is not a major issue because lower layer attacks generally require the adversary to be on the same network as the target host. Additionally, MT6D operating in its stand-alone configuration (described in Chapter 8, Section 8.12.2) hides the actual sender's MAC address behind the MT6D device. This is not the case, however, with MT6D operating in embedded configuration (described in Chapter 8, Section 8.12.1) since the external NIC is the actual NIC of the sender's platform. This NIC contains the actual sender's MAC address. It is possible to obscure the MAC, however, MACs cannot change as often as network addresses.

Another limitation of MT6D is that it is designed to operate on an IPv6 network only. Although MT6D could feasibly be redesigned to operate on an IPv4 network, there is not enough unoccupied address space to facilitate it. Since MT6D "hops" between different addresses, employing it on a densely populated subnet would result in numerous collisions. This is not a concern on IPv6 subnets since even heavily populated subnets will not occupy much of the available address space for the foreseeable future. The likelihood of an address collision on an IPv6 subnet is expressed in Equation 8.4.

Including an unsupported communications path (described in Chapter 8, Section 8.8) in MT6D could result in an adversary learning a host's actual address. In environments requiring a high level of security, the unsupported communications path can be disabled. This configuration would be similar to a classified government network that does not allow communication with the public Internet. Although allowing unsupported traffic could expose a host's identity, protection against targeted network attacks can still be provided by limiting what unsupported traffic is allowed to pass.

MT6D operates similar to VPN technologies in that both endpoints are required to have MT6D operating in order to communicate securely. This requirement means MT6D cannot feasibly protect the public Internet. Protection of the public Internet, however, was never a goal of MT6D. MT6D provides protection for point-to-point communications, similar to IPsec. Neither scheme is designed to be a public infrastructure scheme. This does not mean that MT6D cannot be used to provide secure communications with servers. Even public servers can be configured with a MT6D channel to handle nodes that use MT6D as well as those that do not.

As currently designed, MT6D facilitates point-to-point communications only. This means that MT6D cannot support multicast communications. It is possible to adapt MT6D to support multicast. Host using MT6D could support profiles that utilize group symmetric keys. There are two main disadvantages to using group keys. The first disadvantage is that

the trust model grows from just two hosts to the number of hosts in the multicast group. The second disadvantage is that a key compromise compromises the key for all group members.

If an adversary is able to gain access to a MT6D device, it is possible for the adversary to compromise the keys for all hosts the target device communicates with. Since MT6D communicates point-to-point, other communicating pairs that exclude the target node will not be compromised. Even if access to MT6D data is protected against physical device compromise, it may still be susceptible to a power analysis side-channel attack. An optimized version of MT6D should include either algorithm-level, architecture-level, or hybrid countermeasures [72]; depending on how the optimized version of MT6D is implemented.

Chapter 8, Section 8.12.2 shows how MT6D implemented on individual hosts could hamper a network administrator's ability to monitor the behavior of those hosts. Applying MT6D at the network border could protect internal users while still providing network administrators with the capability to monitor internal traffic. With any security mechanism, however, there are trade-offs that would have to be weighed. Users may choose to use MT6D regardless of whether network administrators allow it. This is particularly true of malicious users. It is important that administrators become aware of technologies like MT6D and how they can impact their networks.

11.4 Future Research Directions

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 should 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 that allows parallel execution. 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.

DSO (described in Chapter 8, Section 8.7) was not included in the prototype design since it adds little to the validity of the proof of concept. Although the concept for DSO was presented in this research, there are likely issues with the design that require future research

and testing. The two most likely issues result from added latency and the assumption of connected trust relationships. Each time a packet is forwarded to another intermediate node, latency is incurred. If many intermediate nodes are involved in forwarding each packet, the added latency may push packet delivery past the current and into the next address rotation interval. As a result, packets would be dropped. If this occurred regularly, network communications would be severely degraded. Limiting the number of intermediate nodes that handle a packet would help ensure packets are not delivered past the current address rotation interval, but would also increase packet loss; possibly past a tolerable level. The lack of a connected trust relationship between hosts may also result in excessive packet loss. Connected trust refers to the assumption that each host, through a chain of other trusted hosts, shares a transitive trust relationship. In reality, there may not be enough hosts using MT6D to establish a connected trust relationship. If no connected trust relationship exists, packets cannot be delivered to their destination. In scenarios where MT6D-enabled hosts frequently communicate, it may be feasible to establish relay points scattered throughout the Internet that ensure connected trust.

11.5 Concluding Thoughts

This work presented a means for dynamically obscuring IPv6 addresses. Obscuring addresses in IPv6 is important due to vulnerabilities that exist in the proposed IPv6 addressing schemes. One objective of this research was to present these vulnerabilities and then analyze and demonstrate them. This was accomplished in Chapter 3. A dynamic addressing solution solves the problems presented by IPv6 addressing. Chapter 5 proved this to be true. Based on this research, a method for dynamically obscuring addresses, called MT6D, was developed. Designing MT6D was the main objective of this research. The design of MT6D was described in Chapter 8. A prototype version of MT6D was developed to prove the validity of the design. The results of preliminary testing were presented in Chapters 9 and 10. The results of this testing proved that MT6D can rotate addresses multiple times per session without disturbing existing sessions.

This research resulted in ten conference publications, two journal articles, two awards, a research gift, and a filed patent. Different aspects of the research related to the flaws in stateless address configuration were presented at the *6th Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)* [32], the *44rd Hawaii International Conference on Systems Science (HICSS-44)* [33], the *Tenth International Conference on Networks (ICN*

2011) [21], and the *Ninth Annual Conference on Communication Networks and Services Research (CNSR 2011)* [24]. Research related to stateful addressing flaws was presented at the *World Congress on Internet Security (WorldCIS-2011)* [35] and the *16th North-East Asia Symposium on Nano, Information Technology and Reliability* [80]. The benefits of dynamic addressing as a security countermeasure were presented at the *6th International Conference on Information Warfare and Security (ICIW 2011)* [34] while the impact of dynamic addressing on network forensics was published in the *Journal of Network Forensics* [20]. The MT6D design and preliminary results using the stand-alone configuration were presented at the *2011 Military Communications Conference (MILCOM)* [22]. Additional aspects of the MT6D design, as well as results from running the integrated configuration, have been accepted for future publication in the *IEEE Security and Privacy Magazine* [23]. Research was also presented at the *2012 IEEE PES Innovative Smart Grid Technologies Conference* [36] on applying MT6D to protect the Smart Grid. An evaluation of the OPNET's capability to pass IPv6 traffic will be presented at the *Eighth Advanced International Conference on Telecommunications (AICT 2012)* [8]. Research on MT6D also won first place at the Virginia Tech 2011 Graduate Student Assembly Research Symposium and third place at the 2011 National Security Innovation Competition. A research award in the amount of $30,000 was presented by SAIC to further MT6D research. Recently, an international patent was filed by Virginia Tech Intellectual Properties on the MT6D technology.

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. MT6D is also able to secure peripherals, such as printers and faxes. Typically, these devices are not protected against attack. As such, they are easy targets for attackers. MT6D can transparently protect these devices without the need for expensive modifications to the peripheral. This technology can also be used to provide secure communications for military/intelligence agents, corporate entities, or for e-commerce. VPN technologies can benefit from MT6D as well. Current VPNs are susceptible to DoS at the endpoints. A VPN using MT6D will have moving endpoint that an attacker statistically cannot target. With migration to IPv6 on the horizon, it is important to prepare the next generation of IPv6 security solutions. MT6D is one of these solutions.

Bibliography

[1] J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971 (Proposed Standard), March 2005.

[2] Attorney General to the Vice President. Cyberstalking: A new challenge for law enforcement and industry. Available at: `http://www.justice.gov/criminal/cybercrime/cyberstalking.htm` accessed on 2 June 2010.

[3] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), March 2005. Updated by RFCs 4581, 4982.

[4] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management part 1: General. In *NIST Special Publication 800-57, National Institute of Standards and Technology. Available at http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part1.pdf*, August 2005.

[5] David Beazley. Understanding the Python GIL. Presented at PyCon 2010, February 2010.

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

[7] A.R. Choudhary. In-depth analysis of IPv6 security posture. In *The 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2009)*, pages 1 –7, November 2009.

[8] Brittany Clore, Matthew Dunlop, Randy Marchany, and Joseph Tront. An evaluation of IPv6 in simulation using OPNET modeler. In *Eighth Advanced International Conference on Telecommunications (AICT 2012), to appear*, Stuttgart, Germany, May 2012.

[9] Brian T. Contos, Colby Derodeff, William P. Crowell, and Dan Dunkel. *Physical and Logical Security Convergence: Powered By Enterprise Security Management*. Syngress, 2007.

[10] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 156–170, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] cProfile. Available at: `http://docs.python.org/release/2.5.2/lib/module-profile.html`, accessed on 28 February 2012.

[12] Cryptography and encryption. Available at: `http://www.cryptoforge.com/security.htm`, accessed on 28 February 2012.

[13] Richard Deal. *Cisco Certified Network Associate Study Guide (Exam 640-802)*. The McGraw-Hill Companies, 2008.

[14] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722.

[15] Department of Homeland Security. Espionage: Reporting potential indicators. Available at: `http://www.dhs.gov/xlibrary/assets/ocso-espionage-brochure.pdf` accessed on 10 June 2010.

[16] Andrew Dowell and Roger Cheng. AT&T Dials Up Limits on Web Data. Available at: `http://online.wsj.com/article/SB10001424052748703561604575282173014134754.html`, June 2010.

[17] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315 (Proposed Standard), July 2003. Updated by RFCs 4361, 5494.

[18] Matthew Dunlop. Introduction to Internet Protocol version 6 (IPv6). Available at: `http://filebox.ece.vt.edu/~jgtront/IPv6/index.html`, December 2009.

[19] Matthew Dunlop. IPv6 addressing. Available at: `http://filebox.ece.vt.edu/~jgtront/IPv6/index.html`, December 2009.

[20] Matthew Dunlop, Stephen Groat, Randy Marchany, and Joseph Tront. IPv6 dynamic network addressing and its impact on network forensic investigations. *Journal of Network Forensics*, 3(1):6–18, 2011.

[21] Matthew Dunlop, Stephen Groat, Randy Marchany, and Joseph Tront. IPv6: Now you see me, now you don't. In *the Tenth International Conference on Networks (ICN 2011)*, January 2011.

[22] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. MT6D: a moving target IPv6 defense. In *the 2011 Military Communications Conference (MILCOM)*, pages 1321–1326, Baltimore, Maryland, November 2011.

[23] Matthew Dunlop, Stephen Groat, William Urbanski, Randy Marchany, and Joseph Tront. The blind man's bluff approach to security using IPv6. *Security Privacy, IEEE*, PP(99):1, 2012.

[24] Matthew Dunlop, Stephen L Groat, Randy Marchany, and Joseph G Tront. The good, the bad, the IPv6. In *Ninth Annual Conference on Communication Networks and Services Research (CNSR 2011)*, Ottawa, Canada, May 2011.

[25] Industrial espionage. Available at `http://www.britannica.com/EBchecked/topic/287033/industrial-espionage` accessed on 10 June 2010. Encyclopedia Britannica.

[26] 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.

[27] Federal Bureau of Investigation. Electronic surveillance needs for carrier grade voice over packet (CGVoP) service, January 2003. CALEA Implementation.

[28] Russell Andrew Fink, Matthew Aloysius Brannigan, Shelby Alana Evans, Aswin Morgan Almeida, and Shelley Anne Ferguson. Method and apparatus for providing adaptive self-synchronized dynamic address translation. Patent, May 2006. US 7,043,633 B1.

[29] Parijat Garg and Rahul Singhai. QoS in VoIP, n.d.

[30] GLORIAD average round trip time - last week. Available at: `http://www.gloriad.org/gloriad/monitor/stats/avg_round_trip_time.week.html` accessed on 11 October 2010.

[31] Kate Greene. Long-distance Wi-Fi. *MIT Technology Review*, March 2008.

[32] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. The privacy implications of stateless IPv6 addressing. In *the 6th Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW)*, April 2010.

[33] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. IPv6: Nowhere to run, nowhere to hide. In *the 44th Hawaii International Conference on Systems Science (HICSS-44 2011)*, January 2011.

[34] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. Using dynamic addressing for a moving target defense. In *the 6th International Conference on Information Warfare and Security (ICIW 2011)*, Washington DC, March 2011.

[35] Stephen Groat, Matthew Dunlop, Randy Marchany, and Joseph Tront. What DHCPv6 says about you. In *the World Congress on Internet Security (WorldCIS-2011)*, February 2011.

[36] Stephen Groat, Matthew Dunlop, William Urbanski, Randy Marchany, and Joseph Tront. Using an IPv6 moving target defense to protect the smart grid. In *IEEE PES Innovative Smart Grid Technologies Conference*, Washington, DC, January 2012.

[37] Patrick Grossetete, Ciprian Popoviciu, and Fred Wettling. *Global IPv6 Strategies: From Business Analysis to Operational Planning*. Cisco Press, May 2008.

[38] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006.

[39] 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.

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

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

[42] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.

[43] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[44] Dorene Kewley, Russ Fink, John Lowry, and Mike Dean. Dynamic approaches to thwart adversary intelligence gathering. *DARPA Information Survivability Conference and Exposition,*, 1:0176, 2001.

[45] David Levin. Lessons learned in using live red teams in IA experiments. *DARPA Information Survivability Conference and Exposition*, 1:110, 2003.

[46] John Leyden. Net shakeup looms as IPv4 resources start running low. *The Register*, June 2010.

[47] Eric Mankin. Last IP address going, going, gone. *Futurity*, February 2011.

[48] Carolyn Duffy Marsan. 10 fool-proof predictions for the Internet in 2020. *Network World*, January 2010.

[49] Ian McCulloh. *Detecting Changes in a Dynamic Social Network*. PhD thesis, Carnegie Mellon University, March 2009. CMU-ISR-09-104.

[50] Robert L. Mitchell. The grill: John Curran. *ComputerWorld*, April 2010.

[51] David Morton. Understanding IPv6. *PC Network Advisor*, 83:17–22, 1997.

[52] Tomasz Mrugalski and Marek Senderski. DHCPv6: Dibbler - a portable DHCPv6. Available at: `http://klub.com.pl/dhcpv6/` accessed on August 2010.

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

[54] Edmund Colby Munger, Douglas Charles Schmidt, Robert Dunham Short III, Victor Larson, and Michael Williamson. Agile network protocol for secure communication with assured system availability. Patent, February 2003. US 2003/0037142 A1.

[55] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.

[56] T. Narten, R. Draves, and S. Krishnan. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941 (Draft Standard), September 2007.

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

[58] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007.

[59] National Archives and Records Administration. *Code of Federal Regulations, Title 28, Section 0.85*, volume 1. U.S. Government Printing Office, July 2001.

[60] National Institute of Standards and Technology (NIST). *Secure Hash Standard*, August 2002. Federal Information Processing Standards Publication 180-2.

[61] ns-3: IPv6 Class Reference. Available at: `http://www.nsnam.org/doxygen/classns3_1_1_ipv6.html`, accessed on 25 January 2012.

[62] OMNeT++ IPv6 Suite. Available at: `http://www.omnetpp.org/omnetpp/doc_details/2137-ipv6suite`, accessed on 25 January 2012.

[63] OPNET: IPv6 for R&D specialized model. Available at: `http://www.opnet.com/solutions/network_rd/simulation_model_library/ipv6.htm`, accessed on 12 December 2011.

[64] OPNET System-in-the-Loop (SITL) module. Available at: `http://www.opnet.com/solutions/network_rd/system_in_the_loop.html`, accessed on 12 December 2011.

[65] OPNET Technologies, Inc. *OPNET Modeler Documentation Set Version 16.0*, October 2010.

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

[67] Fahmida Y. Rashid. IPv4 address depletion adds momentum to IPv6 transition. *eWEEK.com*, February 2011.

[68] Juliano Rizzo and Thai Duong. Practical padding oracle attacks. In *Proceedings of the 4th USENIX conference on Offensive technologies*, WOOT'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[69] RunSnakeRun. Available at: `http://www.vrplumber.com/programming/runsnakerun/`, accessed on 28 February 2012.

[70] Rachid Saadi, Mohammad Ashiqur Rahaman, Valérie Issarny, and Alessandra Toninelli. Composing trust models towards interoperable trust management. In *5th IFIP WG 11.11 International Conference on Trust Management*, volume 358, pages 51–66, Copenhagen, Denmark, June 2011. Springer Boston.

[71] Scapy. Available at: `http://www.secdev.org/projects/scapy/`, accessed on 22 February 2012.

[72] Patrick Schaumont and Zhimin Chen. Side-channel attacks and countermeasures for embedded microcontrollers. In Mohammad Tehranipoor and Cliff Wang, editors, *Introduction to Hardware Security and Trust*, pages 263–282. Springer New York, 2012. 10.1007/978-1-4419-8080-9_11.

[73] Bruce Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.

[74] Claude E. Shannon. A mathematical theory of communications. *The Bell Systems Technical Journal*, 27, 1948.

[75] Victor I. Sheymov. Method and communications and communication network intrusion protection methods and intrusion attempt detection system. Patent, February 2010. US 2010/0042513 A1.

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

[77] William Stallings. *Cryptography and network security (2nd ed.): principles and practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[78] STRATFOR Global Intelligence. The terrorist attack cycle: Selecting the target, September 2005.

[79] tcpdump & libpcap. Available at: `http://www.tcpdump.org/` accessed on 1 February 2012.

[80] Joseph Tront, Stephen Groat, Matthew Dunlop, and Randy Marchany. Security and privacy produced by DHCP unique identifiers. In *the 16th North-East Asia Symposium on Nano, Information Technology and Reliability*, pages 170 – 179, Macau, October 2011.

[81] U.S. Department of Energy. Smart grid. Available at: `http://www.oe.energy.gov/smartgrid.htm` accessed on 9 June 2010.

[82] Iljitsch van Beijnum. River of IPv4 addresses officially runs dry. *Ars Technica*, February 2011.

[83] Introduction to IP version 6. Available at: `http://download.microsoft.com/download/e/9/b/e9bd20d3-cc8d-4162-aa60-3aa3abc2b2e9/ipv6.doc` accessed on 24 May 2010.

[84] U.S. & World population clocks. Available at: `http://www.census.gov/population/www/popclockus.html/` accessed on 4 March 2010.

[85] John Xu. AES CBC or AES CTR mode . Available at: `http://johnx.blogspot.com/2010/10/aes-cbc-or-aes-ctr-mode.html`, October 2010.

Appendix A

IPv6 Network Modeling Using OPNET

OPNET Modeler [65] with its System-in-the-Loop (SITL) module and IPv6 specialized model was used for simulation testing in Chapter 10. OPNET was chosen because it supports both IPv6 in simulation and the ability to connect real devices to a simulation network. This appendix provides a short overview on OPNET followed by a discussion of some of the limitations of its IPv6 model. The limitations described in this appendix are by no means comprehensive. They comprise only those issues identified during testing of MT6D in simulation.

A.1 Overview

OPNET modeler is a tool for network modeling and simulation. It provides the ability to rapidly construct networks of any size and measure network performance under a variety of conditions. OPNET includes a Graphical User Interface (GUI) that makes network configuration easy and intuitive. The GUI also provides visual performance statistics. OPNET also has a number of add-on models to its base Modeler product. One of those models is the IPv6 specialized model [63] taken advantage of in Chapter 10.

OPNET contains a series of process models that represent network links and devices. Each process model contains a series of processes connected by either Interface Control Information structures (ICIs) or event states. ICIs and event states are user-defined data items that transfer data between processes. Processes in OPNET are defined by finite state machines that are represented by state transition diagrams. States and transitions represent

blocks of C code. The benefit of this structure is that states and transitions can be easily reused to form new custom processes. For those states and transitions that do not exist, it is possible for users to write their own C code to develop them. Users can use combinations of custom and predefined processes along with ICIs and event states to define their own process models. Despite the ability to develop new process models and even to customize existing process models, there are a number of proprietary algorithms and models that users cannot modify.

OPNET also includes a SITL module that allows real devices to connect to simulated networks. The SITL module contains a specialized process model that listens on a real network interface. Real traffic from the actual network interface is filtered by the SITL process model and translated into simulated traffic. The opposite occurs for simulated traffic destined for the live network. Filtering is done using the Berkeley Packet Filter syntax [65]. The SITL module currently supports translation of only common protocols including: Ethernet, IPv4, IPv6, ICMP, ICMPv6, TCP, UDP, Open Shortest Path First (OSPF), and Routing Information Protocol (RIP) [65].

The SITL module operates in three different configurations: Live-Sim, Live-Sim-Live, and Sim-Live-Sim [64]. Live-Sim allows a physical device to interface with a simulated network and/or simulated hosts. Live-Sim-Live allows a physical device to connect to another physical device through a simulated network. Sim-Live-Sim allows two simulated networks to connect through one or more physical devices. The last configuration is typically used for hardware testing, in particular scalability testing.

A.2 IPv6 Limitations

OPNET includes basic IPv6 functionality, but has a number of limitations that reduce its ability to be useful for specialized or large-scale IPv6 testing. The first limitation of IPv6 in OPNET is specific to SITL. It is an OPNET coding error that prevents IPv6 from functioning in simulation altogether. This issue is caused by an invalid use of the `type` field in the ICMPv6 packet header. The `type` field is eight bits long as shown in Figure A.1. OPNET statically assigns the different `type` values using an integer data type. Since the integer data type is 32-bits long, OPNET crashes when forming ICMPv6 packets as demonstrated in Figure A.2. It was possible to mask this value to eight bits prior to the ICMPv6 packet forming. The masked `type` value allowed IPv6 to function in OPNET using the SITL module.

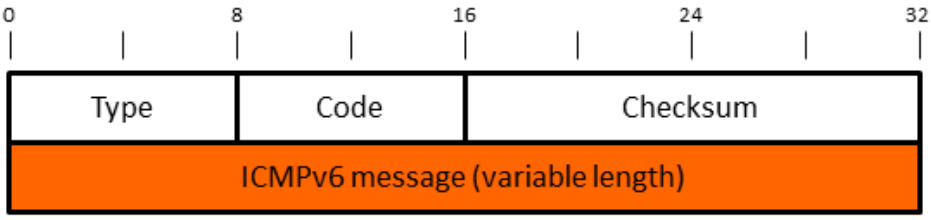


Figure A.1: ICMPv6 header format

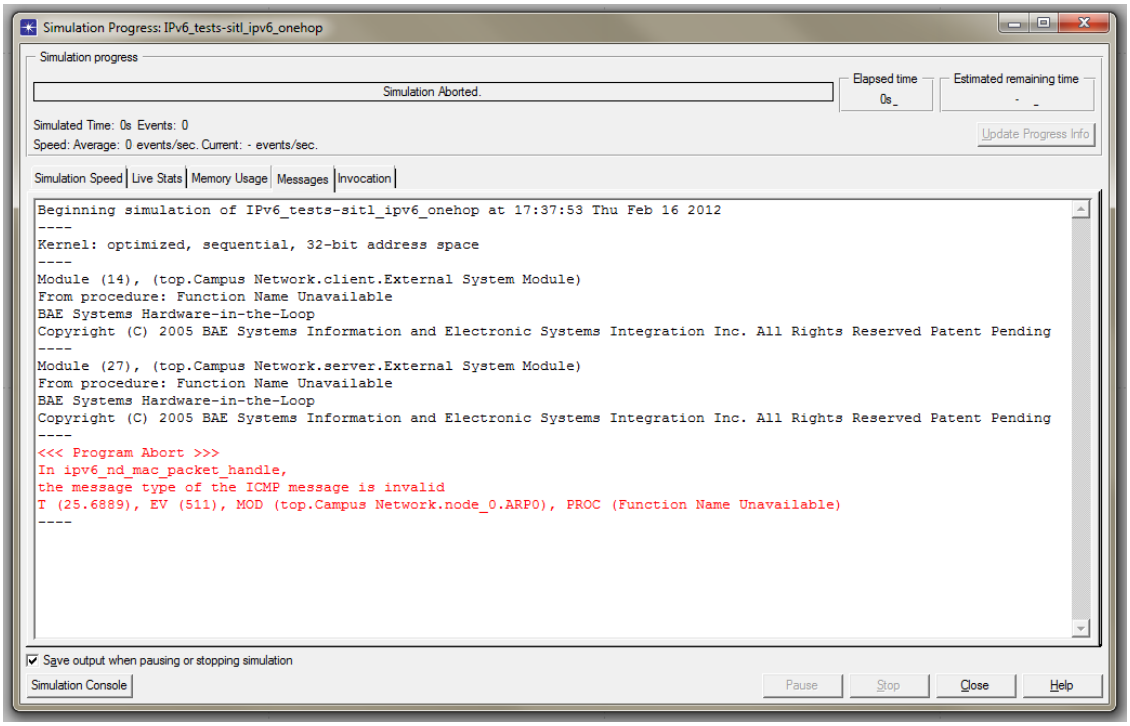


Figure A.2: ICMPv6 messages cause OPNET to abort due to a data type mismatch

Another fairly significant limitation specific to using the OPNET SITL module is that router advertisements are malformed. IPv6 router advertisements include a 32-byte `Prefix Information` field used to notify hosts of the subnet prefix. Router advertisements translated from SITL do not include this field as illustrated in Figure A.3. Router advertisements get dropped by the physical host as a result. The physical host is unable to form a valid global IPv6 address and is also unable to determine a gateway device without router advertisements. This error prevents a host from communicating. Static addresses, gateways, and routes have to be set for physical hosts to overcome this limitation. Each time a new gateway is added

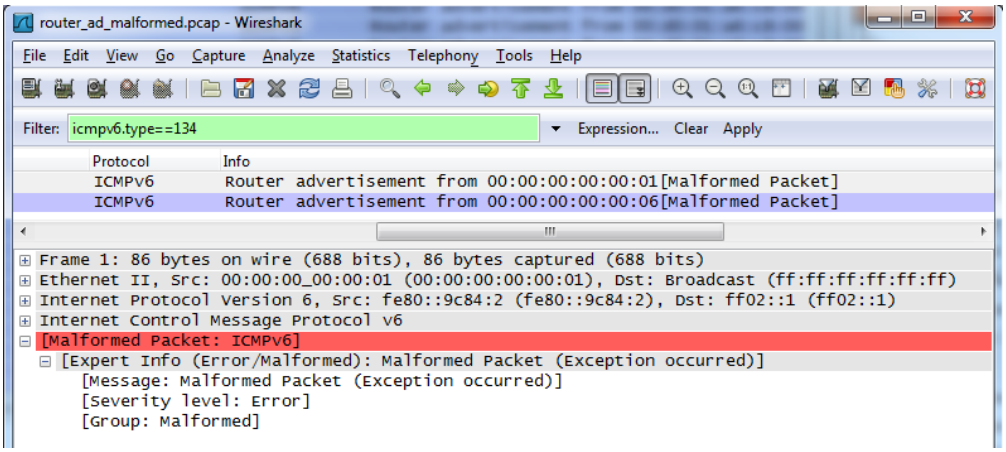


Figure A.3: Malformed IPv6 router advertisement

or removed from a simulation network, the gateway and routes have to be statically set. If a host moves to a new simulation subnet, these values all have to be physically reset. This issue does not occur with hosts within the simulation. When a simulated IPv6 host is configured with only a link local address, it is able to correctly form its global address and send packets to a remote destination. At the start of the simulation, the router sends out an ICMPv6 router advertisement packet with a destination of `ff01::1` indicating the message is for all nodes. Figure A.4 shows the router advertisement contains the fields required by the `Prefix Information` field. It is not clear from the figure how the simulated packet is laid out. Subsequent data packets from the host contain the correct global address indicating that the host did, in fact, get the correct prefix information. It could be that within the simulation the packet is not actually formed and the `Prefix Information` field gets corrupted during SITL translation.

Two other limitations are not code errors, but rather lack of development in the OPNET IPv6 model. The first limitation in development is that OPNET does not support any but the most basic extension headers. The Scapy packet manipulation tool [71] was used to generate packets with various extension headers in order to test OPNET's ability to support IPv6 extension headers. The only confirmed extension headers supported by OPNET are the three upper layer extension headers (TCP, UDP, and ICMPv6). Without support of other extension headers, IPsec will not function, fragmentation is not possible, as well as other important IPv6 functions. The impact on MT6D testing is that none of the features that use the destination options extension header can be tested in simulation. The second limitation, specific to SITL, is that currently no IPv6 routing protocols are supported. This did not

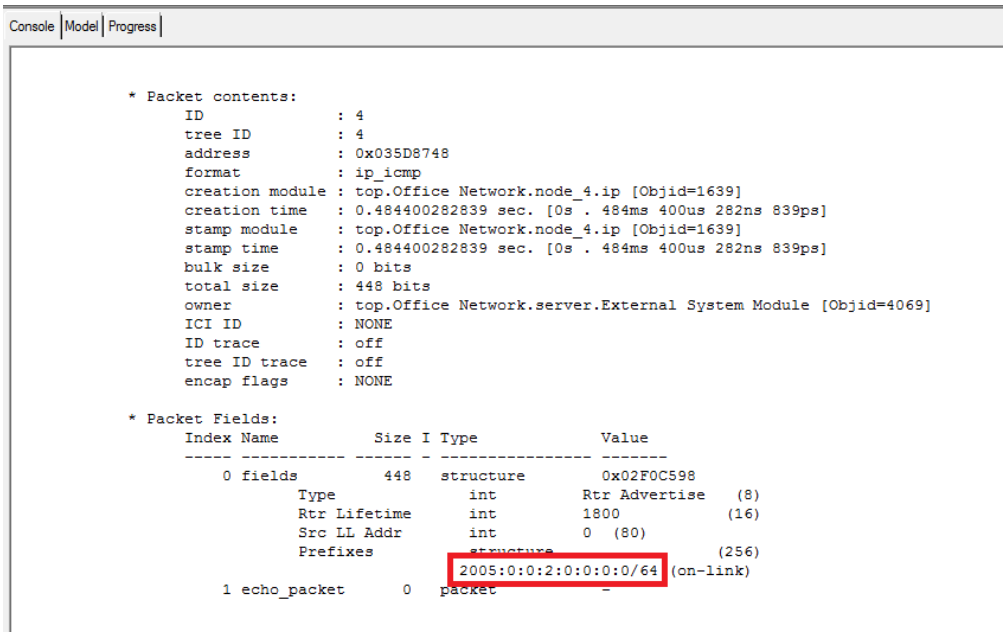
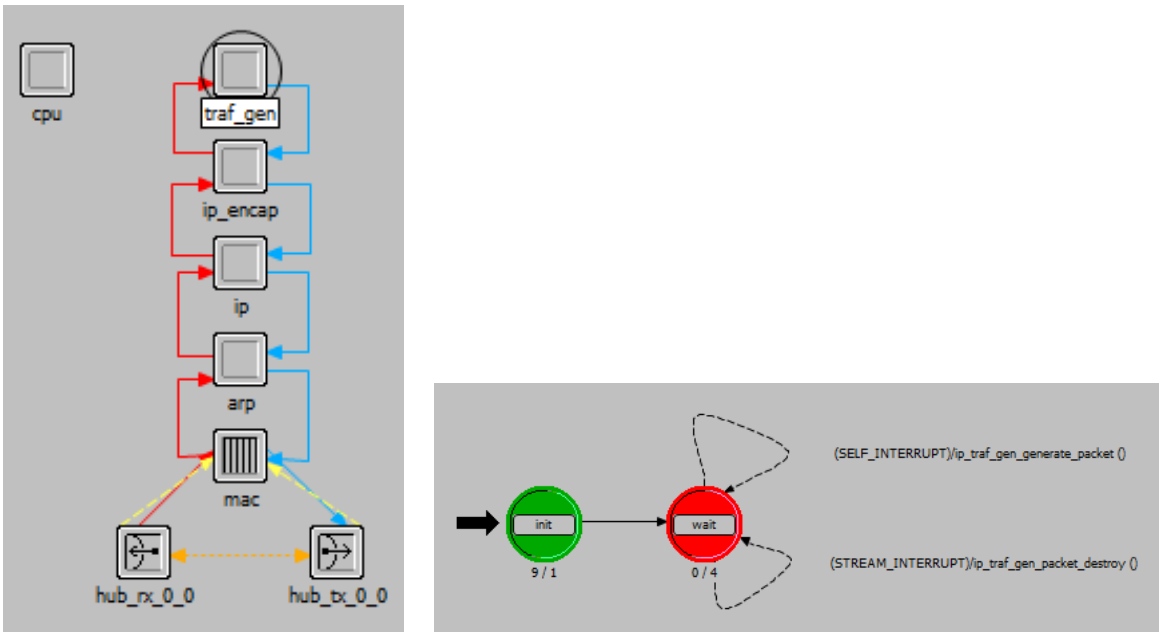


Figure A.4: OPNET simulated router advertisement. The information inside the red box corresponds to the `Prefix` and `Prefix Length` fields within the `Prefix Information` field of an actual router advertisement.

cause an issue with the simulation testing performed in Chapter 10 because all routing was contained within the simulation environment.

A final limitation that impacted testing of MT6D using OPNET has to do with generation of background traffic. Background traffic can be generated in one of three ways; none of which operate in IPv6 without flaws. The primary method of traffic generation is through a flow-based background traffic generator that simulates traffic from groups of nodes. This method does not function at all. When run using IPv6, no traffic is generated and a memory shortage error occurs. The second traffic generation method is through using a traffic generation process (`traf_gen`) within each individual node. The `traf_gen` process is illustrated in Figure A.5(a) while the state diagram for the process is shown in Figure A.5(b). Although setting up background traffic in this manner is tedious, once a node is set up it can be duplicated to increase traffic volume on the network. This technique for traffic generation, does not work, however, in IPv6. Investigation into the issue revealed that the source and destination IPv6 addresses are invalid. The error was caused by improper data typing. Within the code, OPNET is expecting 32-bit integer data types for the address fields. This works fine with IPv4 addresses since they are 32 bits. This does not work for 128-bit



(a) OPNET node process diagram with traffic generation process

(b) State diagram of the traffic generation process

Figure A.5: Process and State diagrams of OPNET's explicit traffic generation capability

IPv6 addresses. OPNET rewrote portions of the code to hold IPv6 addresses in structured data types. The `traf_gen` process was not modified to accept this structure. The final traffic generation technique is application-specific generation processes. OPNET includes the capability to configure application generation processes that can generate traffic for specific applications. This generation technique seems to function with IPv6. When used with SITL, however, it crashes after a while due to an invalid memory access error.

A.3 Summary

Although this appendix highlights the limitations OPNET has with handling different aspects of IPv6, it should be noted that, at the time of this writing, OPNET provides the **most comprehensive** support for IPv6 of any of the simulation tools available. Despite any limitations, OPNET can successfully pass IPv6 traffic. The list of simulation tools claiming this capability is short. One key feature that sets OPNET apart from all other simulation tools is its ability to tie live IPv6 devices into a simulated IPv6 network with its SITL module.

The research in this dissertation is focused on an emerging protocol. There is comparatively little existing research on IPv6. Many applications as well as most firewalls and intrusion detection systems do not currently support IPv6. The challenges OPNET has with dealing with the intricacies of IPv6 are not a shortcoming of OPNET, but rather initiative and vision of the developers to begin preparing for an upcoming market need.