Securing UDT Protocol using Secure Socket Layer

Tatikayala Sai Gopal, Rahul Jain, Reddy Lakshmi Eswari. P, Jyostna. G, Srinivasa Reddy Kamatham
Centre for Development of Advanced Computing (C-DAC)
Hyderabad, India
{fsaigopalt, rahulj, prleswari, gjyostna, ksreddyg}@cdac.in

ABSTRACT: *UDT has emerged as a next generation protocol that effectively utilizes the bandwidth in Long Fat Networks (LFN). Lack of security features in UDT protocol is a major concern which has drawn scientist to explore various security mechanisms. As transport layer security mechanisms are mostly widely accepted mechanism to provide security to data in terms of confidentiality, integrity and Authentication, we have explored various standard mechanisms to provide security to UDT protocol. Through this paper, we propose an approach to integrate Secure Socket Layer with UDT protocol and present detailed analysis report of overhead involved and performance of secured UDT in an emulated testbed.*

Keywords: UDT, UDT protocol, Long Fat Networks, Secure Socket Layer, Authentication

Received: 30 April 2013, Revised 5 June 2013, Accepted 12 June 2013

1. Introduction

Many variant of TCP such as TCP, FAST TCP, BIC TCP and HS TCP has evolved to address the problem of bandwidth utilization in the current high speed networks [1]–[3]. Though there are many variants of TCP,practically all these protocols exhibit few or more problems with fairness, Round Trip Time (RTT) and convergence [1] in high-speed networks.

UDT is an end to end application level protocol which acts as a reliable connection oriented transport protocol [4]–[8]. It is used to transfer bulk amount of data in distributed environments. It is the next generation protocol which effectively utilizes bandwidth in high speed networks including wide area optical networks. UDT is also an alternative transport protocol for GridFTP [9].

Absence of inherent security features in UDT protocol leads researchers to explore various existing mechanisms to secure UDT [10]–[16]. IPsec and SSL are two most widely accepted mechanisms to provide security.IPsec can be used for securing UDT application data as UDT Layer lies above IP [14]. IPsec is a peer to peer protocol which is mostly deployed for Virtual private networks rather than end-to-end security [17]–[19]. As it is deployed at the kernel level, any modification in the IPsec leads to kernel recompilation which cant be done by every end user. Configuration of IPsec and lack of standardization of its API makes it too difficult to be used as standardized solution. Therefore, SSL being at the application level makes it easy to deploy and configure.

In this paper, we present our experiences in the usage of transport level security for UDT protocol. Since UDT provides both reliable and partial reliable transmission of data, transport level security solutions such as TLS/DTLS are considered and explored the feasibility of integration with UDT [14]. TLS is most widely used transport level security solution for providing security to TCP based application whereas DTLS is used for providing security to UDP based applications. DTLS is modification of Transport Layer Security (TLS) that works for unreliable protocols.

In this paper we have presented our approach to integrate TLS/DTLS with UDT Framework and discussed the pros and cons for the same. Section 2 deals with the background details of the UDT framework and its features. Section 3 deals with related work in security for UDT.The details of integrating TLS/DTLS with UDT are but have resulted in several other issues like convergence and RTT problem in high speed networks.

UDP-based Data Transfer (UDT) [1-3, 5-6] is a next generation protocol which overcomes the problems faced by TCP and its variants described in Sections 4. Section 5 shows the experimental results of UDT application after integrating with transport level security solutions and it is comparison with FTP and FTP-SSL file transfer applications using TCP. The paper is concluded in Section 6.

2. Background

This section gives an overview of UDT Layered approach. UDT is built on top of UDP protocol. UDT uses UDP through the socket interface provided by the underlying operating system. Application can call UDT socket API in the same way as they call the system socket APIs. It is a connectionoriented duplex protocol, which supports both reliable and partial reliable messaging.Application can specify a reliable mode by creating the socket with SOCK STREAM and partial reliable mode with SOCK DGRAM flag. UDT socket is a logical entity which is mapped to the underlying UDP socket.

UDT framework consists of three main components UDT buffer management, UDT connection and flow management and UDP multiplexer [4]–[6]. UDT buffer management helps in fragmenting and rearranging the fragments of a message. UDT connection and flow management is designed to provide the reliability and control congestion in the channel. In order to regulate the packet flow, UDT uses rate control as well as a window control mechanism. UDP multiplexer is used for the effective usage of the UDP channel.UDT uses timer based selective acknowledgment in which acknowledgments are generated at fixed intervals [6]. Next section presents related work in the UDT security.

3. Related Work

Bernardo and Hoang emphasized the need to secure the UDT data, as the usage of UDT is progressively increasing for bulk data transfer in the present high-speed network scenario [10], [12]. Bernardo and Hoang presented work on introduction of Authentication Option (AO) and Identity Packet (IP) as part of packet structure for future use to minimize the risk of attack on UDT protocol such as Denial of Service (DOS) attack and also to include a checksum in the UDT design [12], [15].

They also suggested the security feature to be implemented at the application level [12]. An Overview of the proven security mechanisms at various levels was presented [11] and also provided in-depth usage of Generic Security Service Application Program Interface (GSS-API) to provide security to UDT data [14].

According to them, security for UDT can be achieved by the combination of different security solutions such as Simple Authentication and Security Layer (SASL)/GSS-API for authentication and integrity and confidentiality protection provided by DTLS or IPsec [14] and also analyzed correctness of the selected security mechanisms such as UDT-Kerberos (GSS-API), UDT-AO, UDT-DTLS in the symbolic model for UDT and its implementation issues using formal composite logic [16].

4. Integration with TLS /DTLS

In this section, we present our work in integrating TLS/DTLS with UDT to provide security. TLS is widely used to provide secure communication for application that uses reliable protocols such as TCP [20] whereas DTLS provides secure communication for application that uses unreliable transport channel such as UDP [21], [22].UDT supports reliable mode as well as Partial Reliable mode messaging. As a result, TLS is chosen to provide security for reliable mode and DTLS for partial reliable mode of UDT. In case of reliable mode, UDT guarantees the delivery of the message whereas in partial reliable mode either the complete message is delivered or it is discarded completely [4], [6].

In reliable mode, UDT takes care of packet loss and reordering, which pursue similar characteristics as that of TCP. Similarly partial reliable mode of UDT is same as UDP where application can specify a time to live (TTL) for the message and also in-order delivery of the message. TTL indicates life time of the message in the network which is infinite by default. If a specific time is given to packet, it will be retransmitted if NAK is received before TTL of the message expires, otherwise a message drop request will be sent to the peer indicating to discard all the packets of the particular message [5], [6]

There are two ways in which TLS/DTLS can be integrated with UDT.

• TLS/DTLS below UDT framework
• TLS/DTLS above UDT framework

4.1 TLS/DTLS below UDT framework
In Figure 1a, UDT application calls UDT framework through the UDT socket interface, which in turns calls TLS/DTLS functionality. To achieve this, UDT framework has to be modified to call TLS/DTLS rather than UDP. It is not possible to use TLS for securing UDT data, because it cannot be placed on top of UDP, which is an unreliable protocol.Hence DTLS can be chosen as an option for securing the data for UDT.

DTLS handshake takes place before the UDT handshake as it is a layered approach. So, security context is maintained for each UDP socket rather than for each UDT socket. As N UDT sockets are associated with a single UDP socket, they will share single security context. However,to provide security for UDT application, each UDT socket should be
associated with a separate security context rather than sharing a single security context for UDP socket. Therefore, it is an insignificant approach.

4.2 TLS/DTLS above UDT framework
Fig 1b depicts, UDT application calls TLS/DTLS which in turn calls the UDT framework for transmission of data. In order to make this solution feasible, TLS/DTLS has to be modified for UDT functionality rather than TCP/UDP functionality and no modification is required at UDT framework.This can be achieved by the use of Basic Input/output(BIO)object. In this case, UDT handshake is performed prior to TLS/DTLS handshake. Therefore, each UDT socket is mapped with a corresponding SSL structure. With this approach, security is provided for each UDT socket rather than UDP socket.We have implemented this approach, and the details are mentioned in next section.

4.3 Implementation
OpenSSL [23] is a general purpose cryptographic library that implements TLS and DTLS protocol with similar user space API. Basic Input/output (BIO) object provides I/O abstraction to OpenSSL, which hides the details of underlying layer. For each protocol, there exist a corresponding BIO object which acts as a wrapper to the corresponding protocol,i.e. TCP and UDP. A BIO object consists of methods, which hold a function pointer to their respective functionality. Similarly, we have added a BIO object for UDT functionality to make TLS/DTLS to call UDT functionality. Figure 2 shows the mapping of UDT socket with SSL BIO object. In BIO object file, we have created BIO METHOD structure for UDT, which contains a function pointer to the corresponding UDT functionality. This BIO object file can be either added to OpenSSL library or to each UDT application, which requires secure communication. Following is the code snippet that maps UDT socket with SSL object and also makes TLS/DTLS call UDT functionality.

```
/* creating SSL structure */
ssl = SSL new (ctx);
/* creating a bio object for UDT */
bio = BIO new (BIO s udtsock () );
/*mapping BIO object with UDT socket */
BIO_set_fd (bio, udt_session_fd , BIO_NOCLOSE);
```

BIO_s_udtsock () maps the BIO object with the UDT functionality. BIO_METHOD is a structure in BIO object which holds the function pointer to the underlying layer.

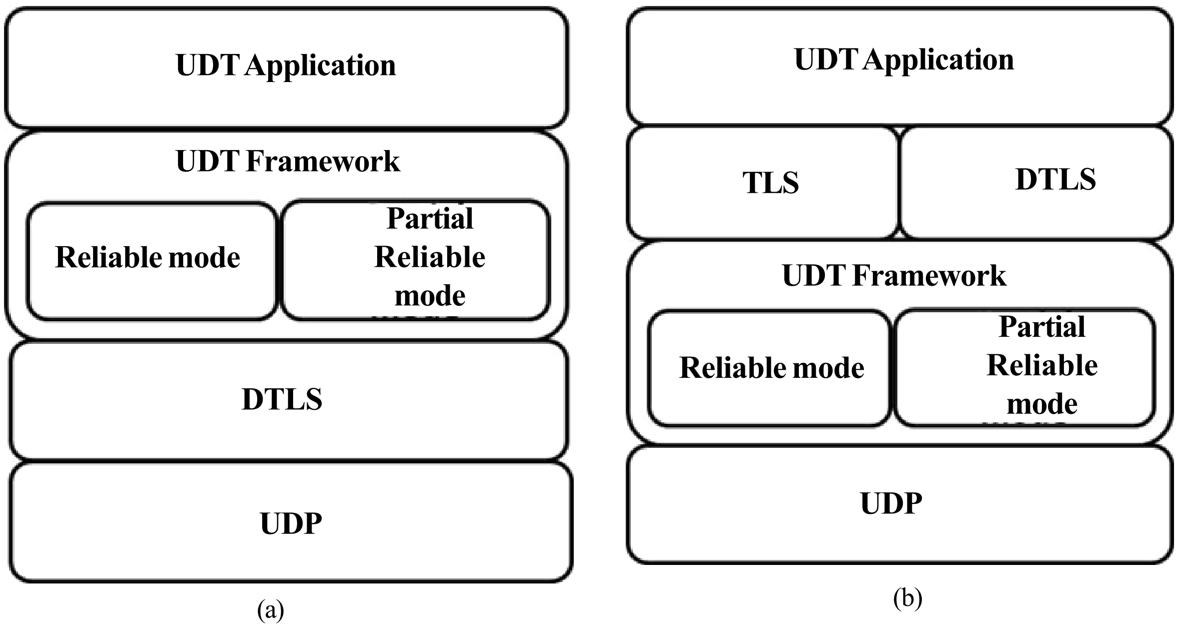

Figure 1. Position of UDT Security Layer in Layered Architecture

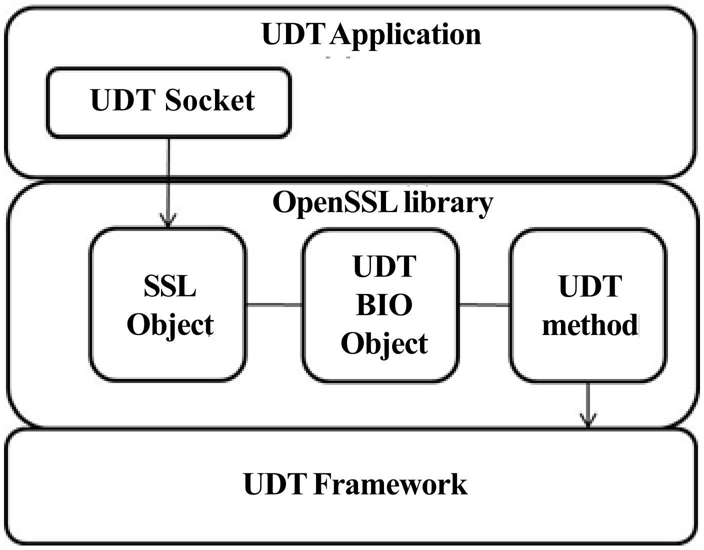

Figure 2. Mapping of UDT socket with SSL BIO object

```
struct bio_method_st {
int type;
const char *name;
int (*bwrite) (BIO *, const char *, int);
int (*bread) (BIO *, char *, int);
int (*bputs) (BIO *, const char *);
int (*bgets) (BIO *, char *, int);
long (*ctrl) (BIO *, int, long, void *);
int (*create) (BIO *);
int (*destroy) (BIO *);
long (*callback ctrl) (BIO *, int, bio info cb *);
} BIO METHOD;
```

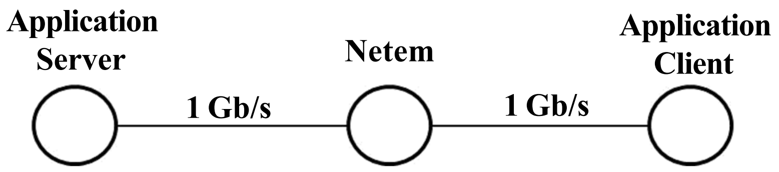

Figure 3. Network test bed

It will fill these function pointer with the address of UDT functions which allows them to call the corresponding functionality. For example, bwrite() and bread() function pointers are mapped to UDT::write() and UDT::read() functionality respectively.

Once the mapping is done between the UDT socket and BIO object, the application client and server call SSL connect () and SSL accept () respectively for TLS/DTLS handshake. UDT handshake is performed prior to the TLS/DTLS handshake as it is a layered approach. During UDT handshake, client and server agree upon sequence number, window size, UDT version, UDT socket type and Maximum Segment Size (MSS) and as part of TLS/DTLS handshake, both the sides negotiate security parameters for the session such as cipher suite, pre-master key, etc. Then security context is established for each UDT socket. TLS/DTLS handshake is successful if all the handshake packets are delivered without any loss [21], [22]. For this, TLS relies on the underlying UDT framework for delivery of these packets using its reliability mode whereas DTLS doesnt rely on lower layer for handshake packet loss as it can handle itself by using a retransmission timer.

Once the TLS/DTLS handshake is completed, application sends the data to OpenSSL, which fragments it, optionally compresses the data, adds Message Authentication Code (MAC), encrypts the UDT data and appends the SSL record header and send it to UDT framework. It adds its own header to the encrypted data and sends the data through the UDP channel. On the other side, UDT header is stripped off and passed to TLS/DTLS where the data is decrypted, verifies MAC, decompresses it if applied, assembles the fragments and pushes the data to the UDT application. Authentication of client server is performed using asymmetric cryptography whereas confidentiality, integrity and message authenticity is provided using symmetric cryptography [21]–[23].

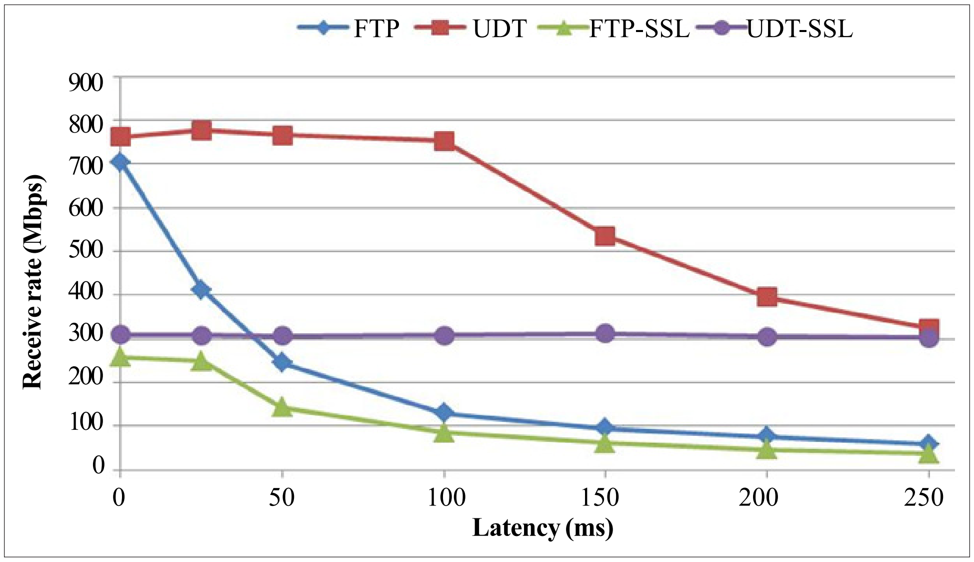

Figure 4. 1 Gbps link - 10GB file transfer

Though DTLS and TLS are used as security solutions for UDT, IP fragmentation can be avoided as UDT will fragment less than 1500 bytes. In the next section, experimental results of the proposed solution are presented.

5. Experimental Results

Figure 3 shows the network test bed for carrying out the experimentation. All the three nodes are connected with 1Gb link on a machine of 1GigE NIC, Intel core i7 3.4GHZ, 8 cores and 8GB RAM. The middle node acts as a router on which Netem tool is installed to test the performance of different applications.

The above implementation is tested, and performance is analyzed by comparing UDT with TCP based solutions. During experimentation, large files (2, 4, 6, 8, 10 GB) are transferred using FTP, FTP with OpenSSL (FTP-SSL), UDT and UDT with OpenSSL (UDT-SSL) based applications, and the results are presented. FTP and FTP-SSL file transfer application uses TCP, UDT and UDT-SSL file transfer application uses UDT protocol as underlying protocol. The file transfer applications such as FTP-SSL, UDT-SSL is tested with the algorithm AES256-SHA and also UDT application and UDT file transfer application with Open-SSL (UDT-SSL) is tested in reliable mode.The standard file transfer application FTP and FTP-SSL uses default congestion control algorithm named CUBIC [24], [25] in Linux operating system . CUBIC is a modification of BIC-TCP.

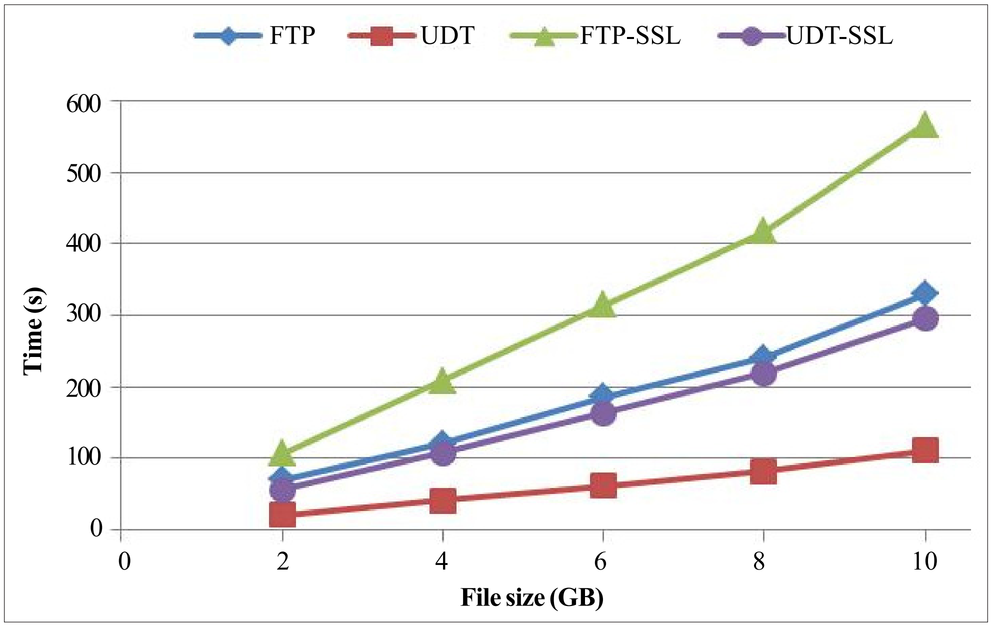

Figure 5. Performance of ftp, ftp-SSL, UDT and UDT-SSL applications with RTT = 50ms

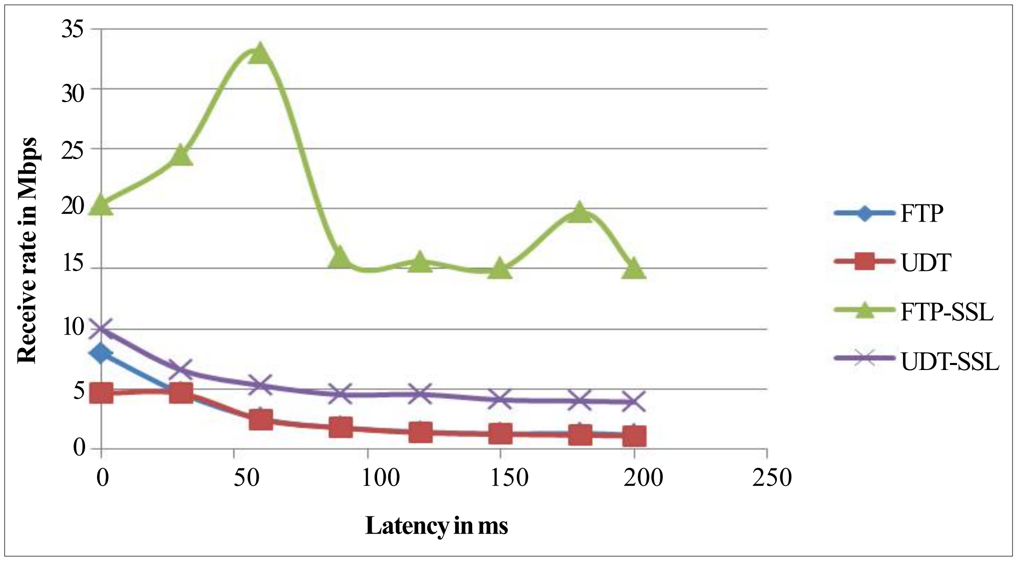

Figure 6. Performance of ftp, ftp-SSL, UDT with 1% loss

Figure 4 shows the throughput results while transferring files with varying RTT.The Throughput of FTP file transfer is high when the RTT factor is less and gradually decreases with the increase of RTT. TCP underutilizes the bandwidth in a high-speed network as Bandwidth Delay Product (BDP) increases [26], [27]. This is because it uses pure window based congestion algorithm.

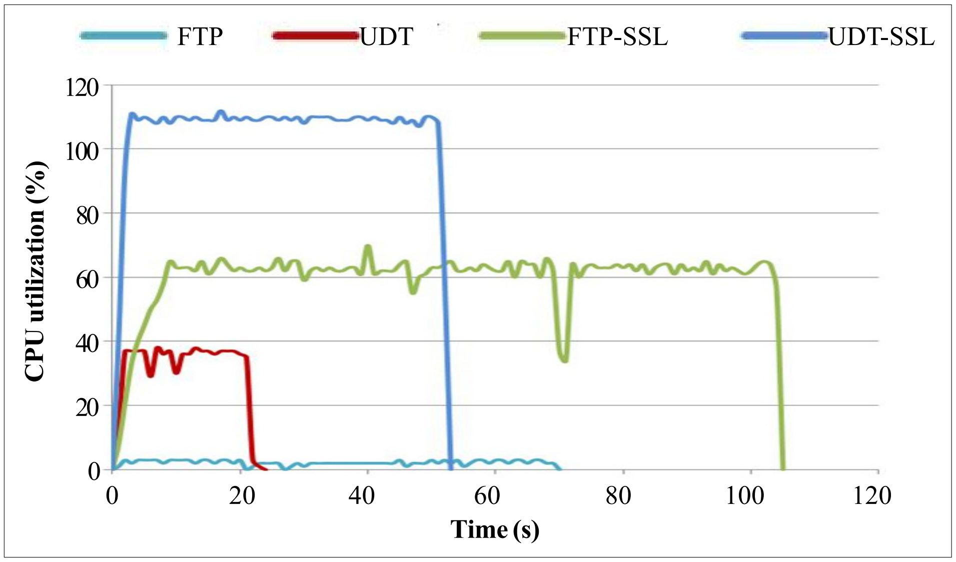

Figure 7. CPU Utilization: Comparison of UDT and UDT-SSL file transfer application with 2GB file and RTT = 50ms

UDT file transfer application performs better than FTP in high-speed networks since the UDT congestion control algorithm uses both window as well as rate based flow control to effectively utilize the bandwidth in the high-speed network. Rate based control is used to tune the packet sending period, where as the window based flow control limits the number of unacknowledged packets. Window control acts as support to rate control mechanism to stop the sender from sending too many packets like TCP flow control. UDT uses an AIMD (additive increase multiplicative decrease) style congestion control algorithm which is designed to utilize high bandwidth efficiently and fairly [5]. UDT Bandwidth estimation technique is used to optimize the parameters dynamically, and random factor is used to remove the negative effect of loss synchronization [5]. When packet loss occurs, CUBIC decreases the window size by 1/5 [24] whereas in UDT, it decreases the sending rate for each congestion period by 1/8 but window size remains constant, and it will be modified only when ACK is received.

The decrease in the throughput for FTP-SSL and UDTSSL file transfer application is due to the encryption/decryption done by the OpenSSL library. From Figure 4, clearly file transfer application using UDT protocol is performing better than TCP protocol.

The Figure 5 shows the timing result when files of varying sizes (2, 4, 6, 8, 10 GB) are sent across the nodes with RTT of 50 ms.Timing results of transferring files indicate that UDT and UDT-SSL are faster when compared with FTP and FTPSSL respectively, because of the above mentioned reasons.

Figure 6 shows the experimental results of ftp, ftp-SSL, UDT and UDT-SSL with configuring 1% loss in the Netem tool. It was observed that throughput of FTP and FTP-SSL is almost similar as the network delays increases. TCP uses CUBIC congestion control algorithm which is extension to Binary increase Congestion (BIC) whereas UDT uses AIMD congestion control algorithm. Unlike TCP congestion control algorithm, UDT decreases the window size between 1/8th to. Another factor that affects the performance of UDT is window updating. In case of TCP, window is updated after every ACK whereas in UDT it is updated after every SYN time (10ms).

Figure 7 shows the CPU utilization (%) for different file transfer applications with respect to time for 2GB file with the delay of 50ms. UDT-SSL and FTP-SSL utilize more CPU cycles because of overhead due to encryption/decryption. The graph illustrates that CPU usage is more when an application uses UDT protocol rather than TCP. The reason behind more utilization of CPU in UDT is due to the design of sender, receiver and multiplexer components [8].

6. Conclusion

UDT is future generation protocol which overcomes the problems of TCP in high speed network. UDT being an application level protocol, it is easy to deploy and modify the protocol. Security is a major concern for UDT. Through this paper, we present the various approaches for providing security to UDT and finally integrated TLS/DTLS protocol with UDT. It is observed from the experimental results that UDT perform far better than the TCP in LFN with and without security.

7. Acknowledgement

Our sincere thanks to Department of Electronics and Information Technology, Ministry of Communications and Information Technology, Government of India for supporting this research work.

References

[1] Yee-Ting Li, Douglas Leith, Robert N. Shorten. (2007). Experimental evaluation of TCP protocols for high speed Networks *IEEE/ACM Transactions on Networking*, Oct.

[2] Les Cottrell, R., Saad Ansari, Parakram Khandpur, Ruchi Gupta, Richard Hughes-Jones, MichaelChen, LarryMacIntosh. (2005). *Characterization and evaluation of TCP and UDP based Transport on Real Networks*, January.

[3] Sangtae Ha,Yusung Kim, Long Le, Injong Rhee, Lisong Xu. (2009). A step toward Realistic performance Evaluation of High-Speed TCP variants, 18th August.

[4] Yunhong Gu, Xinwei Hong, Robert Grossman. (2004). Experiences in Design and Implementation of a High Performance Transport Protocol, Nov.

[5] Yunhong Gu. UDT: A High Performance Data Transport Protocol, UDT thesis.

[6] Yunhong GU, Robert L. Grossman. (2007). UDT: UDP-based Data Transfer for High Speed Wide Area Networks,*The International Journal of Computers and Telecommunications Networking*, 51, May.

[7] Yunhong Gu. (2010). UDT: UDP-based Data Transfer Protocol draft-gg-udt- 03.txt , April,12.

[8] Yunhong Gu, Robert Grossman, UDTv4: Improvements in Performance and Usability.

[9] John Bresnaham, Michael Link, K Rajkumar, Ian Foster. (2009). UDT as an alternative Transport protocol for Grid FTP, International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT), Tokyo, Japan, May.

[10] Bernardo, D. V., Hoang, D. B. (2009). Network security considerations for a New Generation Protocol UDT, *In*: Proceedings of IEEE the 2nd ICCIST Conference.

[11] Bernardo, D.V., Hoang, D. B. (2010). A Pragmatic Approach: Achieving Acceptable Security Mechanisms for High Speed Data Transfer Protocol UDT, *International Journal of Network Security and its Applications*, 4 (3).

[12] Bernardo, D.V., Hoang, D. B. (2009). Security Requirements for UDT, *IETF Internet-Draft*, September.

[13] Bernardo, D.V., Hoang, D. B. (2010). A Conceptual Approach against Next Generation Security Threats: Securing a High Speed Network Protocol-UDT, *Second International Conference on Future Networks*.

[14] Bernardo, D.V., Hoang, D. B. (2010). Protecting Next Generation High Speed Network Protocol- UDT through Generic Security Service Application Program Interface GSS API, 4th IEEE International Conference on Emerging Security Information, *Systems and Technologies, SECUREWARE*.

[15] Bernardo, D.V., Hoang, D. B. (2010). Securing Data Transfer in the Cloud Through Introducing Identification Packet and UDT-Authentication Option Field: A Characterization, *IJNSA*.

[16] Bernardo, D.V., Hoang, D. B. (2012). Compositional Logic for Proof of Correctness of Proposed UDT Security Mechanisms 26th IEEE International Conference on Advanced Information Networking and Applications.

[17] Kent, S., Atkinson, R. (1998). Security Architecture for Internet Protocol, *RFC* 2401.

[18] Heng Yin, Haining Wang. (2005). Building an Application-aware IPsec Policy System, Proceedings of the 14th USENIX Security Symposium.

[19] Arkko, J., Nikander, P. (2003). Limitations of IPsec Policy mechanism, *In*: Proceedings of the 11th International Conference of Security Protocols.

[20] Dierks, T., Rescorla, T. (2008). Transport Layer Security, *RFC* 5246, Aug.

[21] Rescorla, E. (2006). Datagram Transport Layer Security, *RFC* 4347, April.

[22] The Design and Implementation of Datagram TLS, Nagendra Modadugu, Eric Rescorla.

[23] The openSSL Project http://www.openssl.org.

[24] Sangtae Ha, Injong Rhee, Lisong Xu. (2008). CUBIC: A New TCP Friendly High Speed TCP Variant, *ACM SIGOPS Operating System review - Research and development in Linux Kernel*.

[25] Pasi Sarolahti. (2002). Congestion control in Linux TCP, *USENIX annual technical conferenct*.

[26] Shalleeza Sohail, Chun Tung Chou, Salil S.Kanhere, Sanjay Jha. (2005). On Large Scale Deployment of Parallelized File Transfer Protocol, *IEEE*.

[27] Jingsong Zhang, Robert D. McLeod. (2003). A UDP based file transfer protocol with flow control using fuzzy logic approach, *CCGEI*.