A Pragmatic Approach: Achieving Acceptable Security Mechanisms for High Speed Data Transfer Protocol- UDT

Danilo Valeros Bernardo, Doan B Hoang
i-Next Faculty of Engineering and Information Technology, The University of Technology Sydney, Australia
E-mail:bernardan@gmail.com, dhoang@it.uts.edu.au

Abstract

The development of next generation protocols, such as UDT (UDP-based data transfer), promptly addresses various infrastructure requirements for transmitting data in high speed networks. However, this development creates new vulnerabilities when these protocols are designed to solely rely on existing security solutions of existing protocols such as TCP and UDP. It is clear that not all security protocols (such as TLS) can be used to protect UDT, just as security solutions devised for wired networks cannot be used to protect the unwired ones. The development of UDT, similarly in the development of TCP/UDP many years ago, lacked a well-thought security architecture to address the problems that networks are presently experiencing. This paper proposes and analyses practical security mechanisms for UDT.

Keywords: Next Generation, GSS-API, High Speed Bandwidth, UDT, HIP, CGA, SASL, DTLS, AO

1. Introduction

The prevalent and significant development of advanced high speed networks has created opportunities for new technology to prosper.

Recent developments in network research introduced an enhanced version of UDT, considered to be one of the next generation of high performance data transfer protocols [10]. UDT introduces a new three-layer protocol architecture that is composed of a connection flow multiplexer, enhanced congestion control, and resource management. The new design allows protocol to be shared by parallel connections and to be used by future connections. It improves congestion control and reduces connection set-up time.

UDT provides better usability by supporting a variety of network environments and application scenarios [8,9,10,11]. It addresses TCP's limitations by reducing the overhead required to send and receive streams of data.

One compelling example of the implementation of UDT is the Sloan Digital Sky Survey (SDSS) project [46,49], which is mapping in detail one quarter of the entire sky, determining the positions and brightness of more than 300 million celestial objects. It measures distances to more than a million galaxies and quasars. The data from the SDSS project so far has increased to 2 terabytes and continues to grow. Currently, this 2 terabytes of data is being delivered to the Asia-Pacific region, including Australia, Japan, South Korea, and China. Astronomers also want to execute online analysis on multiple datasets stored in geographically distributed locations [17]. This implementation offers a promising direction for future high speed data transfer in various industries.

Securing data during its operations across network layers is therefore imperative in ensuring UDT itself is protected when implemented. The challenge to reduce the cost and complexity of running streaming applications over the Internet and through wireless and mobile devices, while maintaining security and privacy for their communication links, continues to mount.

The absence of a well-thought security mechanism for UDT in its development, however, drives this paper to introduce ways to secure UDT in a few implementation scenarios.

This paper presents application and IP-based mechanisms and a combination of existing security solutions of existing layers that may assist in further enhancing the work earlier presented by Bernardo [8,9,10,11] for UDT.

Bernardo [8] presented a framework which adequately addresses vulnerability issues by implementing security mechanisms in UDT while maintaining transparency in data delivery. The development of the framework was based on the analyses drawn from the source codes of UDT found at SourceForge.net. The source codes were analyzed and tested on Win32 and Linux environments to gain a better understanding on the functions and characteristics of this new protocol.

Network and security simulations using NS2 [37] and the Evaluation Methods for Internet Security Technology tool (EMIST) developed at the Pennsylvania State University with support from the US Department of Homeland Security and the National Science Foundation [38], were performed. Most of the security vulnerability testing, however, was conducted through simple penetration and traffic load tests. The results provided significant groundwork in the development of a proposal for a variety of mechanisms to secure UDT against various adversaries, such as Sybil, addresses, man-in-the-middle and the most common, DoS attacks.

This paper discusses these mechanisms, their simulation and implementation in a controlled environment. The discussion is categorized in the following format. In Section 2 of this paper, the authors present an overview of UDT [17]. More details on UDT and its architecture were discussed by Bernardo in his early works [8,9,10,11]. His works were drawn from and mainly influenced by the works of Gu [17], who developed UDT at the National Data Mining Centre in the US. Also in this section, the descriptions of the proposed security designs and implementation drawn from the initial work performed by Bernardo, and the motivation behind his work, are presented. In Sections 3, 4 and 5, new security approaches are presented. Section 6 discusses the conclusion of the paper and future work.

2. Overview

UDT is a connection-oriented duplex protocol [17], which supports data streaming and partial reliable messaging. It also uses rate-based congestion control (rate control) and window-based flow control to regulate outgoing traffic. This was designed such that rate control updates the packet sending period every constant interval, whereas flow control updates the flow window size each time an acknowledgment packet is received. It was expanded to satisfy more requirements for both network research and applications development [8-11,17]. This expansion is called Composable UDT and is designed to complement the kernel space network stacks. However, this feature is intended for:

- Implementation and deployment of new control algorithms. Data transfer through private links can be implemented using Composable UDT.
- Composable UDT supports application-aware algorithms.

- Ease of testing new algorithms for kernel space when using Composable UDT compared to modifying an OS kernel.

The Composable UDT library implements a standard TCP Congestion Control Algorithm (CTCP). CTCP can be redefined to implement more TCP variants, such as TCP (low-based) and TCP (delay-based). The designers [10] emphasized that the Composable UDT library does not implement the same mechanisms as those in the TCP specification. TCP uses byte-based sequencing, whereas UDT uses packet-based sequencing. This does not prevent CTCP from simulating TCP's congestion avoidance behavior [8-11,17].

UDT was designed with the Configurable Congestion Control (CCC) interface which is composed of four categories: 1) control event handler call backs, 2) protocol behavior configuration, 3) packet extension, and 4) performance monitoring. Its services/features can be used for bulk data transfer and streaming data processing, unlike TCP which cannot be used for this type of processing because it has two problems. Firstly, in TCP, the link must be clean (little packet loss) for it to fully utilize the bandwidth. Secondly, when two TCP streams start at the same time, the stream with longer RTT will be starved due to the RTT bias problem; thus, the data analysis process will have to wait for the slower data stream [8-11].

UDT, moreover, can cater for streaming video to many clients. It can also provide selective streaming for each client when required, whereas TCP cannot send data at a fixed rate. Additionally, in UDP most of the data reliability control work has to be handled by the application.

3. Related Works

Bernardo and Hoang [8-11] present a security framework highlighting the need to secure UDT. The work focuses on UDT's position in the layer architecture which provides a layer-to-layer approach to addressing security. Its implementation relies on proven security mechanisms developed and implemented on existing mature protocols. A summary of security mechanisms and their implementations are presented in fig. 1.

4. Motivations

As UDT is at the application layer or on top of UDP, data are required to be transmitted securely and correctly. This is implemented by each application and not by an operating system or by a separate stack [8]. These implementations may be, and often are, based on generic libraries [17]. The existence of five application-dependent components, such as the API module, the sender, receiver, and UDP channel, as well as four data components: sender's protocol buffer, receiver's protocol buffer, sender's loss list and receiver's loss list [8], require that UDT security features must be implemented on an application basis.

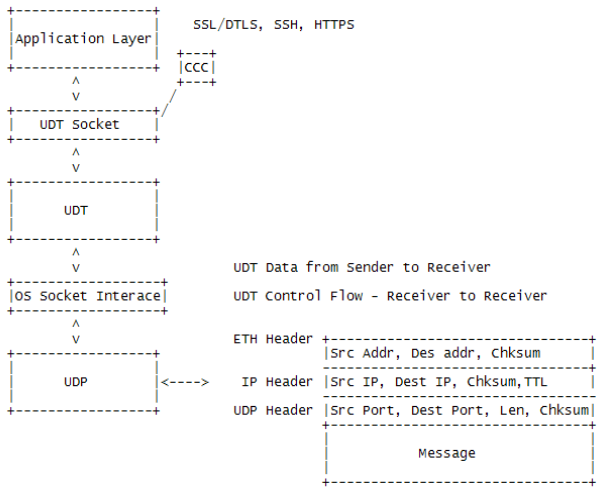

Figure 1: UDT in Layer Architecture. UDT is in the application layer above UDP. The application exchanges its data through UDT socket, which then uses UDP socket to send or receive data [8-11]

4.1 UDT Security Limitations

The contention for the need of security mechanisms of the new UDT is derived from four important observations [2,5].

- Absence of an inherent security mechanism, such as checksum for UDT
- Dependencies on user preferences and implementation on the layer on which it is implemented
- Dependencies on existing security mechanisms of other layers on the stack
- Dependencies on TCP/UDP which are dependent on nodes and their addresses for high speed data transfer protocol leading to a number of attacks such as neighborhood, Sybil and DoS (Denial of Service) attacks.

The presentation of a security framework for UDT supports the need to minimize its sending rates [8-11] in retransmissions and to introduce its own checksum in its design. It also supports the importance of implementing security in UDT. However, the introduction of other security mechanisms to secure UDT is presented to address its vulnerabilities to adversaries exploiting the application, transport, and IP layers.

4.1.1 Possible Security Mechanisms

Previous literature [8-11] presented an overview of the basic security mechanisms for UDT. As the research progresses, the following approaches are further developed.

UDT is designed to run on UDP and is thus dependent on its existing security mechanisms. Consequently, the designers of the applications that use UDT are faced with limited choices to protect the transmission. This paper proposes the following. Firstly, utilising IPsec RFC 2401; however, for a number of reasons, this is only suitable for some applications [7]. Secondly, designing a custom security mechanism on the application layer using API, such as GSS-API [23,29,47], or a custom security mechanism on IP layer, such as

HIP-CGA [1,3,4,5,19,21,25,26,33,43] Thirdly, integrating SASL [31] or DTLS [39] on the transport layer.

These approaches can be significant for application and transport layer- based authentication and end-to-end security for UDT.

- GSS-API - Generic Security Service Application Program Interface [23,29,47]
- Self-certifying addresses using HIP-CGA
- SASL - Simple Authentication and Security Layer (SASL) [31]
- DTLS – Data Transport Layer Security [39]
- IPsec – IP security [7]
- UDT-AO – Authentication Option

They are also applicable to a combination of the following:

* SASL/GSS-API for authentication + channel binding to DTLS, DTLS for data integrity/confidentiality protection

* SASL/GSS-API for authentication + channel binding to IPsec, IPsec for data integrity/confidentiality protection

In this paper, only a brief description of each approach is presented due to space limitations.

5. Securing UDT

Bernardo [8-11] presented an overview on securing UDT implementations in various layers. However securing UDT in application and other layers needs to be explored in future UDT deployments in various applications.

There are application and transport layer-based authentication and end-to-end [12] security options for UDT. This paper also advocates the use of GSS-API in UDT in the development of an application using TCP/UDP. The use of Host Identity Protocol (HIP), a state of the art protocol, combined with Cryptographically Generated Addresses (CGA) is explored to solve the problems of address-related attacks.

5.1 Host Identity Protocol (HIP)

Implementing Host Identity Protocol (HIP) [1,19] is one possible way to secure UDT on top of UDP and IP. This protocol solves the problem of address generation in a different way by removing the dual functionality of IP addresses as both host identifiers and topological locations. In order to do this, a new network layer called the Host Identity is required.

Securing IP addresses plays an important role in networking, especially in the transport layer. Generating a secure IP address can be achieved through HIP. It is considered the building block which is used in other protocols, as well as being a way to secure the address generation in practice [19].

Much literature has been published on the various research on HIP since it was first introduced in RFC 4423. This resulted in a number of new experimental RFCs in April of 2008.

Host identification is attained by using IP addresses that depend on the topological location of the hosts, consequently overloading them. The main motivation behind HIP is to separate the location and host identification information to minimize stressing IP addresses, since they are identifying both hosts and topological locations. HIP introduces a new namespace, cryptographic in nature, for host identities [19]. The IP addresses continue to be used for packet routing.

Using HIP for UDP/TCP in the transport layer of the new network layer, called Host Identity (HI), protects not only the underlying protocol, but UDT as well, since it is running on top of UDP. HI is placed between the IP and transport layer; see fig. 2.

In HIP, the public-key [32,39] of an asymmetric key pair is used as the HI and the host itself is defined as the entity that holds the private-key from the key pair. Application and other higher layer protocols are bound to HI instead of an IP address. The prerequisite for HIP implementation should support RSA and DSA [41] for the public-key cryptography.

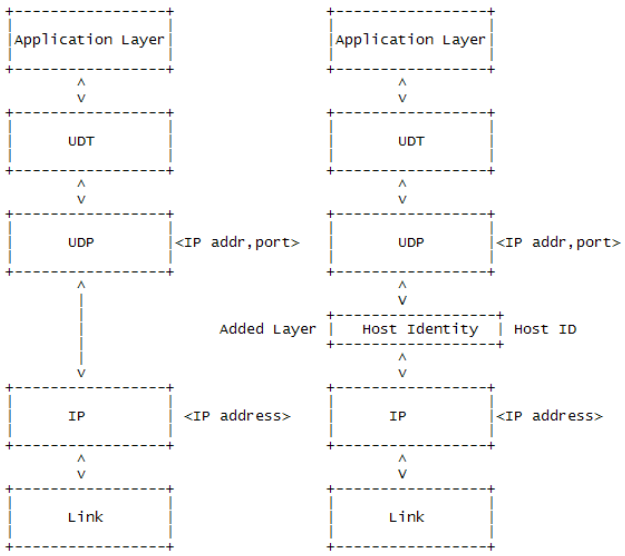

Figure 2: Host Identity Protocol Architecture [19,33].

5.2 Cryptographically Generated Addresses (CGA)

Solving the problems of address-related attacks can be also be achieved by using CGA for address and verification. Self-certifying is widely used and standardized, such as by Host Identity Protocol (HIP) [1,19] and Accountable Internet Protocol (AIP) [2].

CGA [3,4] uses the cryptographic hash of the public key. It is a generic method for self-certifying address generation and verification that can be used for specific purposes. In this paper, the conventions are used to either IPv4 or IPv6.

The simplified setting for CGA is presented in fig. 3. The interface identifier is generated by taking the cryptographic hash [39] of the encoded public-key of the user. Modern cryptography has functions that produce a message digest with more than the required number of bits in CGA. The interface identifier is formed by truncation of the output of the cryptographic hash function to a specific number of bits depending on the leftmost number of bits that form the subnet prefix, i.e., IPv6 addresses are 128-bit data blocks, therefore the

leftmost bits are 64 and the rightmost bits are 64. The prefix is used to determine the location of each node in Internet topology and the interface identifier is used as an identity of the node. Using a cryptographic hash of the public-key is the most effective method to generate self-certifying addresses.

In CGA, the assumption is that each node in the network is equipped with a public-key before generating its address and the underlying public-key cryptosystems have no known weaknesses. Similarly, in UDT, the assumption is that its protection is derived from the security controls implemented on existing transport layers. In this paper, the authors focus on the generic attack models that can be adapted to both UDT and CGA.

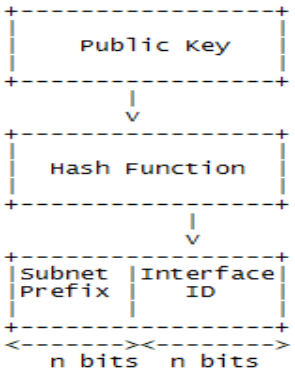

Figure 3: Simplified and modified principle of Cryptographically Generated Addresses [3].

5.2.1 HIP-CGA and UDT

HIP introduces a new namespace, which is cryptographic in nature, for host identifiers. It introduces a way of separating the location and host identity information.

A hashed encoding of the HI, the HIT, is used in protocols to represent the Host Identity.

The HIT is 128 bits long and has the following three properties [1,19]:

- It can be used in address-sized fields in APIs and protocols
- It is self-certifying (i.e., given a HIT, it is computationally hard to find a Host Identity key that matches this HIT)
- The probability of HIT collision between two hosts is very low.

The HITs are self-certifying. This means that no certificates are needed in practice.

The protocol used in HIP to establish an IP-layer communications context, called HIP association, prior to communications is called base exchange [5]. The details are briefly summarized below [33]

- Initiator sends a trigger packet (I1) to responder containing the HIT of the initiator and possibly the HIT of the responder if it is known.

- Next, the responder sends the (R1) packet which contains a puzzle, a cryptographic challenge that the initiator must solve before continuing the exchange. The puzzle mechanism is to protect the responder from a number of DoS threat, see RFC 5201 [33,34]. R1 contains the initial Diffie-Hellman parameters and a signature, covering a part of the message.
- In the I2 packet, the initiator must display the solution to the received puzzle. If an incorrect solution is given, the I2 message is discarded. I2 also contains a Diffie-Hellman parameter that carries needed information for the responder. The packet is signed by the sender.
- The R2 packet finalizes the base exchange and the packet is then signed.

The base exchange protocol is used to establish a pair of IPsec security associations between two hosts for further communication.

HIP introduces a cryptographic namespace for host identifiers to remove the dual functionality of IP addresses as both identifiers and topological locations.

When UDT is implemented on top of UDP, its packets are delivered through HIP. With HIP, the transport layer operates on Host identities instead of using IP addresses as end points. At the same time, the network layer uses IP addresses as pure locators. This provides added protection to the transport layer with applications using UDT's high speed data transmission. With the development of hashed encoding of Host Identifier, a Host Identity Tag can be used in address-sized fields in API's and protocols, including UDT. The hash is truncated to values which are larger in the case of IPv6 implementation, and hence more secure compared with all security levels of CGA [3,4].

HIP uses base exchange protocol [5] to establish a pair of IPsec security associations between two hosts for further communication. The main challenge of implementing HIP is the requirement of a new network layer, called the Host Identity [1], which is difficult to run with existing networking protocols in use.

5.3 Generic Security Service- Application Program Interface (GSS-API)

There are significant application and transport layer based authentication and end-to-end security options for UDT. In this paper, the authors also propose - Generic Security Service Application Program Interface (GSS-API).

The GSS-API is a generic API for carrying out DT client-server authentication. The motivation behind it is that every security system has its own API [26], and the effort in adding different security systems to applications is made extremely difficult by the variance between security APIs. However, with a common API, application vendors could write to the generic API and it could work with any number of security systems, according to [17,29,48]. Vendors can use GSS-API during the UDT implementation. It is considered the easiest to use and implement and implementations exist, such as Kerberos [35].

The Generic Security Service Application Programming Interface provides security services to calling applications. It allows a communicating application to authenticate the user associated with another application, to delegate rights to another application, and to apply security services such as confidentiality and integrity on a per-message basis. Details of GSS-API are discussed in RFC 1964 [29,48].

In summary, the protocol when used in UDT application can be viewed as:

- Authenticate (exchange opaque GSS context) through the user interface and CCC option of UDT.

- The utilize per-message token functions (GSS-API) to protect UDT messages during transmissions.

The GSS-API is a rather large API for some implementations, but for applications using UDT, one need only use a small subset of that API [48].

5.4 Data Transport Layer Security (DTLS)

Another possible mechanism is DTLS. DTLS [40] provides communications privacy for datagram protocols. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DTLS protocol is based on the Transport Layer Security (TLS) [14] protocol and provides equivalent security guarantees. Datagram semantics of the underlying transport are preserved by the DTLS protocol. DTLS is similar to TLS, but DTLS is designed for datagram transport.

High speed data transmission uses datagram transport such as UDP for communication due to the delay-sensitive nature of transported data. The speed of delivery and behavior of applications running UDT are unchanged when DTLS is used to secure communication, since it does not compensate for lost or re-ordered data traffic when applications using UDT running on top of UDP are employed.

DTLS, however, is susceptible to DoS attacks. Such attacks are launched by consuming excessive resources on the server by transmitting a series of handshake initiation requests, and by sending connection initiation messages with a forged source of the victim. The server sends its next message to the victim machine, thus flooding it. In implementing DTLS, designers need to include cookie exchange with every handshake during the implementation of applications using UDT and UDP.

5.5. Internet Protocol Security (IPsec)

Most protocols for application security, such as DTLS, operate at or above the transport layer. This renders the underlying transport connections vulnerable to denial of service attacks, including connection assassination (RFC 3552). IPsec offers the promise of protecting against many denial of service attacks. It also offers other potential benefits. Conventional software-based IPsec implementations isolate applications from the cryptographic keys, improving security by making inadvertent or malicious key exposure more difficult. In addition, specialized hardware may allow encryption keys protected from disclosure within trusted cryptographic units. Also, custom hardware units may well allow for higher performance.

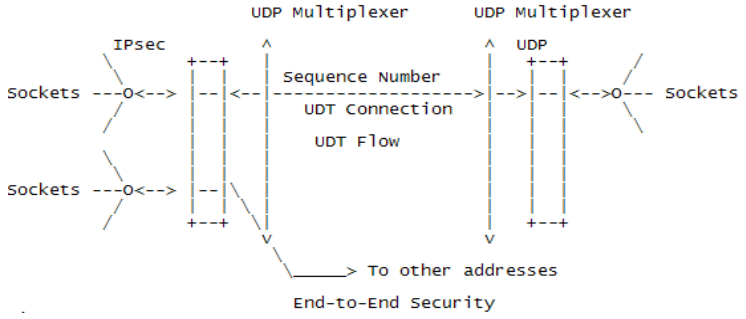

Figure 4: UDT flow using end-to-end security [8-11]. IPsec can be used without modifying UDT and the applications running it.

Implementing UDT running at or above the application layers with IPsec provides adequate protection for data transmission (fig. 4). A datagram-oriented client application using UDT will use the connection-oriented part of its API (because it is using a given datagram socket to talk to a specific server), while the server it is talking to can use the connection-oriented API because it is using a single socket to receive requests from and send replies to a large number of clients.

If nothing else works or is possible in the development of APIs, and in introducing other protocols to protect UDT, IPsec may be a last possible option which provides less overhead in the implementation of applications running UDT.

IPsec can be administered separately and its management can be left to administrators to maintain. It is possible to create a security arrangement to secure UDT connections, such as authentication handled by IPsec. Since it relies on UDP, developers can use UDP encapsulation (see fig. 5) to ensure the connection from UDP is secure. IPsec provides encryption and keying services and offers authentication services; adding ESP extends services to encryption. Specifications on protecting UDP packets can be found on RFC3948.

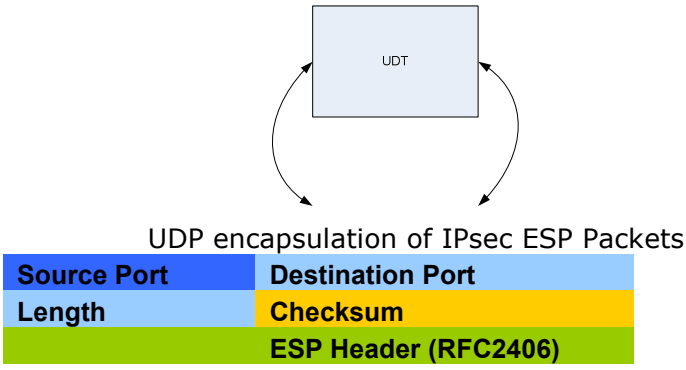

Figure. 5 Schematic diagram of securing UDT on top of UDP [8-11]

5.6. UDT-Authentication Option

UDT-Authentication Option (AO) is introduced to secure UDT -a new UDP-based data transport protocol. UDT is a connection oriented protocol. As such, it needs to include an OPTION for authentication when it is used in data transmission. This is because its connections like TCP, are likely to be spoofed [33].

In TCP, this option is part of the options (0-44 bytes) that occupy space at the end of the TCP header. In utilising the option in TCP [17], this needs to be enabled in the socket. A few systems support this option, which is identified as TCP_MD5SIG option. Note: for the purpose of conceptual analysis, we use MD5 in this paper.

```
int opt = 1;  Enabling this option
setsockopt(sockfd, IPPROTO_TCP, TCP_MD5SIG, &opt,
sizeof(opt));
```

Additionally, the option is included in the checksum. The option may begin on any byte boundary, and the header must be padded with zeros to make the header length a multiple of 32 bits.

Better authentication options in TCP are in progress to address the collision issue in MD5 [34]. However in this paper, the primary motivation is to initially introduce an option to allow UDT to protect itself against the introduction of spoofed segments into the connection stream, regardless what authentication schemes required in a given number of bytes. This option, like in TCP, will be included in the UDP header.

UDT is in user space above the network transport layer of UDP, it is dependent on UDP where this option is not available, thus becomes a challenge in its implementation. UDT, however, provides transport functionalities to applications.

To spoof a connection of UDT, an attacker would not only have to guess UDT's sequence numbers, but would also have had to obtain the password included in the MD5 digest. This password never appears in the connection stream, and the actual form of the password is up to the application, according to RFC 2385. It could during the lifetime of a particular connection so long as this change was synchronised on both ends (although retransmission can become problematical in some implementations with changing passwords) .

To utilise this option in UDT, similar to TCP this needs to be enabled in the socket. A few systems support this option, which can be identified as UDT_MD5SIG or in the case of using SHA, UDT_SHASIG option.

```
int opt = 1;  Enabling this option
setsockopt(sockfd, IPPROTO_UDT, UDT_MD5SIG, &opt,
sizeof(opt));
```

Likewise, the option is included in the UDP checksum. However, there is no negotiation for the use of this option in a connection (also in TCP) rather it is purely a matter of site policy whether or not its connections use the option.

The proposed option can be applied on type 2 of the UDT header. This field is reserved to define specific control packets in the Composable UDT framework.

Every segment sent on a UDT connection to be protected against spoofing will similarly contain the 16-byte MD5 digest produced by applying the MD5 algorithm to these items in the following similar order required for TCP :

1. UDP pseudo header (Source and Destination IP addresses, port number, and segment length)
2. UDT header + UDP (Sequence number and timestamp), and assuming a UDP checksum zero
3. UDT control packet or segment data (if any)
4. Independently-specified key or password, known to both UDTs and presumably connection specific and
5. Connection key

The UDT packet header and UDP pseudo-header are in network byte order. The nature of the key is deliberately left unspecified, but it must be known by both ends of the connection, similar with TCP [19]. A particular UDT implementation will determine what the application may specify as the key.

MD5 algorithm, however, was found to be vulnerable to collision search attacks, and is considered by some to be insufficiently strong for this type of application [22],[28],[34].

However, we specify the MD5 algorithm for this option as basis of our argument to include AO to UDT. Systems that use UDT have been deployed operationally, and there was no "algorithm type" field defined to allow an upgrade using the same option number.

This does not, therefore, prevent the deployment of another similar option which uses another hashing algorithm (like SHA-1, SHA-256). Moreover, should most implementations pad the 18 byte option as defined to 20 bytes anyway, it would be just as well to define a new option which contains an algorithm type field.

In addressing the implication of MD5 collision issue, we recommend using a more secure message algorithm such as SHA-1 or SHA-256.

6. Simulation and Implementation Schemes

In observing the behaviors of UDT in both protected and unprotected settings, (1) the Simulated and, (2) the Implementation schemes are constructed.

The simulated environment operates separately on ns2 [37] and EMIST [38] to provide internal validation. This environment is used to simulate the behavior of data transmission when UDT is used on top of UDP. A test is performed using a new probabilistic packet marking scheme constituted by 3000 nodes. 1000 attackers are selected randomly. To test and determine the number of packets required to reconstruct the attacking path, the selection of one path from all of the attacking paths and its length is w, w=1,2…30. For each value of w, a simultaneous change of values of w is repeatedly changed until the protocol shows a clear attacking path. This allows the simulation to produce a pattern of the behaviors of UDT without any means of protection.

The implementation environment comprises a simple topology. Two honey pot servers (HP1 and HP2) with UDT for windows are installed at two separate locations. They are in a network operating environment running on a 10G pipe trunk 802.1q for tunneling behind firewalls. The attackers are sourced from the Internet. In the first implementation, all traffic is allowed to traverse through any source, destined through any ports on UDP and TCP, and locked to the destination honey pot, where UDT is running on top of UDP. A simple data transfer of 600MB -200GB to another server is then performed. The test is initially performed without any protection. Subsequent tests are performed with the proposed security mechanisms and results are compared.

The following protection schemes are attempted:

(1) A simple authentication scheme using Kerberos [29,35] for GSS-API on an application running UDT and UDP
(2) IPsec between H1 and H2, running the application within the encrypted tunnel
(3) Using VPN SSL connections and running the applications in H1 and H2

7. Results

The number of attacks in figures 6 and 7 is constant in the implementation scheme. The dropped packets were detected when the IDS/IPS was activated on the firewalls. The simple authentication scheme developed to transfer a file via UDT provided by Kerberos using GSS-API on the UDP socket where UDT was operating provided an added protection that sources where the location of the authenticating party was located were assumed to be in the protected environment.

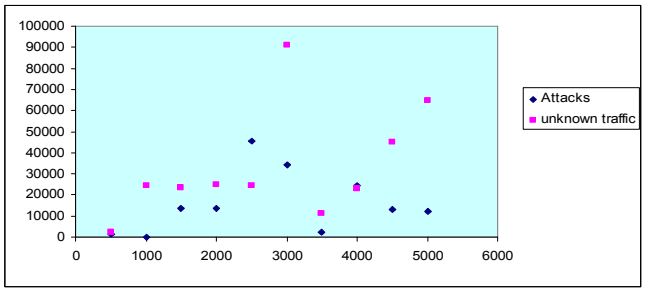

Figure 6: Unprotected environment

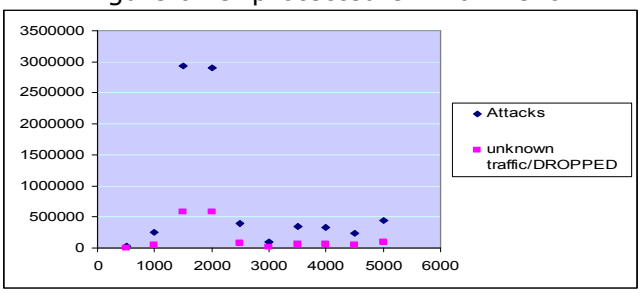

Figure 7: Protected environment

The trend presented in fig. 7 yields significant improvement. End-to-end transfer of data is transparent to the UDT application. The available security mechanisms for UDT that require minimal application and program development are feasible and predominantly applicable to UDT implementation.

In simple file transfer, many available mechanisms for UDP and TCP, and existing security protections for applications are acceptable, i.e., simple authentication. However, for extensive use of UDT, such as SDSS and other large project implementations that require security, UDT requires a security mechanism that is developed and tailored for its behaviors and characteristics based on its design. This paper emphasizes the need, just like the existing mature protocols, for a continuing security evaluation to develop and provide adequate protection, and to maintain integrity and confidentiality against various adversaries and unknown attacks, as well as minimizing dependencies on other security solutions applied on other protocols to ensure minimal overhead in data and message transmission streams.

The limitations of the simulation and implementation schemes constructed may be the simplicity of the applications developed for the tests. Experiments are difficult to perform on the following mechanisms: HIP because of the required additional layer HIT, and DTLS + CGA because of lack of resources. These experiments will be performed in future work.

More extensive development of an application that uses UDT might have yielded more detailed and comprehensive results. Furthermore, the number of false positives and collisions are not considered in the tests. However, the results provide an important indication of how the application which utilizes UDT behaves in such environments.

8. Conclusion and Future Work

Protecting UDT can be achieved by introducing approaches related to self-certifying address generation and verification. A technique which can be applied without major modifications in practice is Cryptographically Generated Addresses (CGA). This technique is standardised in a protocol for IPv6. Similarly, HIP solves the problem of address generation in a different way by removing the functionality of IP addresses as both host identifiers and

topological locations. However in order to achieve this, a new network layer, called Host Identity (HI), is introduced, which makes HIP incompatible with current network protocols.

Protecting UDT by using GSS-API in UDT is another approach; however, this needs to be thoroughly evaluated by application vendors. The use of the GSS-API interface does not in itself provide an absolute security service or assurance; instead, these attributes are dependent on the underlying mechanism(s) of UDT which support a GSS-API implementation to achieve adequate security mechanism.

Another way of protecting UDT is by introducing Authentication-Option (AO), however, this requires changes on the design of UDT to accommodate an AO field. There is also a requirement to use better hashing algorithms to ensure that messages transmitted are duly protected.

In the simulation and implementation schemes, IPsec provides adequate protection on data transfer, and also provides end to end protection on source and destination nodes. In this scheme, the performance of UDT remains the same.

More options remain to be explored such as DTLS – Data Transport Layer Security, SASL - Simple Authentication and Security Layer, and their combinations such as SASL/GSS-API for authentication + channel binding to DTLS; DTLS for data integrity/confidentiality protection; SASL/GSS-API for authentication + channel binding to IPsec; IPsec for data integrity/confidentiality protection.

9. References

1. Al-Shraideh, F. Host Identity Protocol. In ICN/ICONS/MCL, page 203. IEEE Computer Society, 2006.
2. Andersen, D. G., Balakrishnan, H. , Feamster, N. Koponen, T. Moon, D. and Shenker, S.. Accountable Internet Protocol (AIP). In Bahl, V. Wetherall, D. Savage, S. and Stoica, I., editors, SIGCOMM, pages 339–350. ACM, 2008.
3. Aura, T. Cryptographically Generated Addresses (CGA). In C. Boyd andW. Mao, editors, ISC, volume 2851 of Lecture Notes in Computer Science, pages 29–43. Springer, 2003.
4. Aura, T. Cryptographically Generated Addresses (CGA). RFC 3972, IETF, March 2005.
5. Aura, T. Nagarajan, A. and Gurtov. A., Analysis of the HIP Base Exchange Protocol. In In 10th Australasian Conference on Information Security and Privacy ACISP 2005, pages 481–494, 2005.
6. Bellovin, S. " Defending Against Sequence Number Attacks",RFC 1948 1996
7. Bellovin, S."Guidelines for Mandating the Use of IPsec", Work in Progress, IETF, October 2003
8. Bernardo, D.V and Hoang, D., "A Conceptual Approach against Next Generation Security Threats: Securing a High Speed Network Protocol – UDT", *Proc. IEEE the 2nd ICFN 2010, Shanya China*
9. Bernardo, D.V. and Hoang, D., "Security Requirements for UDT", IETF Internet-Draft – working paper, September 2009
10. Bernardo, D.V and Hoang, D., "Network Security Considerations for a New Generation Protocol UDT. Proc. *IEEE the 2nd ICCIST Conference 2009, Beijing China.*
11. Bernardo, D.V and Hoang, D., A Security Framework and its Implementation in Fast Data Transfer Next Generation Protocol UDT, *Journal of Information Assurance and Security* Vol 4(354-360). ISN 1554-1010. 2009
12. Blumenthal, M. and Clark, D. *Rethinking the Design of the Internet: End-to-End Argument vs. the Brave New World*, Proc. ACM Trans Internet Technology, 1, August 2001
13. Clark, D., Sollins, L. Wroclwski, J., Katabi, D., Kulik, J., Yang X., *New Arch: Future Generation Internet Architecture*, Technical Report, DoD – ITO, 2003
14. Dierks , T. and Allen C., "The TLS Protocol Version 1.0", RFC 2246, January 1999
15. Falby, N., Fulp, J. , Clark, P., Cote, R., Irvine, C., Dinolt, G., Levin, T., Rose, M., and Shifflett, D. "*Information assurance capacity building: A case study,*" Proc. 2004 IEEE Workshop on Information Assurance, U.S. Military Academy, June, 2004, 31- 36.
16. Gorodetsky, V., Skormin, V. and Popyack, L. (Eds.),*Information Assurance in Computer Networks: Methods, Models, and Architecture for Network Security*, St. Petersburg, Springer, 2001.
17. Gu, Y., Grossman, R., UDT: UDP-based Data Transfer for High-Speed Wide Area Networks. Computer Networks (Elsevier). Volume 51, Issue 7, 2007.

18. Hamill, J., Deckro, R., and Kloeber, J., "*Evaluating information assurance strategies*," in Decision Support Systems, Vol. 39, Issue 3 (May 2005), 463- 484.
19. H. I. for Information Technology, H. U. of Technology, et al. Infrastructure for HIP, 2008.
20. Harrison, D., RPI NS2 Graphing and Statistics Package, http://networks.ecse.rpi.edu/~harrisod/graph.html
21. Jokela, P., Moskowitz, R.,and Nikander, P., Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). RFC 5202, IETF, April 2008.
22. Joubert, P., King, R., Neves, R., Russinovich, M., and Tracey, J.: Highperformance memory-based web servers: Kernel and user-space performance. *USENIX '01*, Boston, Massachusetts, June 2001.
23. Jray, W., "Generic Security Service API Version 2 :C-bindings", RFC 2744, January 2000.
24. Kent, S., Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, 1998.
25. Laganier, J. and Eggert, L., Host Identity Protocol (HIP) Rendezvous Extension. RFC 5204, IETF, April 2008.
26. Laganier, J., Koponen, T., and Eggert, L., Host Identity Protocol (HIP) Registration Extension. RFC 5203, IETF, April 2008.
27. Leon-Garcia, A., Widjaja, I., *Communication Networks*, McGraw Hill,2000
28. Linn, J. "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
29. Linn, J., "The Kerberos Version 5 GSS-API Mechanism", IETF, RFC 1964, June 1996.
30. Mathis, M., Mahdavi, J., Floyd,S. and Romanow, A.: TCP selective acknowledgment options. *IETF RFC 2018*, April 1996.
31. Melnikov, A., Zeilenga,, K., Simple Authentication and Security Layer (SASL) IETF, RFC 4422, June 2006
32. Menezes, A.J., Oorschot, van P.C. and Vanstone, S. A. ,*Handbook of Applied Cryptography*, CRC Press, 1997
33. Moskowitz, R. and Nikander, P., RFC 4423: Host identity protocol (HIP) architecture, May 2006.
34. R. Moskowitz, R., Nikander, P. Jokela, P., and Henderson, T.,. Host Identity Protocol. RFC 5201, IETF, April 2008.
35. Neuman, C., Yu, T., Hartman, S., Raeburn, K., Kerberos Network Authentication Service (V5), IETF, RFC 1964, June 1996
36. NIST SP 800-37, *Guide for the Security Certification and Accreditation of Federal Information Systems*, May 2004.
37. NS2. http://isi.edu/nsna/ns.
38. PSU Evaluation Methods for Internet Security Technology (EMIST) , 2004, http://emist.ist.psu.edu visited December 2009
39. Rabin, M., "*Digitized signatures and public-key functions as intractable as Factorization*," MIT/LCS Technical Report, TR-212 (1979).
40. Rescorla, E., Modadugu, N., "Datagram Transport Layer Security" RFC 4347, IETF, April 2006
41. Rivest, R.L., Shamir,A., and Adleman, L.M.,"*A method for obtaining digital signature and public-keycryptosystems*," Communication of ACM, 21, (1978),120-126
42. Schwartz, M., *Broadband Integrated Networks*, Prentice Hall, 1996.
43. Stewart , R. (Editor), Stream Control Transmission Protocol, RFC 4960, 2007.
44. Stiemerling, M., Quittek, J., and Eggert, L., NAT and Firewall Traversal Issues of Host Identity Protocol (HIP) Communication. RFC 5207, IETF, April 2008.
45. Stoica, I., Adkins,D.,.Zhuang, S., Shenker, S., Surana, S., *Internet Indirection Infrastructure*, Proc. ACM SIGCOMM 2002, August 2002.
46. Szalay, A., Gray, J., Thakar,A., Kuntz, P., Malik,T., Raddick, J., Stoughton. C., Vandenberg,J.,: The SDSS SkyServer - Public access to the Sloan digital sky server data. *ACM SIGMOD 2002*
47. Wang, G.,and Xia,Y., An NS2 TCP Evaluation Tool, http://labs.nec.com.cn/tcpeval.html
48. Williams, N., "Clarifications and Extensions to th eGeneric Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009
49. Globus XIO: unix.globus.org/toolkit/docs/3.2/xio/index.html. Retrieved on November 1, 2009
50. Zhang, M., Karp, B., Floyd, S., and Peterson,L.,: RR-TCP: A reordering-robust TCP with DSACK. *Proc. the Eleventh IEEE International Conference on Networking Protocols (ICNP 2003)*, Atlanta, GA, November 2003.

Authors

Danilo V Bernardo is presently a non-executive director of a charity organisation, whilst working with a government agency in the State of NSW in Australia. He has authored a book,

co-authored book chapters, and has published conference papers and journals. He has been collaborating with researchers from the National ICT Australia (NICTA) in the area of software engineering. He studied at Harvard, The University of New South Wales, University of California, Berkeley, Cebu University and the University of Technology Sydney. He received academic (a medal for academic excellence) and industry prizes for his work. He won international best paper awards for his research papers in IEEE and Springer Verlag sponsored conferences and received industry awards from a couple of global organisations, such as IBM, the Olympic Technology Office Australia in 2000, and HSBC. His research interests vary from network security, network engineering, software and requirements engineering, business management, to bioinformatics and biotechnology. He also served as Technical Session Chair for LNCS Springer and IEEE ISA 2010 sponsored conference in Japan.

Doan B. Hoang was a Visiting Professor at LIUPPA. He finished his 5-month visiting appointment at Carlos III Universidad de Madrid in 2010. He is a Professor in the School of Computing and Communications, Faculty of Engineering and Information Technology, the University of Technology, Sydney (UTS). Before joining UTS, he was with Basser Department of Computer Science, Sydney University. He also held various visiting positions: Visiting Professorships at the University of California, Berkeley; Nortel Networks Technology Centre in Santa Clara, USA; the University of Waterloo, Canada; Massey University, New Zealand; and Nanyang Technological University, Singapore. He is a director of the UTS Centre for Innovation in IT Services and Applications (iNEXT). He is currently leading research into establishing an innovation culture, reducing the cost of healthcare system through advanced technologies and assistive health Grid/Cloud, and innovative use of the Broadband Internet. He has published over 150 papers in peer reviewed journals, conferences and workshops in his field. He also served as a Technical Program Chair for IEEE Healthcom 2009 in Sydney.