# Network Security Mechanisms and Implementations for the Next Generation Reliable Fast Data Transfer Protocol - UDT

Danilo Valeros Bernardo

A THESIS SUBMITTED AS

A PARTIAL REQUIREMENT FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

OF

THE UNIVERSITY OF TECHNOLOGY – SYDNEY

AUSTRALIA

UNIVERSITY OF TECHNOLOGY SYDNEY

UTS CRICOS Provider Code 00099F

School of Computing and Communications

Faculty of Engineering and Information Technology

The University of Technology – Sydney

Australia

*The shades of night were falling fast,*

*As through an Alpine Village passed*

*A youth, who bore, ‘mid snow and ice,*

*A banner with the strange device,*

*Excelsior!*

*– H. Longfellow*

# CERTIFICATE OF AUTHORSHIP /ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any assistance I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are fully indicated in the thesis.

Signature of Candidate

____________________

# Acknowledgements

Completing a rigorous PhD by research in half the required maximum number of years is a feat that not many can achieve, especially when there are demanding work commitments to meet outside the academia.

This milestone obviously is not possible without the support of many people.

It is therefore only audaciously appropriate to thank the members of the Faculty and i-Next and staff of the University Graduate School (UGS) for their support, and especially for the interests, advice, and assistance they provide to this project, and for their unwavering support in providing funding arrangements that resulted to a string of almost 22 peer-reviewed academic papers written and published about this work.

I wish to thank Prof. Doan B. Hoang, my thesis advisor and supervisor, who fostered good research and development of models, and helped ascertain that the focus of this work was within the specific area of research. His meticulous assessment and his honest feedback have been invaluable, leading to the improvement of this dissertation. His recognition of my strong academic and research capabilities have been instrumental, prompting the University to waive the coursework requirements which would have extended additional years of arduous challenges. His advice on the importance of the use of citations and references, which in their absence can lead to some issues for a student like me (who incidentally has a semi eidetic memory), has left me wondering if this ability of remembering information verbatim is a curse or a gift. Eventually, I obligingly consented to cite someone's work whenever necessary.

In addition, the encouragement and sheer independence I have been given in the development of this dissertation, whilst publishing 2 books and almost 18 additional research papers in other areas of interests, doubtless proved to be invaluable to my present and surely to my future research and consulting work.

Moreover, the work presented in this thesis would also not have been possible without the support, in the form of advice and research direction, of the following:

Dr. Yunhong Gu of the National Data Mining Centre at the University of Illinois, United States, now a software engineer at Google, who provided initial guidance in the implementation of UDT in a test environment, and provided invaluable insights that improved this work.

Nicolas Williams of Sun Corporation, United States, and anonymous reviewers for their constructive comments that helped improve the publications which supported and validated the overall work in the development of a few security mechanisms in this dissertation.

Prof. John Mitchell and Stephan Stiller of Security Laboratory, Stanford University, California, United States, for providing very comprehensive literature about Protocol Composite Logic technique: literature that helped improve the analysis of the proposed mechanisms for UDT.

Dr. Henry Leitner of Harvard University, for his rigorous classes in Computer Science that were instrumental in the development of the Project UDT Tool in Java.

The Australian Government for the Higher Degree Commonwealth Research Scholarship.

Friends and colleagues at NSW State Government.

Lastly and importantly, to my family for their continuing support in the countless journeys I take, including this journey, which I initially thought the longest to complete. Their support has always been the bedrock of my accomplishments.

*Dedicated to DAB†, Linda, Beverly, Jojo, and Bee Bee*

# Abstract

TCP protocol variants (such as FAST, BiC, XCP, Scalable and High Speed) have demonstrated improved performance in simulation and in several limited network experiments. However, practical use of these protocols is still very limited because of implementation and installation difficulties. Users who require to transfer bulk data (e.g., in Cloud/GRID computing) usually turn to application level solutions where these variants do not fair well. Among protocols considered in the application level are User Datagram Protocol (UDP)-based protocols, such as UDT (UDP-based Data Transport Protocol). UDT is one of the most recently developed new transport protocols with congestion control algorithms. It was developed to support next generation high-speed networks, including wide area optical networks. It is considered a state-of-the-art protocol, addressing infrastructure requirements for transmitting data in high-speed networks. Its development, however, creates new vulnerabilities because like many other protocols, it relies solely on the existing security mechanisms for current protocols such as the Transmission Control Protocol (TCP) and UDP. Certainly, both UDT and the decades-old TCP/UDP lack a well-thought-out security architecture that addresses problems in today's networks. In this dissertation, we focus on investigating UDT security issues and offer important contributions to the field of network security. The choice of UDT is significant for several reasons: UDT as a newly designed next generation protocol is considered one of the most promising and fastest protocols ever created that operates on top of the UDP protocol. It is a reliable UDP-based application-level data-transport protocol intended for distributing data intensive applications over wide area high-speed networks. It can transfer data in a highly configurable framework and can accommodate various congestion control algorithms. Its proven success at transferring terabytes of data gathered from outer space across long distances is a testament to its significant commercial promise. In this work, our objective is to examine a range of security methods used on existing mature protocols such as TCP and UDP and evaluate their viability for UDT. We highlight the security limitations of UDT and determine the threshold of feasible security schemes within the constraints under which UDT was designed and developed. Subsequently, we provide ways of securing applications and traffic using UDT protocol, and offer recommendations for securing UDT. We create security mechanisms tailored for UDT and propose a new security architecture that can assist network designers, security investigators, and users who want to incorporate security when implementing UDT across wide area networks.

We then conduct practical experiments on UDT using our security mechanisms and explore the use of other existing security mechanisms used on TCP/UDP for UDT. To analyse the security mechanisms, we carry out a formal proof of correctness to assist us in determining their applicability by using Protocol Composition Logic (PCL). This approach is modular, comprising a separate proof of each protocol section and providing insight into the network environment in which each section can be reliably employed. Moreover, the proof holds for a variety of failure recovery strategies and other implementation and configuration options. We derive our technique from the PCL on TLS and Kerberos in the literature. We maintain, however, the novelty of our work for UDT particularly our newly developed mechanisms such as UDT-AO, UDT-DTLS, UDT-Kerberos (GSS-API) specifically for UDT, which all now form our proposed UDT security architecture.

We further analyse this architecture using rewrite systems and automata. We outline and use symbolic analysis approach to effectively verify our proposed architecture. This approach allows dataflow replication in the implementation of selected mechanisms that are integrated into the proposed architecture. We consider this approach effective by utilising the properties of the rewrite systems to represent specific flows within the architecture to present a theoretical and reliable method to perform the analysis. We introduce abstract representations of the components that compose the architecture and conduct our investigation, through structural, semantics and query analyses.

The result of this work, which is first in the literature, is a more robust theoretical and practical representation of a security architecture of UDT, viable to work with other high speed network protocols.

# Publications

Most of the chapters that are presented in this dissertation have been accepted, published or have been submitted for publication in refereed /peer reviewed research journals and conference proceedings (IEEE, Elsevier, Springer Verlag – LNCS and IETF).

## Research Accomplishments:

Served as Technical Session Chair at the Information Security Assurance Conference in Japan, sponsored by Springer –Verlag Berlin Heidelberg in 2010.

Served as research and technical reviewer for the following international journals:

- Computer & Security –Elsevier 2011, 2012
- International Journal of Earth Science Informatics - Springer Verlag 2012
- Management of Information Systems -MIS Review (MISR) 2012
- Open Access Journals – Network Protocols and Algorithms, sponsored by Polytechnic University of Valencia 2010-2012
- International Journal of Network and Information Security 2009

This research work was supported by UTS FEIT, i-NEXT and Vice Chancellor Travel Grants. This was also partly supported by industry Global Science and Technology Initiatives Grant sponsored by Db2Powerhouse Social Enterprise.

## Best Paper Award

Bernardo, D.V., Hoang, D.B., (2010), 'Security Analysis of Proposed Practical Security Mechanisms for High Speed Data Transfer Protocol' 4th Information Security Assurance 2010, Japan, LNCS Springer –Verlag Berlin Heidelberg, June 23-25, 2010 (Book Chapter).

## Outstanding Research Presentation

Bernardo, D.V., (2012),' Enciphering the Thoughts: Towards Achieving Ultimate Information Security', 3rd International Arts and Sciences, Harvard University Boston, USA, May 27-31, 2012 ,ISSN 1943-6114 (Conference).

## Innovation Patents

Innovation in Encryption systems Patent 2012100172
Innovation in e-information systems Patent 2006100469

Portions of this work are published in the following publications:

## Other/Books

Bernardo, D.V., (2008) i-Think 1st Edition: Selected Works in Business , Technology, Research and Innovation, March 2008, book paperback, Sydney, Singapore, UK and USA, ISBN 978-0-646-486- 543. http://nla.gov.au/anbd.biban42560289

Bernardo, D.V., Hoang, D.B., (2009) Security Requirements for UDT, IETF[3] (working paper), RFC Request for Comments, Internet and Engineering Task Force.

Bernardo, D.V., (2009) i-Think 2nd Edition: Selected Works in Business , Technology, Research and Innovation, March 2008, book paperback, Sydney, Singapore, UK and USA, ISBN 978-0-646-486- 543. http://nla.gov.au/anbd.biban42560289

## International Journals

Bernardo, D.V., Hoang, D.B., (2011) 'Multi-layer Security Analysis and Experimentation of High Speed Protocol Data Transfer for GRID', *Int. Journal of Grid and Utility Computing (IJGUC)*, SN 1741-8488, ISSN 1741-847X. (by invitation)

Bernardo, D.V., Hoang, D.B., (2010), 'A Pragmatic Approach: Achieving Acceptable Security Mechanisms for High Speed Data Transfer Protocol – UDT', SERSC, *International Journal of Security and its Applications* Vol. 4, no. 3, ISSN 1738-9976.(by invitation)

Bernardo, D.V., Hoang, D.B., (2010),'Securing Data Transfer in the Cloud through Introducing Identification Packet, and UDT Authentication Option Field: A Characterization', *International Journal of Network Security and its Applications* ISSN 0974- 9330 and 0975-2307. (by invitation)

Bernardo, D.V., Hoang, D.B., (2009), 'Network Security Considerations for a New Generation Protocol UDT' *JCSIA- Journal of Computer Security and Information Assurance*, Volume 4 –Issue 4, 2009 , Dynamic Publishing, USA, ISSN 1554- 1010 (by invitation)

## Book Chapters

Bernardo, D.V., Hoang, D.B.,(2011), Security Technology 2011," Formalization and Information-Theoretic Soundness in the Development of Security Architecture for Next Generation Protocol – UDT" Jeju Island Korea, LNCS Springer–Verlag Berlin Heidelberg, December 8-10, 2011 (Book Chapter)

Bernardo, D.V., Hoang, D.B.,(2010), International Conference on Future Generation Communication and Networking 2010, "End-to-End Security Methods for UDT Data Transmissions" Jeju Island Korea, LNCS Springer–Verlag Berlin Heidelberg, December 13-15, 2010 (Book Chapter)

Bernardo, D.V., Hoang, D.B.,(2010), 4th Information Security Assurance 2010, "Security Analysis of Proposed Practical Security Mechanisms for High Speed Data Transfer Protocol", Japan, LNCS Springer–Verlag Berlin Heidelberg, June 23-25, 2010 (Book Chapter)

## International Conferences Publications and Proceedings

Bernardo, D.V., Hoang, D.B.,(2012), "Securing the Cloud, Dispelling Fears: An initiative" 16th IEEE Network Based-Information Systems (NBIS) Trustworthy Computing (TwC-2012) Workshop, Melbourne, September 26-28, 2012

Bernardo, D.V., Hoang, D.B.,(2012), "Symbolic Analysis of UDT Security Architecture " 26th IEEE Advanced International Network Information and Application -Workshop , AINA Fukuoka Japan, March 26-29, 2012

Bernardo, D.V., Hoang, D.B.,(2012), "Compositional Logic for Proof of Correctness of Proposed UDT Security Mechanisms " 26th IEEE Advanced International Network Information and Application AINA Fukuoka Japan, March 26-29, 2012

Bernardo, D.V., Hoang, D.B.,(2011), "Empirical Survey: Experimentation and Implementations of High Speed Protocol Data Transfer for GRID " 26th IEEE Advance International Network Information and Application –AINA and 8th FINA Frontiers Information Network and Application, Workshop Singapore, March 22-25, 2011

Bernardo, D.V., Hoang, D.B.,(2010), "A Conceptual Approach against Next Generation Security Threats: Securing High Speed Network Protocols " 2nd IEEE International Conference of Future Networks ICFN 2010 proceedings Sanya, China January 22-24, 2010

Bernardo, D.V., (2010), "UDT –AO Approach" 6th IEEE International Assurance and Security, sponsored by IEEE Intelligent Transportation Systems Society, Atlanta, USA August 23-25, 2010

Bernardo, D.V., Hoang, D.B.,(2009), 2nd IEEE ICCSIT 2009 proceedings re Network Security Considerations for a New Generation Protocol UDT, Volume 3, IEEE ISBN 978-1-4244-4519-6, Beijing China, August 8-11, 2009

Bernardo, D.V., Hoang, D.B.,(2009), 8th IEEE ICISM 2009 proceedings re Quantitative Security Risk Assessment (SRA) Method: An empirical case study , Coimbatore India, December 9 -11, 2009.

Bernardo, D.V., Hoang, D.B.,(2010), 4th IEEE International Conference on Emerging Security Information, Systems and Technologies SECURWARE 2010 re “Protecting Next Generation High Speed Protecting–UDT through Generic Security Service Application Program Interface GSS-API” Venice/Mestre, Italy , July 18-25, 2010.

The research work presented in this thesis has been performed jointly with

Prof. Doan B. Hoang

Head of School

Director, iNext, School of Computing and Communications

Faculty of Engineering and Information Technology

The University of Technology, Sydney

NSW, 2000

Australia.

| Abbreviations | Description |
|---|---|
| ACK2 | Acknowledge of ACK |
| AES | Advanced Encrypt Standard |
| AH | Authentication Header |
| AIMD | Additive Increase/Multiplicative Decrease Algorithm |
| AIP | Accountable Internet Protocol |
| AO | Authentication Option |
| BDP | Bandwidth-Delay Product |
| BiC | Binary Increase Congestion Control |
| CCC | Configurable Congestion Control |
| CGA | Cryptographically Generated Addresses |
| CRS | Constrained Rewrite Systems |
| CTCP | TCP Congestion Control Algorithm |
| DCCP | Datagram Congestion Control Protocol |
| DTLS | Data Transport Layer Security |
| EMIST | Evaluation Methods for Internet Security Technology Tool |
| FAST | FAST TCP Avoidance Algorithm for Long Distance |
| GRS | Growing Rewrite Systems |
| GSS-API | Generic Security Service - Application Program Interface |
| HI /HIT | Host Identifier |
| HIP | Host Identity Protocol |
| IDS | Intrusion Detection System |
| IP | Internet Protocol |
| IPS | Intrusion Prevention System |
| IPSec | Internet Protocol Security |
| IPv4 | Internet Protocol Version 4 |
| IPV6 | Internet Protocol Version 6 |
| KDF | Key Derivation Functions |
| MAC | Message Authentication Code |
| MD5 | Message-Digest algorithm 5 |
| MSS | Maximum Segmentation/Segment Size |
| MTU | Maximum Transmission Unit |
| NAK | Negative Acknowledgement |
| NBN | Network Broadband Network |
| NS2 | Network Simulation 2 |
| PCL | Protocol Composite Logic |

| | |
|---|---|
| P2P | Peer to Peer |
| POP3 | Post Office Protocol 3 |
| PSK | Private Shared Keys |
| RFC | Request For Comments |
| RSA | Rivest, Shamir and Adleman Algorithm |
| RTT | Round-Trip Time |
| SASL | Simple Authentication and Security Layer |
| SDSS | Sloan Digital Sky Survey |
| SHA-1,2,256 | Secure Hash Algorithm |
| SMTP | Simple Mail Transport Protocol |
| STCP | Stream Transport Protocol |
| SYN | Synchronisation/Synchronise |
| TCP | Transmission Control Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| UDT | UDP-based Data Transfer |
| VLAN | Virtual Local Area Network |
| VPN | Virtual Private Network |
| WOAN | Wide Optical Area Network |
| XCP | Xplicit Congestion Protocol |

# Table of Contents

*In Deo confidimus*

# Chapter 1

# Introduction

The rapid growth of advanced high-speed networks has created opportunities for new technology to prosper. With this trend, high-speed networks are becoming increasingly available, such that the Australian government recently initiated the ambitious project of implementing the National Broadband Network (NBN) [120], bringing high-speed networks through optical fibre connections across the country. The utilisation of high-speed networks addresses many of the problems consumers are confronting, such as education disparity, high carbon emissions, and slow delivery of e-health.

The increasing requirements of high-speed networks, meanwhile, have subsequently pushed researchers to develop new protocols that support high density data transmissions in various networks. Many of these protocols are Transport Control Protocol (TCP) [6,46] variants, which have demonstrated better performance in simulation and several limited network experiments. However, they have limited practical applications because of implementation and installation difficulties. Meanwhile, users who need to transfer bulk data (e.g., in Cloud/GRID computing) frequently turn to application level solutions, where these variants become problematic. UDP-based protocols like UDT (UDP-based Data Transport Protocol) are among the protocols considered in the application level solutions for Cloud/GRID computing.

UDT — a fast data transfer protocol — was successfully implemented by capturing data from outer space, gathering terabytes of information, and transferring these across the continents in a high-speed network. This demonstrates a compelling commercial promise in data communications networks.

Whilst many types of protocols solve many of the problems in terms of achieving speed and better network performance, one problem that continues to dominate thus hindering their progression: security.

Today, weak security – or the lack of it – continues to be a perennial challenge to various network implementations.

## 1.1 Contributions

This work introduces for the first time a security architecture for a UDP-based protocol: UDT. In verifying this architecture, extensive reviews, validations, and implementations of security mechanisms are performed. Some of these mechanisms are created and subjected to theoretical and practical validations to achieve proofs of secrecy, authentication, and applicability to sustain the architecture in securing UDT.

In 2009, part of the early work [22] on UDT Security Architecture formed fraction of the proposal that was then put forward to the International Engineering Task Force (IETF) [33]. Further improvement of this proposal, however, will continue through this thesis and its enhanced version will be presented to IETF in the future. The rationale is to standardise UDT Security Requirements and Architecture to support application and network deployments.

The architecture will be exhibited with supplemental information on the schemes that can provide a foundation for basic, if not comprehensive, security of data flow, specifically in the higher-level communication layers.

This work thus presents the following paradigms to the security analysis and implementations of our newly developed and proposed security mechanisms specific to UDT. These mark a first in the literature:

1. Where practical validations pose constraints, formalisation of inductive properties in a set of newly introduced axioms; inference rules and proof

of soundness of the proposed mechanisms over the widely used model (Chapter 5); and the foundational development of these mechanisms for inductive proof analysis and automata of their security properties (Chapters 5 &6) are presented.

2. Formalisation of the proposed architecture through proofs of correctness of mechanisms and secrecy properties in data flow (Chapter 6).

On the practical side, the theoretic approach to the analysis of the real world UDT implementations – with proposed security mechanisms – is extensively applied. These also mark a first in the literature:

1. Systematic investigation of the design and implementation of security – specifically, its absence in UDT.

2. Proofs of authentication and secrecy properties of Generic Security Service – Application Program Interface (GSS-API) and Kerberos, with both symmetric and public-key initialisation in the given theoretic models. (Chapters 5 & 6).

3. Proofs of secrecy and authentication properties of the created UDT-Authentication Option (AO), UDT + Datagram Transport Layer Security (DTLS), and UDT+GSS-API in a real world test environment (Chapters 5 & 6).

4. Development of a proprietary UDT visualisation tool in Java (Chapter 4). While there are existing sophisticated tools available for evaluating the performance of existing protocols, none was available specifically for UDT. We developed a unique Java program that scans UDT static data in a file and demonstrates them in a graph. This is an initial step to developing a clear understanding of how data sets vary when gauging UDT's performance in different scenarios. It is noteworthy that the development of a proprietary tool to assist the evaluation of UDT merits a separate thesis in the research area of network protocols' performance, their evaluations, and simulations.

5. Evaluation of the role of UDT Security Architecture in Cloud/GRID in climate change initiatives (Epilogue).

## 1.2 Organisation

The compositions of each chapter were published in peer-reviewed international conferences, proceedings, and journals. Some of these conferences were sponsored by Springer Verlag Lecture Series in Computer Science (LNCS) and the Institute of Electrical and Electronics Engineers (IEEE).

Considering this thesis was published in multiple peer-reviewed conferences and publications, the chapters that form this work are organised  that incorporates the comments of various reviewers. In Chapter 2, we present an overview of UDT [82]. We review existing research on UDT [22-33, 82]. We review existing literature and present our proposed security designs and implementations. We then outline the motivation behind this work. In Chapters 3, 4, and 5, we present existing and new approaches to secure UDT. These chapters also describe theoretic analyses, experiments, simulations, and implementations of these approaches. In Chapter 6, we outline the architecture, analysing it through rewrite systems and automata. In Chapter 7, we conclude the dissertation and describe possible directions for future work. In the Epilogue, we present additional contribution of our work to climate change initiatives.

## 1.3 Background

Recent developments in network research introduced UDT, which is considered to be one of the next-generation of high performance data transfer protocols [82]. UDT introduces a new three-layer protocol architecture 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 also provides better usability by supporting a variety of network environments and application scenarios [22-33]. It addresses TCP's limitations by reducing the overhead required to send and receive streams of data.

One example of the implementation of UDT is the Sloan Digital Sky Survey (SDSS) project [82, 79, 139], which involves mapping in detail one quarter of the entire sky and determining the positions and brightness of more than 300 million celestial objects. The project measures distances to more than a million galaxies and quasars. The data from the SDSS project have so far increased to 2 terabytes, and this number continues to grow [82,139]. Currently, these terabytes of data are delivered to Europe via Chicago, then to the Asia-Pacific region, including Australia, Japan, South Korea, and China. Astronomers execute online analyses on multiple datasets stored in geographically distributed locations [79].

This implementation offers a promising direction for the future deployment of high-speed data transfer in various industries. However, for the industries to benefit from this technology, it is of utmost importance that the data must be secured and UDT itself must be protected in wide area networks.

However, the present challenge of reducing the cost and complexity of running streaming applications over the Internet as well as through wireless and mobile devices – all while maintaining security and privacy for their communication links – continues to mount.

This challenge is compounded by the absence of well-thought-out security mechanisms for protocols (such as UDT) during its early stage of development; it is this that drives this dissertation to introduce novel ways of securing UDT in extensive implementation scenarios.

The goal is to introduce an architecture that supports the modularity and structure of a protocol such as UDT. To develop this architecture, and to further enhance our work, we introduce application and IP-based mechanisms as well as a combination of existing security solutions of existing layers.

The proposed [22-33] architecture will adequately address vulnerability issues by implementing security mechanisms in UDT while maintaining transparency in data delivery. Its development is based on the analyses drawn from the source codes of UDT found at SourceForge.net. The source codes are analysed and tested on Windows and Linux environments to gain a better understanding of the functions and characteristics of this new protocol. A data analysis tool developed to visualise UDT data transmission in either secure or non-secure environments will be used.

Also to be performed are network and security simulations such as NS2 [86,123] 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. Furthermore, we will survey and use other available security network devices and tools (e.g., firewalls) that are widely used in the industry.

Most of the security vulnerability testing, meanwhile, will be conducted through penetration and traffic load tests. The results will provide significant groundwork for the development of a proposal and, eventually, of an architecture encompassing a variety of mechanisms designed to secure UDT against various adversaries, such as Sybil, man-in-the-middle, and the most common, Denial-of Service (DoS) attacks.

## 1.4 Overview

In this section, we discuss our research on developing a unified security architecture for UDT. We use the terms ‘approach,’ ‘methodology,’ ‘method,’ ‘framework,’ and ‘architecture’ interchangeably in this dissertation as these terms share connotations in both our past and present publications.

In a part of this dissertation published in [22-33], we highlighted the security limitations of UDT and determined the threshold of feasible security schemes within the constraints under which UDT was designed and developed. We introduced a method of securing applications and traffic using the UDT protocol and offered recommendations to meet security requirements for UDT.

Here, we summarise the breadth of security methods proposed for UDT and review the results. We present an improved security methodology and architecture after extensive specification and conformance tests. The results from these tests can assist network and security investigators, designers, and users who consider and incorporate security when implementing UDT across wide area networks.

## 1.5 Related Works

We present a security architecture with various feasible mechanisms that can secure UDT [22-27, 33]. This architecture focuses on UDT's position in the Open Systems Interface (OSI) layer architecture, which can provide a layer-to-layer approach to address security. We develop the architecture with the knowledge that UDT security relies mainly on the security in existing mature protocols.

A summary of security mechanisms and their implementations is presented in Figure 1-1. This summary is used as a basis to create a comprehensive security architecture, which is presented in Chapter 6.

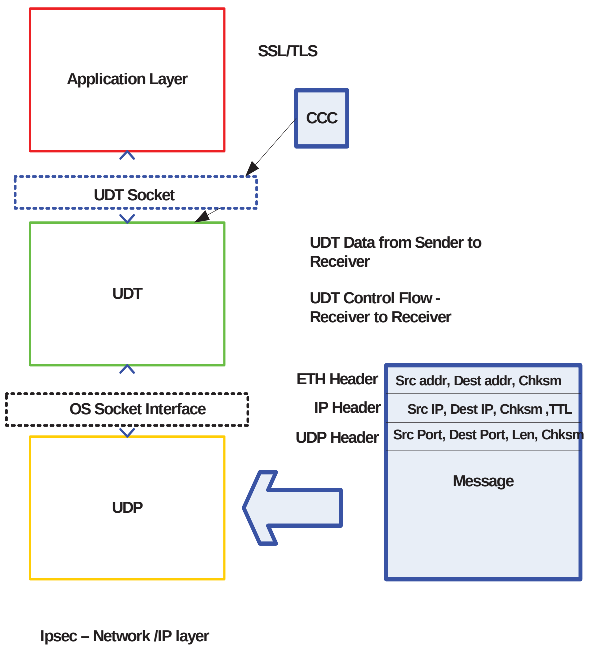

Application Layer
SSL/TLS
CCC
UDT Socket
UDT
UDT Data from Sender to Receiver
UDT Control Flow - Receiver to Receiver
ETH Header
IP Header
UDP Header
Src addr, Dest addr, Chksm
Src IP, Dest IP, Chksm ,TTL
Src Port, Dest Port, Len, Chksm
Message
OS Socket Interface
UDP
Ipsec – Network /IP layer

Figure 1-1: UDT in Layer Architecture. UDT is in the application layer above UDP. The application exchanges its data through the UDT socket, which then uses the UDP socket to send or receive data [22-33].

Because UDT operates between the application and transport layers running on top of UDP, data being carried must be transmitted securely and correctly. This must be implemented by each application, an operating system, and, whenever possible, by proprietary mechanisms using a separate stack [22-33].

The implementation must be based on generic libraries [30] and supported by the application-dependent components, such as the API module, the sender, receiver, and UDP channel. It also relies on the sender's protocol buffer, receiver's protocol buffer, sender's loss list, and receiver's loss list [24].

## 1.6 Constraints and Hypotheses

UDT is a connection-oriented duplex protocol [30], 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 has been designed so that rate control updates the packet sending period at constant intervals, while flow control updates the flow window size each time an acknowledgment packet is received. The protocol has also been expanded to satisfy additional requirements for both network research and applications development [21-31]. This expansion is called Composable UDT and is designed to complement kernel-space network stacks. However, this expansion is intended for:

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

- Support of application-aware algorithms;

- Ease of testing new algorithms for kernel space.

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 [33] emphasise 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, therefore, does not prevent UDT from simulating TCP's congestion avoidance behaviour [23, 30, 32-33].

UDT, moreover, is designed with the Configurable Congestion Control (CCC) interface that uses the following techniques: 1) control event handler call backs, 2) protocol behaviour configuration, 3) packet extension, and 4) performance monitoring. Its features can be used for bulk data transfer and streaming data processing, unlike TCP, which cannot be used for this type of processing because of two impediments: firstly, in TCP the link must be clean (with little packet loss) for it to fully utilise the bandwidth; secondly, when two TCP streams start at the same time, the stream with the longer Round-Trip Time (RTT) will be starved, due to the RTT bias problem [26, 28]; the data analysis process will thus have to wait for the slower data stream.

Since UDT does not have well-thought-out security mechanisms, we approach the development of these mechanisms in three phases (Figure 1-2): first, by developing research questions based on the building blocks (the objectives and aims of this work); second, by drawing research outcomes based on the results of analyses and methods; and third, by confirming the techniques and strategies (such as simulation, analysis, experimentation, implementation, and evaluation) that can be used in this development.

The need for security mechanisms for UDT is derived [22-33] from the following observations about UDT:

- Its dependencies on user preferences and implementation on the layer on which it is implemented;

- Its dependencies on existing security mechanisms of other layers on the stack;

- Its dependencies on TCP/UDP, which are dependent on nodes and their addresses for high-speed data transfer protocols.

This research, therefore, explores the existing security tools and determines which of these can best secure UDT in a networked environment. The following research questions are investigated in this work:

- *Can UDT address existing and future network adversaries and threats?*
  **Hypothesis:** UDT can practically be secured through a variety of security mechanisms.

- *Are the proposed methods of securing UDT, UDP, and TCP materialising possible on the application, presentation, transport, and network layers?*
  **Hypothesis:** The proposed methods can be implemented on selected layers.

- *Which are the best and most practical methods to secure UDT?*
  **Hypothesis:** Both commercially proven and proprietarily developed security mechanisms can best secure UDT.

- *Are the methods applicable to existing protocols, and can they be used in the development of new fast data protocols?*
  **Hypothesis:** Methods of securing UDT on the application, session, transport, and Internet Protocol (IP) can be used in future protocols.

These questions are addressed using various approaches, including theoretic inductive proofs, simulation, and experimentation.

With the aforementioned taken into consideration, this work investigates a way to secure UDT, its practical use in networked environments, and its contribution to future applications and networks. We explore and analyse various security mechanisms, such as GSS-API [23,99,109-110,148], UDT-AO [19,32,36], Cryptographically Generated Addresses (CGA) [11, 22-33], Host Identity Protocol (HIP) [7, 12, 83, 96, 105-106,118,137], DTLS/TLS [59-60, 128], Internet Protocol Security (IPSec) [21-33] and propose the best method to secure UDT.

The following are the potential applications of this research:

- Techniques for the development of security mechanisms for protocol libraries;

- Provision of well-thought-out security mechanisms and architecture for existing and future protocols; and

- Deployment of more secure data transfer in a Cloud/GRID.

The techniques and methods for securing UDT (and other future protocols), as presented in this dissertation, provide a foundation for developers seeking to secure next-generation protocols.

### 1.6.1 Research Objectives and Scope

This dissertation focuses on UDT and proposes a practical security architecture for the protocol.

#### 1.6.1.1 Scope

The scope of this work is to develop mechanisms suitable for UDT and other fast data protocols, which currently have no security mechanisms in place. This work attempts to address at least three research problems:

1. The introduction of new techniques, which can be achieved through characterisation and utilisation of implementation as a validation scheme;
2. The creation of an empirical model, which can be achieved by addressing the question of generalisability and by using analysis as validation; and
3. The realisation of an analytic model and architecture, which can be achieved by addressing the question of selection, and by using experience, simulation, and experimentation as validation schemes based on the activities and implementations performed in a controlled environment.

#### 1.6.1.2 Key Research Objectives

The key research objectives of this work are framed by the research model (Figure 1-2):

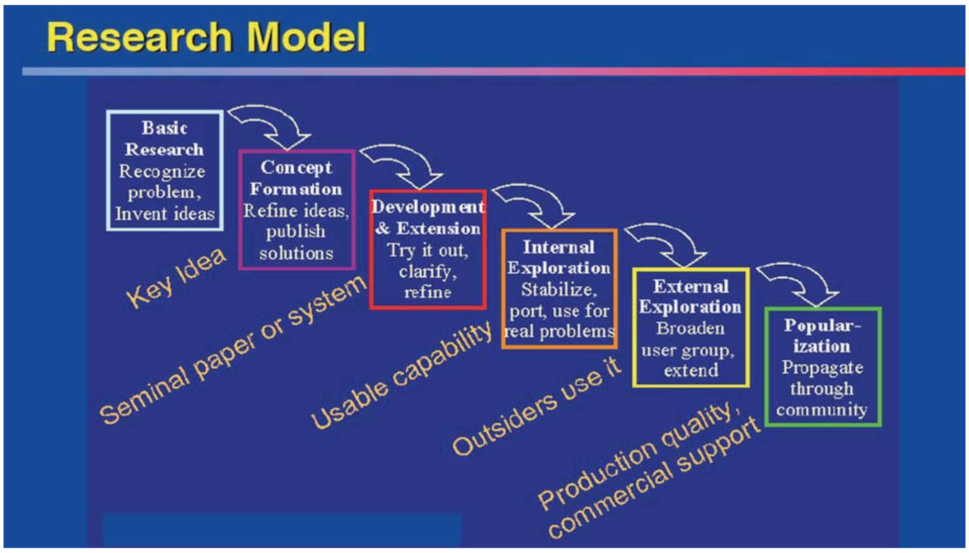

Research Model
Basic Research Recognize problem, Invent ideas
Concept Formation Refine ideas, publish solutions
Development & Extension Try it out, clarify, refine
Internal Exploration Stabilize, port, use for real problems
External Exploration Broaden user group, extend
Popular-ization Propagate through community
Key Idea
Seminal paper or system
Usable capability
Outsiders use it
Production quality, commercial support

Figure 1-2: Research Model

- To explore the various security mechanisms available, as well as their uses to existing protocols;
- To conduct a comprehensive security analysis on UDT;
- To provide mechanisms suitable for securing UDT in a networked environment; and
- To establish the practicality of such mechanisms for other fast protocols.

The scope and objectives of this work are determined by the research questions, by the strategies for achieving research outcomes, and by the use of a few validation schemes for achieving practicability. The guiding principles of this work are based on the research phases (Figure 1-3), which contributed to the development of the comprehensive building blocks of this work.

We demonstrate a comprehensive way of achieving research outcomes. Since the focus of the research is on technology – and not on computer science alone – the identification of some key features to describe modern technology is performed.

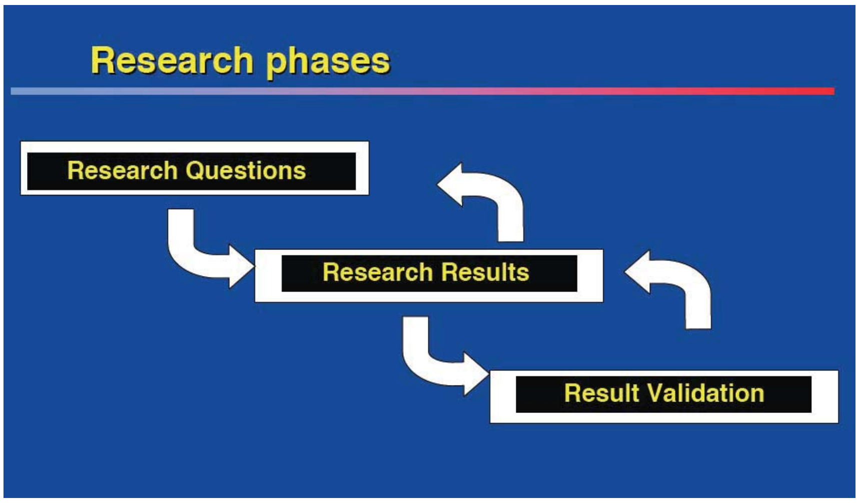


Figure 1-3: Research Phases

Since UDT is designed to run on UDP [82], it depends on UDP's existing security mechanisms. Consequently, designers of the applications using UDT are faced with limited security choices.

In our published works [22-33], we presented an overview of the basic security mechanisms for UDT. As the research progressed, we achieved the following:

Firstly, we modified the UDT codes – changing the Maximum Segment Size (MSS) values, introducing a checksum, and using Message-Digest Algorithm 5 (MD5) [88, 107, 146] in its codes. However, this is only suitable for some applications.

Secondly, we designed custom security mechanisms on the application layer, using API (such as GSS-API) or custom security mechanisms on the IP layer (such as HIP-CGA or IPSec).

Thirdly, we introduced UDT-AO for secrecy and authentication in data transmission.

Finally, we integrated existing transport layer implementation schemes, such as DTLS [22-33, 114].

The following mechanisms can be significant for application- and transport-layer-based authentication and end-to-end security for UDT. These are as follows:

**Security through:**

**IP Layer**

- HIP – Host Identification Packet [7, 83,96,105-106,118,137]
- Cryptographically Generated Address (CGA) [11, 22-33]
- Self-certifying addresses using HIP-CGA [7, 11, 22-33, 83, 96, 105-106, 118, 137]
- IPSec – IP security [21-33]

**Session/Application and Transport Layers**

- GSS-API - Generic Security Service Application Program Interface [22-31, 99,109-110,148]
- UDT-AO - Authentication Option [19, 32, 36]
- SASL - Simple Authentication and Security Layer (SASL) [114]
- DTLS – Data Transport Layer Security [59-60, 128]

In addressing UDT's security requirements, we present powerful paradigms for the security analysis of the newly developed and proposed security mechanisms through formal language and practical implementations. These paradigms support the development of the security architecture of UDT while achieving substantial and compositional verification of each of the proposed mechanism in isolation.

# Chapter 2

# State-of-the-Art Protocol

The growth of network bandwidths since the introduction of packet switching has contributed to the significant rise in Internet traffic. In recent years, new applications such as peer-to-peer (P2P) file sharing, multimedia, and mobile computing have increased users' expectations, motivating new designs in which various communication links, such as GRID [81], satellite, wireless and mobile computing, can securely participate and handle traffic at higher layers of the protocol stack.

These new applications vary in traffic, connection characteristics, and communication links. While most of these applications still use TCP for data transfer because of its reliability and stability, performance issues have been noted in the implementation of large networks that require high bandwidths.

These issues have led to the development of new and different schemes with more reliable characteristics and better congestion control. One example is XCP (eXplicit Congestion control Protocol) [47,100], which demonstrates good performance characteristics when tested on routers and satellite systems [100]; other variants, meanwhile, such as STCP and DCCP, are designed to improve congestion control. However, a number of these new schemes face challenges on deployment because they require changes in the routers as well as the operating systems of end hosts. Recent studies have shown that the gradual deployment to update Internet-facing routers results in a significant performance drop. XCP, with characteristics similar to TCP, has also exhibited a number of security flaws.

Apart from congestion control and performance, for which TCP variants were originally developed, security considerations also need to be included in the architectural designs of the new generation of protocols [24].

Developments in 2007 introduced the state-of-the-art UDT, a next generation of high-performance data transfer protocol. UDT introduces a new three-layer protocol architecture composed of a connection flow multiplexer, enhanced congestion control and resource management. The design allows the protocol to be shared by parallel connections and by future connections. It also improves congestion control and reduces connection set-up time. Moreover, UDT provides better usability by supporting a variety of network environments and application scenarios [22]. It addresses TCP's limitations by reducing the overhead required to send and receive streams of data. However, the pressure to reduce the cost and complexity of running streaming applications over the Internet and through wireless and mobile devices continues to mount. Users have also expressed the demand for better security and privacy for their communication links. Despite being widely used, existing protocols, e.g., TCP and UDP, have a number of inherent serious security flaws.

This work focuses on UDT's security requirements, based on existing network protocols. It is aimed at determining and developing security mechanisms to form a robust security architecture that will preserve the security and privacy of the data flow.

Since UDT relies on UDP to check IP streams, it is susceptible to attacks such as snooping, packet interception, and IP masquerading. Its objective is to deliver bandwidth-intensive applications over a protocol that carries a minimal amount of overhead (such as UDP), but it cannot guarantee that it will avoid compromising the security, privacy, and data integrity desired by users.

Furthermore, UDT is a UDP-based approach [31-33] and is considered to be the only UDP-based protocol that employs a congestion control algorithm targeting shared networks. It is a new application-level protocol with support for user-configurable control algorithms and more powerful APIs.

## 2.1 Transport Protocols and Network Congestion Control

In this literature review, we discuss existing Internet transport protocols and network congestion control algorithms. We briefly demonstrate the layered architecture of Transmission Control Protocol/Internet Protocol (TCP/IP) [6,61,62,69] and discuss UDT based on the existing literature [72], which we fully acknowledge in this section.

In order to provide various functionalities to applications, including but not limited to data delivery, data reliability control, and streaming or messaging service, transport protocols are designed and created with four fundamental objectives usually transparent to such applications: efficiency, fairness, convergence, and distributedness [81-82].

[146,151] highlighted that transport protocol also needs to be efficient: it needs to utilise the available bandwidth as efficiently as possible. To be efficient, as further explained by [72] and supported by [135], a protocol must accomplish the following two tasks in a short time: a) probe the maximum available bandwidth, and b) recover to maximum speed when congestion or packet loss causes a drop in the sending rate. Once it reaches maximum speed, it should remain at its current state until the network situation changes, i.e., oscillations should be as small as possible [82].

On the other hand, the network bandwidth is expected to be shared fairly among all concurrent flows. The measurement of fairness can have different standards. The most common one is the max-min fairness, the objective of which is to maximise the minimum throughput [81-82].

Literature [81-82,135,151-152] defines intra-protocol fairness of a protocol as a fairness property among all flows belonging to the same protocol. In particular, RTT (Round-Trip Time) independence is used to describe the special case of fairness over topology with different RTTs; this is not satisfied by TCP [152]. The fairness problem becomes more difficult when heterogeneous protocols coexist. A new transport protocol is required to consider the situation wherein it coexists with TCP before it is widely deployed on the Internet. The fairness between TCP and the new protocol is called TCP friendliness [151-152].

According to [82], the data sending rate must converge to a unique equilibrium from any starting point, given any specific network situation. Because binary feedbacks are typically used to notify changes in the network situation, it is thus acceptable that the throughput oscillates around a fixed point [81-82]. This is the global stability property of Internet transport protocols.

Finally, because the Internet is such a large distributed system, it is impossible to have a server dispatch the bandwidth. It is at the end hosts that transport protocols [151] must control their data sending rate, with or without assistance from the routers through which the traffic passes. The end-to-end principle [27,35,73,84] states that, whenever possible, transport protocol operations should only occur at end hosts in order to increase the system's scalability. It is also necessary that end hosts have congestion control functionalities [30], even with the existence of gateway operators.

In order to achieve these objectives, congestion control is utilised in the transport protocol. The transport protocol adjusts the data sending rate using a certain congestion control algorithm, which functions as a feedback system and produces feedback that can either be explicitly generated from intermediate nodes such as routers; or estimated by packet losses, increase trends in packet delay, or timeout events [27,82]. Explicit feedback from routers brings more accurate information, but it also requires higher computation and deployment costs. The data sending rate can be tuned through either the inter-packet time or the number of outstanding packets. The former method is called rate-based congestion control while the latter is called window-based congestion control. Both methods can be applied at the same time. A linear system is often applied in a control scheme to tune these parameters because of its simplicity [84]. The most famous control algorithm is TCP's AIMD algorithm [6], or additive increase/multiplicative decrease algorithm [81-82].

### 2.1.1 TCP's Constraints

TCP has been widely adopted as a data transfer protocol for high-speed networks. However, many literature reviews [6,81-82,151-152] emphasise that TCP substantially underutilises network bandwidth over high-speed connections. TCP [6,46] increases its congestion window by one at the length of Round-Trip Time (RTT) and reduces it by half at a loss event [81]. As discussed in the works of [6,81-82], in order for TCP to increase its window for full ulitilisation of 10 Gbps, for example, with 1.5 kilobyte packets, it requires over 83,333 RTTs. Moreover, with 100ms RTT, it takes approximately 1.5 hours for full utilisation in steady state according to Gu [82]; therefore, the loss rate cannot be more than 1 loss even per 5 Gbyte packets, which is less than the theoretical limit of the network's bit error rates.

TCP's AIMD-based control algorithm [6,46] increases the sending rate (via congestion window size) by approximately 1 segment per RTT, but halves it once there is a loss event.

The throughput of a TCP flow can be approximately modeled by [6,151-152]

$$T = \frac{S}{R\sqrt{\frac{2p}{3}} + t_{RTO}\left(3\sqrt{\frac{3p}{8}}\right)p\left(1+32p^2\right)}$$

where $S$ is the TCP segment size, $R$ is the network RTT, $p$ is the loss rate, and $tRTO$ is the TCP timeout value.

A number of proposals [47,100] have been presented to fine-tune TCP parameters. One of these proposes an increase in packet size by setting the jumbo packet option to up to 64k bits, with multiple TCP connections in use according to [6,46]. This model indicates that TCP becomes ineffective as the network bandwidth and delay both increase [6, 22-33,81-82,151-152].

On the other hand, the existence of the RTT in the TCP throughput model means that concurrent flows with different RTTs may have different throughputs: a manifestation commonly known as RTT bias.

After acknowledging TCP's limitation, researchers responded by introducing several promising new protocols i.e., XCP and UDT [81-82]. These protocols – with the exception of XCP, a router-assisted protocol [100] – adaptively adjust their increase rates based on the current window size. Consequently, the larger the congestion window is, the faster it grows. These protocols are designed to be TCP-friendly [6,151-152] in high loss rate environments and highly scalable in low loss environments.

### 2.1.2 UDT – An Alternative

The widespread presence of short-lived, web-like flows on the Internet and TCP's stability drive the success of the use of Transmission Control Protocol. However, it has been noted [6,81-82,151-152] that the usage of network resources in high-performance distributed data-intensive applications is quite different from that of traditional Internet applications because of the following reasons: first, the data transfer often lasts a very long time at very high speeds; second, distributed applications need cooperation among multiple data connections. Therefore, fairness between flows with different start times and network delays is desirable.

Finally, in GRID computing over high-performance networks, the abundant optical bandwidth is usually shared by only a small number of bulk sources. The concurrency is much smaller than that on the Internet [6,22-33,100].

Here we adopt an example presented by [81-82]. It presented a simple but typical example application, called the streaming join. The main contention was assuming that the real-time data streams come from a remote machine A and a local machine B, which were joined by another local machine C with a window-based join algorithm [81].

It was assumed that the two data streams were composed of records of the same size.

Figure 2-1 illustrates the network topology (Figure courtesy of Gu [81-82]).

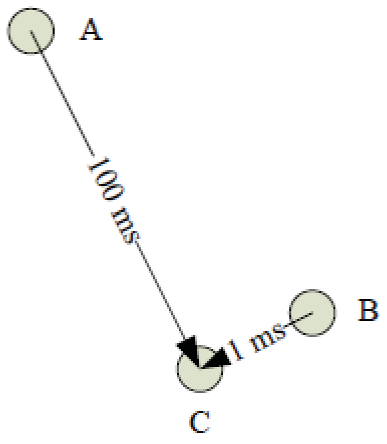


Figure 2-1: A streaming join example. The two data streams from A and B are sent to C and converged there. The RTT between A and C is 100ms, whereas it is only 1ms between B and C. Both links share a 1Gb/s bottleneck at C.[82]

In the experiment [82], TCP is used to transfer the streams, in both a real network and the simulated environment using NS-2 simulator. It is observed that the throughputs of the two streams are 3.52 and 863 Mb/s in the real network and 80.5 and 807 Mb/s in the simulation environment, respectively. The slower stream (AC), according to [82] limits the join throughput to AC*2, or 7 Mb/s in the real network and 160 Mb/s in the simulation environment (out of the 1 Gb/s maximum possible throughput). Although applications can sometimes tune the data source rate to alleviate this problem, this needs global knowledge of the network topology and static network environment, which is unrealistic in most cases.

In this dissertation, we analyse UDT in order to develop a comprehensive security architecture for distributed data-intensive applications in wide-area high-speed networks.

UDT addresses the solution by investigating two orthogonal research problems [72]: 1) the design and implementation of transport protocols with respect to throughput and CPU usage; and 2) the Internet congestion control algorithm with respect to efficiency, fairness, and stability.

UDT is an application-level, end-to-end, unicast, reliable, connection-oriented streaming data transport protocol. The UDT protocol is completely at user space above UDP, i.e., it uses UDP to transfer user data and protocol control information. UDT uses packet-based sequencing to check packet loss and guarantee data reliability. It is specially designed for high-speed bulk data transfer by aiming to remove or reduce the overhead of memory copy, loss information processing, acknowledging, etc. UDT provides reliable streaming data transfer service, similar to TCP.

The UDT protocol supports a large variety of control algorithms. Moreover, it supports congestion control algorithms to be configured at run time; each UDT flow can thus have its own control algorithm, and it can change the algorithm at any time.

The built-in (default) UDT congestion control algorithm is proposed to utilise high bandwidth efficiently and fairly. The UDT algorithm uses a loss-based AIMD mechanism. Bandwidth estimation technique is used to optimize its increase parameter dynamically. A random decrease factor is used to remove the negative effect of loss synchronisation [82].

According to [81-82], UDT is not used to replace TCP on the Internet, where the bottleneck bandwidth is relatively small and there are large amounts of multiplexed short life flows.

It must be emphasised that UDT, when coexisting with TCP flows, is designed not to occupy more bandwidth than does TCP, unless the TCP flows fail to utilise their fair share due to TCP's efficiency problems in high bandwidth-delay product (BDP) environments. TCP will still be used in these high BDP networks, and an application that uses UDT may sometimes run on public networks.

UDT is defined and distinguished by its three major aspects [82]:

・UDT is at the application level , thus promotes better deployment method than in kernel protocols. UDT is designed with an efficient and fair congestion control algorithm, which is considered a better approach than other UDP-based protocols that very limited congestion control capabilities [82].

- UDT itself is also a protocol framework with configurable congestion control, which according to [82] both support application awareness and support evaluation of new congestion control algorithms.

The development of UDT protocol addresses numerous research problems in data transport protocols [82]. This development makes the following specific contributions:

- UDT provides a timely and practical solution to the problem of transferring bulk data in high-speed wide-area networks.

Therefore, UDT is easily deployable. There are only four versions of TCP that have been widely deployed in the past three decades, and, according to [82], this is because of the long time lag of standardisation, implementation, and deployment of kernel space protocols. While there were numerous TCP variants proposed at the same time that UDT was developed, these protocols are not expected to be deployed widely in the near future. In addition, bandwidth estimation techniques are used in the UDT congestion control mechanism such that there is no need for the manual tuning of control parameters.

- Gu's [82] work systematically investigated the design and implementation issues of high-performance data transport protocol at the application level [22-33,81-82].

While often neglected, protocol design and implementation have a significant impact on efficiency. In the UDT project, we identified the overhead arising from acknowledgments, loss processing, threading, and memory copy, and to these we proposed appropriate solutions.

- UDT's congestion control algorithm addresses both efficiency and fairness objectives [81-82].

Therefore, UDT's algorithm takes approximately a constant time to converge to 90% of the available bandwidth. UDT flows are fair to each other, even if they have different RTTs. While UDT is highly efficient, it is not that aggressive. It is friendly to concurrent TCP flows. Furthermore, the UDT algorithm solves the loss synchronisation problem by using a random decreasing method.

Finally, UDT can also handle limited non-congestion packet losses.

- UDT's approach is highly scalable. Given that there is enough CPU power, UDT can support unlimited bandwidth within terrestrial ranges. No matter how fast the data transfer rate is, the timer-based selective acknowledgment generates a constant number of acknowledgments (ACKs). The congestion control algorithm and the bandwidth estimation technique, meanwhile, allow UDT to increase to 90% of the available bandwidth no matter how large it is. In addition, the constant rate control interval helps realize RTT fairness.
- Composable UDT offers more to application development and network research by allowing configurable congestion control algorithms. This feature enables easy development of application- or network-specific control mechanisms, as well as easy evaluation of new control algorithms.
- Finally, Gu [82] developed a productivity quality open source UDT library that can be used in real world applications and research work [81-82].

## 2.2 The UDT Protocol

In this chapter we describe how UDT works through its design and implementation. After our overview of the UDT protocol in section 2.1, we describe in detail the UDT protocol, including packet structures, connection maintenance, packet sequencing, acknowledging, and reliability control. We also introduce UDT's flow and congestion control in this section, followed by an analysis of the control algorithm in the next chapter. Finally, we present brief concluding remarks in section 2.8.

### 2.2.1 Overview

UDT adapts itself into the layered network protocol architecture (Figure 2-2), and uses UDP through the socket interface provided by operating systems. Meanwhile, it provides a UDT socket interface to applications, which can then call the UDT socket API in the same way they call the system socket API.

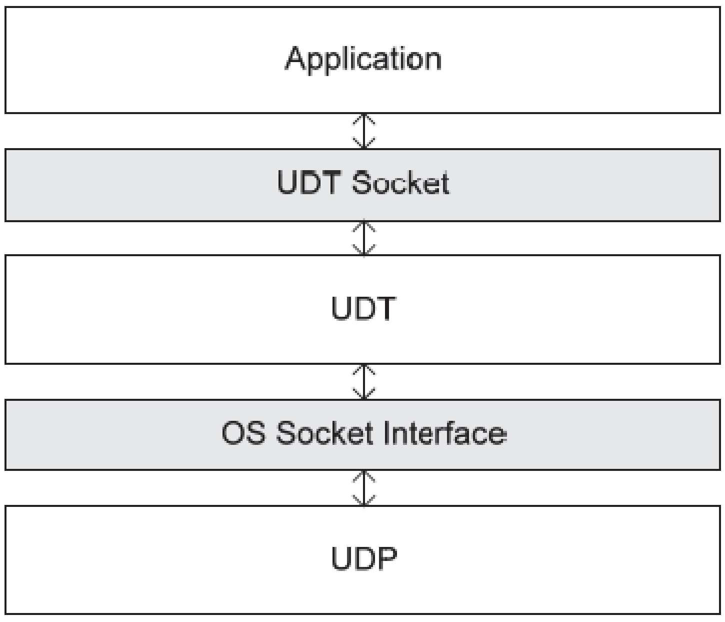


Figure 2-2 Layered architecture of UDT (courtesy of Gu [82]). In this layered architecture, the UDT layer is completely in user space above the network transport layer of UDP, whereas the UDT layer itself provides transport functionalities to applications.

UDT is a duplex transport protocol. Each UDT entity has two logical parts: the sender and the receiver. The sender sends (and retransmits) application data according to flow control and rate control. The receiver, meanwhile, receives both data packets and control packets, and also sends out control packets according to the received packets.

Figure 2-3 describes the relationship between the UDT sender and the receiver. In Figure 2-3, the UDT entity A sends application data to the UDT entity B. The data is sent from A's sender to B's receiver, whereas the control flow is exchanged between the two receivers.

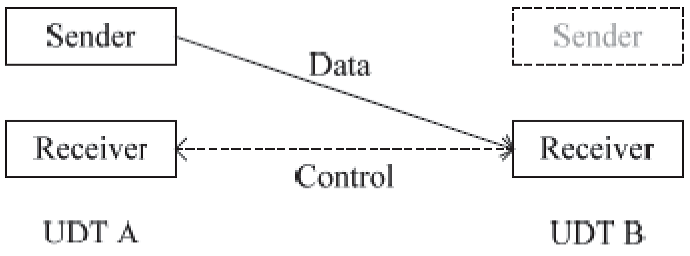


Figure 2-3: Relationship between UDT sender and receiver (courtesy of Gu [82]). All UDT entities have the same architecture, each having both a sender and receiver. This figure demonstrates the situation wherein a UDT entity A sends data to another UDT entity B. Data is transferred from A's sender to B's receiver, whereas control information is exchanged between the two receivers.

The receiver is also responsible for triggering and processing all control events, including congestion control and reliability control, as well as their related mechanisms.

UDT uses rate-based congestion control (rate control) and window-based flow control to regulate the outgoing data traffic. 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. UDT always tries to pack application data into fixed-size packets, unless there is not enough data to be sent. Since UDT is supposed to be used to transfer bulk data streams, we assume that there is only a very small portion of irregularly sized packets in a UDT session. The fixed size can be set up by applications and the optimal value is the path MTU (including all packet headers). The actual size of a UDT packet can be known from the UDP header [82].

## 2.2.2 Congestion Control

The congestion window size (*W*) is dynamically updated according to the product of packet arrival speed (*AS*) and the sum of SYN and RTT: $W = AS * (SYN + RTT)$. Here, SYN is the constant rate control interval, which is defined as 0.01 seconds in the current protocol specification.

For protocols that acknowledge every data packet, the maximum amount of data packets on the fly is the product of sending speed and RTT. In UDT, however, acknowledgment is triggered every SYN time, so that the value should be the product of sending rate and (SYN + RTT). In addition, we use the receiving speed instead of the sending speed, because the former can reflect the network situation more precisely.

UDT uses a modified AIMD algorithm [82], for which the formula is as follows [22-33,82].

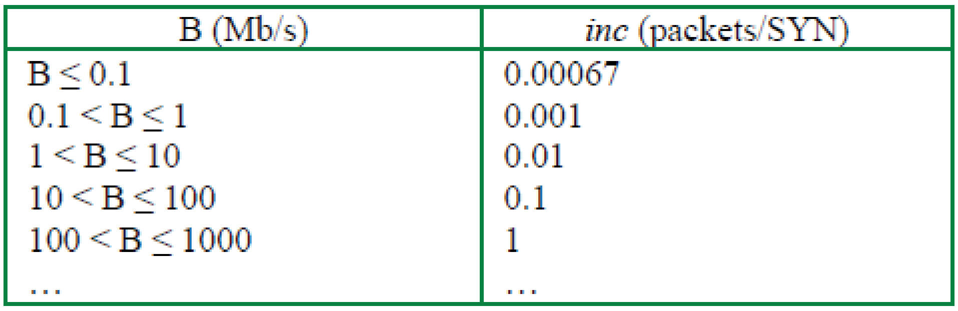
Every SYN time, if there is no NAK, but there are ACKs received in the previous SYN time, the number of packets to be increased in the next SYN time (*inc*) is calculated by:

(1) $$inc = \max(10^{\lceil \log_{10} B \rceil - 9},\ 1/1500) \times 1500 / MSS$$

where $B$ is the estimated available bandwidth in bits per second and *MSS* is the maximum segmentation size in bytes [82], which is also the fixed UDT packet size.

The easiest way to understand (1) is through Table 2-1, which gives examples of *inc*, wherein *MSS* is 1500 bytes. If *MSS* is not 1500 bytes, the increments listed in Table 2-1 will be corrected by the ratio of 1500/*MSS*.

Table 2-1: UDT increase parameter computation example (courtesy of Gu [82]). The first column represents the estimated available bandwidth and the second column represents the increase in packets per SYN. While the available bandwidth increases to the next scope of 10's integral power, the increase parameter also increases by 10 times.



| B (Mb/s) | *inc* (packets/SYN) |
|---|---|
| $B \leq 0.1$ | 0.00067 |
| $0.1 < B \leq 1$ | 0.001 |
| $1 < B \leq 10$ | 0.01 |
| $10 < B \leq 100$ | 0.1 |
| $100 < B \leq 1000$ | 1 |
| … | … |

B (Mb/s) *inc* (packets/SYN) [82]

$B \leq 0.1$ 0.00067

$0.1 < B \leq 1$ 0.001

$1 < B \leq 10$ 0.01

$10 < B \leq 100$ 0.1

$100 < B \leq 1000$ 1

… …

The packet sending period $P$ is then recalculated according to equation (2), where $P'$ is the current packet sending period [72]:

(2) $SYN / P = SYN / P' + inc$

Once a NAK is received, the packet-sending period is increased by 1/8:

(3) $P = P' * 1.125$

If the largest sequence number in this NAK is greater than the largest sequence number sent when the last decrease occurred according to [82], the sender stops sending packets in the next SYN time to help clear the congestion.

The UDT congestion control described above is not enabled until the first NAK is received or the flow window size has reached the maximum flow window size. This is the slow start period of the UDT congestion control. During this time, the inter-packet time is kept as zero. The initial flow window size is 2 and it is doubled each time an ACK is received. The slow start only happens at the beginning of a UDT connection, and once the above congestion control scheme is enabled, it will not happen again.

However, UDT was developed without a well-thought-out security architecture. Unlike TCP, many security mechanisms and architectures were developed to secure data, information, and communications.

## 2.3 UDT Packet Structures

UDT is designed to have two packet structures: the data packets and the control packets. These are distinguished by the first bit (flag bit) of the packet header. The data packet header starts with 0, while the control packet starts with 1.

**Data Packet**

| | | |
|---|---|---|
| 0 or 1 | Sequence Number | 0-31 bit |
| FF | Message Number | 29 bit |
| | Time Stamp | 32 bit |

Control Packet

| Flag 0 / 1 | Packet type | Information 0- 15 | User defined types | 0-31 |
|---|---|---|---|---|
| 1 | Type | | Extended Type | 31 bit |
| | | ACK Sub –Sequence Number | | |
| | | Time Stamp | | |
| | | Control Information | | |

Figure 2-4: UDT packet header structures [82]. The first bit of the packet header is a flag that indicates whether this is a data packet or a control packet. Data packets contain a 31-bit sequence number, a 29-bit message number, and a 32-bit timestamp. A control packet header, on the other hand, uses 1-15 bit for the packet type information, as well as 16-31 for user defined types. The detailed control information depends on the packet type [22-33,82].

The packet sequence number uses 31 bits after the flag bit. It uses packet-based sequencing, which means that the sequence number increases by 1 for each sent data packet in the order of packet sending. The Sequence Number is wrapped once it has increased to reach the maximum number ($2^{31}$ -1) [20,82].

As in other protocols such as DCCP, the sequence number is used to arrange packets into sequence, to detect loss [6,46] and network duplicates, and to protect against attackers, half-open connections, and delivery of very old packets. Every packet carries a Sequence Number; most packet types also include an Acknowledgment Number, which is carried in a control packet, the second packet structure of UDT. The control packet is parsed according to the structure if the flag bit of a UDT packet is 1.

UDT is a connection-oriented duplex protocol, 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 at constant intervals, whereas flow control updates the flow window size each time an acknowledgment packet is received. It has since expanded to satisfy additional requirements of both network research and applications development. This expansion is called Composable UDT, and it 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 the 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 [6,61,81] emphasised that a Composable UDT library does not implement the same mechanisms as in the TCP specification. TCP uses byte-based sequencing, whereas UDT uses packet-based sequencing. It was stressed that this does not prevent CTCP from simulating TCP's congestion avoidance behaviour [6,37,61].

## 2.4 UDT and Application Programming Interface

Application programming interfaces allow developers to write applications that can make use of UDT services. In this chapter, we provide an overview of the most common APIs for IP applications. We then present an approach in securing UDT by interfacing with APIs, using GSS-API to meet UDT security requirements.

The socket interface is one of several application programming interfaces (APIs) used in communication protocols. Designed to be a generic communication programming interface, it was first introduced by the 4.2 BSD UNIX system. Although it has not been standardised, it has become a de facto industry standard.

The socket interface is differentiated by the services that are provided to applications: stream sockets (connection-oriented), datagram sockets (connectionless), and raw sockets (direct access to lower layer protocols) services.

A variation of the BSD sockets interface is provided by the Winsock interface developed by Microsoft and other vendors to support TCP/IP applications on Windows operating systems. Winsock provides a more generalised interface, allowing applications to communicate with any available transport layer protocol and underlying network services, including, but not limited to, TCP/IP.

## 2.5 Uses of API

The following lists some common and basic socket interface calls. In the next section, we see an example scenario of using these socket interface calls [65].

### 2.5.1 Intitalise a socket

FORMAT

*int sockfd= socket(**int** family, int type, int protocol)*

*Where:*

- family stands for addressing family. It can take on values such as AF_UNIX, AF_INET, AF_OS2, AF_NS and AF_IUCV. Its purpose is to specify the method of addressing used by the socket.
- type stands for the type of socket interface to be used. It can take on values such as SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, and SOCK_SEQPACKET.
- protocol can be UDP, TCP, IP or ICMP or any other existing variants such as UDT.
- sockfd is an integer (similar to a file descriptor) returned by the socket call.

### 2.5.2. Bind (register) a socket to a port address

FORMAT

*int bind(int sockfd, struct sockaddr * localaddr, int addrlen)*

*Where:*

- sockfd is the same integer returned by the socket call.
- localaddr is the local address returned by the bind call.

Note that after the bind call, we now have the values for the first three parameters inside our 5-tuple association: *{protocol, local-address, local-process, foreign-address, foreign-process)*

### 2.5.3 Indicate readiness to receive connections

FORMAT

*int listen(int sockfd, int queue-size)*

*Where:*

- sockfd is the same integer returned by the socket call.
- queue-size indicates the number of connection requests that can be queued by the system while the local process has not yet issued the accept call.

### 2.5.4 Accept a connection

FORMAT

*int accept(int sockfd, struct sockaddr * foreign-address,*

*int addrlen)*

*Where:*

- sockfd is the same integer returned by the socket call.
- foreign-address is the address of the foreign (client) process returned by the accept call.

Note that this accept call is issued by a server process rather than a client process. If there is a connection request waiting on the queue for this socket connection, accept takes the first request on the queue and creates another socket with the same properties as sockfd; otherwise, accept will block the caller process until a connection request arrives.

### 2.5.5 Request connection to the server

FORMAT

*int connect(int sockfd, struct sockaddr * foreign-address,*

*int addrlen)*

*Where:*

- sockfd is the same integer returned by the socket call.
- foreign-address is the address of the foreign (server) process returned by the connect call.

Note that this call is issued by a client process rather than a server process.

### 2.5.6 Send and /or receive data.

The read(), readv(sockfd, char *buffer int addrlen), recv(), readfrom(), send(sockfd, msg, len, flags) and write() calls can be used to receive and send data in an established socket association (or connection).

Note that these calls are similar to the standard read and write file I/O system calls.

### 2.5.7 Close a socket.

FORMAT

*int close(int sockfd)*

*Where:*

- sockfd is the same integer returned by the socket call.

## Example Scenario

As an example, consider the socket system calls commonly presented for a connection-oriented protocol in Figure 2-5.

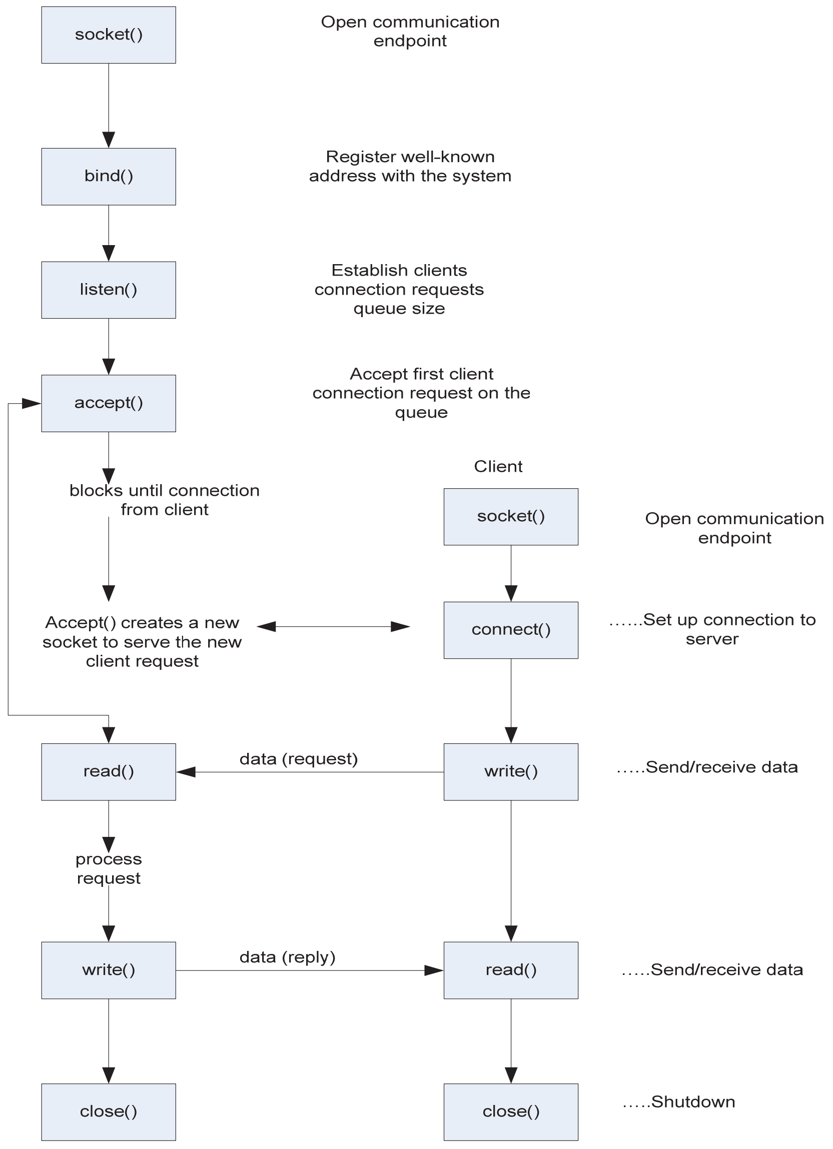

socket()
Open communication endpoint
bind()
Register well-known address with the system
listen()
Establish clients connection requests queue size
accept()
Accept first client connection request on the queue
blocks until connection from client
Client
socket()
Open communication endpoint
Accept() creates a new socket to serve the new client request
connect()
......Set up connection to server
read()
data (request)
write()
.....Send/receive data
process request
write()
data (reply)
read()
.....Send/receive data
close()
close()
.....Shutdown

Figure 2-5: Socket System Calls for Connection-Oriented Protocol

Consider the previous socket system calls in terms of specifying the elements of the association:

| | Protocol | Local Address, | Local Process | Foreign Address, | Foreign Process |
|---|---|---|---|---|---|
| connection-oriented server | Socket() | bind() | | Listen() | Accept() |
| connection-oriented client | Socket() | | connect() | | |
| connectionless server | Socket() | bind() | | recvfrom() | |
| Connectionless client | Socket() | bind() | | Sendto() | |

Figure 2-6: Socket System Calls and Association

The socket interface is differentiated by the different services that are provided. Stream, datagram, and raw sockets each define a different service available to applications.

1. Stream socket interface (SOCK_STREAM): It defines a reliable connection-oriented service (over TCP for example). Data is sent without errors or duplication and is received in the same order as it is sent. Flow control is built-in to avoid data overruns. No boundaries are imposed on the exchanged data, which is considered to be a stream of bytes. An example of an application that uses stream sockets is the File Transfer Program (FTP).

2. Datagram socket interface (SOCK_DGRAM): It defines a connectionless service (over UDP for example). Datagrams are sent as independent packets. The service provides no guarantees; data can be lost or duplicated, and datagrams can arrive out of order. No disassembly and reassembly of packets is performed. An example of an application that uses datagram sockets is the Network File System (NFS).

3. Raw socket interface (SOCK_RAW): It allows direct access to lower later protocols such as IP and ICMP. This interface is often used for testing new protocol implementations. An example of an application that uses raw sockets is the Ping command.

## 2.6 UDT Application Socket Interface

UDT uses UDP through the socket interface provided by operating systems. It provides its own UDT socket interface to applications [82].

Applications can call the UDT socket API in the same way they call the system socket API.

Since UDT is a duplex transport protocol, according to Gu [82] each UDT entity has two logical parts: the sender and the receiver. The sender sends (and retransmits) application data according to flow control and rate control. The receiver receives both data packets and control packets, and sends out control packets according to the received packets as well.

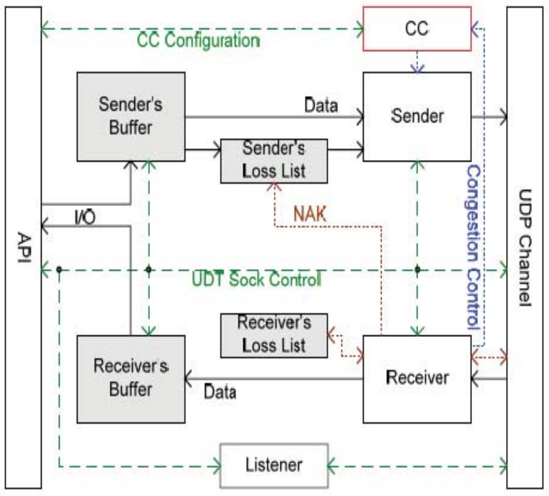


Figure 2-7: The solid line represents the data flow, and the dashed line represents the control flow. The shading blocks (buffers and loss lists) are the four data components, whereas the blank blocks (API, UDP channel, sender, receiver, and listener) are function components [82]. Details presented in this chapter were a review of the works of Gu [82].

### 2.6.1 Implementation

According to Gu [82], the special difficulty in processing Gb/s speed data transfer was noticed a decade ago. Gu [82] contended that although the need for additional processor and hardware overhead no longer required today, the implementation of an application level transport protocol is still sensitive to its performance. Overheads of memory copies and context switches bring more difficulty for application level implementations.

### 2.6.2 Software Architecture

We review the architecture presented by [82]. The Figure 2-2 depicts the UDT software architecture, which highlights the UDT layer that has five function components: the API module, the sender, the receiver, the listener, and the 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 [20,22,81-82]. Because UDT is bi-directional, all UDT entities have the same structure.

The API module is responsible for interacting with applications. The data to be sent is passed to the sender's buffer and sent out by the sender into the UDP channel [82]. At the other side of the connection (not shown in this figure but it has the same architecture), the receiver reads data from the UDP channel into the receiver's buffer, reorders the data, and checks packet losses. Applications can read the received data from the receiver's buffer.

The receiver also processes received control information. It will update the sender's loss list (when NAK is received) and the receiver's loss list (when loss is detected). Certain control events will trigger the receiver to update the congestion control module, which is in charge of the sender's packet sending.

The UDT socket options are passed to the sender/receiver (synchronization mode), the buffer management modules (buffer size), the UDP channel (UDP socket option), the listener (backlog), and CC (the congestion control algorithm, which is only used in Composable UDT). Options can also be read from these modules and provided to applications by the API module [81].

### 2.6.3 User Interface

The API (application programming interface) is an important consideration when implementing a transport protocol. Generally, it is a good practice to comply with the socket semantics. However, due to the special requirements and use scenarios in high performance applications, additional modifications to the original API are necessary according to Gu [81-82] and Bernardo [22-33].

In the past several years, network programmers have welcomed the new *sendfile* method [22-33,81-82]. It is also an important method in data intensive applications, as these are often involved with disk-network IO. In addition to *sendifle*, a new *recvfile* method is also added, to receive data directly onto disk. The *sendfile/recvfile* interfaces and *send/recv* interfaces are orthogonal [82].

UDT also implements overlapped IO at both the sender and the receiver sides. Related functions and parameters are added into the API.

Some lower level APIs should be exposed to applications by an upper level protocol. For example, if the transport layer knows whether a packet loss is due to congestion or link error from the network layer, it will be very helpful for congestion control on links with high bit error rates. UDT exposes many UDP interfaces to give applications the most flexibility for configuring their transport facilities.

An application can make use of the UDT library in a few ways according to Gu [81]. The library provides a set of C++ API that is very similar to the system socket API. Network programmers can learn it easily and use it in a similar way as using TCP sockets.

When used in applications written by languages other than C/C++, an API wrapper can be used. So far, both Java and Python UDT API wrappers have been developed [22,82].

Certain applications have a data transport middleware to make use of multiple transport protocols. In this situation, a new UDT driver can be added to this middleware, and then used by the applications transparently. For example, a

UDT XIO driver has been developed so that the library can be used in Globus [79] applications.

Finally, the library also provides a set of C API that has exactly the same semantics as the system socket API. An existing application can be re-compiled and linked against the UDT/CCC C library. In this way, the applications use our library transparently [82] without any changes to the source codes. There is one limitation, though. UDT does not support multi-process models (e.g., using *fork* system call) due to efficiency considerations, so this method does not work if the existing application uses the same sockets in multiple processes.

### 2.6.4 Protocol Configuration

To accommodate certain control algorithms, some of the protocol behaviour has to be customized. For example, a control algorithm may be sensitive to the way that data packets are acknowledged. UDT/CCC provides necessary protocol configuration APIs for these purposes.

It allows users to define how to acknowledge received packets at the receiver side. The functions of *setACKTimer* and *setACKInterval* determine how often an acknowledgment is sent [82], in elapsed time and the number of arrived packets, respectively.

The method of *sendCustomMsg* sends out a user-defined control packet to the peer side of a UDT connection, where it is processed by callback functions *processCustomMsg*.

Finally, UDT/CCC [82] also allows users to modify the values of RTT and RTO. A new congestion control class can choose to use either the RTT value provided by UDT, or its own calculated value. Similarly, the RTO value can also be redefined.

There are other features of the UDT protocol that are either not related to congestion control or are helpful to most control algorithms. These features, such as selective acknowledgment (SACK) [6,82] and robust reordering (RR) [82], cannot be configured by CCC users, although some of the features can be configured through UDT interfaces.

An application exchanges data through the UDT socket but relies on the UDP socket to send and receive data. This results problems during data transmission from the UDP socket, such as unreliable data, and security flaws [22]. UDT's Sequence Number is 31 and a bit long. This is a small sequence space that does not effectively protect connections against some blind attacks, such as the injection of resets into the connection. It does not have a feature that avoids sequence number attacks, where an attacker can guess the sequence numbers that a future connection would use [20].

The distinction between UDT and other protocols, such as UDP, TCP, STCP and DCCP, is that UDT does not have a reliable checksum algorithm [22-33]. For most protocols, checksum is applied to the protocol header. It applies strong integrity checks, which are available in other protocols (e.g. DCCP). They use the same algorithm to generate the IP checksum to generate this number. The checksum can be included for the segment in UDT, in addition to providing the information contained, and to prevent packets from being incorrectly forwarded by UDP. This provides an added security feature to ensure segment integrity.

As a Fast Data Transfer protocol, UDT additionally needs to provide a mechanism to limit the potential impact of some denial-of-service attacks. It needs to provide limitations on requests, processing options and ICMP messages, and excessive packet generation avoidance on the servers. Because it was designed as an application level protocol that is intended for delivery of data in high speed networks, the need to establish how it handles QoS is essential [2,3]. It is a relatively new protocol, tested in limited production cases in 2004 and focused on performance between long distance links, before UDTv4 was developed and introduced in 2007 [22-33,81-82].

## 2.7 Approaches

There are research works that use discrete mathematics, set theory (computer logic), and so forth to prove their hypotheses. Many of these works have a theoretical component, but their uses differ in the various works. For instance, we employ formal methods in verifying our work, but we do so in two basic but unique ways. First, while most works deal only with mechanisms that already exist, the components that now become the techniques we use in this work will encompass all mechanisms that do exist, will exist, and can ever be thought of and conceptualised. Second, most works are interested in how best to do things; while we are not at all interested in optimality and performance (although in our work we address them through simulations and experimentations), we are concerned with the question of feasibility: what can and what cannot be done in the given topology. We shall look at this from the perspective of which language structures and formal methods the mechanisms and the architecture we describe in this work can and cannot describe and accept, and what possible meaning their output may have.

Thus, in this dissertation, we shall not only perform the simulations and experiments to validate our work; we also verify by describing, specifying, and proving our proposed mechanisms employing Protocol Composite Logic (PCL), and by using formal methods for verification to form the proposed architecture.

PCL is employed to theoretically and compositionally analyse each security mechanism we introduced. This technique was developed at Stanford Security Laboratory and has since gained momentum in the research community in the field of theoretical computer science. New notations have since been produced as of this writing, however, and they continue to be tested in existing security protocols.

The technique we use, therefore, will be based on the existing notations, which were already extensively used to prove mature security protocols.

### 2.7.1 PCL

PCL is a formal language for describing protocols. It uses terms and actions instead of informal arrows-and-message notation. It provides operational semantics that provide description of protocol executions. However the main idea of PCL is to provide protocol logic, which states security properties in terms of secrecy and authentication. Its proof system is comprehensive that specifies axioms and inference rules for formally proving security properties.

We present an example of PCL in Challenge-Response Threads

## Challenge – Response Threads

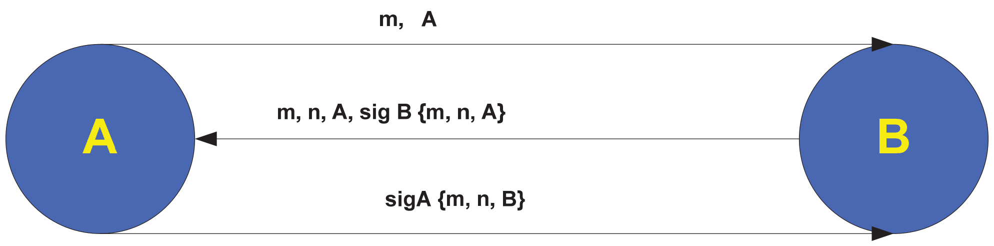


Figure 2-8: Challenge Response. A signature of signed message and signature on the message yield m,n,A, sig B{m,n,A}.

Table 2-2: Challenge Init and Response

| | |
|---|---|
| InitCR(A, X) = [ | RespCR(B) = [ |
| new m; | receive Y, B, {y, Y}; |
| send A, X, {m, A}; | new n; |
| receive X, A, {x, sigX{m, x, A}}; | send B, Y, {n, sigB{y, n, Y}}; |
| send A, X, sigA{m, x, X}; | receive Y, B, sigY{y, n, B}; |
| ] | ] |

**Shared secret in key establishment**

KE |= [ InitKE(A, B) ] A  Honest(B) ⊃

(Has(X, m) ⊃ X=A ∨ X=B )

After **InitKE(A,B)** initiates this mechanism

If **B** is Honest…

Then if some party **X** knows secret **m**, then **X** can only be either **A**, or **B**

**Initiator authentication in Challenge-Response**

CR |= [ InitCR(A, B) ]A  Honest(B) ⊃ ActionsInOrder(

Send(A, {A,B,m}),

Receive(B, {A,B,m}),

Send(B, {B,A,{n, sigB{m, n, A}}}),

Receive(A, {B,A,{n, sigB{m, n, A}}})

)

After initiator executes his program --- **InitCR(A, B)**

If **B** is honest…

…then **msg** sends and receives must have happened in order prescribed by protocol spec

**Correctness of Challenge-Response**

```
InitCR(A, X) = [
        new m;
        send A, X, {m, A};
        receive X, A, {x, sigX{m, x, A}};
        send A, X, sigA{m, x, X};
]
```

```
RespCR(B) = [
        receive Y, B, {y, Y};
        new n;
        send B, Y, {n, sigB{y, n, Y}};
        receive Y, B, sigY{y, n, B}};
]
```

CR |- [ InitCR(A, B) ]A Honest(B) ⊃ ActionsInOrder(

Send(A, {A,B,m}),

Receive(B, {A,B,m}),

Send(B, {B,A,{n, sigB {m, n, A}}}),

Receive(A, {B,A,{n, sigB {m, n, A}}})

)

### 2.7.1.1 PCL Notations

The following notations are employed in this work. Just like mathematical notations, the analysis and verifications require basic understanding of proofs inferences, definitions, axioms, and set theory.

Proof - formally prove properties of security protocols

Axioms - simple formulas that are provable by hand

Inference rules - proof steps

Theorem - a formula obtained from axioms by application of inference rules

*Properties of Proof System*

Soundness

- If $\phi$ is a theorem, then $\phi$ is a valid formula
  - $Q \vdash \phi$ implies $Q \models \phi$
- Informally: if we can prove something in the logic, then it is actually true

Proved formula holds in any step of any run

- There is no bound on the number of sessions
- Unlike finite-state checking, the proved property is true for the entire protocol, not for specific session(s)

*Sample Axioms*

New data

- [ new x ]P  Has(P,x)
- [ new x ]P  Has(Y,x) $\supset$ Y=P

Acquiring new knowledge

- [ receive m ]P  Has(P,m)

Performing actions

- [ send m ]P  ◇Send(P,m)
- [ match x/sigX{m} ] P  ◇Verify(P,m)

*Reasoning About Cryptography*

Pairing

- Has(X, {m,n}) $\supset$ Has(X, m) $\wedge$ Has(X, n)

Symmetric encryption

- Has(X, encK(m)) $\wedge$ Has(X, K-1) $\supset$ Has(X, m)

Public-key encryption

- Honest(X) $\wedge$ ◇Decrypt(Y, encX{m}) $\supset$ X=Y

Signatures

- Honest(X) $\wedge$ ◇Verify(Y, sigX{m}) $\supset$ $\exists$ m' (◇Send(X, m') $\wedge$ Contains(m', sigX{m})

*Honesty rule*

$\forall$roles R of Q. $\forall$ initial segments A $\subseteq$ R.

Q |- [ A ]X $\phi$

Q |- Honest(X) $\supset$ $\phi$

*Terms (letter assignment is assignable)*

| | | |
|---|---|---|
| t ::= | c \| | constant |
| | x \| | variable |
| | N \| | name |
| | K \| | key |
| | t, t \| | tuple |
| | sigK{t} \| | signature |
| | encK{t} | encryption |

These notations are rigorously used to achieve verification and compositionality of security mechanisms.

### 2.7.2 Rewrite Systems and Automata

Another approach we use is rewrite systems and automata. We specify and analyse our proposed security mechanisms in the UDT security architecture and show that the specifications allow us to validate their viability through analysing the data flow. Furthermore, we conduct structural, semantic and query analyses and describe the security mechanisms' data flow through formal methods to verify our architecture. The properties of rewrite systems are related to the security data and network flows and therefore, classical theoretical and practical analysis can be conducted the same way they are used to specify these flows across network topologies.

The increasing complexity of developing and validating a security architecture has led to less extensive practical experiments being performed as single-faceted approach. Topologies composed of numerous devices in various networks are used to reflect an extensive representation of a specific environment. However, because of the heterogeneity of devices required across multiple-environments, it is difficult to analyse the security mechanisms' functionalities. The formal specifications of the security mechanism's data flow, therefore, are crucial. We use formal methods to specify their data flow within the architecture, thus, allow us to carry out analyses across the architecture reflecting with lesser constraints.

In rewrite system, we define constant signature with arity, and positions of mechanisms and data flow in terms and variables. We introduce set of rewrite rules, with syntactic classes. We look at rewrite closure and automata with the defined constraints, examples:

Given the substitutions $\rightarrow f(x, g(c, y))$

These are represented in rewrite rules

$l \rightarrow r$

$f(x, g(c, y)) \rightarrow g(x, y)$

$f(x, x) \rightarrow c$

Rewriting Closure, note the idea: $*\!\rightarrow = 1^{o} \rightarrow 2$, where $\rightarrow 1$ and $\rightarrow 2$ might not be defined as rewriting relations but are in some way easier to analyse, which is specifically used with:

$\rightarrow 1$ - constrained decreasing rewriting ($\leq$)

$\rightarrow 2$ - increasing ground rewriting ($>$)

$l \rightarrow r$ if [condition]

Conditions generally on reachability and joinability on variables from l:

$x \mid y$

$c \xrightarrow{*} z$

$f(x, g(y, z)) \rightarrow c$ if $[x \mid y, d \xrightarrow{*} z]$

Moreover, we use automata to achieve decidability in rewrite systems. We note that the emptiness of a language accepted by a reduction automaton is decidable. This class of reduction automata is closed under union and intersection, where there is a construction for the union that preserves determinism.

## 2.8 Concluding Remarks

We briefly discussed UDT and highlighted the absence of a well-thought-out security. We reviewed existing literature based on the work of Gu [81-82]. We also presented a brief example of how a socket creates a connection, which proved to be useful in creating mechanisms as an add-on library to secure UDT. We also provided a brief description of the approaches we use to achieve formal verification of our mechanisms and architecture in this dissertation.

These approaches will be briefly outlined in Chapters 5 and 6.

In the following chapters, we look into the schemes available and develop a novel security architecture specifically for UDT.

# Chapter 3

# Security Mechanisms

Network protocols do not rely solely on the lower layers of the OSI stack for security; they also rely on other layers. Like other new high-speed protocols, UDT relies on the Application, Transport, IP, and Network layers for data delivery and protection. Like other existing protocols, UDT also has a socket interface for linking with API, a feature that makes it flexible in implementation. In this chapter, we view at how a UDT user can achieve security by using another application service interface. By implementing adequate security mechanisms, UDT can achieve authentication, maintain confidentiality and integrity during data transmission. The rationale is to provide a new way of securing high-speed network protocols such as UDT when implemented in various network environments.

## 3.1 UDT-Authentication Option Field

In this section, we create and introduce UDT-Authentication Option (AO) [19, 32, 36] as another way of securing UDT. We call this AO to differentiate this mechanism in the introduction of a UDT extension to achieve security. We evaluate UDT-AO [19,32] through the use of existing message authenticity for other protocols such as TCP. We review existing message protection that can act like a signature for UDT segments, incorporating information known only to the connection endpoints. Since UDT operates on UDP for high-speed data transfer, we propose the creation of a new option in UDT that can significantly reduce the danger of attacks on applications running UDT. This can maintain message integrity during data transmissions on high-speed networks.

A few security mechanisms proposed are application and IP-based. We present a combination of existing security solutions on various layers [22-33] for UDT.

### 3.1.1 UDT Option for Authentication

UDT is a connection-oriented protocol, and therefore it needs to include an option for authentication. In TCP, this is part of the options (0-44 bytes) that occupy space at the end of the TCP header.

Similarly, to use the option in TCP (RFC 2385) [88], it needs to be enabled in the socket. A few systems support this option, which is identified as the TCP_MD5SIG option.

```
int opt = 1;  Enabling this option

setsockopt(sockfd, IPPROTO_UDT, UDT_MD5SIG, &opt, sizeof(opt));
```

The option can be included in the checksum. However, there is no negotiation for the use of this option in a connection; rather, it is purely a matter of site requirement as to whether or not its connections use the option.

### 3.1.2 Syntax for UDT Option

We propose an option that can be applied to Type 2 of the UDT header. This field is reserved for defining specific control packets in the Composable UDT framework.

Every segment sent on a UDT connection (if it is to be protected against spoofing) will contain the 16-byte MD5 [88, 107, 146] digest produced by applying the MD5 algorithm to these items in the following order, similar to that 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
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. A particular UDT implementation will determine what the application may specify as the key.

In order to calculate checksum, a "pseudo header" is added to the UDP message header. This includes:

```
IP Source Address        4 bytes

IP Destination Address   4 bytes

Protocol                  2 bytes

UDP Length                2 bytes
```

The checksum is calculated over all the octets of the pseudo header, UDP header, and data.

If the data contains an odd number of octets a pad, zero octet is added to the end of data. The pseudo header and the pad are not transmitted with the packet.

In the example code,

```
u16 buff[] is an array containing all the octets in the UDP header and data.

u16 len_udp is the length (number of octets) of the UDP header and data.

BOOL padding is 1 if data has an even number of octets and 0 for an odd number.
```

u16 src_addr [4] and u16 dest_addr [4] are the IP source and destination address octets

```
/*

************************************************************************

Function: udp_sum_calc() modified

Description: Calculate UDP checksum

*************************************************************************

*/

typedef unsigned short u16;

typedef unsigned long u32;

u16 udp_sum_calc(u16 len_udp, u16 src_addr [],u16 dest_addr [], BOOL padding, u16 buff[])

{

u16 prot_udp=17;

u16 padd=0;

u16 word16;

u32 sum;

// Find out if the length of data is even or odd number. If odd,

// add a padding byte = 0 at the end of packet

if (padding&1==1){

padd=1;

buff[len_udp]=0;

}

//initialize sum to zero

sum=0;

// make 16 bit words out of every two adjacent 8 bit words and
// calculate the sum of all 16 vit words

for (i=0;i<len_udp+padd;i=i+2){

word16 =((buff[i]<<8)&0xFF00)+(buff[i+1]&0xFF);

sum = sum + (unsigned long)word16;

}

// add the UDP pseudo header which contains the IP source and destinationn addresses

for (i=0;i<4;i=i+2){

word16 =((src_addr[i]<<8)&0xFF00)+(src_addr[i+1]&0xFF);

sum=sum+word16;

}

for (i=0;i<4;i=i+2){

word16 =((dest_addr[i]<<8)&0xFF00)+(dest_addr[i+1]&0xFF);

sum=sum+word16;

}

// the protocol number and the length of the UDP packet

sum = sum + prot_udp + len_udp;

// keep only the last 16 bits of the 32 bit calculated sum and add the carries

while (sum>>16)

sum = (sum & 0xFFFF)+(sum >> 16);

// Take the one's complement of sum

sum = ~sum;

return ((u16) sum);

}
```

Upon receipt of the signed segment, the receiver must validate it by calculating its own digest from the same data (using its own key) and comparing the two digests. A failed comparison must result in the segment being dropped, and must not produce any response back to the sender. Logging the failure is recommended.

Unlike other TCP extensions (e.g., the Window Scale option [RFC1323]), the absence of the option in the SYN-ACK segment must not cause the sender to disable its sending of signatures. This negotiation is typically done to prevent some TCP implementations from misbehaving upon receiving options in non-SYN segments. In UDT, it is ACK2 (ACK of ACK).

This is not a problem for this option, since, similarly, the SYN-ACK sent during connection negotiation will not be signed and will thus be ignored. The same applies to ACK2 for UDT: the connection will never be made, and non-SYN segments (which do not exist in UDP) with options will never be sent. More importantly, the sending of signatures must be under the complete control of the application, not at the mercy of the remote host failing to recognise and understand the option.

The proposed option has the following format:

```
+---------+---------+-------------------+

| Kind=19 |Length=18| MD5 digest... |

+---------+---------+-------------------+
```

The MD5 digest is always 16 bytes in length, and the option will appear in every segment of a connection.

### 3.1.3 Implications

#### 3.1.3.1 Header Size

As with other options that are added to every segment, the size of the MD5 option in TCP must be factored into the MSS offered to the other side during connection negotiation. Specifically, the size of the header to subtract from the MTU (whether it is the MTU of the outgoing interface or IP’s minimal MTU of 576 bytes) is at least 18 bytes larger in TCP.

On the other hand, the UDP header specifies where segment data starts with a 4-bit field, which gives the total size of the header (including options) in a 32-byte word. This means that the total size of the header plus option must be less than or equal to 60 bytes — this leaves 40 bytes for options.

As a concrete example, existing BSD defaults to sending window-scaling and timestamp information for the connections it initiates. The most loaded segment will be the initial SYN packet that starts the connection. With MD5 signatures, the SYN packet will contain the following:

```
-- 1 byte packet type of control packet of UDT

-- 8 bytes for sequence number from the data packet

-- 8 bytes for timestamp  (data packet)

-- 16 bytes for MD5 digest
```

This adds up to 33 bytes

#### 3.1.3.2 Hashing Algorithm

MD5 [88, 107, 146] algorithm has been found to be vulnerable to collision search attacks, and it is considered to be insufficiently strong for this type of application. However, we specify the MD5 algorithm for this option as a basis of our argument to include AO in UDT. Systems that use UDT have been deployed operationally, and no "algorithm type" field has been defined to allow an upgrade using the same option number. Therefore, this does not prevent the deployment of another similar option that 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 best to define a new option that contains an algorithm type field. To address this, we recommend using a more secure message algorithm such as SHA-1 or SHA-256.

#### 3.1.3.3 Key configuration

It should be noted that the key configuration mechanism of routers may restrict the possible keys used between peers. It is strongly recommended that an implementation be able to support, at minimum, a key composed of a string of printable ASCII of 80 bytes or less, which is also the current practice in TCP.

## 3.2 Generic Security Service - Application Program Interface (GSS-API)

The GSS-API [23, 99] is a generic API for performing UDT client-server authentication. The motivation behind it is that every security system has its own API [99], and there are difficulties involved in adding different security systems to applications due to the variance between security APIs. However, with a common API, application vendors can write to the generic API, which works with any number of security systems [23, 99, 109-110,148], and use GSS-API during the UDT implementation. It is also considered the easiest to use and implement with other schemes, such as Kerberos [18, 39-40, 45, 110, 121].

The GSS-API 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. Notably, the GSS-API [23, 99, 109-110, 148] is used in four stages:

> Firstly, the application acquires a set of credentials with which it may prove its identity to other processes. These credentials confirm the application's global identity, which may or may not be related to any local username under which it may be running.
>
> Secondly, a pair of communicating applications establishes a joint security context using these credentials. The security context is a pair of GSS-API data structures containing shared state information, which is required in order for the per-message security services to be provided. Examples of state information that may be shared between applications as part of a security context are cryptographic keys and message sequence numbers. As part of the establishment of a security context, the initiator is authenticated to the responder, and may require that the responder is authenticated in return. As an option, the initiator may give the responder the right to initiate further

security contexts, acting as an agent or delegate of the initiator. This transfer of rights is termed delegation, and it is achieved by creating a set of credentials similar to that used by the initiating application, but which may also be used by the responder [99, 109].

To establish and maintain the shared information that makes up the security context, certain GSS-API calls will return a token data structure, which is an opaque data type that may contain cryptographically protected data. The caller of a GSS-API [110,148] routine is responsible for transferring the token to the peer application, encapsulated, if necessary, in an application-application protocol. Upon receipt of such a token, the peer application should pass it to a corresponding GSS-API routine, which will then decode the token and extract the information, updating the security context state information accordingly.

Thirdly, per-message services are invoked to apply either integrity and data origin authentication or confidentiality, integrity and data origin authentication to application data, which are treated by GSS-API as arbitrary octet strings. An application transmitting a message that it wishes to protect will call the appropriate GSS-API routine (gss_get_mic or gss_wrap) to apply protection [23, 99, 109] – specifying the appropriate security context – and send the resulting token to the receiving application. The receiver will pass the received token (and, in the case of data protected by gss_get_mic, the accompanying message-data) to the corresponding decoding routine (gss_verify_mic or gss_unwrap) to remove the protection and validate the data.

Lastly, at the completion of a communications session (which may extend across several transport connections) [109-110], each application calls a GSS-API routine to delete the security context. Multiple contexts may also be used either successively or simultaneously within a single communications association, at the option of the applications [23,99,109-110].

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

The GSS-API is a rather large API. For applications using UDT, one need only use a small subset of that API.

## 3.3 Identity Packet within UDT

In today's Internet, the first packet (eg. the TCP SYN) carries no higher-level data that provides adequate sender's information. On the initial transmission, it only carries the source IP address (network layer) and an initial sequence number (transport layer). The high-level data can only be exchanged or transmitted after the complete ACK has been instantiated. Consequently, the receivers will not be able to establish who is sending the data without using additional overhead.

In TCP, the first packet of interaction should carry identity information. Therefore, we propose the use of an Identity Packet within UDT. UDT, like TCP, contains no data which can be used to identify a user (except such information as contained within the [unencrypted] data part of the packet). While the source and destination ports (TCP/UDP), in cooperation with the IP address of the sender and receiver, can identify both participating parties in the lower level, UDT carries no higher-level data that can identify the source before an application processes the packets received.

Network protocols like UDT, meanwhile, have a Sequence Number Field that provides identification; the initial value, however, is determined by the implementer, who decides how the initial sequence number is chosen, e.g., randomly. The same is true, for instance, for the Window field used for congestion control. Since congestion control has a key influence on the overall performance of the protocol, operating system manufacturers have made many attempts to optimise it.

The lack of identity at the lower-level (network layer) in existing network protocols has made achieving network security difficult. Several protocols developed and implemented on top of transport and IP layers have usually created overhead and network incompatibilities. It is a challenge to develop a new technology that includes identity directly into the packets while retaining backward compatibility. Some have placed encrypted and digitally signed identity information into the packets by developing applications that become part of the stack, adding digitally signed identity information to the packets and decoding the information of any incoming connection attempts.

In order for UDT to be used in tomorrow's Internet, it has to be able to accommodate higher-level data association while also maintaining its dependency on low-level protocols such as TCP and UDP. The initial packet of any association, which is called the rendezvous packet, carries high-level information to initiate the association. This provides the receiving entity with the information that enables it to decide whether or not to process the first packet of an association. This information can be delivered in a reliable manner - that is, by cryptographically protecting it prior to and during the transmission.

A mechanism for "First Packet Identity" (see Figure. 3-1) within UDT should be devised, and it should be robust enough that a receiver cannot be flooded by requests to take action before they have verified the identity and trust at the application level. This information can be created using user-defined types field + information. It is possible to delegate this first-packet-identity decision-making to a guard machine that can take on the burden as well as the risk of overload.

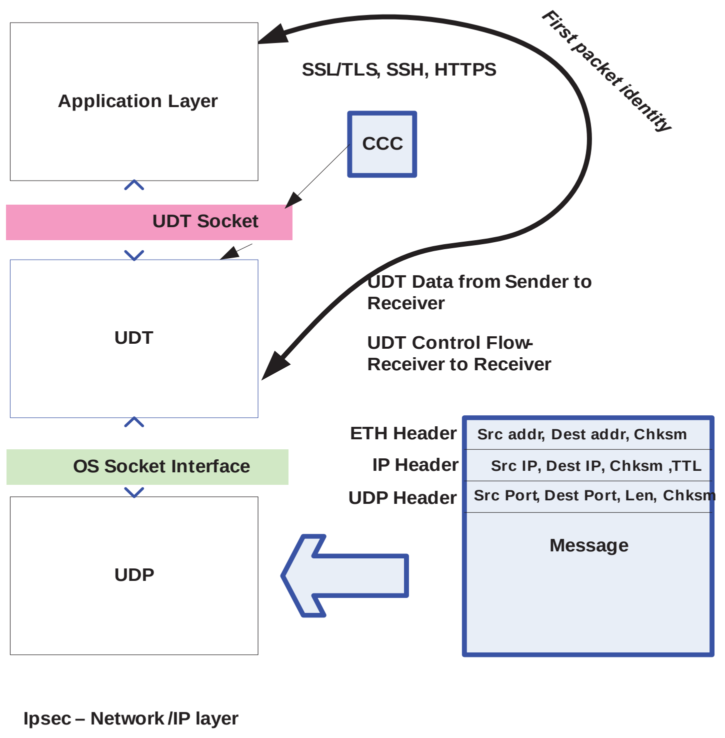

First packet identity
SSL/TLS, SSH, HTTPS
Application Layer
CCC
UDT Socket
UDT
UDT Data from Sender to Receiver
UDT Control Flow-Receiver to Receiver
OS Socket Interface
ETH Header
IP Header
UDP Header
Src addr, Dest addr, Chksm
Src IP, Dest IP, Chksm ,TTL
Src Port, Dest Port, Len, Chksm
Message
UDP
Ipsec – Network /IP layer

Figure 3-1. First Packet Identity in UDT in Layer Architecture. The application exchanges its data through the UDT socket, which then uses the UDP socket to send or receive data through an encrypted mechanism [22-33].

UDT Packet

| 0 1 | Packet type | Information 15 | User defined types | |
|---|---|---|---|---|
| 1 | Type | *IDENTITY packet* | Extended Type | 31 bit |
| X | | ACK Sub – Sequence Number | | |
| | | Time Stamp | | |
| | | Control Information | | |

Figure 3-2. UDT Packet composes of Identity Packet.

UDT needs to build on the identity representation used at the application level, because even though the data may not be visible to the routes (e.g., it may be encrypted), it may still reveal too many attributes of the user. Moreover, it may not be associated with each transmission unit, though applications may be willing to install more state than routers are. It needs to be robust to deal with resource usage and flooding attacks.

Implementing Host Identity Protocol (HIP) [7, 12, 73, 96, 105-106, 118, 137] 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 achieve this, a new network layer called the Host Identity is required.

Furthermore, it is important to highlight the problem of securing IP addresses, which plays an important role in networking, especially in the transport layer. Generating a secure IP address can be achieved through HIP, which is considered a building block for IP security used in other protocols. It is considered another way of securing the address generation in practice.

More works [5, 34-35, 44, 47, 55, 74] have been published in connection with this issue; these include various research projects on HIP since it was first introduced in RFC 4423 [118]. 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 in order to minimise stressing IP addresses, which typically identify both hosts and topological locations. HIP introduces a new namespace, cryptographic in nature, for host identities. The IP addresses, meanwhile, continue to be used for packet routing.

The use of 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 Figure 3-2.

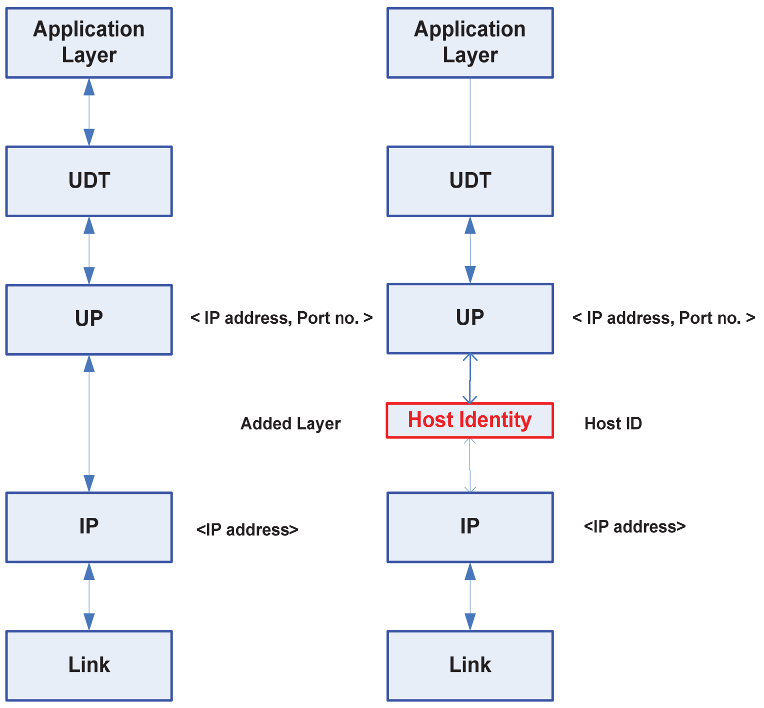


Figure 3-3: Host Identity Protocol Architecture [22-33,105-106,118].

In HIP, the public-key of an asymmetric key pair is used as the HI and the host itself is defined as the entity that holds the private-key of the key pair. Application and other higher layer protocols are bound to HI – and not to an IP address. The prerequisite for HIP implementation should support RSA and DSA for the public-key cryptography.

## 3.4 Other Mechanisms

In this section, we survey and present other viable mechanisms for securing UDT. In previous studies [22-33], we presented an overview of ways to secure 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 [22-33] security options for UDT. We advocate the use of GSS-API in UDT in the development of an application using TCP/UDP. The use of HIP, a state-of-the-art protocol, combined with CGA, is explored to solve the problems of address-related attacks.

### 3.4.1 Diminishing MSS

Here we consider the phenomenon of UDT diminishing its sending rates in the presence of retransmission time-outs and the arrival of duplicate acknowledgments. We note that an attacker can impair its connection by either causing data packets or their acknowledgments to be lost, or by forging excessive duplicate acknowledgments. Causing three congestion control events back-to-back will often cut the *ss* threshold to its minimum value of 3*MSS, causing the connection to enter the slower-performing congestion avoidance mode.

Here is the pseudo-code of the fast retransmit and fast recovery algorithm, with UDT's CTCP redefined two handlers: *onACK and onTimeout*.

```
Virtual void onACK (cons tint&ack)
{
  if(three duplicate ACK detected)
 {
  //ssthresh=max{flight_size /2,3}
  // cwnd=ssthresh + 3* MSS
  }
  else if (further duplicate ACK detected)
 {
  //cwnd=cwnd + MSS
  }
  else if (end fast recovery)
  {
   // cwnd=ssthresh
  }
  else
 {
 //cwnd=cwnd+1/cwnd
 }
}
```

It is important to ensure that the sending rates do not cause a slower performing congestion avoidance phase.

### 3.4.2 Cryptographically Generated Addresses (CGA)

Solving the problems of address-related attacks can also be achieved by using CGA for address generation and verification. Self-certifying is widely used and standardized, such as by HIP [7, 12, 83, 96, 105-106, 118,137] and Accountable Internet Protocol (AIP) [8].

CGA 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 thesis, the conventions used are either Internet Protocol Version 4 or 6 (IPv4) or (IPv6).

The simplified setting for CGA [11] is presented in Figure 3-3. The interface identifier is generated by taking the cryptographic hash 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 truncating 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, e.g., 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 for generating 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 thesis, we consider evaluating the generic attack models that can be adapted to both UDT and CGA.

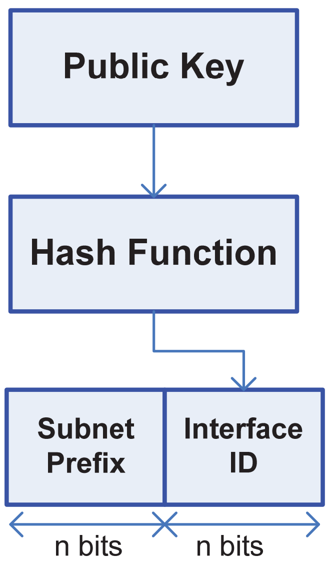


Figure 3-4: Simplified and modified principle of Cryptographically Generated Addresses.

### 3.4.3 HIP-CGA and UDT

HIP introduces a new namespace, which is cryptographic in nature for host identifiers. Furthermore, 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 [7, 12, 83, 96, 105-106]:

- It can be used in address-sized fields in APIs and protocols;

- It is self-certifying (i.e., it is computationally hard to find a Host Identity key that matches a given HIT);

- The probability of HIT collision between two hosts is very low.

As stated above, the HITs are self-certifying. This means that no certificates are needed in practice.

In order to establish an IP-layer communications context, an association needs to be created. This is called HIP association, which is being utilised for base-exchange protocol [7, 12, 83, 96, 105-106, 118, 137]. The details are briefly summarised below:

- Initiator sends to the responder a trigger packet (I1) 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 serves to protect the responder from a number of DoS threats; see RFC 5201 [119]. 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 information needed by 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. This is important since 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 the HI, a HI Tag can be used in address-sized fields in APIs and protocols, including UDT. The hash is truncated to values which are larger in the case of IPv6 implementation, and thus more secure compared to all security levels of CGA.

Since HIP uses base exchange protocol [105-106] to establish a pair of IPsec security associations between two hosts for further communication, the main challenge of its implementation is the requirement of a new network layer, called the HI. This is difficult to run with existing networking protocols in use.

### 3.4.4 Data Transport Layer Security (DTLS)

Another proposed mechanism is DTLS [59-60, 128]. DTLS 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); however, unlike TLS, it is designed for datagram transport. TLS [60] protocol provides equivalent security guarantees. On the other hand, datagram semantics of the underlying transport are preserved by the DTLS protocol.

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 behaviour 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 that use 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 via the transmission of 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's machine, thus flooding it. In implementing DTLS, during the implementation of applications using UDT and UDP, designers need to include cookie exchange with every handshake.

### 3.4.5 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 [21] offers the promise of protecting against many denials of service attacks. It also offers other potential benefits. Conventional software-based IPsec implementations, for example, isolate applications from the cryptographic keys, thus 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. Custom hardware units, moreover, may well lead to higher performance.

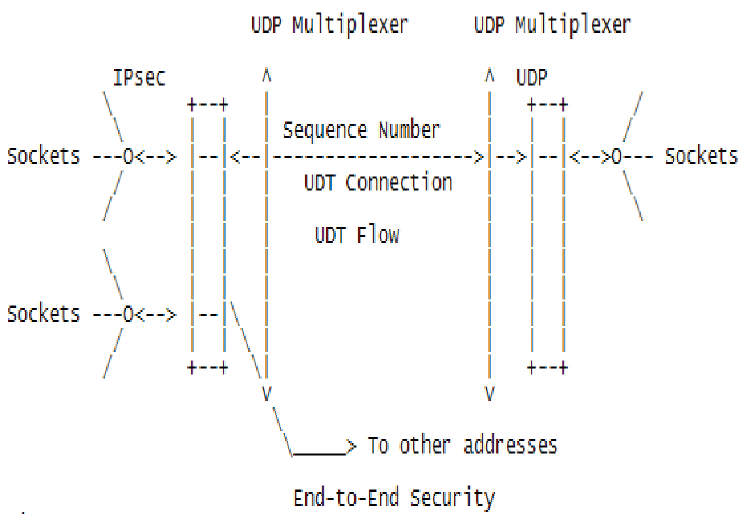


Figure 3-5: UDT flow using end-to-end security [21-33]. IPsec can be used without modifying UDT and the applications running it.

Implementing UDT running at or above the application layers with IPsec provides adequate protection for data transmission (Figure. 3-4). 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.

IPsec can be administered separately and its management can be left to administrators to maintain. It is possible to create an arrangement for securing UDT connections, such as authentication handled by IPsec. Since IPsec relies on UDP, developers can use UDP encapsulation (see Figure. 3-5) to ensure that 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.

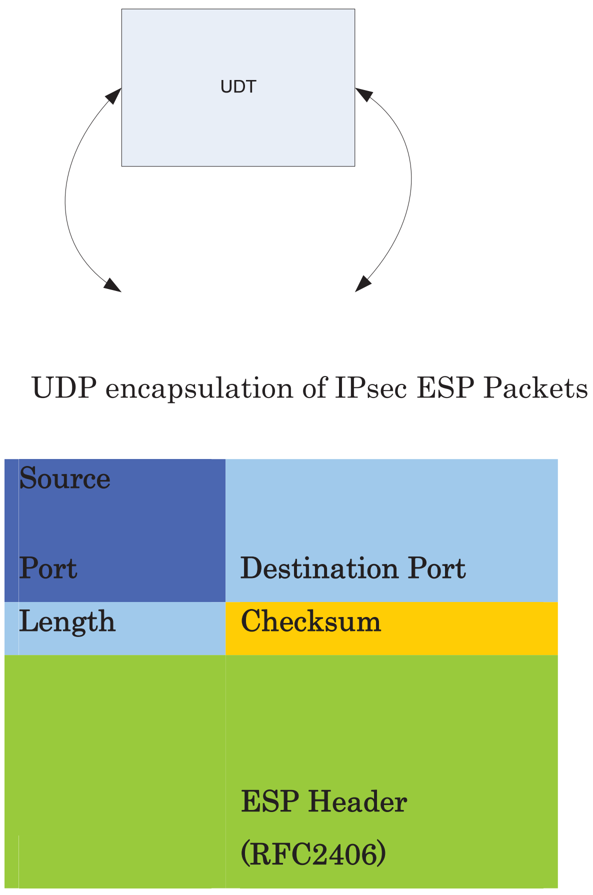


Figure 3-6: Schematic diagram of securing UDT on top of UDP [22-33]

## 3.5 Concluding Remarks

### 3.5.1 Summary of GSS-API

The utilisation of GSS-API to secure UDT needs to be thoroughly evaluated by application vendors. However, the use of the GSS-API interface does not, in itself, provide an absolute security service or assurance; rather, such assurance is dependent on the underlying mechanism(s) of UDT.

### 3.5.2 Summary of UDT-AO

A security option for UDT has been proposed in an attempt to improve the current situation, wherein UDT lacks any form of security. While the MD5 option will soon be replaced by SHA-1 or above, the MD5 option for security remains a significant requirement for the internetworking. Many of the existing network and security systems still use MD5, and this is why the approaches to address its vulnerabilities are important in the implementation phase.

We use UDT-AO instead of IPsec because of rudimentary reason: UDT-AO can support routing protocols, in some cases, connections where keys need to be assigned with individual transport sessions that handle large data transmissions. Moreover, it includes a socket pair which can be used as a security parameter index, rather than using a separate field as an index (IPsec's Security Paramater Index (SPI)).

AO is intended to protect the UDT protocol itself from attacks that other data stream protection mechanisms cannot. However, when there is a level of security to protect the UDT congestion control attack due to UDT's sequence number, IPsec is recommended.

In the preceding chapters, we explained that UDT provides a Type 2 field, which is reserved for user-defined control packets in Composable UDT. The detailed control information carried by these packets varies and depends on the packet types; however, since the UDT/CC library is designed to focus on congestion control algorithms, this field has limited customisation ability. Therefore, there is a need to introduce and expand this customisation ability to include security algorithms.

### 3.5.3 Summary of the UDT-Identification Packet

'First packet identity' needs to be instituted, and devised in such a way that it is robust enough that a receiver cannot be flooded with requests requiring them to take excessive action(s) before verifying the identity and trust at the application level.

The preceding discussions in this thesis have focused on the conceptual low-level protection of the end node. Fundamentally, UDT relies on TCP and UDP for data delivery, which can include data identity in terms of its packet header before the transmission is validated at the application level.

### 3.5.4 Summary of the other mechanisms

Securing UDT can be achieved by introducing approaches related to self-certifying address generation and verification. A technique that 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, and it does so by removing the functionality of IP addresses as both host identifiers and topological locations. To achieve this, however, a new network layer called Host Identity must be introduced. This makes HIP incompatible with current network protocols.

During the simulation of implementation schemes, such as those that will be presented in Chapter 4, it is noted that IPsec provides basic protection on UDT data transfer, as well as end-to-end protection on source and destination nodes. In this scheme, the performance of UDT remains the same. However, we propose other mechanisms that can provide security on UDT connections (e.g., UDT-AO, GSS-API) where keys need to be assigned with individual transport sessions that handle large data transmissions.

# Chapter 4

# Experimental Validations and Practical Implementation

In this chapter, we work on validating the applicability of our proposed mechanisms for the UDT architecture. We focus on practical mechanisms and their applications to UDT, while also considering their limitations.

In the course of our validations, we develop a program to aid our experiment on UDT. This program is developed to support our data gathering activities, since UDT, a new protocol, has no proprietary software tools readily available. Furthermore, we create tailored environments in which to conduct experiments on UDT in several important steps and scenarios.

We create a secure environment wherein UDT operates outside a secure perimeter. We then operate UDT inside a secure perimeter, this time within various security devices. These devices include firewalls, anti-virus software, and intrusion detection systems. Subsequently, we introduce our proposed mechanisms and eliminate those that are not theoretically viable for UDT. Our basis for elimination is drawn from the UDT design, its compatibility with existing mechanisms, and complexity. Finally, we select the proposed mechanisms we developed and attempt to expand their practical applications. We then theoretically achieve proof of correctness of our created mechanisms for implementations in Chapter 5.

## 4.1 Outcomes

We published part of this work in [22-33]. We highlighted UDT security vulnerabilities and evaluated the protocol in an environment with commercially available equipment and tools to support the experiments.

We introduce a data gathering tool we developed to capture the data transmitted by UDT and to show the results in a visual presentation. We create, for the first time in the literature, a tool (Project UDT) specifically for UDT, which can be used in other experiments on other network protocols. The algorithms designed to capture and interpret data are written to accommodate other protocols for data capturing and presentation. Our intention is to analyse data and protocol performance for the sake of UDT security.

Project UDT can be used to capture data across high-speed networks in long distances, and to interpret such data. It provides straightforward data gathering and simulations. To create this tool, we investigate how UDT captures data and how these data vary as the distance increases. We have written an algorithm to assist us in the analysis and development of an architecture that is useful in various UDT implementations. It is initiated with supplemental information on the schemes, which can provide basic if not comprehensive security of data flow from lower-level to higher-level network communication layers.

### 4.1.1 Overview and Environment

We designed and built a private high-speed WAN extended across metropolitan areas within the city of Sydney to the Western suburbs, NSW, Australia. We provisioned two links (internal and external) across the Virtual Local Area Network (VLAN) (Figure 4-1). This was our physical model. To compare the results, we conducted actual experiments and, at the same time, executed simulations based on our assumptions.

The assumptions resulted to the creation of robust environments supporting high-speed network data transmission. On the right in Figure 4-1 is the primary (located in Sydney) and on the left is the secondary (located in Western Sydney), set 40 km apart from each other.

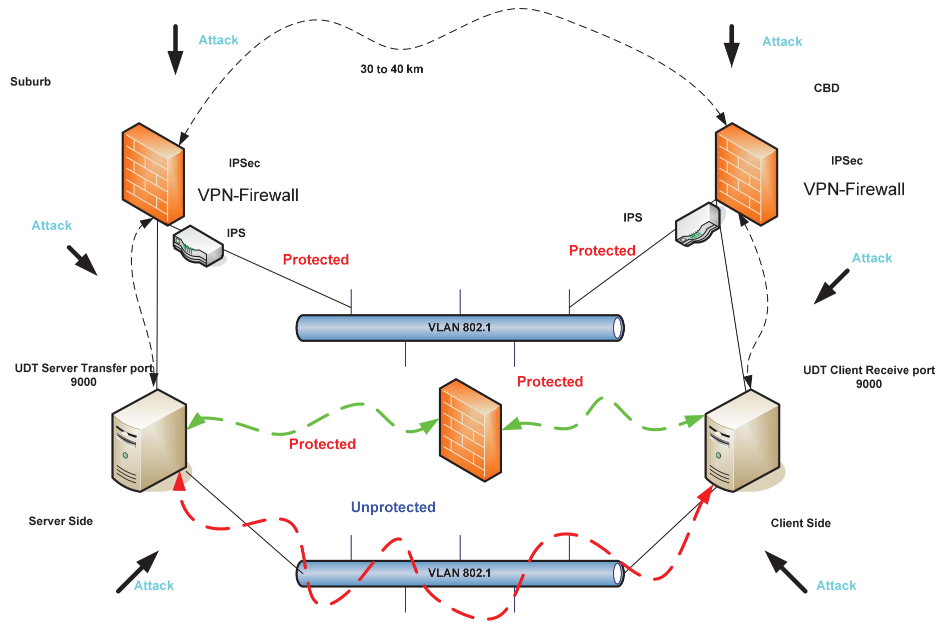


Figure 4-1: Built Environment. It supports high-speed network data transmissions.

Our security model was composed of two high-speed security devices and routers. To analyse the network performance using realistic assumptions, a series of queues were used. In the assumptions, we considered validation of Poisson arrivals in this specific network and security model – this time in both unprotected and protected environments. While the simulation was running, we operated the real transmissions of data through the high-speed data transfer protocol and noted that as the number of connections on the link increased, the superposition pushed the arrivals towards Poisson status [17].

We consider these transmissions, as independent assumptions, taking into consideration new packets (specifically length) of UDT is independently chosen for the packet each time the packet was received at one node in both secured and unsecured connections. It is also based to obtain expected packet waits and expected number of packets.

In the experiment, we sent a large amount of data from one location to another based on two scenarios. In the first scenario, the data (>10 Tb) was sent through the unprotected environment through the internal network. During the transmission, TCP flooding and simulated attacks were performed. In the second scenario, the data was transmitted through the protected environment. We gathered both results and captured them in a UDT txt file for analysis. We implemented the mechanisms (UDT-AO, DTLS, and Kerberos) in limited but practical scenarios, and simulated the data transmissions with basic authentication schemes on our routers for AO as well as on servers (in the case of Kerberos). We tested DTLS by implementing it with UDP datagram, while also performing basic functionality tests during UDT data transmissions. We used Kerberos to simulate GSS-API, using Microsoft's SSP architecture version 5. We used keys for basic authentications from the UDT client workstation to the UDT server. Modifications were made in the UDT codes, although we experienced C++ compilations run-time issues, as expected, which we eventually fixed with add-on C++ libraries.

In the end, we relied more on the initial assumptions made about the above scenarios running multiple mechanisms at the same time, albeit in the given constraints. We focused on packet transmissions and basic modified UDT codes to meet our basic assumptions. The packets traversed to various scenarios in parallel with the simulations (e.g. attacks and mechanisms), providing adequate results to support our hypotheses.

### 4.1.2 Proprietary Tool

In this section, we briefly introduce our program called Project UDT. It is used mainly to capture the packet transmitted from a static file, and to give a visual representation of how the packet transmission behaved. This program can be implemented in both protected and unprotected environments to gauge the security mechanisms introduced for UDT.

### 4.1.3 Methodology

The modelling methodology, which the program formed, is based on the assumptions presented. The program relies on the methodology, and scales the analysis from the data captured in a text file. It then plots a graph based on the values captured to a file. The ratio of the slopes and values scales the graph to achieve a visual representation of the data captured. The program also performs the analysis based on the Pollaczek-Khintchine formula [22-33] to obtain the expected data packet waits and expected number of packets transmitted. This formula is not dependent on the type of network protocol used; neither is it dependent on whether the network protocol used, e.g., UDT, for data transmission is protected or unprotected. We then obtain second moments of these distributions mathematically. To capture the rate of link time, we send the results with the rate and speed of data UDT transmission to a flat file.

Java Code: //values are read from the UDT file and presented in a graph

```
for (int j = 0; j < dataValues.size(); j++) {

        int valueP = j * bar_width + 1;

        /*

        if (j%2 ==0){

           valueP += 50;

        }*/

// capture the data and calculate their performance based on the
time they were transmitted.



// optional -System.out.println("valueP: valueP);

    int valueQ = title_height;

    int height = (int) (dataValues.get(j) * graphScale);

     if (dataValues.get(j) >= 0)

      valueQ += (int) ((maxDataValue - dataValues.get(j))

           * graphScale);

     else {
```

```
                valueQ += (int) (maxDataValue * graphScale);

                 height = -height;

               }

          // calculate the performance based on the algorithm that supports
          the waiting time queue (Laplace transform)
```

The project UDT program analyses the distributions from the flat file and characterises arrival or service processes. The well-known problem in studying distribution systems is that they do not possess a closed form Laplace transform – negating the direct application of results from queuing theory [22-33].

A recursion in the data transmission is thus developed to analyse a three-point discrete distribution of packet sizes, based on the assumptions of the discrete queuing systems assumed for a specific environment. Therefore, the algorithms for the program are created to numerically study link congestion and generate all link waiting time distributions on these links.

Consider the standard queuing system [22-33]: if *Wq (t)* is the probability that a system waiting time in the queue is less than or equal to *t*, then its Laplace transform is given by :

$$W_q^*(s) = \int_0^\infty e^{-st}\, W_q(t)\, dt = \frac{(1-\rho)}{s-\lambda(1-B^*(s))}$$

Typically, one plugs in the Laplace transform B*(s) of the service time then inverts Wq* (s) either analytically or numerically. When the service time distribution is heavy-tailed, B*(s) does not exist in closed form, and transform approximation is used to numerically approximate B*(s) with a discrete approximation.

An initial version of UDT-data capture is developed to analyse data packets, and to present these for data modelling and interpretation (see Figure.4-2).

Input File: UDTFile.txt (tested data transmission)

| SendRate(Mb/s) | RTT(ms) | CWnd | PktSndPeriod(us) | RecvACK | RecvNAK |
|---|---|---|---|---|---|
| 3.00126 | 12.832 | 307 | 3872.83 | 28 | 12 |
| 1.88694 | 0.734 | 194 | 4536.45 | 29 | 2 |
| 2.07238 | 1.268 | 13 | 2191.21 | 29 | 3 |
| 4.07238 | 4.278 | 11 | 2132.13 | 29 | 3 |
| 6.07238 | 1.268 | 14 | 2110.34 | 29 | 3 |
| 7.07238 | 1.268 | 12 | 3121.16 | 29 | 3 |
| 1.07238 | 3.268 | 11 | 2112.28 | 29 | 3 |
| 2.07238 | 1000 | 16 | 1192.10 | 29 | 3 |

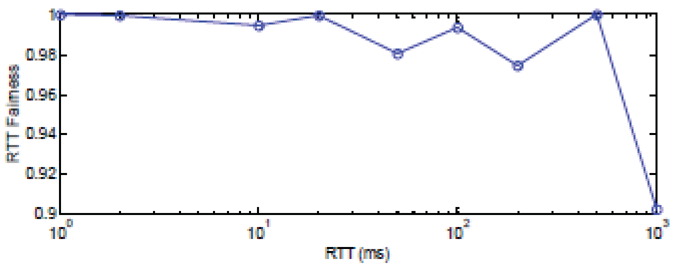


Figure 4-2: The figure presents UDT RTT fairness; an average of throughput (given a data size of 100mb~1TB) . Two concurrent UDT flows are simulated in the above fig. 1 topology, with one link having an RTT of 1ms and the other having an RTT of 1000ms. This result yields the same output [22-33]

The algorithm of the program is explicit and novel to cater to data gathering and interpretation. A file captures the data from either protected or unprotected environment in the simulated environment based on the assumptions presented in this section.

The data that will be captured include two important scenarios: (1) attacks on data transmitted by UDT with its underlying security mechanisms, and (2) attacks on data transmitted by UDT without any security mechanisms.

These scenarios consider physical and digital attacks; for example, in the physical attack to a link or node, the link is disabled. The traffic is re-routed by an alternative link in the given environment. UDT, however, does not estimate any performance gathered between time intervals during data transmission. The tool thus models the data captured and displays it for estimation and interpretation. The physical or cyber attacks will not significantly affect UDT data gathering.

### 4.1.4 Data Collection

The task of collecting and interpreting data is undoubtedly extensive. This is due to the sporadic behaviour of packets in high-speed networks. Here, a tool is developed to specifically capture and interpret the data transmitted through UDT. The objective is to collect and interpret the data in the file, with appended packets transmitted on a high-speed network either in encrypted or unencrypted mode in various proposed schemes (e.g., UDT-AO, UDT+TLS, UDT+GSS-API) that use a newly developed high-speed data transfer protocol UDT. The program developed has become a key tool in data analysis, and replaces the existing manual collection of information of the data transferred for statistical analysis.

### 4.1.5 Description of Tool

The tool is developed in Java. It has a visualisation capability to present transmitted data. Its basic functionalities allow displaying a set of data in a file and representing this in a graph.

The tool is dependent on two sub-components (ProjectUDT.java -390 lines and GenerateBarGraph.java -150 lines).

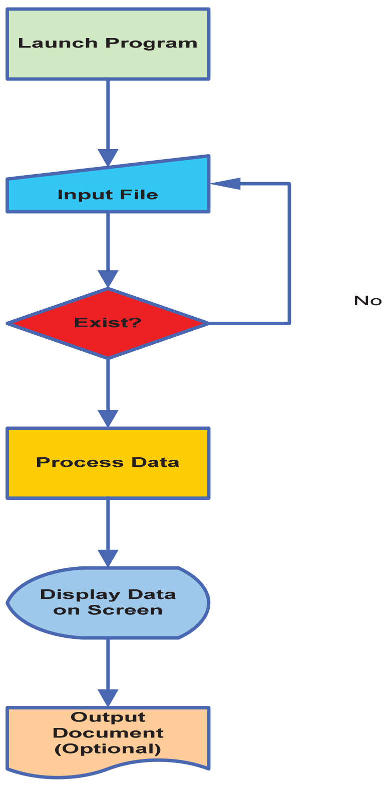
Project UDT is the main program. It starts with a Graphics Unit Interface (GUI) Menu.

- The menu-driven options provide choices such as "Process a File," "Exit," and "Help".
- "Process a File" accepts file in text format only. In this program, the file name of this text file is UDTFile.txt. This is a file with raw entries captured from a live data transmission across multiple sites running on a high-speed optical network/WAN.

- The tool then processes the two important entries of the file and displays these in a GUI format. An option is then provided to interpret the entries in a bar graph.
- The entries are then presented by a graph. This graph is derived from the gif file.

The following flow chart summarizes the above process flow.

Figure 4-3: Program Process



The following is a list of inclusions:

- Menu
- Help Section
- File Processor (Input/upload a file)
- Data graph
- Output a file (a readable)

## 4.1.6 Program and Image Files

- ProjectUDT.java

| | |
|---|---|
| Number of Constructs/Methods | 8 |
| Number of Classes | 1 |
| Number of Variables/Imports | 12 approx and 36 imports |
| Number of Codes | 386 less 15% spaces and comments |

- GenerateBarGraph.java

| | |
|---|---|
| Number of Constructs/Methods | 2 |
| Number of Classes | 1 |
| Number of Variables/Imports | 15 approx and 7 imports |
| Number of Codes | 144 less 15% spaces and comments |

- Image Files

  - image.gif ( project)

  - globe.jpg (supplementary project)

- Input file (see Appendix for full entries)

  - UDTFile.txt

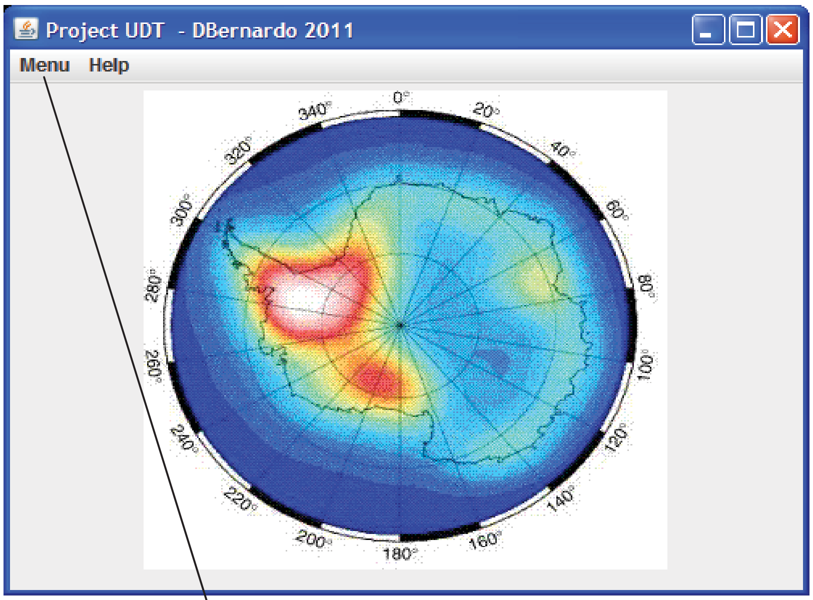
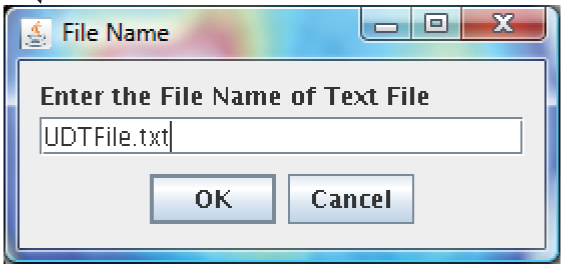


Figure 4-4: Main Menu of Project UDT is composed of the following options: Menu, Help Section, File Processor (Input/upload a file), Data graph, Output a file (ASCII).


File Name

Enter the File Name of Text File

UDTFile.txt

OK Cancel

Figure 4-5: Accept text file

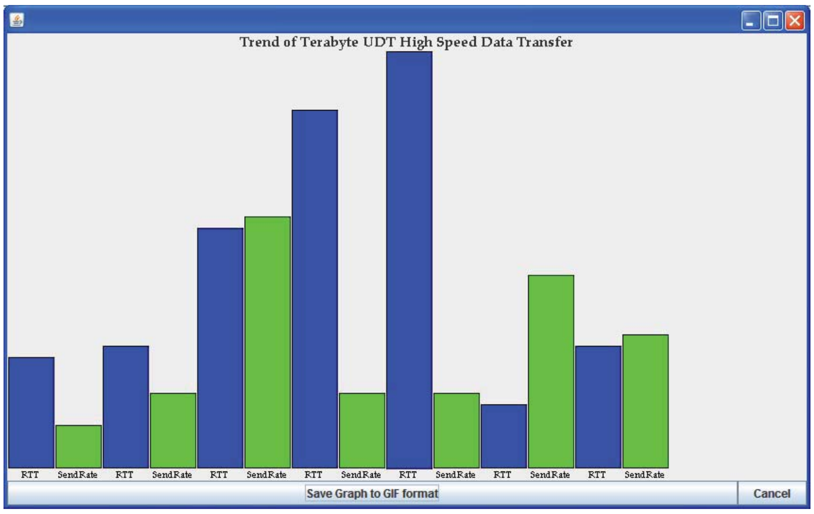


Figure 4-6: Output graph.gif

The graph of the given data visually shows that the RTT is higher than the SendRate in a given distance of 40 km (i.e., from CBD to Western Sydney). This means that the SendRate is directly proportional to the distance and resistance of existing network connections. This trend changes when tested on the different volatile network connections (e.g., wireless, optical, and satellite) based on the following equation [82]:

$$\frac{RTT}{SYN} \cdot 10^{\frac{e-9}{2}} \cdot \sqrt{6} \leq 1$$

The use of data under baseline conditions provides an initial analysis that yields mean and standard deviations of any delays during data transmissions in a protected environment. Protected environment means that data are either transmitted with encrypted mechanisms (AO and GSS-API), or transmitted in protected communication channels (such as IPSec and DTLS).

### 4.1.7 Summary

Project UDT gathers and represents data from the initial test results in different scenarios. These scenarios are practical: first, under a normal environment; and second, under a DoS attack. The components in various scenarios are subjected to an attack across locations in the outskirts of Sydney, about 30-40 km apart. Agents for this attack are simulated packet transmissions using UDT. The path of the attack traffic is from various locations. The attack traffic causes the saturation of unprotected links across locations.

Overall, the simulation comparisons have indicated that the analytic methodology approximates the actual environment and the assumptions defined. Both results provide a clear indication of how the tested security mechanisms behave when used with UDT. In a DoS scenario, the link overloading has highlighted interesting effects on the network through packet transmission and packet size distribution. These scenarios aid in determining what kind of modifications, if any, can be made to better capture the results.

In addition to further research on the specific model for UDT [22-33], the validations of the strength of security mechanisms against other attacks, such as worms, have been confidently dealt with and neutralised in the end-to-end security architecture in a real practical scenario, with UDT integrated and implemented with selected security mechanisms.

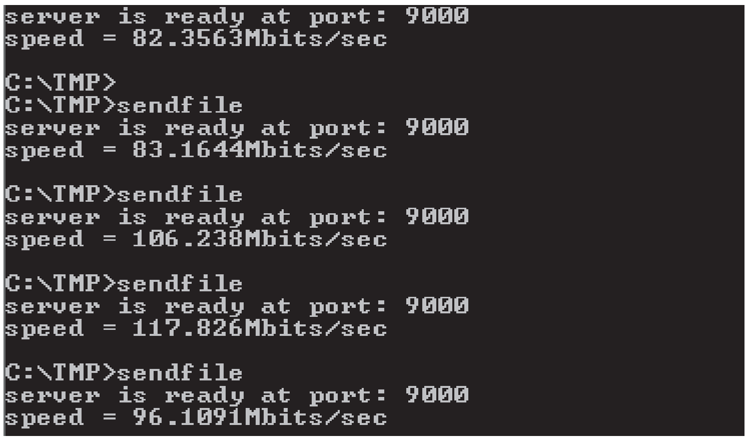
In the next sections, experimentations and practical simulations are presented using the commercially available tools. To support the theoretical proofs, other security mechanisms are also attempted. These are performed in Chapters 5 and 6.

## 4.2 Practical Validations

In this section, we employ existing commercial tools and devices to represent a testbed environment. We use common methods for analysis, from layer 2 to layer 5 of the TCP/IP layers.

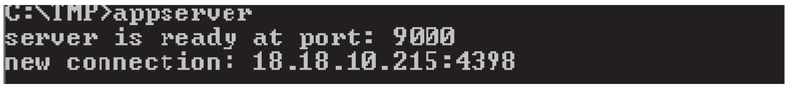
Like many TCP and UDP implementations, there are various mechanisms considered unsuitable for UDT (for example, using TLS on layer 2 for UDT and UDP). In this section, we attempt to establish suitable scenarios that are practical and cost-effective for UDT experimentation and implementation – by exploring security methods operating on layer 3.

For the UDT data file transfer tests, the protocol developments provided by FreeBSD Release and Windows were briefly covered. The *'sendfile.cpp'* and 'recvfile.cpp' (on port 9000) were executed on each client and server nodes (see Figure 4-7 and Figure 4-8). Moreover, short samples of data (from 100meg, 1G and 4G, gradually increasing the size to 13 terabytes) were used. The data transfer was performed from one data centre to another – situated in two geographical locations, roughly 20 to 40 km apart – via 1Gb/s link through the cloud.



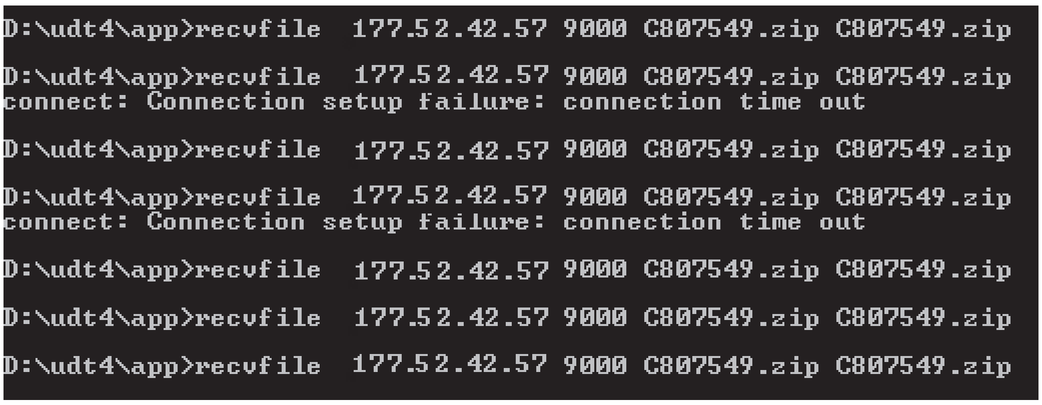
```
server is ready at port: 9000
speed = 82.3563Mbits/sec

C:\TMP>
C:\TMP>sendfile
server is ready at port: 9000
speed = 83.1644Mbits/sec

C:\TMP>sendfile
server is ready at port: 9000
speed = 106.238Mbits/sec

C:\TMP>sendfile
server is ready at port: 9000
speed = 117.826Mbits/sec

C:\TMP>sendfile
server is ready at port: 9000
speed = 96.1091Mbits/sec
```

Appserver

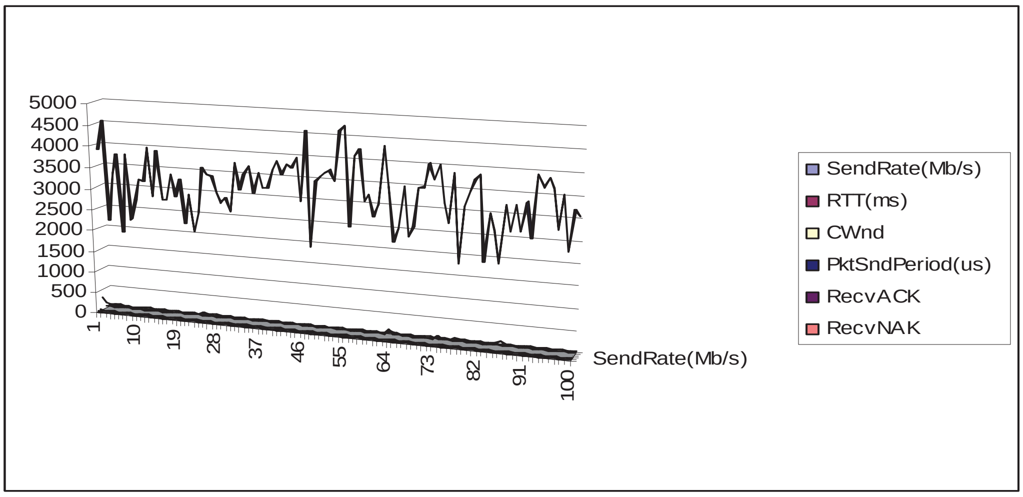


```
C:\TMP>appserver
server is ready at port: 9000
new connection: 18.18.10.215:4398
```

Figure 4-7: Send Packet Connection (Server Side)



```
D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip
connect: Connection setup failure: connection time out

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip
connect: Connection setup failure: connection time out

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip

D:\udt4\app>recvfile  177.52.42.57 9000 C807549.zip C807549.zip
```



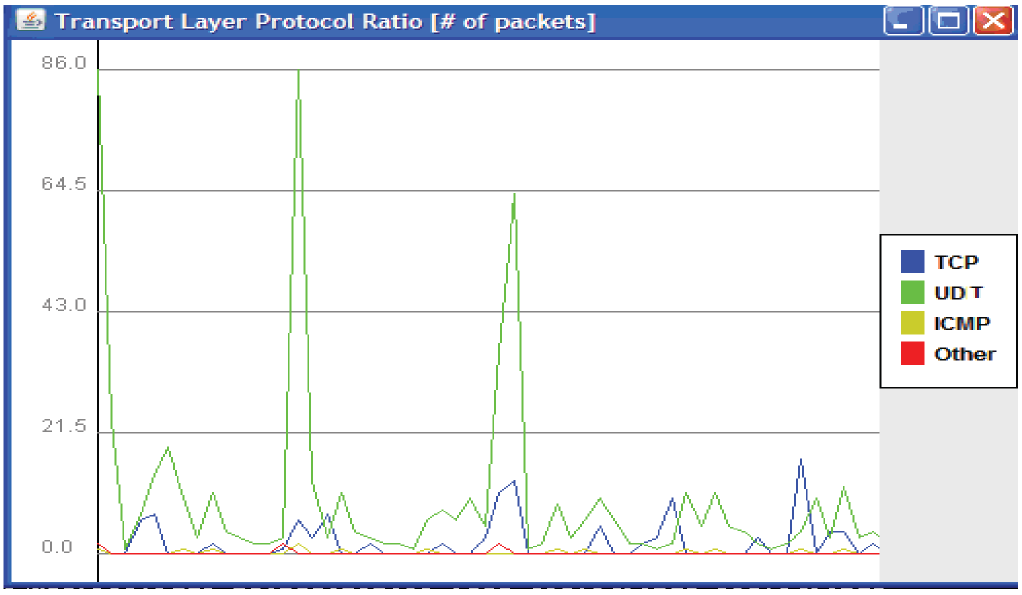


Figure 4-8: Receive Packet Connection (Client Side). Attempts were also made while the port 9000 was not listening. The connection setup failure message should be displayed if this port is not listening on the server. The graph represents when Appclient IP ADDRESS 9000 was operated, with results on SendRate(Mb/s) RTT(ms) CWnd PktSndPeriod(us)RecvACK-RecvNAK. Details found in Appendix B.

The experiments were conducted using a gateway-to-gateway encryption method in two separate data centres. We attempted GSS-API + UDT (Kerberos ticketing systems) along with UDT-AO and UDT+DTLS scenarios. In the tests, UDT performance in gateway-to-gateway mode was evaluated, and UDT was tested using commercial security products with IPSec capability in this mode. In addition, a network comprising two gateway hosts was created, and two firewalls at the gateways were utilised, with each configured for site-to-site VPN. Furthermore, the *clients* used for the test were composed of 100TB storage and *servers* were running on Windows OS, with a sample application to handle data transfer using UDT behind the gateways.

The intention was to transfer data across both secured and unsecured environments, i.e., encrypted and unencrypted links, and to conduct simulated attacks on the application that was running UDT.

### 4.2.1 Measurement Schemes and Results

In order to test the performance, the data file transfer was repeated using several measurement tools (e.g., ArcSight, AlgoSec, PCAP, HP OpenView, Statscout, Juniper VPN-Firewalls, NS-2, and Wireshark) which entailed measuring the size of the files and documenting the time in which they reached their destinations. In order to test security, vulnerability tests, such as conducting simulated DoS attacks (e.g. sending continuous ICMP packets, TCP SYN flooding, transferring virus infected files and so forth) were carried out. Moreover, a proprietary vulnerability assessment testing tool was used (e.g., Nessus) ; in order to simulate a real-life scenario, while also determining how traffic could influence normal network activity, the testing tool included a regularly updated database – of the latest threats– which served to ensure that existing adversaries and the most prominent potential threats were considered.

Also utilised were tools that allowed the generation of line rate stateful traffic at up to 10 Gbps, thereby allowing trunk ports to be directly tested while also determining the impact of multiple GigE ports being aggregated over 10 Gbps. The tools used are considered to be the industry's most comprehensive layer 4-7 applications testing solution, supporting all major protocols – including UDP, TCP, streaming media, IPv6, IPSec, and custom protocols – as well as a number of enterprise applications via the capture/replay function.

In the test network, encryption was performed on the higher specification gateways and firewalls (Figure. 4-1, page 69).

The resulting performance was acceptable with encryption. The significant result was that the encrypted throughput was seen to be acceptable with the rate of the non-encrypted throughput (Figure 4-9 and Figure 4-10). Detailed results can be observed in Table 4-1, which reflects the use of high-speed bandwidth and high-end firewall capabilities on the edge of the networks.

Finally, *'data file transfer'* tests were performed between the two gateways, with the response time being measured in milliseconds.

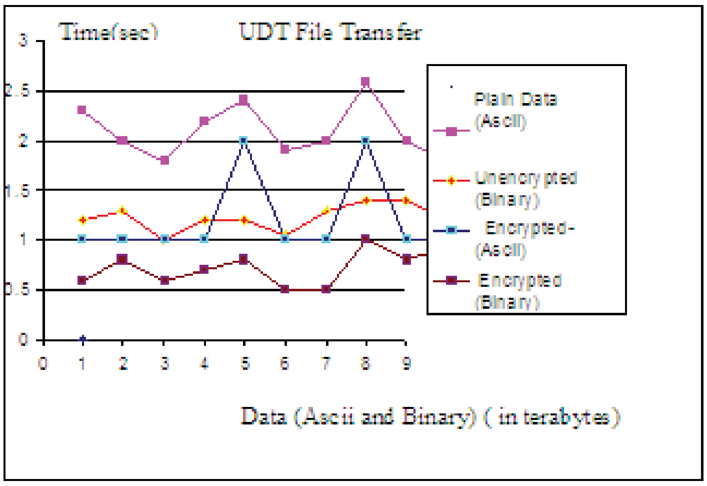


Figure 4-9: Secured Data Transfer Graph results (In the secured environment we tested file encryption to both ASCII and Binary Data streams to determine if there was impact to the speed transfer when using UDT)

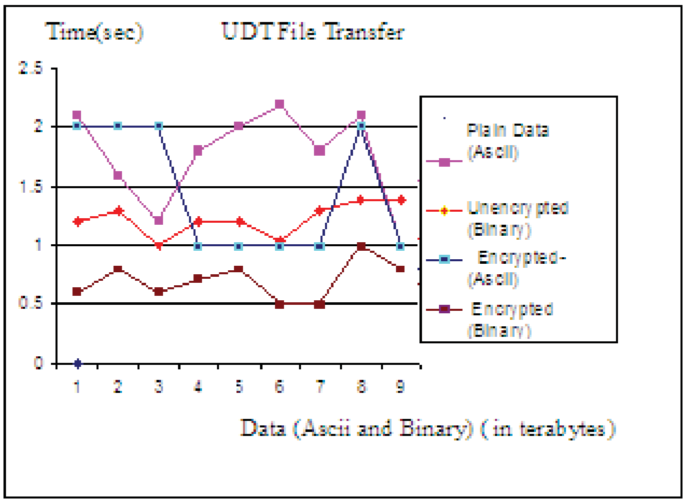


Figure 4-10: Unsecured Data Transfer Graph results (In the unsecured environment we tested file encryption to both ASCII and Binary Data streams to determine if there was impact to the speed transfer when using UDT)

Table 4-1. Secured Transmission. Results with security/ encryption

### Secured transmission

(10 repeats)

| Measurement | Average Plain | Average with Encryption |
|---|---:|---:|
| TCP throughput | 200.9 ms | 180.4 ms |
| UDP throughput | 140.2 ms | 130.3 ms |
| UDT throughput | 9000 ms | 6000 ms |
| TCP response time | 1/1/1 ms min/avg/max | .8/1/1.2 ms min/avg/max |
| UDP response time | 1/1/1 ms min/avg/max | 1/1/1 ms min/avg/max |
| UDT response time | 1/1/1 ms min/avg/max | .9/.95/1 ms min/avg/max |

Table 4-2. Unprotected Transmission. Results without security / encryption

### Unprotected Transmission

(10 repeats)

| Measurement | Average Plain | Average |
|---|---:|---:|
| TCP throughput | 187 ms | 162 ms |
| UDP throughput | 120.1 ms | 113 ms |
| UDT throughput | 7000 ms | 4000 ms |
| TCP response time | 1/1/1 ms min/avg/max | 1.2/2.1/3 ms min/avg/max |
| UDP response time | 1/1/1 ms min/avg/max | 1.2/2.1/3 ms min/avg/max |
| UDT response time | 1/1/1 ms min/avg/max | 1.2/2.1/3 ms min/avg/max |

Table 4-3. UDT test results with (encrypted data) and without (or plain data) encryption.

### Secured UDT Data File Transfer

| UDT | Time (sec) | Plain Mbytes | Time (sec) | Encrypted Mbyte/s | Repeat |
|---|---|---|---|---|---|
| Server-gateway-gateway-server | 2.3 | 1.2 | 1 | 0.6 | 1 |
| | 2 | 1.3 | 1 | 0.8 | 2 |
| | 1.8 | 1 | 1 | 0.6 | 3 |
| | 2.2 | 1.2 | 1 | 0.7 | 4 |
| | 2.4 | 1.2 | 2 | 0.8 | 5 |
| | 1.9 | 1.05 | 1 | 0.5 | 6 |
| | 2 | 1.3 | 1 | 0.5 | 7 |
| | 2.6 | 1.4 | 2 | 1 | 8 |
| | 2 | 1.4 | 1 | 0.8 | 9 |
| | 1.8 | 1.2 | 1 | 0.9 | 10 |

Table 4-4. UDT test results with (file encryption) and without (or plain data) encryption.

### Unsecured UDT Data File Transfer

| UDT | Time (sec) | Plain Mbyte/s | Time (sec) | Encrypted Mbyte/s | Repeat |
|---|---|---|---|---|---|
| Server-gateway-gateway-server | 2.1 | 1.2 | 2 | 0.6 | 1 |
| | 1.6 | 1.3 | 2 | 0.8 | 2 |
| | 1.2 | 1 | 2 | 0.6 | 3 |
| | 1.8 | 1.2 | 1 | 0.7 | 4 |
| | 2 | 1.2 | 1 | 0.8 | 5 |
| | 2.2 | 1.05 | 1 | 0.5 | 6 |
| | 1.8 | 1.3 | 1 | 0.5 | 7 |
| | 2.1 | 1.4 | 2 | 1 | 8 |
| | 1 | 1.4 | 1 | 0.8 | 9 |
| | 1.8 | 1.2 | 1 | 0.9 | 10 |

The raw response times were found to be poorer for the non-encrypted (clear) test, but it was presumed that the intervening gateways without security were responsible for such an effect. The unsecured environment was susceptible to DoS attacks on the gateways and servers, which therefore affected data transfer performance and integrity. Results, however, can be improved significantly, especially for much larger packet sizes, wherein fragmentation occurs in a protected link, such as IPSec.

Overall, the results established throughout this experiment illustrate the improved performance that can be delivered by the use of a higher specification encrypting device with higher bandwidth links. The results of the various throughput tests performed suggest that, at the bandwidth levels requiring fast data transfer, the protected environment would appear to offer a scenario that does not have a significant impact in terms of latency or session quality.

### 4.2.2 Impact on Performance

The trials of both secured and unsecured environments imply that encryption at the network layer does not impose significant performance problems in the middle of attacks. Perceived latency is very similar in these tests, and the empirical results imply that the overhead at the network layer is in the order of a handful of milliseconds.

It should be noted that, in addition to the encryption overhead on the CPU, the encrypted packets will also be larger, owing to the additional AH/ESP data being sent, and that the packet re-assembly at the far end will take longer, due to delays in the passing of encrypted data beyond just the raw computational burden.

In terms of the algorithms, one may assume that weaker algorithms are less computationally expensive; however, existing encryption algorithms such as AES can offer improved encryption in comparison to 3DES (which the commercial IPSec VPN product uses), and for less processor effort, too (an important consideration when encryption is required on miniature smart type devices).

Coupled with advances in processor and bandwidth speed, the latency penalty for encryption will continue to fall as a percentage of the time and bandwidth required for high data transfer. The same, of course, may not be true in the case of UDT implementations over a low bandwidth network like a cellular wireless network (where the packet size overhead is far more significant).

### 4.2.3 Socket and Application Layer UDT Protection

The simulated and implementation schemes based on the previous tests created for the network and IP layers were performed in order to observe the behaviours of UDT in both secured and unsecured settings in the application and socket layers. Notably, the simulated environment operated separately on NS2 and EMIST, so as to provide internal validation. This environment was used in order to simulate the behaviours of data transmission in cases wherein UDT is used on top of UDP. A test was also performed using a new probabilistic packet marking scheme and other commercially available tools (eg. IPS) constituted by 3,000 nodes; 1,000 attackers were selected randomly.

In order 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 was defined as *w*, *w*=1,2…30. For each number of paths, a simultaneous change of values of *w* was repeatedly changed until the protocol showed a clear attacking path; this permitted the simulation to produce a pattern of the behaviours of UDT without any means of protection.

The implementation environment featured a simple topology. Two honeypot servers (HP1 and HP2) with UDT for Windows were installed at two separate locations. They were in a network operating environment running on a 10G pipe trunk 802.1q for tunnelling behind firewalls. The attackers were sourced from the Internet. In the first implementation, all traffic was permitted to traverse through any source, destined through any port on UDP and TCP, and locked to the destination honey pot, where UDT was running on top of UDP. A simple data transfer of 600MB–200GB to 2TB to another server was then performed. The test was initially performed without any protection, while subsequent tests were performed with the proposed security mechanisms in place. The results were then compared.

The following protection schemes were attempted:

1. A simple authentication scheme using Kerberos [42-43] for GSS-API on an application running UDT and UDP;
2. UDT-AO and UDT-DTLS across border gateway routers;
3. Using VPN SSL connections and running the applications in H1 and H2.

### 4.2.4 Results

The number of attacks in Figures 4-11 and 4-12 was constant in the implementation scheme. The dropped packets were detected when the IDS/IPS was activated on the firewalls. The simple authentication scheme – which was developed to transfer a file via UDT, provided by Kerberos using GSS-API on the UDP socket where UDT was operating – provided added protection that sourced the location of the authenticating party in the protected environment.

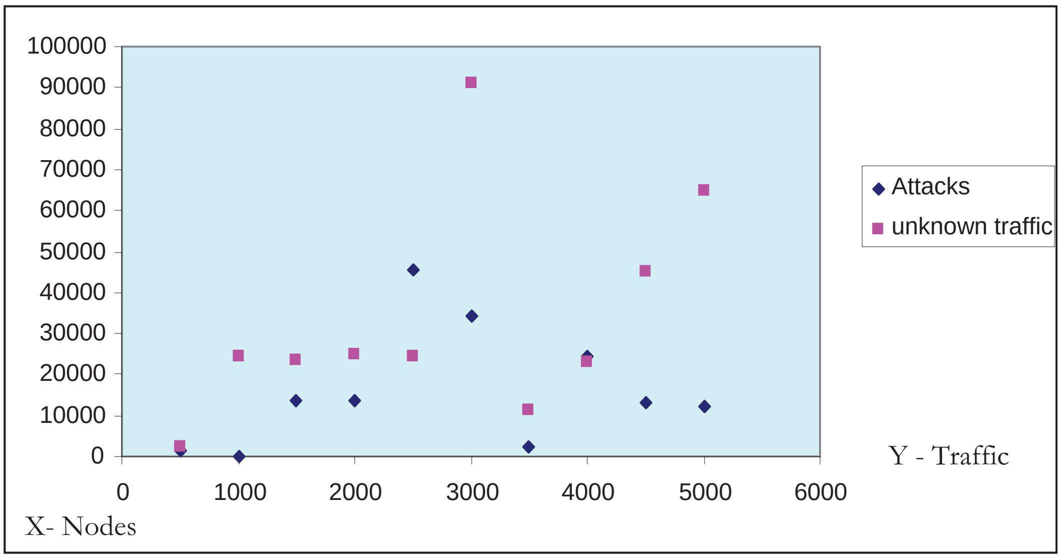


Figure 4-11: Unsecured environment

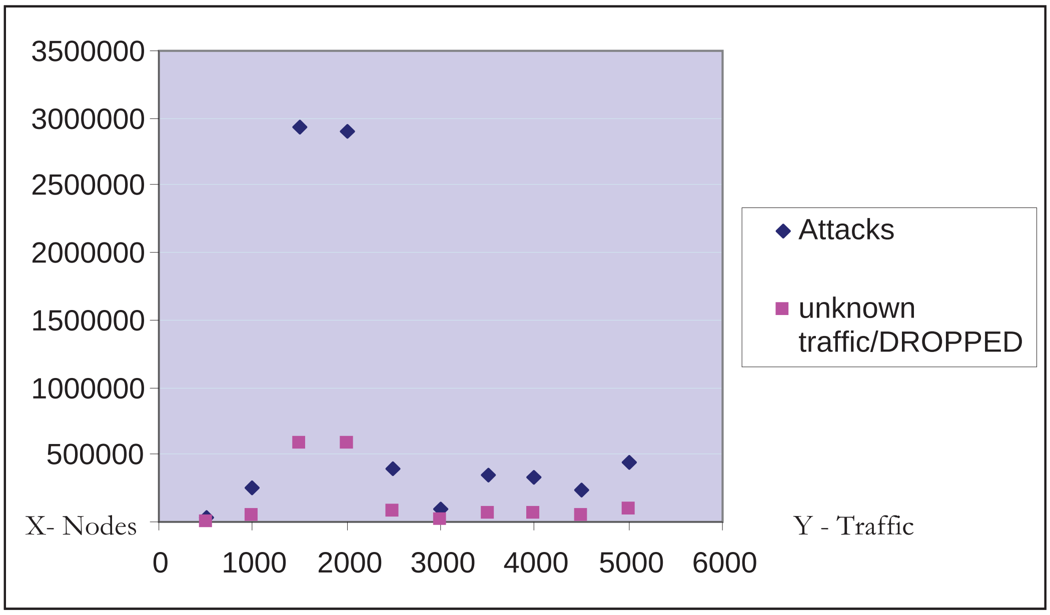


Figure 4-12: Secured environment (with UDT-AO, UDT-DTLS and GSS-API implementations)

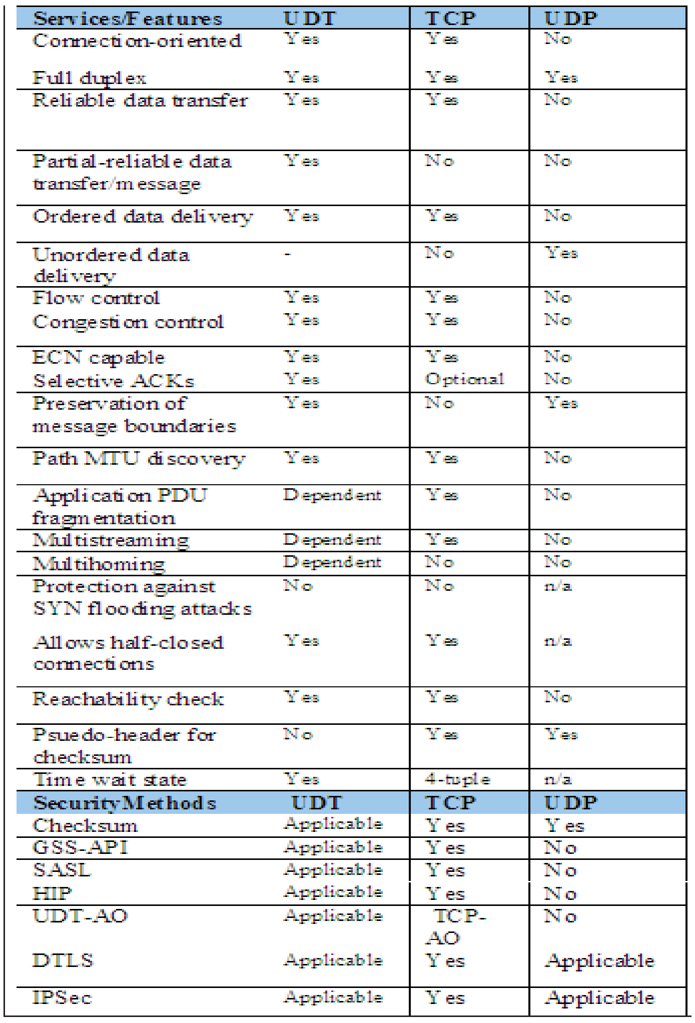
The trend presented in Figure 4-12 yielded significant improvements. Moreover, the end-to-end transfer of data was transparent to the UDT application. The available security mechanisms for UDT requiring minimal application and program development are feasible and predominantly applicable to UDT implementation.

In the case of simple file transfer, many available mechanisms for UDP and TCP, as well as existing security protections for applications, are acceptable, i.e., simple authentication. However, in cases of extensive use of UDT — such as in SDSS and other large project implementations requiring security — UDT requires a security mechanism that is developed and tailored for its behaviours and characteristics based on its design. This work emphasises the need for UDT — just like the existing mature protocols — to be subjected to continuing security evaluations. Amendments based on the evaluations will aim to develop and provide adequate protection, thereby maintaining integrity and confidentiality against various adversaries and unknown attacks. To ensure minimal overhead in data and message transmission streams, these amendments will also minimise dependencies on other security solutions applied on some protocols.

The limitations of the simulation and implementation schemes constructed may be related to the simplicity of the applications developed for the tests.

Experiments are difficult to perform on the following mechanisms: HIP, owing to the required additional layer HIT; and DTLS + CGA.

Table 4-5. Transport Protocol Matrix



| Services/Features | UDT | TCP | UDP |
|---|---|---|---|
| Connection-oriented | Yes | Yes | No |
| Full duplex | Yes | Yes | Yes |
| Reliable data transfer | Yes | Yes | No |
| Partial-reliable data transfer/message | Yes | No | No |
| Ordered data delivery | Yes | Yes | No |
| Unordered data delivery | - | No | Yes |
| Flow control | Yes | Yes | No |
| Congestion control | Yes | Yes | No |
| ECN capable | Yes | Yes | No |
| Selective ACKs | Yes | Optional | No |
| Preservation of message boundaries | Yes | No | Yes |
| Path MTU discovery | Yes | Yes | No |
| Application PDU fragmentation | Dependent | Yes | No |
| Multistreaming | Dependent | Yes | No |
| Multihoming | Dependent | No | No |
| Protection against SYN flooding attacks | No | No | n/a |
| Allows half-closed connections | Yes | Yes | n/a |
| Reachability check | Yes | Yes | No |
| Psuedo-header for checksum | No | Yes | Yes |
| Time wait state | Yes | 4-tuple | n/a |
| **Security Methods** | **UDT** | **TCP** | **UDP** |
| Checksum | Applicable | Yes | Yes |
| GSS-API | Applicable | Yes | No |
| SASL | Applicable | Yes | No |
| HIP | Applicable | Yes | No |
| UDT-AO | Applicable | TCP-AO | No |
| DTLS | Applicable | Yes | Applicable |
| IPSec | Applicable | Yes | Applicable |

A more extensive development of an application using UDT may yield more detailed and comprehensive results (Table 4-5); in addition, the number of false positives and collisions was not considered in the tests. The results nonetheless provide an important indication of how the application using UDT behaves in such environments. Accordingly, protecting UDT can be achieved by introducing approaches related to self-certifying address-generation and verification. A technique that can be applied without major modifications in practice is CGA, which 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. To achieve this, however, a new network layer known as HI has been introduced, making HIP incompatible with current network protocols (Table 4-6).

The protection of UDT through the use of GSS-API in UDT, meanwhile, represents another approach; however, this requires thorough evaluation by application vendors.

Table 4-6. Summary of Schemes

| Method | Complexity | Practicality | New Layer required | Analysis | Simulation/Experiment |
|---|---|---|---|---|---|
| Checksum | High | Moderate | NA | Completed | Application dependent |
| GSS-API | Medium | Moderate | NA | Completed | Application dependent |
| SASL | Medium | Moderate | NA | Completed | NA |
| HIP | High | Moderate | HIT | Completed | Application dependent |
| UDT-AO | High | Moderate | NA | Completed | Algorithmic dependent |
| DTLS | High | Low | NA | Completed | NA |
| IPSec | Medium | Moderate | NA | Completed | Operations dependent |

The use of the GSS-API interface does not in itself provide absolute security or assurance; rather, such attributes are dependent on the underlying mechanisms of UDT, which support a GSS-API implementation to achieve an adequate security mechanism. Another method of ensuring UDT protection is through the introduction of the Authentication-Option (AO); this, however, requires changes to the design of UDT, so as to accommodate an AO field. There is also the requirement to use a better hashing algorithm to ensure that messages transmitted are duly protected.

With the results, we can draw our experience from the comparisons between the various scenarios. Since UDT is a new protocol, there has been little adoption of it; therefore, no security mechanisms are so far available for the application layer. Initially, we considered host-to-host encryption as a feasible solution, depending on the operating system and method desired; however, it was noted that this was likely to require some expertise in the end user of the end host's system, and would further cause various problems related to firewalls, because stateful inspection for UDP — on which UDT runs — cannot be performed on an encrypted session.

The gateway-to-gateway encryption appears to offer a flexible and relatively efficient means to encrypt UDT and UDP data over the public element of a session connection, although it remains vulnerable to interference and probing on the internal site network behind the gateway – unless, that is, appropriate security solutions are taken.

Additionally, it has been observed, based on limited tests performed on entry-level high-speed gateway devices, that the latency effects of encryption do not appear to be significant. Increasing commodity CPU power is making encryption even more viable for reasonable UDT data transfer. Notably, opportunistic encryption is desirable. The VPN firewalls include support for this, but this so far has not been tested in detail: the scaling issues may be significant and should be tested further if stateful/proxy types of firewalls are to be considered for wider UDT deployment.

Gateway encryption (IPSec) – via the existing security devices for higher bandwidth and larger environments – is presently an effective method. Such a device and its configurations, however, require wider-scale testing prior to potential use. Moreover, end-to-end encryption through gateway-to-gateway encryption offers security for UDT, as it does to other protocols.

In the simulation and implementation schemes, IPSec provided adequate protection in terms of data transfer, and similarly provided end-to-end protection on source and destination nodes. In this scheme, the performance of UDT remained the same.

## 4.3 Concluding Remarks

Experiments and simulations were performed for the proposed mechanisms for practical reasons: first, to determine the viability of the selected mechanisms; second, to determine if UDT is applicable to practical applications with the support of resources given the current high-speed network specifications; and third, to subject UDT in rigorous tests within both unprotected and protected environments, in the given limited and isolated constraints.

In this chapter, we presented our own UDT proprietary tool, which was developed to capture UDT traffic. Using this tool to capture data transmissions, we also conducted a comprehensive test and analysed the traffic of data in various environments using the proposed mechanisms.

In the results, it was noted that various mechanisms worked for the existing protocols — such as TCP and UDP — but did not, as expected, work for UDT (see Table 4-5). There were, however, various mechanisms found to be viable for UDT but not for UDP, such as AO and GSS-API. The combination of UDT and UDP provided UDT with the connection and flow control that it required to operate in the selected mechanisms.

The approach presented in this chapter emphasised the following:

(1) Most mechanisms presented were experimentally validated on connection-based protocols.

(2) The method of security connections through IPSec and DTLS worked well for UDP and UDT.

(3) The use of GSS-API, HIP, and CGA were complex and costly, but provided security solutions for the new protocol. Proof of methods through formalisation will be used for verification.

(4) An option such as AO is suitable for UDT, provided that an alternative or a combination of hash methods is used.

(5) Practical solutions include running IPSec on top of UDP, and SSL. However, these solutions can sometimes be detrimental to business needs, such as creating constraints on data flow access

and performance [19] in a more open and trusted environment wherein IPSec is not a requirement.

(6) Public Key can be used, but is also costly when the application relies on certificate authorities.

(7) The methodology, as observed in Table 4-6, provides a new opportunity to address security for UDT and other protocols. The results can assist network and security investigators, designers, and users, all of whom consider and incorporate security when implementing UDT across wide area networks. These can also support the security architectural designs of UDP-based protocols, as well as assist in the future development of other state-of-the-art fast data transfer protocols.

This experiment is extended in the next chapter to rigorous theoretic proof of correctness. It will focus on three mechanisms found provable within UDT implementations. By provable, we mean that these mechanisms are successfully simulated and tested in UDT practical environments, albeit with minor changes required. While the work focuses on limited and minimal resources as well as on the prevention of dramatic changes to the UDT design, we yield promising results from the proposed mechanisms – with, of course, selected techniques in the design and implementation put in place.

Three of these mechanisms are found to be practically viable for UDT due to three important reasons: first, the design of UDT is closely based on TCP and UDP, and therefore much of the design of the security for UDT should also be viable to TCP and UDP; second, existing security mechanisms (e.g., IPSec, DTLS, and Kerberos) have proven to fair well in the TCP and UDP implementations; third, the data flow and multiple connection characteristics of UDT require features that these mechanisms provide – features such as authentication, confidentiality, and data integrity – proving these to be useful and compatible with existing protocols.

# Chapter 5

# Proof of Correctness of the Selected UDT Security Mechanisms

In this chapter, an approach analysing the applicability and secrecy properties of the selected security mechanisms when implemented with UDT is introduced. In this approach, a formal proof of correctness, thereby determining applicability with formal composition logic is carried out. This approach is modular; it has a separate proof for each protocol section that provides insight into the network environment in which each section can be reliably employed. Moreover, the proof holds for a variety of failure recovery strategies and other implementation and configuration options. The technique is derived from Protocol Composite Logic (PCL) on TLS and Kerberos in the literature. The novelty of this work on UDT is maintained: specifically on the developed mechanisms such as UDT-AO, UDT+DTLS, UDT+GSS-API.

## 5.1 Overview of Proof Method

To analyse the protocol based on formal language and logic, we employ the PCL method.

PCL [52] entails reasoning about properties achieved by formalised steps in a setting that does not compel explicit reasoning about attacker actions. Many literatures define PCL as a formal approach for proving security properties of a class of network protocols. According to [1,2,44,52-55,66-65,87,115-116], the central question addressed by PCL is whether it is possible to prove properties of security protocols compositionally, by using reasoning steps that do not explicitly mention attacker actions. In order to reason about the protocols, the proof of properties of one sequence of actions by one agent involves not only local reasoning about the security goal of that component [113,115-116], but also about environmental conditions that prevent destructive interference from other actors that may use the same certificates of key materials, according to [115].

These environmental conditions are generally formulated as protocol invariants. These are properties true for all of the roles of the protocol at hand, and according to [114,115-116], there are properties that may be required in any other protocol operating in the same environment.

Various versions of PCL investigated in past work proved to be sound for protocol executions that employ any number of principals and sessions, over both symbolic models, and over more traditional cryptographic assumptions [1,2, 113,115-116].

Several groups of researchers, according to [2], have taken measures to concatenate the symbolic model to the probabilistic polynomial-time computational model used in security protocol studies, e.g. [44,52-55,65-67,87,113,115-116].

In this chapter, we will prove correctness of each selected security mechanisms for UDT in the symbolic model [30], and determine any issues in its implementation through employing formal logic. Connections between symbolic trace properties and computational soundness properties are achieved in [37]. All these efforts have been directed at proving security properties for well-established mechanisms.

### 5.1.1 Significance

The major milestone of this technique is its use in the integration of methods of security mechanism analysis into the verification process prior to their deployment in UDT.

### 5.1.2 PCL Method

We begin with a brief discussion of PCL relevant to the analysis, [1,2,54]. We base our discussion on the original proponents of PCL [54], which we fully acknowledge in this chapter.

To model protocols, we need to define a protocol by a set of roles, [1,2,52-55] each specifying a sequence of actions to be executed by an honest agent. In PCL, protocol roles are characterised employing a simple 'protocol programming language' according to [54-55] based on *cords*. The possible protocol actions include nonce generation, signatures and encryption, communication steps, and decryption and signature verification via pattern matching [54]. Programs can also rely on input parameters that are typically decided by context, or are the result of set-up operations, and supply output parameters to subsequent operations.

In this Chapter, we outline the proof system and the proof of soundness of the axioms [54] and the rules [1,2,44,52-55,65-67,87,113,115-116]. Most protocol proofs employ formulas of the form $\theta[P]X\varphi$, which expresses that initiating from a state where formula $\theta$ is true, after actions $P$ are determined and executed by the thread $X$, the formula $\varphi$ is true in the resulting state. Formulas $\varphi$ and $\theta$ typically create assertions about temporal order of actions and the data accessible to various principals that are useful for stating secrecy [115-116].

The proof system extends first-order logic with axioms and proof rules for protocol actions, temporal reasoning, knowledge, and a specialised form of invariance rule called the *honesty rule* [52-55,65-67,87,113,115-116]. The honesty rule is essential for merging facts about one role with inferred actions of other roles, in the presence of attackers. Intuitively, according to [115-116] if Alice receives a response from a message sent to Bob, the honesty rule obtains Alice's ability to exercise properties of Bob's role to assert about how Bob created his reply. In short, if Alice contends that Bob is honest [52-54], she may inherit Bob's role to reason from this assumption.

## 5.2 Proof of UDT-AO Protocol

The first mechanism we propose is UDT-AO protocol. It is a lightweight protocol part of our ongoing IETF review process for UDT. We show how UDT-AO is intended to secure long-lived connections for UDT when used in various routing protocols. It is not intended to replace IPsec suite to secure connections. Hence, we analyse UDT-AO protocol for consideration in the development of a viable security architecture. We employ a finite-state method to ascertain that this protocol does not have any flaws. We also substantiate the protocol utilising a protocol verification logic. We use formal proof to verify the viability of this protocol to secure UDT transmission.

The UDT-AO is an authentication framework that was proposed to the Internet Engineering Task Force (IETF). It operates on the transport layer and supports a variety of mechanisms for two entities to authenticate themselves to each other.

UDT is a connection-oriented protocol. As such, it requires 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 [142].

The proposed option can be implemented on Type 2 of the UDT header. This field is reserved to determine specific control packets in the Composable UDT framework. Every segment sent on a UDT connection to be secured against spoofing will similarly contain the 16-byte MD5 digest achieved by applying the MD5 algorithm to these items, in the following (Table 5-1) similar order required for UDT:

Table 5-1: UDT + UDP Process

```
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 [34,36-37,61,151]. However, a particular UDT implementation will determine what the application may specify as the key.

The focus is on validating the protocols and their applicability to UDT by determining if errors and incompatibility problems exist, and, therefore, in their absence re-enforce the viability of AO for UDT security architecture.

UDT-AO provides message authentication verification between two end points [22-33]. This message authentication function protects a message's data integrity [33]. In order to accomplish this function, Message Authentication Codes (MAC) are utilised, which rely on Shared Keys (SK). There are various ways to generate MACs. The general requirements are outline for  MACs used in UDT-AO, both for currently specified MACs and for any future specified MACs. Two MACs algorithms selected that are necessary in all UDT-AO implementations. Moreover, two Key Derivation Functions (KDFs) employed to create traffic keys used by the MACs are introduced. These KDFs are required by all UDT-AO implementations, as presented (Table 5-2) below.

Table 5-2: Successful Message Exchange in UDT-AO

```
[Message 1:S . P]: SNonce, S, AlgocryptList

[Message 2:P . S]: P, S, PNonce, SNonce, AlgocryptList, AlgocryptSel,

                   {Payload}KDF(PKEY),MACSK

[Message 3:S . P]: PNonce, SNonce, AlgocryptSel, {Payload}KDF

                   (PKEY),MACSK

[Message 4: P . S]:{Payload}KDF (PKEY),MACSK
```

Successful UDT-AO message transfer exchange.

The Keys Shared (SKey), (PKey), and Private Keys (PSK) are derived from a key derivation function KDF, which names a Pseudorandom Function (PRF) and uses a Master_Key (MKey) and some connection-specific input with that PRF to generate Traffic_Keys (TKey), the keys suitable for authenticating and integrity-checking individual UDT segments.

## 5.2.1 UDT-AO Description

MKey is generated as seed for the KDF. It is similarly assumed this is a readable PSK; thus, it is also considered, which based on the characteristics of existing protocols, it is of variable length. MKey should be random, but in some cases when chosen by the user, it might not be. For interoperability, the management interface by which the PSK is configured [117] must acknowledge ASCII strings, and must also permit for configuration of any arbitrary binary string in hexadecimal form.

The assumption is that KDF-X selects two arguments, a key and a seed, and outputs a bit string of length X [2]. The notation KDF-X(Y,Z) [i..j] constitutes the *i'th* through *j'th* octets (8 bits) of the output of the KDF-X. The PSK has length PL, while the SKey and PKey have length KS, which is a value specified by the example Algocrypt (Table 5-3).

Table 5-3: KDF-AES-128-CMAC

```
KDF-AES-128-CMAC

Input : MKey (Skey or PKey length KS)

          :I (input data of the PRF)

          :  MKeyLen (length of MKey in octets)

          :  len ( length in octets)

Output: TKey (Traffic_key, 128-bit Pseudo-Random Variable)

Variable: Key (128-bit for AES-CMAC)

Step 1.   If MKeyLen is equal to 16

Step 1a.  then
```

```
          K :=MKey;

Step 1b.  else

          K: = AES-CMAC(0^128, MKey, MKeylen);

Step 2.   TKey : AES-CMAC(K,I, len);

           Return TKey;
```

Key derivation is defined as follows:

```
inHEX=0x00.

inputString = PNonce || P || SNonce || S.

MKey = KDF-KS(inHEX value, PL || PSK || AlgocryptSel || inputString)[0..KS-
1].

Assumption 160 bits in case of  KDF_ HMAC_SHA1

SKey = KDF-{160+2*KS}(MKey, inputString)[160..159+KS].

PKey = KDF-{160+2*KS}(MKey, inputString)[160+KS..159+2*KS].

Assumption 128 bits in case of KDF_AES128_CMAC based on AES-CMA-PRF-128

SKey = KDF-{128+2*KS}(MKey, inputString)[128..127+KS].

PKey = KDF-{128+2*KS}(MKey, inputString)[128+KS..127+2*KS].
```

Similarly, the first 128 octets of KDF-{128+2*KS}(MKey, inputString)[128+KS..127+2*KS] are divided into two keys [117] which are exported as part of the protocol. They may be employed for key derivation in higher level protocols [1,2]. Every AO method which supports key derivation is needed to export such keys, but they have been omitted because they are not relevant to the current analysis.

UDT-AO is designed to equip mutual basic authentication between the peer and the server (end-to-end). The successful message exchange decides the authenticity of the peer by the use of key SKey, which is deduced from the long-term key PSK for MAC in Message 2 in the algorithm found in Table 5.2. The successful message exchange purports [117] the authenticity of the server by the use of SKey for the MAC in Message 3. AO is also designed to cater for session independence. This means that even if there is a weakened exchange, this prevents the attacker from compromising past or future sessions. AO is a symmetric key authentication protocol, and therefore the secrecy of long-term key PSK is essential for all the above introduced properties to hold [52-54].

Both the peer and the server are dependent upon to silently dispose of any message which is unexpected (e.g., receiving Message 4 instead of Message 2), doesn't parse (e.g. the wrong nonce is returned), or whose MAC is invalid. The only exception is for Message 2. If the server acquires an invalid MAC then it must respond with an UDT-AO failure message. The peer must always be willing to accept Message 1 from a server since there is no integrity protection [1,2,52-55].

The analysis that was determined confirms the basic requisite for securing UDT. The non-standard use of a key derivation function [117] which is exhausted to create session keys is a fundamental weakness, but does not accommodate an obvious attack detrimental to UDT data transmission. In addition, the difficulties which such non-standard usage composes when trying to validate the protocol's correctness should not be rejected. This is because of the fact that the messages are not encrypted, and therefore a basic authentication is more so a necessity. Using a modular approach to protect UDT data transmission by combining other security mechanisms to guarantee a secure UDT implementation, without computation overload, therefore, is prescribed.

### 5.2.2 UDT-AO Proof of Correctness

In this section, the introduction of a formal correctness proof of UDT-AO using a formal language method that executes any number of principals and sessions, over both symbolic models and over more traditional cryptographic assumptions is presented.

## 5.2.3 Formal Description of UDT-AO in the Formal Language

Table 5-4: Formal Description of UDT-AO

| **UDT: Server** ≡ [ | **UDT: Peer** ≡ [ |
|---|---|
| new *SNonce*; | receive *SNonce*. ˆ *S.ALGOCRYPTLIST*; |
| send *SNonce*. ˆ *S.ALGOCRYPTLIST*; | new *PNonce*; |
| receive ˆ *P*. ˆ *S.PNonce.SNonce*. | *MKEY* := prg *PSK*; |
| *ALGOCRYPTLIST.ALGOCRYPTSEL.enc*1.*mac*1; | *InputString* := *PNonce*. ˆ *P.SNonce*. ˆ*S*; |
| *MKEY* := prg *PSK*; | *SKEY* := kdf1 *InputString*,*MKEY*; |
| *InputString* := *PNonce*. ˆ *P.SNonce*. ˆ*S*; | *PKEY* := kdf2 *InputString*,*MKEY*; |
| *SKEY* := kdf1 *InputString*,*MKEY*; | *enc*1 := symenc *pl*1, *PKEY*; |
| *PKEY* := kdf2 *InputString*,*MKEY*; | *mac*1 := mac ˆ *P*. ˆ *S.PNonce.SNonce*. |
| *pl*1 := symdec *enc*1, *PKEY*; | *ALGOCRYPTLIST.ALGOCRYPTSEL.enc*1; |
| verifymac *mac*1, ˆ *P*. ˆ *S.PNonce.SNonce*. | send ˆ *P*. ˆ *S.PNonce.SNonce*. |
| *ALGOCRYPTLIST.ALGOCRYPTSEL.enc*1,*SKEY*; | *ALGOCRYPTLIST.ALGOCRYPTSEL.enc*1.*mac*1; |
| *enc*2 := symenc *pl*2, *PKEY*; | receive *PNonce.SNonce*. ˆ *S.ALGOCRYPTLIST*. |
| *mac*2 := mac *PNonce.SNonce*. | *enc*2.*mac*2; |
| *ALGOCRYPTLIST.enc*2, *SKEY*; | verifymac *mac*2, *PNonce.SNonce*. |
| send *PNonce.SNonce*. ˆ *S.ALGOCRYPTLIST.enc*2.*mac*2; | *ALGOCRYPTLIST.enc*2, *SKEY*; |
| receive *enc*3.*mac*3; | *pl*2 := symdec *enc*2; |
| verifymac *mac*3, *enc*3, *SKEY*; | *enc*3 := symenc *pl*3; |
| *pl*3 := symdec *enc*3, *PKEY*; | *mac*3 := mac *enc*3, *SKEY*; |
| ]*S* | send *enc*3.*mac*3; |
|  | ]*P* |

## 5.2.4 UDT-AO Security Properties

The properties that UDT-AO ought to satisfy include:

*Setup Assumption.* To establish security properties of the UDT-AO protocol, [1,2,52-55], it is deduced that the Server$\hat{}S$ and the Peer$\hat{}P$ in consideration are both honest, and are the only parties which recognise the corresponding shared *PSK*. However, this acquiesces all other principals in the network to be potentially malicious and capable of reading, blocking and changing messages transmitted to the network.

**Definition 1 (Secrecy).** *The Server to Peer is said to exchange key secrecy, where defined as:*

φsetup ≡ Honest( ˆ P) ^ Honest( ˆ S) ^ (Has(X, PSK) ⊃ ˆX = ˆ S v ˆX = ˆ P)

*Security Theorems.* The secrecy theorem for UDT-AO inculcates that the signing and encryption keys SKey and PKey should not be obvious and known to any principal other than the peer and the server. For server$\hat{}S$ and peer$\hat{}P$, this property is used as *SECudt-ao*(*S,P*) defined as:

SECudt-ao(S, P) ≡ (Has(X, PKey) v Has(X, SKey)) v (ˆX = ˆ S ⊃ˆX = ˆ P)

**Theorem 1 (AO-Secrecy).** *On execution of the server role, key secrecy holds. Similarly for the peer role. Formally,* UDT-AO _ *SECserver pkey,skey , SECpeer pkey,skey, where*

*SECserver pkey,skey*≡ [**UDT**: **Server**]*S SECudt-ao*(*S, P*)

*SECpeer pkey,skey*≡ [**UDT**: **Peer**]*P SECudt-ao*(*S, P*)

**Proof Sketch.** *Proof intuition is as follows:*

*PSK* is considered to be known to ˆ*P* and ˆ*S* only. The keys *SKey,PKey* are determined by employing *PSK* in a key derivation function (*MKEY* could be a truncation of *PSK* or generated by application of a PRG to *PSK*, according to the length needed). The honest parties employ *SKey,PKey* as only encryption or signature keys - none of the payloads are determined by a KDF application. This is the intuition why *SKey,PKey* remain secrets. A rigorous proof would utilise a stronger induction hypothesis and induction over all honest party actions [54].

The authentication theorem for UDT-AO creates that on completion of the protocol, the principals accede on each other's identity, protocol completion status, the cryptographic suite list and selection, and each other's nonces. The authentication property for UDT-AO is determined in terms of matching conversations [2]. The basic idea of matching conversations is that on execution of a server role, we corroborate that there exists a role of the designated peer with a corresponding view of the interaction [117].

For server $\hat{}S$, communicating with client $\hat{}P$, matching conversations are created as *AUTHudt-ao*(*S, P*) defined below:

```
AUTHudt-ao(S, P) ≡ (Send(S, msg1) < Receive(P,msg1))^

                   (Receive(P,msg1) < Send(P,msg2))^

                   (Send(P,msg2) < Receive(S, msg2))^

                   (Receive(S, msg2) < Send(S, msg3)
```

**Definition 2.** *Server is said to execute and formulate authentication session for the authenticator.*

**Theorem 2 (AO-Authentication).** *On formulation of the server role, authentication holds. Similarly for the peer role. Formally,* UDT-AO _ *AUTHserver peer ,AUTHpeer server, where*

```
AUTHserverpeer ≡ [UDT: Server] S ∃η. P = ( ˆ P, η) ^ AUTHudt-ao(S, P)
```

AUTHpeer server≡ [UDT: Peer] P ∃η. S = (ˆ S, η) ^ AUTHudt-ao(S, P)

**Proof Sketch.** The formal proof in PCL is in Section 5.2.3. We formulate the proof intuition here. We required to add two new axioms **MAC0** and **VMAC** (see Section 5.2.5 ) to the extant PCL proof system in order to contend about MACS. Axiom **MAC0** says that anybody calculating a *mac* on a message *m* with key *k* must include both *m* and *k*. Axiom **VMAC** says that if a *mac* is proven to be correct, it must have been generated by a *mac* action.

***AUTHserver peer*:** The Server validates the *mac*1 on *msg*2 to be a mac with the key *SK*. By axiom **VMAC**, it must have been formulated by a *mac* action and by **MAC0**, it must be by someone who has *SKey*. Hence by secrecy, it is either *P* or *S* and therefore, in either case, an honest party. It is an invariant of the protocol that a *mac* action on a message of the form *X̂.Y.XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc* is executed by a thread of ˆ*X*, captured by *Γ*1 - hence it must be a thread of ˆ*P*, say *P*. Also using *Γ*1 it ascertain that *P* received the first message and generated nonce *PNonce [117]* and sent it out first in the message *msg*2. From the actions of *S*, its newly determined *SNonce* is sent out first in *msg*1. Employing this information and axioms **FS1***,* **FS2**, the sequence order then actions as receive and send as described in *AUTHserver peer*.

*AUTHpeer server* : The Peer verifies the *mac*2 on *msg*3 to be a *mac* with the key *SK*. By axiom **VMAC**, it must have been determined by a *mac* action and by **MAC0**, it must be by someone who has *SKey [117]*. Hence by secrecy, it is some thread of either ˆ*P* or ˆ*S* and therefore, in either case, an honest party. It is an invariant of the protocol that a *mac* action on a message of the form *YNonce.XNonce.*ˆ*Y.ALGOCRYPTLIST.enc*, is performed by a thread of ˆ*Y* , captured by *Γ*2 - hence it must be a thread of ˆ*S*, say *S*.

However, this *mac* according to [117] does not restrict the variables *ALGOCRYPTSEL* and *enc*1 sent in *msg*2. So to ascertain that *S* received the exact same message that *P* sent, we utilise *Γ*2 to further reason that *S* verified a *mac* on a message of the form of *msg*2. And axioms **VMAC***,* **MAC0** repeat to deduce that this *mac* [117] was generated by threads of ˆ*S* or ˆ*P*. Now, it is possible to use *Γ*1 and the form of *msg*2 to contend that a thread of ˆ*P* did it, which also generated *PNonce* - hence by **AN1**, it must be *P* itself. Now an invariant can be managed that states that a thread initiating such a *mac* does it uniquely, captured by *Γ*3, thus binding *ALGOCRYPTSEL, enc*1. Moreover, **FS1***,* **FS2** can now be managed as in the previous proof to create the order described in *AUTHpeer server*.

## 5.2.5 UDT-AO Axioms

The proof system enhances first-order logic with axioms and proof rules for protocol actions, temporal reasoning, properties of security (e.g., cryptographic) primitives, and a specialised form of program invariance rule called *honesty rule* [54-55]. Below is the list of axioms employed in this thesis.

### Formal Proofs

#### *New Axioms*

**MAC0** Mac(X, m, k) ⊃Has(X,m) ^ Has(X, k) *means anybody computing a mac on a message m with k must possess both m and k.*

**VMAC** VerifyMac(X, m',m, k) ⊃∃Y. Mac(Y,m, k) ^m' = MAC[k](m) *states that if a mac is verified to be correct, it must have been generated by a mac action*

*Note: Extant PCL proof system reason about MACS through the new axioms MAC0 and VMAC*

#### *Invariants*

Γ1 ≡ Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc,K) ⊃ ˆ Z = ˆX ^ (Receive(Z, Y Nonce. ˆY .ALGOCRYPTLIST) - *assign Γ1 invariant for AUTHserver peer. – it captures the mac action on a message of the form XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc, which is performed by a thread X.ˆ, captured by Γ1*

< Send(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc.mac)) ^

mac = MAC[K]( ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc) ^

FirstSend(Z,XNonce, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc.mac)

Γ2 ≡ Mac(Z, Y Nonce.XNonce. ˆY .ALGOCRYPTLIST.enc,SKEY) ^SKEY = KDF1[K](YNonce. ˆY .XNonce.ˆX)⊃

ˆZ = ˆ Y ^ ∃ ALGOCRYPTSEL', enc1. (Send(Z, Y Nonce. ˆY .ALGOCRYPTLIST) <

Receive(Z, Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1.mac1) <

Send(Z, Y Nonce.XNonce.ALGOCRYPTLIST.enc.mac)) ^

mac1 = MAC[SKEY](Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1) ^

mac = MAC[SKEY](Y Nonce.XNonce.ALGOCRYPTLIST.enc) ^

VerifyMac(Z,mac1, Yˆ .Xˆ.Y Nonce.XNonce.ALGOCRYPTLIST.ALGOCRYPTSEL' .enc1, SKEY) ^

FirstSend(Z, Y Nonce, Y Nonce. ˆY .ALGOCRYPTLIST) - *assign Γ2 invariant for AUTHpeerserver*

Γ3 ≡ Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc,K) ^

Mac(Z, ˆ X.ˆY .XNonce.YNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc' ,K) ⊃ ALGOCRYPTSEL =

ALGOCRYPTSEL' ^ enc = enc' *assign Γ3 invariant for AUTHpeerserver*

---

*Formal Proof of AUTHserver*

*peer*

---

**AA1** [**UDT-AO : Server**]S VerifyMac(S, mac1, ˆ P. ˆ S.PNonce.SNonce. (1)

ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY)

*Axiom VMAC is generated by a mac action and by MAC0, it be someone with SKEY*

SECserver **VMAC** [**UDT-AO : Server**]S ∃X. ( ˆX = ˆ P ∨ ˆX = ˆ S) ^ (2)

pkey,SKey , Mac(X,
P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY)

*AUTHserver peer verifies the mac1 on msg2 to be a mac with key SKEY*

Γ1 [**UDT-AO : Server**]S ∃η. P0 = (ˆ P, η) ^

Mac(P0, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) ^

*By using Γ1 we prove that P received the first message and generated nonce PNonce and sent it out first in the message msg2.*

Receive(P0,msg1) < Send(P0,msg2) ^

FirstSend(P0, PNonce,msg2) (3)

*(3) temporary predicate requires only until the same nonce used by the peer succeeds in completion*

InstP0→P[**UDT-AO:Server**]SMac(P,
ˆP.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) ^

Receive(P,msg1) < Send(P,msg2) ^

FirstSend(P, PNonce,msg2) (4)

*(4) temporary predicate requires only until the same nonce used by the peer succeeds in completion*

**FS1 [UDT-AO : Server]**S FirstSend(S, SNonce,msg1) *order the receives and sends* (5)

**FS2**, **[UDT-AO : Server]**S (Send(S,msg1) < Receive(P,msg1)) ^ *order the receives and sends*

(Receive(P,msg1) < Send(P,msg2)) ^

(Send(P,msg2) < Receive(S,msg2)) (6)

**AA4 [UDT-AO : Server]**S (Receive(S,msg2) < Send(S,msg3)) (7)

AUTHserver (8)

peer

## *Formal Proof of AUTHpeerserver*

**AA1 [UDT-AO : Peer]**P VerifyMac(P,mac2, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2, SKEY) (9)

SECserver **VMAC** [UDT-AO : Peer]P $\exists$X. ( ˆX = ˆ P $\vee$ ˆX = ˆS) ^

pkey,SKey , Mac(X, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2, SKEY) (10)

Γ2, **[UDT-AO : Peer]**P $\exists\eta$. S0 = (ˆ S, $\eta$) ^

$\exists$ALGOCRYPTSEL', enc1'. (Send(S0, SNonce. ˆ S.ALGOCRYPTLIST) <

Receive(S0, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1'.mac1) <

Send(S0,PNonce.SNonce.ALGOCRYPTLIST.enc2.mac)) ^

mac1 = MAC[SKEY]( ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1') ^

mac = MAC[SKEY](PNonce.SNonce.ALGOCRYPTLIST.ˆS.enc2) ^

FirstSend(S0, SNonce, SNonce. ˆ S.ALGOCRYPTLIST) (11)

Inst S0 $\longrightarrow$ S **[UDT-AO : Peer]**P $\exists$ALGOCRYPTSEL', enc1'. (Send(S, SNonce. ˆ S.ALGOCRYPTLIST) <

Receive(S, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1'.mac1) <

Send(S, PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2.mac)) ^

mac1 =MAC[SKEY]( ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1') ^

mac = MAC[SKEY](PNonce.SNonce. ˆ S.ALGOCRYPTLIST.enc2) ^

VerifyMac(S,mac1, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1', SKEY) ^

FirstSend(S, SNonce, SNonce. ˆ S.ALGOCRYPTLIST) (12)

Inst ALGOCRYPTSEL', enc1', [**UDT-AO : Peer**]P ∃X. ( ˆX = ˆ P ∨ X = ˆS) ^

**VMAC,MAC0** Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1',SKEY) (13)

Γ1,**AA1**, [**UDT-AO : Peer**]P New(X, PNonce) ^ New(P, PNonce) (14)

**AN1**, [**UDT-AO : Peer**]P X = P *AN1 generated by PNonce for P thread* (15)

**AA1**, [**UDT-AO : Peer**]P Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL'.enc1', SKEY) ^ Mac(X, ˆ P.ˆS.PNonce.SNonce.ALGOCRYPTLIST.ALGOCRYPTSEL.enc1,SKEY) (16)

Γ3, [**UDT-AO : Peer**]P ALGOCRYPTSEL' = ALGOCRYPTSEL ^ enc1' = enc1 (17)

[**UDT-AO : Peer**]P (Send(S,msg1) < Receive(S,msg2) < Send(S,msg3)) (18)

**FS1** [**UDT-AO : Peer**]P FirstSend(P, PNonce,msg2) *order receives and sends* (19)

**FS2**, [UDT-AO : Peer]P (Send(S,msg1) < Receive(P,msg1)) ^ *order receives and sends*

(Send(P,msg2) < Receive(S,msg2)) ^

(Receive(S,msg2) < Send(S,msg3)) (20)

**AA4** [UDT-AO : Peer]P (Receive(P,msg1) < Send(P,msg2)) (21)

AUTHpeer (22)

server

Axioms define general truths applicable to every protocol [54-55]. For instance, the axiom VMAC encodes the common property of signatures [1,2,54] that if a thread verifies that a message *x* is assigned by a principal *Y^,* it must be *Y^* signature key used to generate the signature. Further, if the agent Y^ is honest, no one else has access to this key, implying that there exists a thread of the agent *Y* that did indeed sign the term *x,* according to [52-55].

## 5.2.6 UDT-AO Operating Environment

The formal proof outlined above applies to the case where fresh nonces for UDT-AO are generated every time. When the peer employs the same nonce repeatedly until it succeeds in completion [117], a different form of reasoning needs to be utilised to ascertain the intended message ordering. Specifically, the predicate

FirstSend(*P, PNonce,msg*2) does not necessarily hold anymore [117]. However, it is possible to still appeal to the fact that a MAC must have been generated and distributed before it could be received and verified, to be able to sequentially order messages. Therefore, formalising this requires the new axiom VMAC:

```
VMAC' Receive(X,m2) ^ Contains(m2,m') ^VerifyMac(X,m',m, k) ^

Mac(X,m, k)⊃ ∃Y,m1. Mac(Y,m, k) ^ Contains(m1,m') ^

(Send(Y,m1) < Receive(X,m2))
```

The proof above uses axioms previously proved sound in the symbolic model.

## 5.3 Proof of UDT+DTLS Protocol

In this section, we outline the DTLS with UDT protocol. In UDT+DTLS, we focus on two principals called the UDT+DTLS client and the UDT+DTLS server. In a way correlative to TLS, DTLS guarantees mutual authentication and establishes a shared key between these two principals. The proof of UDT+DTLS, therefore, lies on the authentication property. The identification of any UDT+DTLS program invariants also emphasises the security properties of UDT+DTLS as part of the development of the security architecture.

### 5.3.1 UDT-DTLS Description

We outline DTLS protocol in the formal language we introduced in the earlier sections. DTLS protocol provides end-to-end security; it is selectively deployed on the Internet in some security and e-commerce systems. We focus on how DTLS can be used to mutually authenticate the supplicant and the authenticator, and to derive a shared secret key [1,2,30,52-54] to add security in UDT data transmissions.  We will be proving DTLS in isolation and will be identifying conditions under which other protocols may operate concurrently without introducing any vulnerabilities. Identifying such conditions appears valuable, given the promising deployment of DTLS on UDT. We employ the terms client and server for DTLS protocol participants, and similarly, we adhere to the proof of correctness of DTLS based on TLS, when deploying UDT.

### 5.3.2 UDT-DTLS Proof of Correctness

DTLS has many possible modes of operation. Similarly, we limit our attention to the mode where both the server and the client have certificates, since this mode satisfies the mutual authentication property. DTLS is developed to construct over datagram to cater for unreliable packet transmission, retransmission and reordering. To the greatest extent, DTLS is identical to TLS, however unlike TLS, DTLS adds explicit state to records and adds explicit sequence numbers to secure datagrams.

The DTLS utilises a simple retransmission timer to handle packet loss. The description in 5.3.3 illustrates the basic concept using the first phase of DTLS handshake. Server programs are elucidated in the following section, where Vy and Vx exhibits the protocol version and cipher suite, Ky is the server's public key, V is the client's verification key. We employ the match action to check signatures, verify keyed hashes and generate decryption. Observe that the terms handShake1 and handShake2 exhibit the concatenation of all the terms sent and received by a principal up to the point it is used in the program.

### 5.3.3 Formal Description of UDT+DTLS in the Formal Language

Table 5-5: Formal Description of UDT+DTLS

| | |
|---|---|
| ***UDT:DTLS Server*** *= [* | ***UDT: DTLS Client = [*** |
| *DTLS : Server = (Y, Vy)[* | *DTLS : Client = (X, ˆ Y , Vx)[* |
| *receive ˆX. ˆ Y .nx.Vx; new ny;* | *new nx; send ˆX. ˆ Y .nx.Vx;* |
| *send ˆ Y . ˆX .ny.Vy;* | *receive ˆ Y . ˆX .ny.Vy;* |
| *receive ˆX. ˆ Y .encky.sig.hc200;* | *new secret;* |
| *sigterm := ˆX . ˆ Y .nx.Vx · ˆ Y . ˆX .ny.Vy·* | *encky := pkeyenc secret, ˆ Y ;* |
| *encky;* | *sigterm := ˆX . ˆ Y .nx.Vx · ˆ Y . ˆX .ny.Vy·* |
| *verify sig, sigterm, ˆX;* | *encky;* |
| *secret := pkeydec encky, ˆ Y ;* | *sigvx := sign sigterm, ˆX;* |
| *hc20 := ˆX. ˆ Y .nx.Vx · ˆ Y . ˆX .ny.Vy·* | *hc2 := hash ˆX. ˆ Y .nx.Vx · ˆ Y . ˆX.ny.Vy·* |

| | |
|---|---|
| `encky · sig · “client”;`<br><br>`verifyhash hc200, hc20, secret;`<br><br>`hs := hash ˆX . ˆ Y .nx.Vx · ˆ Y . ˆX .ny.Vy · ˆX . ˆ Y .`<br><br>`encky · sig · “server”, secret;`<br><br>`send ˆ Y . ˆX .hs;`<br><br>`]Y hY, ˆX , secretiY` | `encky · sigvx · “client”, secret;`<br><br>`send ˆX . ˆ Y .encky.sigvx.hc2;`<br><br>`receive ˆ Y . ˆX .hs00;`<br><br>`hs0 := ˆX . ˆ Y .nx.Vx · ˆ Y . ˆX .ny.Vy · ˆX. ˆ Y .`<br><br>`encky · sigvx · “server”;`<br><br>`verifyhash hs00, hs0, secret;`<br><br>`]XhX, ˆ Y , secretiX` |

### 5.3.4 UDT-DTLS Security Properties

The properties that DTLS (based on TLS) ought to satisfy include:

1. Like TLS, the DTLS principals accede on each other's identity [128] protocol completion status, the values of the protocol version, cryptographic suite, and the secret that the client sends to the server. For server ˆY communicating with client ˆX, this property is formulated in Definition 3.

2. The secret that the client formulates should not be known to any principal other than the client and the server [1,2,22-33,54,128]. For server ˆY and client ˆX , this property is generated in Definition 4.

**Definition 3. (DTLS Authentication, similar to TLS** [60]**)**

DTLS is said to formulate session authentication for the server role if $_{\text{Dtls,auth}}$ holds, where

Table 5-6: Honest Rule

```
Dts,auth ::= Honest( ˆX ) ^ Honest( ˆ Y ) ⊃ ∃X.ActionsInOrder(

Send(X, ˆX , ˆY ,m1),

Receive(Y, ˆX, ˆY ,m1),

Send(Y, ˆ Y , ˆX ,m2),

Receive(X, ˆ Y , ˆX ,m2),

Send(X, ˆX , ˆY ,m3),

Receive(Y, ˆX, ˆY ,m3),

Send(Y, ˆ Y , ˆX ,m4))

and m1, m2, m3, m4 represent the corresponding DTLS messages.
```

**Definition 4 (DTLS Key Secrecy).** DTLS is said to provide secrecy if $_{\text{Dtls,sec}}$ holds, where

```
 Dtls,sec ::= Honest( ˆX) ^ Honest( ˆ Y ) ⊃

 Has( ˆX , secret) ^

 Has( ˆ Y , secret) ^

(Has( ˆ Z, secret) ⊃ ˆ Z = ˆX v ˆ Z = ˆ Y )
```

The proof system is used to prove guarantees for both the client and the server. Due to space constraints, the list only includes the guarantee for the authenticator in Theorem 3. The client guarantee is similar. The secrecy of the exchanged key material in TLS is established by combining local reasoning based on the client's actions with global reasoning about actions of honest agents. Intuitively, a client that generates the secret only sends it out either encrypted with an honest party's public key or uses it as a key for a keyed hash (this is captured by the predicate NonceSource). Furthermore, no honest user will ever decrypt the secret and send it in the clear. Specifically, an honest party can send the secret in the clear only if it receives it in the clear first. Secrecy follows directly from these two facts.

**Theorem 3 (DTLS Server Guarantee).**

(1) On execution of the server role, key secrecy and session authentication are guaranteed if the formulas in (2) hold.

Formally,

```
Dtls,1  ^ Dtls,,2 |-

DTLS:Server]X Dtls,auth  ^ Dtls,,sec
```

(2) The formulas in below are invariants of DTLS.

Formally, DTLS Invariants:

Table 5-7: DTLS Invariants

```
Dtls,1  :=  ¬∃m.Send(X,m)  ^   (Contains(m,HASHsecret(handShake1,  “server”))∨
Contains(m,HASHsecret(handShake2,    “client”))    ∨    Contains(m,    SIGV
x(handShake1)))

Dtls,2 := Honest( ˆ Y ) ^  Send(Y,m) ^  ContainsOut(m, secret,ENCKy(secret))
⊃(¬Decrypts(Y,m′)  ^  Contains(m′, secret))v (Receives(Y,m′′) < FirstSend(Y,
secret) ^  ContainsOut(m′′, secret,ENCKy(secret)))
```

## 5.3.5 UDT-DTLS Operating Environment

We now characterise the class of protocols that safely constitutes with DTLS. As in the preceding section, we relate DTLS invariants to deployment considerations.

DTLS,1 states that messages of a certain format should not be sent out by any protocol that executes in the same environment as TLS. One set of terms exhibit keyed hashes of the handshake, where the key is the shared secret established by a DTLS session; [22-33,54,60] another set refers to signatures on the handshake messages. A client running a protocol that signs messages indiscriminately could instigate the loss of the authentication property. Such an attack would only be possible if the client certificate used by DTLS was shared with other protocols and was infringed by them.

DTLS rules out an undesirable sequence of actions that may allow an intruder to learn the shared secret. Intuitively, if an honest principal is tricked into decrypting a term containing the secret using its private key, after which it sends out the contents of the encryption [60], the secrecy property of DTLS is lost. Clearly, if principals utilise an exclusive public/private key pair for DTLS, such an attack is not possible. However, since another protocol may employ the same public/private key pair as DTLS, it is important to check that these formulas are invariants of any other protocol.

## 5.4 Proof of UDT+GSS-API (Kerberos) Protocol

The third mechanism proposed is UDT+GSS-API. We illustrate the proof system in this section using GSS-API and we focus on Kerberos V5 [39-40,45,110], proven to any protocols, which GSS-API uses. In this section we illustrate how Kerberos is formalised to achieve proofs of secrecy and authentication.

### 5.4.1 UDT+GSS-API (Kerberos) Description

The formulation is based on the A level formalisation of Kerberos V5 in [110]. Kerberos provides mutual authentication and establishes keys between Clients and application Servers, employing a sequence of two message interactions with trusted parties called the Kerberos Authentication Server (KAS), and the Ticket Granting Server (TGS) [35,37,43].

### 5.4.2 Proof of UDT+GSS-API through Kerberos

Mechanisms are denoted in a process calculus by defining a set of roles [54], such as ‘Client’, or ‘Server.’ Each role is provided by a sequence of actions such as sending or receiving a message, generating a new nonce, or decrypting or encrypting a message [110]. In a run of a mechanism, a principal may execute one or more instances of each role, each execution constituting a thread identified by a pair ($\hat{}X;\eta$), where $\hat{}X$ is a principal and $\eta$ is a unique session identifier.

Kerberos has four roles [110]: Client, KAS, TGS and Server. The pre-shared long-term keys between the client and KAS, the KAS and TGS, and the TGS and the application server, will be written as k $_{X;Y}$ $^{type}$ where X and Y are the principals sharing the key. The type appearing in the superscript indicates the relationship between X and Y: c→k indicates that X is acting as a Client and Y is acting as a KAS, t→k for TGS and KAS and s→t for application server and TGS.

In the first stage, the Client (C) generates a nonce (represented by new n1) and sends it to the KAS (K) along with the identities of the TGS (T) and itself. The KAS generates a new nonce (AKey - Authentication Key) [115] to be utilised as a session key between the Client and the TGS. It then sends this key along with some other fields to the client encrypted under two different keys- one it shares with the Client ($k^{c\rightarrow k}_{C,K}$) and one it shares with the TGS($k^{t\rightarrow k}_{T,K}$). The encryption with $k^{t\rightarrow k}_{T,K}$ is called the Ticket Granting Ticket (tgt). The Client extracts AKey by decrypting the component encrypted with $k^{c\rightarrow k}_{C,K}$ and recovering its parts using the match action which deconstructs $text_{kc}$ and associates the parts of this plaintext with AKey, $\eta$1, and T^. The ellipses (...) indicates further Client steps for interacting with KAS, TGS.

In the second stage, the Client gets a new session key (SKey - Service Key) and a service ticket (st) to converse with the application server S which takes place in the third stage. The control flow of Kerberos exhibits a staged architecture where once one stage has been completed successfully, the subsequent stages can be performed multiple times, or aborted and started over for handling errors.

### 5.4.3 Formal Description of UDT + GSS-API in the Formal Language

Table 5-8: Formal Description of UDT + GSS-API

| | |
|---|---|
| **Client** = (C; ^K ; ^ T; ^ S; t) [ | **KAS** = (K) [ |
| new n1; | receive ^ C: ^ T:n1; |
| send ^ C: ^ T:n1; | new AKey; |
| receive ^ C:tgt:enckc; | tgt := symenc AKey: ^ C; kt!k |
| textkc := symdec enckc; kc!k | T;K ; |
| C;K ; | enckc := symenc AKey:n1: ^ T; kc!k |
| match textkc as AKey:n1: ^ T; | C;K ; |
| _ _ _ stage boundary _ _ _ | send ^ C:tgt:enckc; |
| new n2; | ]K |
| encct := symenc ^ C;AKey; | **TGS** = (T; ^K) [ |
| send tgt:encct: ^ C: ^ S; n2; | receive tgt:encct: ^ C: ^ S:n2; |
| receive ^ C:st:enctc; | texttgt := symdec tgt; kt!k |
| texttc := symdec enctc;AKey; | T;K ; |
| match texttc as SKey:n2: ^ S; | match texttgt as AKey: ^ C; |
| _ _ _ stage boundary _ _ _ | textct := symdec encct;AKey; |
| enccs := symenc ^ C:t; SKey; | match textct as ^ C; |
| send st:enccs; | new SKey; |
| receive encsc; | st := symenc SKey: ^ C; ks!t |
| textsc := symdec encsc; SKey; | S;T ; |
| match textsc as t; | enctc := symenc SKey:n2: ^ S;AKey; |
| ]C | send ^ C:st:enctc; |
| | ]T |
| | **Server** = (S; ^ T) [ |
| | receive st:enccs; |
| | textst := symdec st; ks!t |
| | S;T ; |

| | |
|---|---|
| | `match textst as SKey: ^ C;`<br><br>`textcs := symdec enccs; SKey;`<br><br>`match textcs as ^ C:t;`<br><br>`encsc := symenc t; SKey;`<br><br>`send encsc;`<br><br>`]S` |

### 5.4.4 GSS-API Kerberos Properties and Operating Environment

The security objectives of proving Kerberos are of two types: authentication and secrecy. The authentication objectives take the form that a message of a certain format was indeed sent by some thread of the expected principal. The secrecy objectives achieve the form that a putative secret is a good key for certain principals. For example, $AUTH^{client}_{kas}$ outlines that when C completes executing the Client role, some thread of K^ indeed sent the expected message; $SEC^{client}_{akey}$ outlines that the authorisation key is good after execution of the Client role by C; the other security properties are related.

The proof of Kerberos Security properties clearly underscores and demonstrates an interleaving of authentication and secrecy properties, reflecting the institution behind the proposed mechanism.

## 5.5 Concluding Remarks

In this chapter, 3 mechanisms were selected and analysed: the UDT-AO, UDT+DTLS and UDT+GSS-API, Kerberos [22-33,39-40,110] authentication protocols.

We found a few anomalies that are widely found in authentication protocols [72,73]. Similarly, we raised the issues of a repairable DoS attack, an anomaly in the derivation of the master key MKey, and a potential algocrypt or in some cases simple cipher suite downgrading attack. While the third anomaly is unavoidable, proper awareness of an attacker's ability to weaken algocryptList in *Message 1* and provision of appropriate measures to address it should prevent problems from arising. We found that by flooding the network with fake *Message 1*'s, an attacker can force a peer to re-compute the MAC key SKey, causing the peer to be unable to correctly process *Message 3* from a legitimate server. This attack is especially problematic because UDT-AO [22-33] is designed to work on devices such as routers with easily exhaustible limited memory. To minimise the chance of a DoS attack, we propose a fix that allows the peer to maintain state per connection and session, instead of state per message [43,142]. We also identified an anomaly in the derivation of the master key MKey, similar to the findings of [48-50]. Specifically, MKey was derived using a KDF; most of the key derivation is assigned with initial constant key value, though this does not provide an obvious way for an attacker to reliably learn session keys [1,2,22-33,44], but it is better to use a more standard implementation, such as key reset and variable assignment of initial KDF [110].

In case of DTLS, we identified the secrecy and authentication properties of this mechanism viable with UDT through the use of sequence numbers. Properties of secrecy, however, rely on how data are transmitted based on a presumed secrecy of long-tem shared symmetric keys [1,2,22-33,52,84].

In Kerberos, the secrecy of encryption keys [18,39-40,45,110,121] allows the establishment of authentication, which is achieved by virtue of ciphertext integrity offered by the symmetric encryption scheme. Similarly, it can be understood that a ciphertext could have been produced only by one of the possessors of the corresponding key.

The execution of Client role within UDT environment by a principal is guaranteed, because there is an asymptotically overwhelming probability and that the intended ticket sent the expected response, assuming that the client is trustworthy before the data are transmitted through UDT.

The theoretic and discussed proofs of secrecy and authentication of UDT-AO, UDT+DTLS, and UDT+GSS-API, Kerberos [22-33] demonstrate they are useful mechanisms for UDT, provided that appropriate techniques are supplemented with extensive practical validations in UDT implementations.

# Chapter 6

# High Speed Data Transfer Security Architecture

The primary objective of the architecture includes the management of messages through the proposed security mechanisms and cryptographic keys, the security of data communications, and the integration of data protection enhancing technologies. Our approach is based on the results of our work which formulated on the enhancement of existing schemes to create a novel approach to secure UDT. They rely upon well-discussed schemes that can be upgraded to provide improved security and primary protection in future extensive UDT deployments.

Essentially, the idea is to use a common baseline design that, on one hand, provides a sufficient level of protection of data and communication, but which, on the other hand, is deemed practical and deployable to UDT and to other similar protocols. The design will rely on well-established and understood cryptographic primitives, which are fully scrutinised, thus sufficiently trusted and implementable in various environments.

## 6.1 Framework Objectives

The goals our architecture seek to achieve include: using a cryptographic key management for messages, privacy and data integrity, and secure communication. The architecture focuses on the upper layers, from IP through to the application layers of the stack. In addition, our architecture will also seek to address the basic fundamental design of UDT and to introduce a security mechanism that is not available at the packet level.

The design is partially drawn from the architecture of TCP and UDP. We introduce a proprietary design intended for UDT basing on the foundation laid by Gu [82]. The design is novel and applicable to the future design of UDP-reliant protocols.

### 6.1.1 Milestone

We present a practical security architecture for UDT that is also applicable to other high-speed data transfer protocols. We also describe scalable mechanisms to achieve the desired protection. To develop this architecture, extensive reviews and validations were conducted, as well as implementations of existing mechanisms on UDT. The results were presented in the preceding chapters.

The architecture is useful in several ways. It is presented with supplemental information on the schemes, which can provide a basis for basic, if not comprehensive, security of data flow — specifically in the higher-level communication layers.

### 6.1.2 Summary of Work

The importance of studying UDT is highlighted owing to two reasons: firstly, it has potential commercial promise; secondly, it is one of the Fastest Data Transfer protocol available but has no security to protect data transfer.

We introduce an approach securing UDT-implementations in various layers [22-33]. However, securing UDT in terms of both application and other layers needs to be further explored in future UDT deployments in various applications. It is important to note that there are applications, transport layer-based authentications and end-to-end security options for UDT.

In this chapter, we discuss the results of our work by presenting an experimentally validated framework to secure UDT. There are five important areas which the framework highlights:

- Security at the application and session layers via UDT extensions require client and servers, and significant changes to applications to accommodate security features;

- Security on the layers 2-3. The encryption is performed, and abstracted from the UDT application, eg., via gateway-to-gateway, Virtual Private Networks (VPNs), when security on the application layer becomes too complex to develop;

- Other mechanisms that are available - such as IPSec- can protect data traffic. However, the development of specific mechanisms for UDT minimises reliance on mechanisms that can affect performance and add overhead and complexity in UDT implementations;

- Introduction of viable mechanisms UDT-AO, UDT+DTLS, and UDT+GSS-API or UDT-Kerberos. These mechanisms achieved proof of correctness and are therefore suitable for UDT;

- The inclusion of all existing mechanisms for the data flow within the architecture provides an extensive analysis of other mechanisms that can be implemented with UDT.

## 6.2 Architecture

Based on the schemes reviewed, the following layer-to-layer architecture is presented for UDT.

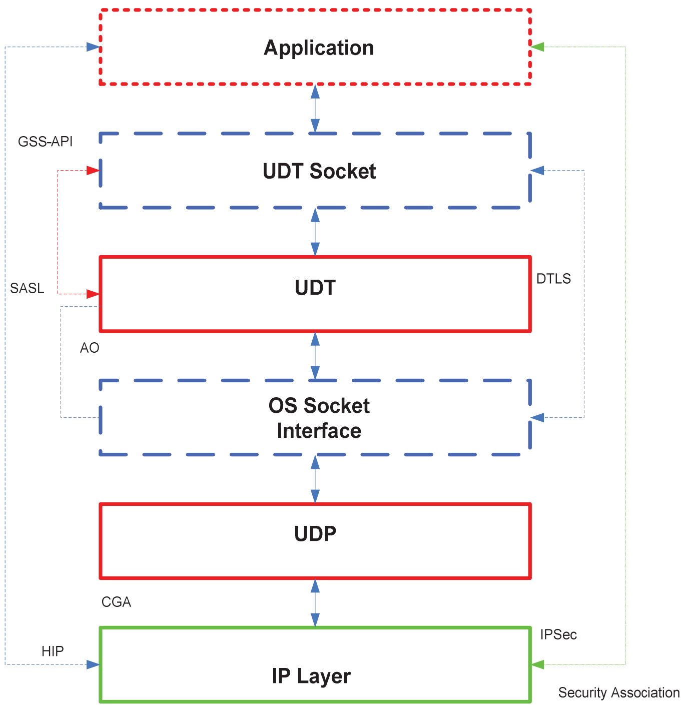


Figure 6-1. Layer-to-Layer Architecture. In this architecture, the UDT layer provides transport functionalities to applications. The security schemes that can be implemented on this layer are DTLS and SASL for the upper layer. The layer above using UDT Socket can be implemented with GSS-API. UDT, however, can implement AO at the transport layer. The remaining lower layer can be protected using IPSec (securing end-to-end), HIP through the Application layer. CGA is specifically implemented on the IP layer provided HIP is not binding to the UDT socket.

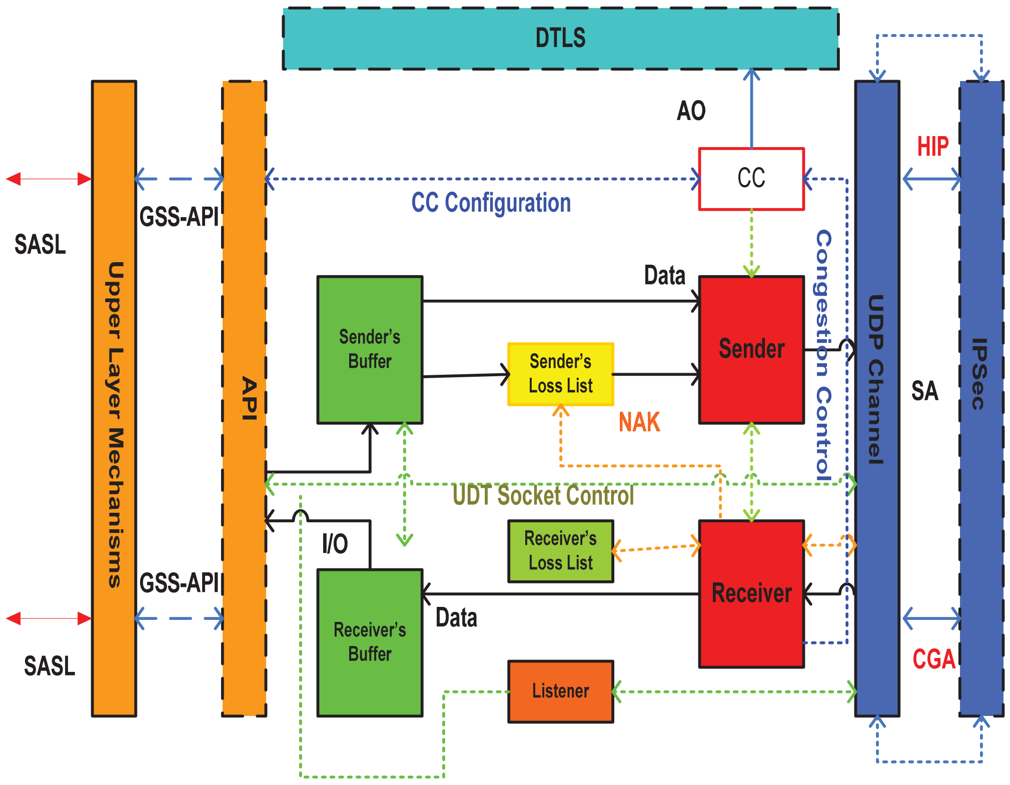


Figure 6-2: Proposed UDT Security Architecture

From Figure 6-1, the detailed data flows with the proposed mechanisms are presented in Figure 6-2. With the introduction of IPSec, GSS-API, SA, SASL (a standard mechanism required to manage secret encryption and authorisation keys), a generic key management API is proposed, which can be used for IPSec and other existing security services. Similar to using sockets, this specific API creates a new protocol family — the PF_Key domain. This must be constant and must be used with key management sockets, according to RFC 2367.

It is important to note that IPsec provides services to packets based on the Security Association (SA) [20,22-33], which is stored for use in the Security Association Database (SADB) [21-33]. This can be used for other routing protocols.

Key operations are supported on key management sockets, such as:

1. UDT can request a key from a key management daemon. The process /application that uses UDT can send a message to the kernel with open key management sockets by writing to a key management socket.

2. A process can read a message from the kernel (that UDT operates). The kernel uses this facility to request that a key management daemon install an SA for a new UDT connection.

The security architecture has been introduced following an extensive review, which included a design for a security-specific modular structure for the UDT protocol, which is also practical for other data transfer protocols.

## 6.3 Synopsis

We have presented our methods by using existing security mechanisms and developed one for UDT with the specifications of TCP, UDP, and Sockets API. Our design is the result of extensive experiments and implementations, and – using existing high speed appliances to support our schemes – has practical application.

The implementation of this architecture is not limited to UDT. It is designed to be adaptable and to work with other fast data transfer protocols.

Special consideration on different layers of the stacks and the introduction of either proprietary or commercially designed types to meet stringent security requirements can achieve enhanced security.

## 6.4 Symbolic Analysis of Proposed UDT Security Architecture

In this section, we introduce an approach using rewrite based systems and automata in order to specify and analyse the selected security mechanisms and data flow. We outline and employ this approach to verify more effectively our proposed architecture.

This approach is closely replicating dataflow to allow a real representation of the implementation of selected mechanisms integrated into the architecture. We consider this approach effective in corresponding with the properties of the rewrite systems; the specifications of the architecture to utilise theoretical but proven approach to perform the analysis.

There are not many verification tools available and specifically tailored to analyse our proposed architecture and its underlying components. We state abstract representation of the components that compose the architecture and conduct our analysis, through extensive analyses. These analyses will examine the relationship between components within the architecture. These will also examine the possibility of issues (i.e., design anomalies, adversarial attacks). We employ these analyses as a way to avoid or mitigate successful attacks.

The use of automata to achieve decidability in rewrite systems will highlight an important class of reduction automata which is closed under union and intersection. With each reduction automaton we can associate a complete reduction automaton that accepts the same language. This construction preserves determinism. The class of complete deterministic reduction automata is closed under complement. These important properties assist us in determining the data flow confluence of the security mechanisms analysed through the rewrite systems.

The possible inputs and data flow on which these mechanisms can work are determined through the use of the above approaches. They can assist us analyse the strengths and weaknesses of the proposed mechanisms. We then arrive at what the results which lead to conclude to be the most feasible mechanisms possible.  When we do, we shall expect to determine tasks that even it cannot perform. This will be our ultimate result, that no matter what mechanisms and architecture we build, there will always be questions that are simple to state that our approaches can either accept or reject. Along the way, we attempt to evaluate some of the issues, such as optimality through practical simulations and experimentations in a real environment.

As we conduct important analyses: structural, semantic, and query analyses. Structural and semantic analyses examine the relationships that the mechanisms have with other mechanisms, and examine the decision processes within the architecture. Query analysis on the other hand provides analysis such where does the data flow across the UDT architecture come from and from which a security mechanism it passes through. In this work, we concentrate on tracing the data flow to ensure that anomalies can be detected.

## 6.5 Approach

A brief description of the basic notions related to the terms ‘algebra’ (i.e., terms, substitutions, positions), ‘rewriting systems’ (i.e., reduction relation, confluence, termination) [49, 64] and ‘tree automata’ is discussed. In the succeeding sections of this chapter, we discuss and outline some basic notions that we used and. We also introduce the notion of constrained rewrite systems. The main interests in the sections are based on formal languages and rewrite based systems.

### 6.5.1 Term Algebra

Many sorted signatures of the form (F, S) consisting of a set of sorts S and a set of function symbols F have been considered. Symbols of F are denoted by bold characters **f**, **g**, and so on, and their profiles are denoted as follows **f** : s1 x…x sn →s where s1,…,s are sorts of S and n is the arity of **f**. The set of terms of sort s built [49,64,71] out of symbols from F and of sorted variables from a set $\mathcal{X}$ is denoted by $\mathcal{T}_{\mathcal{X}}^{s}$ and the set of ground terms of sort s is denoted by $\check{\mathcal{T}}^{s}$. For any $t \in \mathcal{T}_{\mathcal{X}} = \cup_{s \in S} \mathcal{T}_{\mathcal{X}}^{s}$, $\mathcal{V}ar(t)$ denotes the variables occurring in t. If any variable of t occurs only once in t, then t is said to linear. A position within t is a sequence $\omega$ of integers describing the path from the root of t (seen as a finite labelled tree) to the root of the subterm at that position, denoted by $t_{|\omega}$. We use $\varepsilon$ for the empty sequence $\omega$ . | $\omega$ | is the length of the position. Pos(t) denotes the set of position of t. t($\omega$) is the symbol of t at position $\omega$ and tt[s]$_{w}$ the term t with the subterm at position $\omega$ replaced by s. A substitution $\sigma$ is a mapping from $\mathcal{X}$ to $\bar{\mathcal{T}}_{\mathcal{X}}$ which is the identity except over a finite set of variables (its domain) and which is extended to an endomorphism of $\bar{\mathcal{T}}_{\mathcal{X}}$. A substitution is said ground if all the variables of its domain are mapped to ground terms. A term t matches a term t’ iff $\sigma$ (t’)=t for some substitution $\sigma$. Two terms t and t’ are unifiable *iff* $\sigma$ (t’) = $\sigma$ (t) from some substitution $\sigma$ [42, 49-50, 63-64, 78, 104,141].

## 6.5.2 Tree Automata

A tree automaton is a triple $A = (Q, Q_F, \Delta)$ where Q is a finite set of symbols called states disjoint from F, $Q_F \subseteq Q$ is the set of final states and $\Delta$ is a finite set of transitions of the form f (q1,…,qn) $\rightarrow_\Delta$ where q1,…,qn, q $\in Q$ and n is the arity of f .$\rightarrow_\Delta$is extended to $\rightarrow_\Delta$* as follows [49-50,64,104,141]: if if $\forall i, t_i \rightarrow^*_\Delta q_i$ and f (q1,…,qn)$\rightarrow_\Delta$ q, then f (t1,…,tn)$\rightarrow^*_\Delta$ q. The language recognised by $A = (Q, Q_F, \Delta)$ is $\mathcal{L}(A) = \{t \in \mathcal{T} \mid \exists q \in Q_F, t \rightarrow^*_\Delta q\}$. A set (or a language) of terms recognised by a tree automaton is said regular. A relation R is regular if there exists an automaton recognising $\{\tilde{t} \mid t \in R\}$ where for any t = (t1,…,tn) and $\omega \in \cup_i \mathcal{P}os(t_i)$, $\tilde{t}(\omega) = (t_1[\omega], \ldots, t_n[\omega])$ with $t_i[\omega) = t_i(\omega)$ if $\omega \in \mathcal{P}os(t_i)$ and the special symbol $\Lambda$ otherwise. Boolean operations, Cartesian product, projection and cylindrification preserve regularity. It can be stated that a set or a relation is effectively regular *iff* it is regular and can compute an automaton which recognises it.

## 6.5.3 Rewrite Systems

A rewrite rule is a pair of terms *l* → *r*. The terms *l* and *r* are respectively called the left-hand side and right-hand side of the rule. A rewrite system [139,142] R is a finite set of rewrite rules. Any rewrite system R induces binary relation over terms denoted by →R as follows: for any terms t, t', t →R t' if there exist a rule *l* → *r* of R, $\omega \in \mathcal{P}os(t)$ and a substitution $\sigma$ such that $t|_\omega = \sigma(l)$ and $t' = t[\sigma(r)]_\omega$. A rewrite rule is linear iff its left-hand side and right-hand side are linear. A rewrite system is linear if all its rules are linear. A Growing Rewrite System (GRS) [42, 50, 97, 104, 141] is a linear rewrite system such that for every rule *l*→*r*, $l(\omega) = r(\omega') \in \mathcal{X}$ for some positions $\omega, \omega'$ then $|\omega| \leq 1$.

### 6.5.4 Extension to Rewrite Systems

Without restriction, we can consider that any GRS (Growing Rewrite Systems) is a set of rules of one of the following forms (extension of Jacquemard's [49, 95]):

$$\begin{array}{rcll} x \rightarrow r[x] & \| & x \in A & (1) \\ f(x_1,\ldots,x_n) \rightarrow r[x_1,\ldots,x_n] & \| & x_1 \in A_1,\ldots,x_n \in A_n & (2) \end{array}$$

Knowing that any unconstrained variable $x$ can be seen as a variable constrained by $x \in$ A where A is the automaton recognising all ground terms. Thus, we consider that any variable is constrained.

Let be $L$ a regular language recognised by A$L$ and R a CGRS. The automaton recognising $(\rightarrow_{\mathrm{R}}^{*})^{-1}(L)$ is built as follows:

$$\mathcal{R}each_0 = (Q, Q_0, \Delta_0) := \biguplus_{(l \rightarrow r \| C) \in \mathrm{R}} \left( \biguplus_{(x \in A) \in C} A \right) \uplus A_L$$

where the disjoint sum $(\uplus)$ of two automata over the same signature is the automaton whose set of states, set of final states and set of rules are the union of corresponding sets of the two automata, provided that they are all disjoint [95]. Then, we transform $\Delta_k$ $(k \geq 0)$ into $\Delta_{k+1}$ by applying the following rules:

$$(i) \ \frac{\left\{ \begin{array}{l} \left(f(x_1,\ldots,x_n) \rightarrow g(r_1,\ldots,r_m) \| \bigwedge_i x_i \in A_i\right) \in \mathrm{R} \\ g(q_1,\ldots,q_m) \rightarrow q \in \Delta_k \end{array} \right.}{f(q'_1,\ldots,q'_n) \rightarrow q \in \Delta_{k+1}}$$

with the conditions:

1. for all 1 $\leq$ $i \leq n$, $q_i'$ is a final state of A$i$

2. for all 1 $\leq$ $j \leq$ $m$, there exists a substitution $\theta : \mathcal{X} \to Q$ such that $\theta(r_j) \xrightarrow{*}_{\Delta_k} q_j$ and for each $x_i$ occurring in $g(r1,\ldots,rm)$, we have _ $\theta(x_i) = q_i'$.

$$(ii)\ \frac{\left(f(x_1,\ldots,x_n) \to x \| \bigwedge_i x_i \in A_i\right) \in \mathrm{R}\ ;\ \ q \in Q}{f(q_1',\ldots,q_n') \to q \in \Delta_{k+1}}$$

with the conditions:
1. $x = x_i$ for some 1 $\leq i \leq n$
2. for all 1 $\leq$ $i \leq n$, if $x_i = x$ then $q_i' = q$, otherwise $q_i'$ is a final state of A$i$.

The desired automaton is $\mathcal{R}each = (Q, Q_L, \Delta)$ where $Q_{\mathcal{L}}$ is the set of final states of A$\mathcal{L}$.

An ordered rewrite system is a rewrite system in which rules are ordered. For an ordered rewrite system R, → R is defined as follows: for any terms t, t', t→R t' if there exists a rule $l$ → $r$ of R, $\omega \in \mathcal{P}os(t)$ and a substitution $\sigma$ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_{\omega}$, and such that there is no prior rule $l'$→$r'$ such that $t_{|\omega'} = \sigma'(l')$ for some $\omega'$ and $\sigma'$ [64, 95, 104].

A constrained rewrite system (CRS) is a rewrite system [42, 50] that defines a constrained such that for every rule $l$ → $r$ is associated to a set of membership constraints x $\in$ A where x is in Var($l$) and A is a regular tree language. If l → r is associated to $\{x_1 \in A_1, \ldots, x_n \in A_n\}$, we write $l$ → $r$ | | x1 $\in$ A1,…,xn $\in$ An. The binary relation → R induced by a constrained rewrite system R is de_ned as follows: for any terms t, t', t →R t' iff_there exists a rule l →r | | x1 $\in$ A1,…,xn $\in$ An of R, $\omega \in \mathcal{P}os(t)$ and a substitution $\sigma'$ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_{\omega}$, and such that $\sigma(x_i) \in A_i$ for every i.

Given a rewrite system R, $\rightarrow_R^*$ denotes the reflexive transitive closure of the relation induced by R. For any term v, $\rightarrow_R^{-1}$ (v) denotes the set $\{u \mid u \rightarrow_R v\}$. For any set of ground terms $\mathcal{U} \subseteq \mathcal{T}$, $\rightarrow_R^{-1} (\mathcal{U})$ denotes the set $\{u \mid \exists v \in \mathcal{U}, u \rightarrow_R v\}$. A rewrite system R is confluent iff for any terms *u*,w, v, if *u* $\rightarrow_R^*$v and *u* $\rightarrow_R^*$ w, then there exists t such that v$\rightarrow_R^*$ t and w$\rightarrow_R^*$ t. *u* is irreducible w.r.t R iff there is no v such that *u* $\rightarrow$R v. If *u* $\rightarrow_R$ v and v is irreducible w.r.t R, then v is a normal form of *u.*

For any linear (constrained or not) rewrite system R and rule r of R, it is denoted by rec(r) the regular set of ground terms that are reducible by r. If R is an ordered rewrite systems, it is denoted by rec(r/R) the set of terms that are reducible by r and by no rule prior to r in R.

## 6.6 Formalisation

In this section, we formalise the proposed architecture based on specified rewrite rules. We exhibit the flow as rewrite rules. These specifications are available in the flow of data through the proposed security mechanisms.

Formalisation achieves validation without system overload. The theoretical proofs of the mechanisms operating within the proposed architecture are adequate in this work to substantiate the correct data flow within compositional layers of the mechanisms. These either functional in isolation or in a group with other mechanisms defined in the design.

**Example 1**. We base our example on Figure 6-3. Data flow from either $\leftarrow$left to $\rightarrow$right, or up to down, and vice versa. Mechanisms put in place monitors the flow and accept and exit flows. Reject flows only occur if an anomaly is detected in either in the flow-source or flow-destinations within the data flow of UDT.

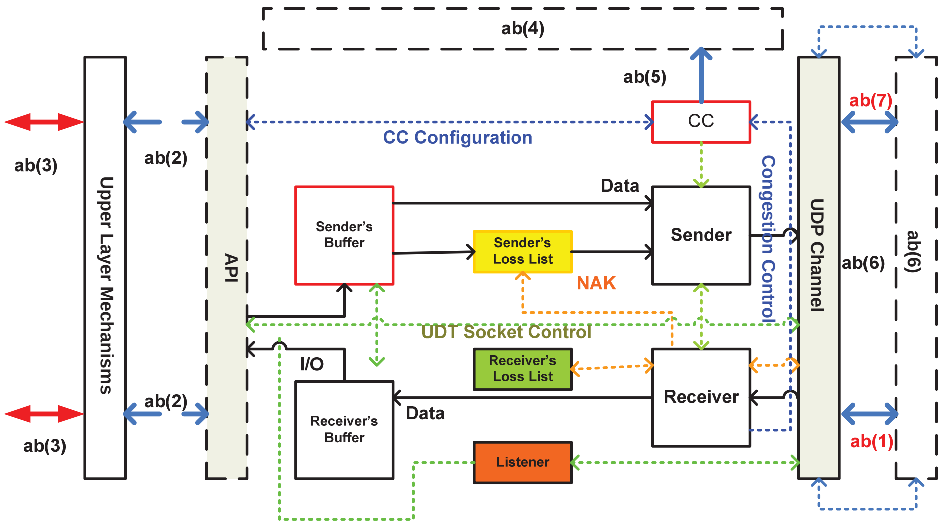


Figure 6-3: Data Flow through mechanisms (m) with in architecture.

## 6.6.1 Data Flow

In our approach, data flow is demonstrated as an algebraic term. The selected symbolic representation of data flow is based on the following signatures:

*Note: (from(ab,x),dest(ab,x)) represents a dataflow (from(ab(1),x),dest(ab(1),x')) to the set of data whose flow-source comes from a security mechanism ab# and flow-destination exits*

```
Left, right:   data → data   ab(#)data flow from left to right
Right, left:   data → data   ab(#)data flow from right to left
Up, down    :  data → data   ab(#)data flow from up to down (as above Figure 6-3)
Down, up    :  data → data   ab(#)data flow from down to up (as above Figure 6-3)
#           :         → data   value of # such that a(1)…a(7)
From        :  data x data  → flow-source
Dest        :  data x data  → flow-destination
Dataflow    : flow-source x flow-destination → Dataflow (as above Figure 6-3)
a,b         :  mechanism → mechanism (as above Figure 6-3)
```

We label the mechanisms as ***ab(#),*** such that term ***t=ab(#)*** and where ***#*** is a value.

**mechanisms** : *ab(1)* CGA, *ab(2)* GSS-API, *ab(3)* SASL, *ab(4)* UDT+DTLS, *ab(5)* UDT-AO,*ab(6)* SA, *ab(7)* HIP

There are various possibilities to describe how the data flow passes through the mechanisms. The flow, however, only relies on the given flow source. To distinguish which mechanisms are used, we represent these as words over {a,b} and a value *# (1,2..n).* We restrict our representation of data as dataflow, flow-source, flow-destination, and # to specific mechanisms we proposed.

For example the term *t=ab(#).* # identifies as a mechanism; example *ab(1)* is assigned to a specific mechanism CGA. This representation allows us to build a tree automata that recognises data flow and mechanisms to analyse the architecture as a state.

Data are terms of sort Dataflow for example, the term dataflow *(from(ab,x),dest(ab,x))* represents a dataflow *(from(ab(1),x),dest(ab(1),x'))* to the set of data whose flow-source comes from a security mechanism ab# and flow-destination exits to the same mechanism is a given flow.

In the preceding section, we define how the flow can be secured across *ab(#)* mechanisms (using the symbols flow-source and flow-destination), thus allowing us to encode and assume a symbolic attack by appropriate tree automata, which we use for the analysis.

## 6.6.2 Architecture Flow

Furthermore, to extend the data flow in the proposed architecture, the following symbols are added.

*Enter or accept, Exit or reject : → Decision*

From a rewriting point of view, the flow rewrites a dataflow into enter, exit or reject.

**Definition 1 (Mechanism)**. A mechanism is composed of ordered rewrite systems *Prem, Processm,* and *Exitm* such that:

- rules of *Prem* are of the form of p→ d where p is a linear term of sort Dataflow and d a (ground) term of sort Decision;
- rules of *Prem* and *Exitm* are, respectively, of the form:

```
From(mechanism, x) → From(mechanism', x')
Dest(mechanism, x) → Dest(mechanism', x')
```

*Where mechanism, x are linear terms and mechanism', x' are ground terms.*

**Example 2**. The mechanism described in Example 1 can be specified as follows:

$$\left\{ \text{Dataflow} \left( \begin{array}{ll} \text{From} & (ab[x],data) \\ \text{Dest} & (ab[y],z) \end{array} \right) \begin{array}{c} \rightarrow \\ \text{accept} \end{array} \right.$$

$$\textit{Dataflow} \quad (x,y) \rightarrow \textit{exit}$$

**Definition 2 (Semantics)**. For any mechanism *[1...n]*, its semantics is denoted by [m] and defined as follows:

*[m] = [m]accept U [m]exit OR [m] reject*

With R:{x→x} is the rewrite system R in which the rule x→x has been added as the last rule.

$[m]\ \text{accept} = \{(t,u) \in T^{dataflow} \times T^{dataflow} \mid \exists\, v \in T^{dataflow},\ t \rightarrow_{Prem;\{x \rightarrow x\}} v \rightarrow_{Processm} \textbf{enter} \wedge v \rightarrow_{Exitm;\{x \rightarrow x\}} u\}$

$[m]\ exit = \{(t, \mathbf{exit}) \in T^{dataflow} \times T^{Decision} \mid \exists\ v \in T^{dataflow},\ t \rightarrow_{Prem;\{x \rightarrow x\}} u \rightarrow_{Processm} \mathbf{exit}\}$

From an abstract point of view, a mechanism can be distinguished as a partial or total function which takes an input (data / dataflow) and returns either exit/reject.

## 6.7 Analysis of the Architecture

It can be observed that this rewrite specification not only allows automatic checking of properties of semantics of a mechanism, as part of the overall architecture; it also permits structural and query analysis on the architecture itself.

### 6.7.1 Semantics Analysis

A mechanism can be observed as a decision process that allows dataflow with data that can be either accepted or rejected. Therefore, the following properties require verification: *consistency,* which indicates that at most, one decision is taken for a given data flow*;* *termination,* which ensures that a mechanism computes a decision in a finite time; and *completeness,* which signifies that for any dataflow, the mechanism returns a decision.

By construction, any mechanism that denotes a terminating and consistent decision process is a function. Completeness can be therefore defined as follows:

**Definition 3 (Completeness)**. It states that a mechanism (m) as part of the whole architecture (a), is a complete *iff* [m], which is a total function.

The particular shape of the rules defining a mechanism allows it to represent the semantics of a mechanism as a regular relation and to verify its completeness.

**Proposition 1**. *Completeness is decidable.*

**Proof.** The proof relies on the regularity of the relations involved in the definition of the semantics of a mechanism as part of the architecture. Since the left-hand sides of all the rewrite rules composing a particular mechanism are linear and share no variable with their right-hand sides, we can show that $\rightarrow_{Prem;\{}$and $\rightarrow_{Exitm;\{}$are regular trees relations. Since the identity is a regular relation, it follows that $_{Prem;\{ x \rightarrow x \}}$ and $_{Exitm;\{ x \rightarrow x \}}$ are also regular. By composition and restriction, we obtain that $[m]^{accept}$ and $[m]^{exit/reject}$ are regular tree (functional) relations. Subsequently, *[m]* is a regular tree (functional) relation. The completeness can be tested by checking that the first projection of *[m]* covers the (regular) set of all possible incoming data transported by the dataflow.

In case of a complete architecture - yet selecting only the applicable mechanisms in either in isolation or in compositional groups, where these are used at the same time, but independently operate on a given speed, it is important to determine if the chosen mechanism is applicably stronger than the others. This can be accomplished, through verification, based on specifications of the mechanisms.

**Definition 4 (Order).** A partial order can be defined over complete mechanism within a complete architecture as follows: for any *m* and $m', m \preceq m'$ (*m'* is more permissive than *m*) *iff* $[m]^{accept} \subseteq [m']^{accept}$. We write $m \approx m'$ *iff* $m \preceq m'$ and $m' \preceq m$.

A mechanism *m'* is thus permissive than a mechanism **m** if it accepts all data flow that **m** accepts. Note that $m \preceq m'$ *iff* $[m]=[m']$.

For the same reasons, it can decided whether a mechanism is more or less permissive than the other within the architecture.

**Proposition 2.** *The order relation* $\preceq$ *is decidable.*

**Proof.** As it has already been shown, for any mechanism *[m], [m]*$^{accept/enter}$ and *[m]*$^{exit/reject}$ and *[m]* are regular relation. Consequently, the inclusion *[m]*$^{accept/enter} \subseteq$ *[m']*$^{accept/enter}$ is decidable. Note that two mechanisms may have the same semantics even if their rules are different. This is particularly interesting since it allows to simplify and optimise the flows of a mechanism and check if the succeeding mechanism has the same semantics as the preceding one.

### 6.7.2 Structural Analysis

Structural analysis refers to the detection of so-called anomalies [64] in the implementations of a particular mechanism. These anomalies are looked at as properties expressed as relationships between the rules of mechanisms *(rm)* within the proposed architecture. Examples of anomalies are superseding (an *rm* leads to decisions contradictory to decisions of prior rules of prior mechanisms), redundancy (an *rm* can be removed without any impact to other mechanisms and data flow), and generalisation (an *rm* matches a superset of the set of data matched by a prior *rm* with a different decision). It should be mentioned that, while several approaches have been developed for the detection of the above anomalies, these approaches are often intentionally introduced in order to obtain more compact or more effective *rm* sets. Detecting anomalies is still interesting since it can outline some mitigation. Only discussed here is the approach for detecting superseding; the other anomalies can be treated in a similar way. Recall the definition of the superseding anomaly in this context: it can be said that a mechanism superseding *iff* it contains at least one rule, such that all dataflow it allows and accepts are rejected by a prior *rm*. In such a case, the concerned *rm* is said to be superseded.

The detection of the superseded *rms*, as well as of the other anomalies, is based on the regularity of the sets of terms associated to a given *rm*. More precisely, each rule *rm =r* is associated to several sets: *rec(r),* denoting the set of data matching *r; rec(r=Process$_m$)*, denoting the set of data matching r that match no S prior rule of Process$_m$ (i.e. *rec(r)*\ $\bigcup_{r'<r}$ *rec(r'))* and *rec(r=Process$_m$ [d])* denoting the set of data matching r that match no other rule of *Process$_m$* associated to the decision d. Since the left-hand sides of the processing rules are linear terms, all the sets *rec(r)* are regular; the other sets are also regular since they can be built starting from *rec(r)* and using operations that preserve regularity. Anomalies can then be detected using inclusion or emptiness tests. For example, to detect if a rule *r* is superseded, it suffices to check the emptiness of *rec(r=Process$_m$[accept/enter])* if the right-hand side of *r* is dropped while the emptiness of *rec(r= Process$_m$[exit/reject])* is not dropped.

It is well-known that operations over tree automata are highly complex. In this case, the complexity of the needed operations strongly depends on the representation of data and, in particular, on the representation of dataflow. The choice of dataflow as words over *{a,b}* or *{0,1}* (or equivalently as terms built from the monadic symbols 0 and 1, a and b, and the constant #) is indeed made in order to obtain efficient implementations of the corresponding automata operations.

To simplify, consider the word automata; the correspondence with tree automata is straightforward. Due to the representation of ab(i…n) ranges, we are confronted with n-prefix (or simply prefix) languages, i.e. regular languages of the form $\alpha_1.\{0,1\}^* \cup \ldots \cup \alpha_n.\{0,1\}^*$ or *(a,b}**. A good property of the manipulated ranges is that corresponding minimal and deterministic automata have no loop except at their unique final state, which loops over itself for any word.

Moreover, as discussed, the sets of dataflow of a given mechanism are 1-prefix. It follows that *rec(r), rec(r=Process$_m$),…,* are prefix languages. Consequently, anomalies can be efficiently detected using our approach.

Query analysis is another kind of analysis attempted. This kind of analysis has assisted us in understanding the behaviour of a mechanism within the architecture through deriving outputs based on pre-determined and pre-defined queries, such as “which mechanism will receive the data flowing from left to the right side of the architecture?” It was introduced earlier the semantics of the mechanisms and the architecture as regular relation.

## 6.8 Concluding Remarks

In verifying our proposed architecture, we use symbolic analysis of each mechanism within the edge of UDT data flow. Then we analyse the overall architecture with the mechanism’s data flow.

By using this approach, we show the viability of these mechanisms for this architecture. We describe the architecture using structural and query analyses, and automata, aside from using rewrite systems, which interpret relevant properties and semantics through classical theoretical and practical way. We show that these approaches and analyses highlight any foreseen anomalies that require decision processes, thereby assisting in solving and mitigating risks.  Mitigations may include additional security mechanisms, proprietary layer 7 and layer 3-4 devices and so on. We use formalisation to understand the flow of data from and within the architecture.

The approach we use allows us to address implementation scenarios from theoretical and practical point of views. Using automata operations and performing basic practical applications and implementations, as described in the preceding chapters, allow us to understand and address any issues in the mechanisms underlying the security architecture. We find that rewrite systems, combined with structural and query analyses, effectively highlight any vulnerabilities and anomalies in the composition of a viable UDT security architecture.

# Chapter 7

# Conclusion and Scope for Future Work

The final chapter of this dissertation summarises and assesses the original contributions of this work. This final section describes the continuing security evaluation of UDT and continuing improvement of the proposed architecture, and presents the scope for future work.

## 7.1 Summary

In Chapter 1, we presented concepts and hypotheses of this work. We introduced the specific network protocol of interest and discussed its security limitations. In Chapter 2, we outlined the existing literature written and published about the network protocol. In Chapter 3, we proposed security mechanisms, and modified important variables in the UDT structure itself by adjusting the values of MSS, header, and size, in various scenarios to facilitate performance evaluation within both secure and unsecure environments. In Chapter 4, we presented and described the design and implementation of a unique visualisation tool—Project UDT—which constitutes a multi-faceted tool to assist in UDT analysis. Then we presented our experiments that aimed to derive practical verifications of our mechanisms.

By using readily available security devices to build an infrastructure for conducting experiments, we examined activity at the given protocol level with an accurate knowledge of events at other levels.

In Chapter 5, we used information-theoretic and proof of correctness and verifications of selected mechanisms to further substantiate the results of our simulations and experiments.

In Chapter 6, we presented the architecture with these mechanisms and analysed the architecture's data flow, traceability, applicability, and security by using rewrite systems and automata.

## 7.2 Assessment

This section of the dissertation measures our work described in the previous chapters towards the thesis and the hypotheses presented in Chapter 1.

The thesis of this dissertation may be summarised as: outcome of the observation of UDT data flow and the determination of the absence of its security functions which relied upon existing mechanisms of other protocols and in which, largely for practical reasons, are of limited capability. No research relying on these mechanisms may in consequence be restricted in its scope or accuracy, or even determined by the available mechanisms with which they can be adopted. New mechanisms and techniques are needed which, by supporting a more robust and viable security architecture, will contribute to better understanding of UDT's practical applications and the other protocols and its systems as a whole, its constituent components, and in particular, the interaction of security mechanisms to secure UDT data transfer within and across the network protocol stack.

The hypotheses follow that UDT is distinctive and not secure compared to other known network protocols, and these are tested through experimentation and implementation to confirm the contention. The hypotheses follow that the proposed mechanisms to secure UDT can be implemented on the selected layers of the protocol stack, and are also tested through information-theoretic, symbolic analysis, compositional proof of correctness, automata and rewrite systems. Moreover, the hypotheses follow that, using the commercially proven and proprietarily developed security mechanisms can protect not only UDT but also other protocols, and are also tested through experimentation and implementation to confirm our contentions. Lastly, the hypotheses also follow that securing UDT on all protocol layers using our proposed security mechanisms is the best technique, which is also applicable to existing protocols, and are also tested through symbolic, information-theoretic, proof of correctness, and practical implementations, which in their absence would not have been possible to develop the security UDT architecture.

We achieve important research outcomes and highlight these in fourfold:

1. Development of security mechanisms to form a novel security architecture for UDT;
2. Introduction of new axioms for inductive proofs to verify proofs of secrecy, authentication, and applicability of the proposed mechanisms;
3. Use of techniques to verify traceability, applicability, secrecy, and authentication of security properties of mechanisms within the proposed architecture to ensure a secure data flow, and lastly;
4. Development of a proprietary program for UDT data transmission analysis and visualisation. All of these are first in the literature, in addition to the practical security implementations on UDT itself to form a reliable, secure data integrity and data flow through the architecture.

## 7.3 Conclusion

In validating our proposals, we developed and created techniques which merit a separate dissertation. Although there is a body of research dedicated to these techniques, we highlighted them in this work.

We presented an approach to the formal verification of cryptographic authentication protocols, specifically on our proposed mechanisms. We illustrated the mechanisms in single and multi-session capabilities. We created our own axioms and techniques to form a reliable proof simplification and proof automaton. We used PCL for the proof of authentication protocols (for our proposed and developed UDT-AO, UDT+DTLS, and UDT+GSS-API).

To achieve applicability of the proposed mechanisms and architecture to other network protocols, we recommend that verification activity be conducted and reused and tested in a practical context.

The reuse from one mechanism to another occurs when each activity is independent of the mechanism itself. This happens in the case of an anomaly and adversary. Moreover, activities such as identifying and proving invariants, or formalising and proving secrecy and authentication properties, are similar from one architecture to another and can be reused. In Chapter 5, we showed how the confidentiality of some keys was used in the mechanisms for UDT.

The main aim of this work has been to address the absence of security for UDT. Our proofs of secrecy and authentication of security properties of the selected mechanisms, including those practically developed for UDT, provided support based on the strength of these mechanisms and based on their applicability to other high speed network protocols that rely on the speed of UDP.

In order to model our mechanisms we used a powerful paradigm (eg. rewrite systems and formalisms) sometimes referred to as the chemical abstract machine paradigm: it means that a system is a set of atomic actions which may be applied repeatedly, and in any order, and whenever the proper pre-condition holds.

We used states and relations and formalisms to present semantics which we represent in simple and basic mathematical notation. We applied rewrite systems and automata to achieve theoretic proof of security properties, proof of secrecy, authentication, and applicability of mechanisms within the architecture. We analysed and interpreted the likelihood of data flow failure that can be caused by anomalies or adversaries.

Lastly, we expressed the secrecy and authentication properties that we wanted the protocol to satisfy and observe the complete view of the proposed mechanisms and the proposed architecture data flow based on the understanding of trustable principles to guarantee data integrity and security. Our modeling of the adversary (in the form of an intruder) provides the possibility of a compromised mechanism in the architecture design. And by identifying constraints, we applied a mechanism that prevents this from occurring. We imposed a sequencing constraint which amounts to forbid multi-session capabilities when an attack occurs. We also imposed a parallel multi-session that can still allow the data to flow across other mechanisms securely.

The potential of our tool, techniques, mechanisms, and the architecture in itself, created a breadth of interesting novel and multi-faceted research which drew positive reviews from various independent international academic conferences and journals.

## 7.4 Future Work

It is interesting to present discussion of the scope for future work in the form of further investigation into the UDT implementations and UDT security architecture to support its objectives and continuing Project UDT development, with further analysis of existing and new mechanisms for UDT and other protocols.

### 7.4.1 Further Analysis: Proposed UDT Security Mechanisms

The transition from low-speed to high-speed networking in some sectors, therefore, comes at a time when societal reliance upon networks is greater than its already significant level. The assured use and access of these networks in a private, protected and reliable manner, and also with appropriate service guarantees, is deemed fundamental, which has motivated this thesis to investigate the available mechanisms which are widely used, and to develop the security architecture for UDT.

The security architecture is introduced following extensive review, which included a design for a security-specific modular structure for the UDT protocol which is also considered to be practical for other data transfer protocols.

Moreover, methods using existing security mechanisms have been introduced, and a framework for UDT with the specification of TCP, UDP and UDT, and Sockets API has been developed. The design is the result of extensive experiments and implementations, and has broad coverage in terms of practical applications with the use of existing high-speed appliances to support the schemes. In addition, the framework has been presented with notable schemes, all of which aim to ensure utility and value to users and designers of high-speed network and data transfer protocols.

The practicality of our design architecture for UDT suggests that similar designs should be created applicable and desirable for future protocol design, thereby leading to a more secure-capable protocol with higher-quality implementation security schemes.

Whilst the mechanisms in our works have been introduced so as to develop a comprehensive architecture for UDT and have therefore been subject to extensive validation and annotation, and eventually implementation and deployment, the architecture nevertheless still requires improvement, and is not as clearly presented as it might be. With this in mind, it would be interesting to see how it can be used with other fast data-transfer protocols.

Since the architecture has been developed based on particular implementations and with reference to the UDT source code and existing mechanisms, we have aimed to make it sufficiently flexible so as to permit other implementation and deployment schemes created in relation to other protocols. Notably, it would be interesting to use the framework to guide the fresh implementation and deployment of new protocols so as to determine how much implementation security-specific change is required.

Essentially, UDT does not have a clear modular structure, but rather has accreted functionality following the new versions being released; nevertheless, security mechanisms remain notably absent. Due to this situation, it is recognised that any improvement to this structure would be worthwhile. For instance, future work focuses on good security and modular structure that would introduce an improved checksum and authentication option without redesigning the entire structure of a protocol when deploying in IP V6.

### 7.4.2 Project UDT Tool Enhancement

Current work is focused on expanding the Project UDT tool to real-time and performance analysis. Options to allow additional protocol-based data extraction modules and the scope for further work using this tool is wide—the limits of existing powerful tools for UDT lead to further innovative research. Future work may include investigations of new mechanisms and DoS attack detection, traffic engineering issues, and Cloud/GRID security.

The use of the tool, mechanisms, and architecture is not limited to the observation of well-established protocols and their security mechanisms: our work and techniques are valuable to other researchers and the industry in the development of new high-speed protocols and their security.

# Chapter 8

# Epilogue

In our published work, we presented the important socio-economic relationship of IT Security to Climate Change. We emphasised the importance of developing a tailored security architecture for new high-speed network protocols that operate on Cloud/GRID (C/G) computing. In this dissertation, we presented techniques that assisted us in developing a security architecture for UDT. Our verification techniques in the development of this architecture are applicable to other protocols; these require, however, continual attention and improvement given the fast development of network and security technologies as well as the prevalence of cyber adversaries. We published our works and extended our contributions to Environmental Science and Technology. The succeeding sections highlight the relationship of IT Security and our developed architecture, which we used as an example, to support climate change initiatives.

## 8.1 Future Direction: Securing the Cloud, Dispelling Fears: Ways to Combat Climate Change

High-speed network protocols operate on high-speed networks, which offer easy and low-cost access to education, e-health, communication and business services. Going forward, one of their most valuable features is their capacity to replace physical goods with virtual ones (dematerialisation). Research shows that digital goods are superior in terms of minimising the use of energy and carbon dioxide (CO2) versus their physical counterparts, including physical activities (e.g., travel). But dematerialisation of goods comes with its own challenges. One of them is security. Because of cyber adversaries, various educational, military, government and medical institutions have been reluctant to adopt the full dematerialisation of their specific goods (medical records, and classified military and government information) and delivery of services. For these organisations, transmitting sensitive data is as important as national security.

In this work, we present situations that reflect user's fears and reluctance about fully adopting technology in order to achieve its benefits. We surveyed a few organisations to highlight the reasons of low and poor C/G adoption. Our findings highlight and confirm the importance of our work on developing a security architecture tailored for a specific network protocol that runs on a Smart GRID. We introduce this architecture as one of the many solutions to address the security challenge. We deviate from using technical jargon to describe this architecture; however, we will briefly present this architecture to demonstrate how it can protect a Smart GRID protocol.

Clearly, increasing the level of awareness as well as increasing security technologies to protect data transfer may dispel the security fears of organisations. Similar security designs and solutions that will be introduced to fast-growing high-speed networks, such as the NBN in Australia, address users' confidence and increase C/G adoption, therefore also increasing the dematerialisation of objects and services and helping minimise CO2 and energy consumption. Underscoring the importance of securing C/G through IT security in the field of environmental science and technology is another first in the literature.

### 8.1.1 Introduction

Climate change has become one of the most challenging problems faced by society today. Undoubtedly, organisations are working on addressing this problem through various means. Universities and research and non-for-profit institutions, for instance, have been investing a significant amount of resources and time to find solutions that address this ongoing problem, which, according to UNESCO's report [144], has affected and continues to affect various countries around the world. From the American continents, Africa, Australia-Asia Pacific to Europe, weather changes – and their impact on the environment, on the world's ecology, on resources such as food, and on human security itself – have become a major concern for many governments. In recent years, many researches have started focusing on the use of technology to tackle and minimise this problem.

Major steps were taken and the introduction of green technology, environmental science, and technology provided promising directions to assist in minimising climate change. On one hand, new interdisciplinary courses are now being offered in various institutions to raise the awareness of climate change. Government initiatives, on the other hand, introduced ways to expand network infrastructures, such as the NBN in Australia, and have taken important steps in implementing this high-speed network in various selected schools, health centres, homes, and industries. These have created new opportunities to provide socio-economic solutions through the Internet and C/G computing.

No doubt, the Internet and C/G computing, its applications, and IT in general play an increasingly important role today, more so than in previous years. Examples of this have brought social and political changes (e.g., Asia's Philippine revolutions and the Arab Spring Uprising, to name a few). Technology and its applications have aided civil societies and provided innovative reforms, through e-voting and e-governments.

Without a doubt, they have also become tools for tackling climate change. With the right framework and a clear understanding of technology's capabilities, minimising the impact of climate change can be achieved. However, while technology can be a great tool to address global problems, many organisations are still reluctant to fully embrace its potential.

We contend that one of the reasons that explain this problem of low C/G adoption is cyber threats. The attacks reported in [10,48,51] have increased and these continually stalled organisations to fully digitise their objects and services. Securing high-speed networks, therefore, has since pushed researchers to develop new architectures for high density data transmissions in WAN [22-33]. Many of these protocols are developed based on different technology variants, which have demonstrated better performance in simulation and in several network experiments but have limited practical applications because of implementation and installation difficulties [24-26].

Some organisations wanting to digitise goods (dematerialisation of goods) and virtualise service delivery (through transferring bulk data over long distances, thus minimising energy and CO2 for travelling) turn to application level solutions where these variants do not fair well. Many examples of this technology considered in the application level solutions are UDP-based protocols, such as UDT for C/G computing [15]. UDT was developed in a research laboratory at the National Data Mining Center, University of Illinois, Chicago [22-23,82]. However, the absence of security features for this new technology stalls its full deployment to various implementation scenarios.

We analyse common issues of poor C/G adoption across organisations in Europe, US, and Australia. We focus on the most common and predominant issue and present an approach that can increase users' confidence towards C/G adoption, and serve to support existing climate change initiatives.

In the succeeding sections of this work, we highlight our contributions. We then outline the issues of users' reluctance towards C/G adoption; we focus on the role of security and introduce a basic example of a security architecture that we developed for high-speed network protocol that operates on a Smart GRID. We present our discussions in Section 8.2. Finally, we end this chapter with the conclusion and future work in Section 8.3.

### 8.1.2 Contributions

In our works [22-33], we presented our important contributions to securing high-speed networks that use a specific variant, called UDT. We conducted formalisation to develop the architecture through proofs of mechanisms and secrecy properties in data flow, and provided a clearer understanding of how they work and secure protocols without using extensive resources to achieve verification on its applicability and adaptability to networks. For detailed information, we encourage the readers to read [22-33].

In this work, we present and extend our contributions to environmental science and technology, by highlighting the importance of the security[1] to climate change. We argue that security does not only protect infrastructures that support climate change initiatives; it also significantly and indirectly decreases environmental impact through the minimisation of carbon emissions and energy consumption.

The major contribution of this work is to underscore the important role of security in achieving climate change initiatives.

### 8.1.3 Security in High-Speed Networks

As an example, we selected a future generation GRID protocol that runs on high-speed networks. This protocol was designed to transfer a large amount of data sets across long distances. It was used in collecting large data sets from Outer Space, which were then transferred across Chicago, New York, Amsterdam, Korea, Japan, and Australia. The development of UDT, a fast data transfer protocol for C/G computing, holds great promise to the future progression of data-intensive network capabilities. This protocol was successfully implemented by capturing data, gathering terabytes of information, and transferring these across the continents in a high-speed network. This provided a compelling commercial promise in Wide Optical Area Network (WOAN) [22] and NBN[120]. Whilst many types of protocols solve many of the problems related to achieving speed, performance, and environmental issues (minimisation of carbon emissions), one common problem remains: security.

Securing protocols, such as UDT, to achieve privacy, confidentiality, and data integrity in wide optical area networks is a challenge against existing anomalies. The study of the security of UDT is new and presents interesting challenges. Many of the security problems present in existing protocols like TCP and UDP are also applicable to UDT. Moreover, many of the traditional security mechanisms, e.g., end-to-end encryption, may be applicable to UDT implementation and, in certain cases, may be even more necessary[24-26].

We will not discuss the technical aspects [22-33] of this protocol. Instead we will use the architecture we designed for this protocol as an enabling attribute to exemplify the need to secure Smart GRID. We believe that securing the protocols that operate in the GRID will create opportunities for organisations and individuals to adopt new technologies.

In this work, we survey and review existing literature that will assist in determining the relationship between C/G adoption and security. We present the relationship between security and climate change, and the significance of this relationship.

We then discuss our proposed approach in addressing the problem of slow C/G adoption: by introducing a comprehensive security architecture for GRID protocol, UDT, which we worked on for 3 years. We present this as part of our approach to secure the C/G and to dispel fears related to the adoption of this technology in the wider community.

### 8.1.4 Current Trend

There is an increasing number of organisations and citizens capitalising the Internet by joining online communities (through fixed and wireless connections) using a high-speed network (Figure. 8-1).

**Volume of data transmission by Australian Internet users**

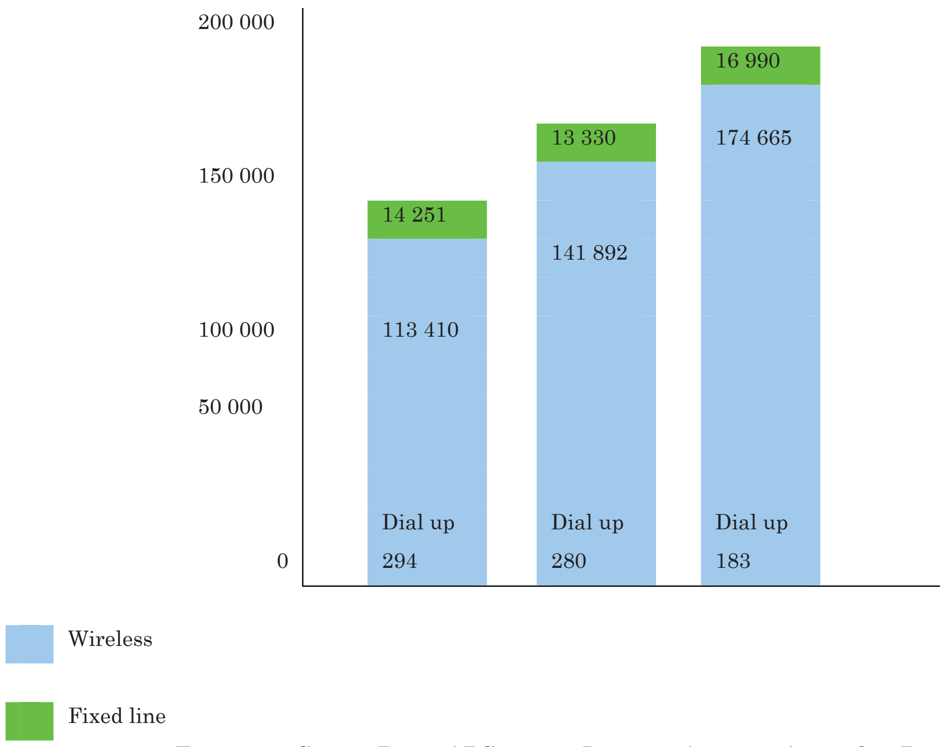


Figure 8-1 Source Data: ABS,8153.0 Internet Activity, Australia, December 2010[13]

In 2010 alone, there were almost 11 million active Internet subscribers in Australia [13]. According to the report [13, 120], 93% of Internet connections had more than 1.5 megabits of download capabilities. Over time, Internet activity will increase its consumption of resources, e.g., data consumption. In 2010, there were 191 terabytes of data downloaded [120]: a significant increase from the 99 terabytes of downloaded data recorded in the previous year. Based on this trend, the online capacity and requirements will only continue to increase and become even more data-intensive in the future. However, a few companies and individuals continue to avoid dematerialising their objects (data) and services; and many still send their data through physical media (e.g., DVD, CD, and USB) because of their fear of cyber threats[10,15,51,93].

We present a few situations to highlight this challenge.

#### 8.1.4.1 Securing e-Health

A major health company [120] outside of Sydney transmits data across long distances. These data are health records of their patients, some of which have to be transmitted in bulk across Sydney to Singapore for analysis and medical treatment. Technologies for data transmission and performance are now available, but this company faces a new challenge: securing the data being transferred across long distances.

In the past, this company had to courier these data by mail. Now that high-speed networks are available through NBN, the company, which wants to capitalise on the new technology, continues to express apprehensions about security.

*"In the past it was too hard to physically send data over long distances; often, they had to be delivered by hand. Security was not an issue then. Now, over the high-speed network, location and distance are no longer an issue; the size of data is no longer a problem either. The greater obstacle to us is security. We want to make sure that large amounts of data are secure when delivered over the Internet. We cannot afford to expose our patients' sensitive information."*

### 8.1.4.2 Securing Smart GRID, Smart City

In alliance with the Federal Government, a state government in Australia has pooled up to 100 million dollars [120] for the Smart GRID, Smart City [29] initiative, which will demonstrate an electricity system of the future. This initiative will employ various technologies and provide customers with informed choices about their energy use.

One non-profit organisation involved in the evaluation of the technology raises the issue of threats: threats that do not only fall under the physical threat to the GRID and infrastructure itself, but also threats that can occur when data transmitted across smart meter applications and delivered across at least 3000 homes / end points [120].

*"The government must consider the risks involved in deploying these types of technologies and address them. We believe that technologies can be detrimental when used and abused by adversaries. Lessons must be learned from the 9/11 attack in the United States. The data – such as information of customers, their addresses – and security features of these smart applications must be introduced and evaluated."*

### 8.1.4.3 Priorities and Barriers to Dematerialisation

Additionally, European organisations are continually making the transition to C/G computing. However, in the 2010 survey [10,13,120], a range of factors considered to be barriers to C/G-based services included geographical location of services, contract lock-ins, to name a few.

According to [48], in 2011, security remained the biggest identified barrier to adoption (63%), followed by integration issues (57%) and performance / reliability concerns (55%). Furthermore, the same survey revealed that security was deemed an ongoing issue that went beyond C/G (74%), but was certainly a priority in evaluating and managing IT delivered via C/G (80%). Figure 8-2 also shows that it remained the single biggest hurdle impacting C/G take-up (63%), despite being less significant than in the previous year's survey (71%). Security was a particular issue for companies in the United Kingdom (UK) (74%) and Germany (70%) [48].

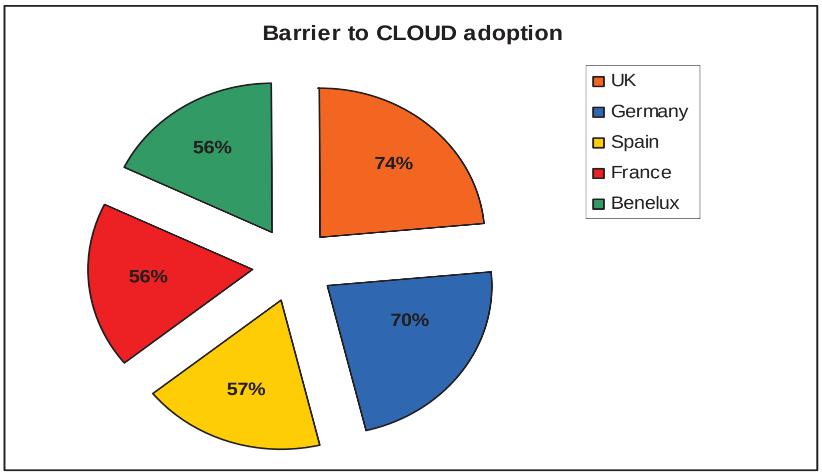


Figure 8-2: Source Data: Colt CIO Cloud Survey, May 2011 [48]

#### 8.1.4.4 User impact

According to a UK [10] study commissioned by a security software maker, more than 1,400 regular Internet users stated that cybercrime was the UK's most feared crime, outranking physical burglary, assault, and robbery [10, 48]. The study also found that 87% of the participants were worried about the threat of cybercrime, 33% were not convinced they had adequate measures in place to protect themselves, and 25% said there was not enough information available on cybercrime to protect themselves effectively. That left a significant percentage of people (62%) who, while they could find enough information to protect themselves, were still inherently worried about the threat of cybercrime.

An organisation, Db2Powerhouse, quoted a research: *"Furedi (2002) and others have portrayed the general culture of fear as a type of psychological fear of fear (Phobophobia), which can lead to stress, intense anxiety, and unrealistic and persistent public fear of crime and danger, regardless of the actual presence of such fear factors. Garland (2002) describes this phenomenon as the crime complex, a societal state where public anxiety about crime is the norm and has been imprinted in people's everyday lives as an established and expected societal* *aspect. In a 2003 survey (PewInternet, 2003), 49% of US citizens said that they were afraid of cyber assaults on key parts of the US economy. Lack of control over a situation that is perceived as threatening or dangerous gives rise to feelings of emotional distress, fear, and insecurity. Such strong emotions can inhibit flexible thinking and lead to irrational behaviour (Sutherland, 2007, p. 89) or other equally strong reactions. The effects of cybercrime and cyberterrorism-related discourse and the induced fear in the public can be seen by acts such as the need for more laws protecting against illegal cyber activity and the giving away of people's own privacy in exchange for better security."*

Results show that organisations are still uncertain whether or not they can secure their data as these are transmitted across high-speed networks. These valid concerns merit considerations to meet and maximise the benefit of high-speed networks empowering digital economy. Given that digital economy is dynamic, it is important to recognise the need of additional and continuing requirements of citizens. Investigating the issue of security and cyber threats should be closely monitored with new initiatives being introduced.

## 8.2 Discussions

In this work, we reaffirm that cyber threats remain a major inhibitor to the adoption of Internet technology and C/G across industries and schools, specifically in Europe, US, and Australia. There are, of course, huge cultural differences between these countries, and we did not attempt to directly compare these differences due to the constrained scope of research.

To support the findings and the situations presented, we looked at various avenues ranging from socio-economic to technical perspectives. We also visited existing literature [10,13,15,48,51,76-77,145] and surveyed 4 Australian organisations (schools and research institutes, non-profit organisations, a government agency, and a university; labelled as A, B C and D on Table 8-1). We used existing research approaches: the Theory of Reasoned Action, the Theory of Planned Behaviour, the Technology Acceptance Model (TAM) [4-5, 56-57] and Diffusion of Innovations. From these approaches we derived how the respondents perceived C/G to improve their organisational performance. Three important issues emerged: user trust, commitment, and satisfaction. All of the respondents from these organisations revealed their reluctance to adopting the C/G (see Table 8-1). This suggests that organisations perceive that the dangers of Internet C/G adoption outweigh the benefits, particularly in the absence of a comprehensive security architecture to protect information being transmitted across the GRID.

Table 8-1 * = Inhibiting Issue + = Enabling Issue. Internet Security is considered an inhibiting issue to full acceptance and deployment of C/G in Australia.

| Probing Issues | Emergent Themes | Org. A | Org. A | Org. B | Org. B | Org. C | Org. C | Org. D | Org. D |
|---|---|---|---|---|---|---|---|---|---|
| User Trust | Internet* Security | 78 % |  | 76% |  | 80% |  | 67 % |  |
|  | Legislation * | 1% |  |  | 4% | 2% |  | 6% |  |
|  | Reliability + | 2% |  | 3% |  |  | 3% |  | 4% |
|  | Knowledge and Skills + | 1% | 2% | 5% |  |  | 5% |  | 3% |
| Commitment | Loyalty |  | 1 % |  | 5% |  | 4% |  | 2% |
|  | Communication of Service |  | 10% | 2% |  |  | 2% |  | 4% |
| Satisfaction | Conflict Reduction |  | 3.5% |  | 4% |  | 2% |  | 4% |
|  | Risk Reduction | 1.5% |  |  | 1% | 2% |  | 10% |  |

To dispel this fear, we introduced a foundational framework applicable across existing IT initiatives:

- Introduction of a security architecture, presented in Figure 5-6. This sample presents security architecture of a specific protocol that can be deployed within a next-generation high-speed network data transmission operating on high-speed Smart GRID;
- Introduction of appropriate security policies and standards (i.e., ISO 27001/2) [94] and implementations of security mechanisms infrastructures within the organisations, based on these policies and standards;
- Increased level of security awareness to individuals and organisations;
- Institutional initiative that promotes security as a vital component to addressing climate change.

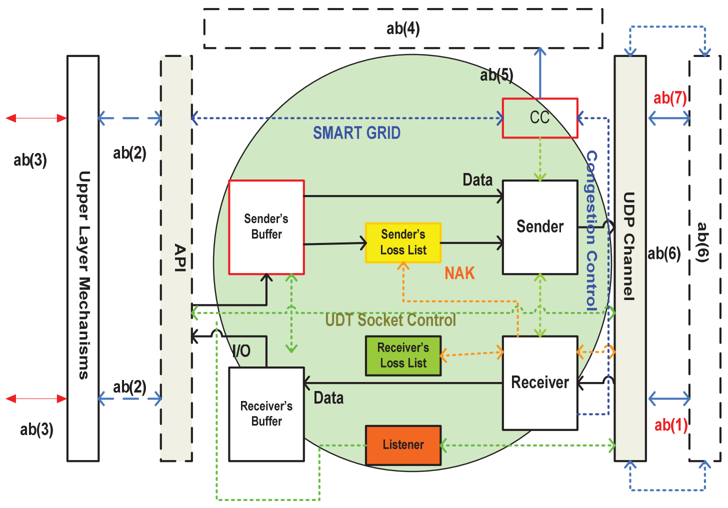


Figure 8-3: Layer-to-Layer GRID UDT Architecture. In our proposed architecture, the UDT layer provides transport functionalities to applications (Smart GRID metering applications, data and image transmissions) with security schemes that can be implemented. We encourage the readers to visit [22-33] for more technical details of the architecture. Attributes *ab*(1 to 7) are security mechanisms.

The framework presented underscores the relationships of network security technologies in tackling the inhibitor of C/G adoption. The framework is comprised of increasing awareness, implementing initiatives, and security primitives across vital components of the network through which data will be intensively transmitted, thereby increasing the use and deployment of virtual objects, accelerating the replacement of physical objects and activities by virtual services, and moving away from energy waste and intensive resource consumption (see Figures 8-4 and 8-5).

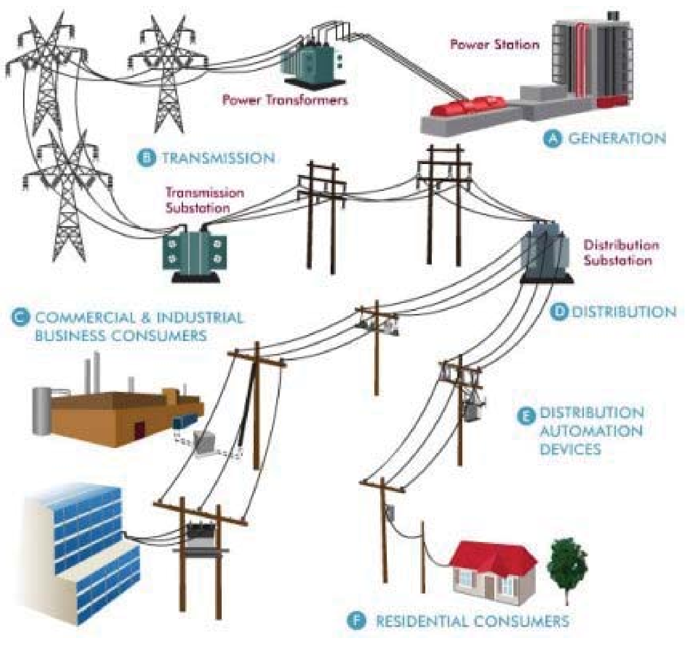


Figure 8-4: Example of Smart GRID ( A to F ) Flow Courtesy of [91,93]

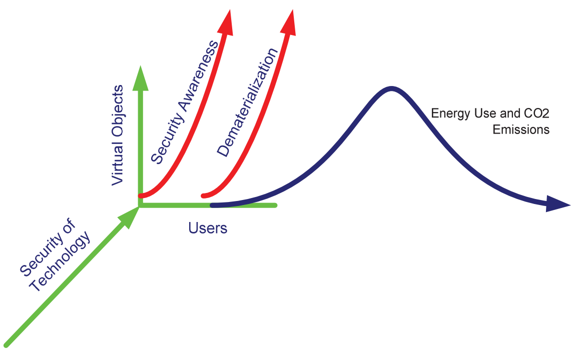


Figure 8-5: Securing the technology applied to the protocol that operates within a high-speed network increases the onset of virtual objects and proportionally decreases energy and CO2 consumption, thereby addressing climate change.

## 8.3 Conclusion and Future Work

In this work, we reviewed the existing concerns – including inhibitors to C/G adoption – and the obvious benefits of high-speed networks such as NBN to combat climate change. However, if cyber threats continue to be a major factor of slow technology adoption, the benefits of using this next-generation technology will create little impact. We looked at introducing security as a major component in addressing these issues. We investigated the growing concerns of cyber threats that slow down the maximisation of the benefits of NBN. We introduced security and highlighted its role in achieving proprietary technology such as UDT to address climate change.

We also introduced techniques and mechanisms that can protect existing high-speed data network transfer protocols with limited interdependencies [22-33]. Our security architecture and its underlying techniques can increase user confidence in NBN and potentially move users to embrace full dematerialisation of physical objects and activities in service delivery. By providing security on protocols that run on high-speed networks, we believe that our contribution can assist not only in raising awareness of the important link that network security has, but also in maximising the use of various innovative technologies that can assist in combating climate change.

Future work focuses on developing international and industry standards for security in high-speed network data transfer protocols, such as Internet Engineering Task Force (IETF) and International Standard Organisations (ISO) [94] – specifically addressing continuing challenges that involve the role of technology and important components like network security in order to address climate change.

Questions on the applicability of the architecture across high-speed network protocols running on NBN can be further addressed. The most important of these are exploratory questions on how technology and the use of it – through dematerialising objects and physical services moving towards a full deployment of C/G computing – can continue to progress across regional areas and cities in Europe, Asia, US, Australia, and beyond.

# Bibliography

[1.] Abadi, M., Rogaway,P. (2002), Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology 15, 103–127.

[2.] Adao, P., Bana, G., Scedrov, A.(2005), Computational and information-theoretic soundness and completeness of formal encryption. CSFW18, 170–184.

[3.] Ajzen, I. (1991), "The Theory of Planned Behavior." Organizational Behavior and Human Decision Processes 50(2): 179-211.

[4.] Ajzen, I. and Fishbein, M. (1980), Understanding Attitudes and Predicting Social Behavior. London, Prentice-Hall, Englewood Cliffs.

[5.] Ajzen, I. and Madden, T. (1986), "Prediction of Goal-Directed Behavior: Attitudes, Intentions, and Perceived Behavioral Control." Journal of Experimental Social Psychology 22: 453-474.

[6.] Allman, M., Paxson, V. and Stevens, W.(2009): TCP congestion control. IETF, RFC 2581, April 1999.

[7.] Al-Shraideh, F. (2006), Host Identity Protocol. In ICN/ICONS/MCL, page 203. IEEE Computer Society.

[8.] Andersen, D.G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D. and Shenker, S. (2008), Accountable Internet Protocol (AIP). In Bahl, V. Wetherall, D. Savage, S. and Stoica, I., editors, SIGCOMM, pages 339–350. ACM.

[9.] Aoto, T., Yoshida, J., and Toyama, Y.,Proving confluence of term rewriting systems automatically. In Rewriting Techniques and Applications, pages 93-102. Springer, 2009.

[10.] Ashford, W. (2007, October 29), Cybercrime is biggest UK fear. Computer Weekly. Retrieved from: http://www.computerweekly.com/;accessed on October 1, 2010.

[11.] Aura, T. (2005), Cryptographically Generated Addresses (CGA). RFC 3972, IETF.

[12.] Aura, T., Nagarajan, A., and Gurtov A., (2005), Analysis of the HIP Base Exchange Protocol. In 10th Australasian Conference on Information Security and Privacy ACISP, pages 481–494.

[13.] Australian Bureau of Statistics, 8153.0 – Internet Activity, Australia, December (2010) http://www.abs.gov.au/ausstats/abs@.nsf/mf/8153.0I; last accessed Oct 16, 2011.

[14.] Backes, M., Pfitzmann, B., Waidner, M. (2003), A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015.

[15.] Baranetsky, V. (2009, November 6). What is cyberterrorism? Even experts can't agree. Harvard Law Record. Retrieved from: http://www.hlrecord.org/news/what-is-cyberterrorism-even-experts-can-t-agree-1.861186 ;accessed Nov. 6, 2009.

[16.] Baudet, M., Cortier, V., Kremer,S., Computationally Sound Implementations of Equational Theories against Passive Adversaries. In, Caires, L., Italiano, G.F.,Monteiro, L., Palamidessi,

[17.] Baudrillard, J. (1994), Simulacra and Simulation. Ann Arbor: University of Michigan Press.

[18.] Bella, G., Paulson, L.C. (1998), Kerberos version IV, Inductive analysis of the secrecy goals. In, Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 361–375. Springer, Heidelberg.

[19.] Bellare, M., Rogaway,P. (1994), Entity authentication and key distribution. In, Stinson, D.R. (ed.) CRYPTO 1994. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg.

[20.] Bellovin, S. (1996), " Defending Against Sequence Number Attacks", RFC 1948.

[21.] Bellovin, S.(2003), "Guidelines for Mandating the Use of IPsec", Work in Progress, IETF.

[22.] Bernardo, D.V and Hoang, D. (2008), "Network Security Considerations for a New Generation Protocol UDT. Presented at Proc. IEEE the 2nd ICCIST Conference, Beijing China.

[23.] Bernardo, D.V. and Hoang, D. (2010), "Protecting Next Generation High Speed Protecting–UDT through Generic Security Service Application Program Interface GSS-API". Presented at 4th IEEE International Conference on Emerging Security Information, Systems and Technologies SECURWARE 2010 Venice/Mestre, Italy.

[24.] Bernardo, D.V. and Hoang, D. (2009), 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.

[25.] Bernardo, D.V. and Hoang, D. (2010), "A Conceptual Approach against Next Generation Security Threats: Securing a High Speed Network Protocol – UDT", Proc. IEEE the 2nd ICFN 2010, Shanya China.

[26.] Bernardo, D.V. and Hoang, D. (2010), A Pragmatic Approach: Achieving Acceptable Security Mechanisms for High Speed Data Transfer Protocol- UDT SERSC. International Journal of Security and Its Applications Vol. 4, No. 4, October, 2010.

[27.] Bernardo, D.V. and Hoang, D. (2010), End-to-End Security Methods for UDT Data Transmissions. FGIT 2010, Korea: 383-393 LNCS, Springer, Heidelberg.

[28.] Bernardo, D.V. and Hoang, D. (2010), Security Analysis of the Proposed Practical Security Mechanisms for High Speed Data Transfer Protocol. AST/UCMA/ISA/ACN 2010: 100-114, Japan, LNCS Springer –Verlag Germany.

[29.] Bernardo, D.V. and Hoang, D. (2011), " Empirical Survey: Experimentation and Implementations of High Speed Protocol Data Transfer for Grid, 25th IEEE AINA Workshop 2011, pp. 335-340.

[30.] Bernardo, D.V. and Hoang, D. (2011), "Formalisation and Information-Theoretic Soundness in the Development of Security Architecture for Next Generation Network Protocol – UDT", SECTECH Conference, Jeju Island, Korea 2011 LNCS Springer, Heidelberg.

[31.] Bernardo, D.V. and Hoang, D. (2011), Multi-layer Security Analysis and Experimentation of High Speed Protocol Data Transfer for GRID, International Journal of Grid and Utility Computing, in the press, October, 2011.

[32.] Bernardo, D.V., "UDT (2010) -Authentication Option field, An approach " Presented at 6th IEEE International Conference of Information Assurance and Security (IAS), Atlanta, USA, August 23-25, 2010.

[33.] Bernardo, D.V., and D. Hoang (2009) "Security Architecture for UDT", Work in Progress, IETF.

[34.] Bishop., S., Fairbairn, M., Norrish, P., Sewell, P., Smith, M., and Wansbrough, K., TCP, UDP, and Sockets:rigorous and experimentally-validated behavioural specification: Volume 2: The Specification. Technical Report UCAM-CL-TR-625, Computer Laboratory, University of Cambridge, Mar. 2005. 386pp.

[35.] Blumenthal, M., and Clark, D., (2001). Rethinking the Design of the Internet, End-to-End Argument vs. the Brave New World, Presented in Proc. ACM Trans Internet Technology, 1

[36.] Bonica, R. ,Weis B., Viswanathan, S., Lange, A., Wheeler, O. (2007), "Authentication for TCP-based Routing and Management Protocols, " draft-bonica-tcp-auth-06, (work in progress), Feb. 2007.

[37.] Brackmo, L., O'Malley, S. and Peterson, L. (1994) "TCP Vegas: New Techniques for congestion detection and avoidance", 1994 ACM SIGCOMM Conference, pages 24-25.

[38.] Braun, T. and Diot, C. (1995) : Protocol implementation using integrated layer processing. ACM SIGCOMM '95, Cambridge, MA, Aug. 28 - Sep. 1, 1995.

[39.] Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A. (2006), Verifying confidentiality and authentication in kerberos 5. In, Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 1–24. Springer, Heidelberg.

[40.] Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A. (2002), A Formal Analysis of Some Properties of Kerberos 5 UsingMSR. In, Fifteenth Computer Security FoundationsWorkshop—CSFW-15, Cape Breton, NS, Canada, pp. 175–190. IEEE Computer Society Press, Los Alamitos.

[41.] Canetti, R., Herzog, J. (2006), Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In, Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg.

[42.] Caron, A.C., "Linear bounded automata and rewrite systems: Influence of initial configurations on decision properties" TAPSOFT '91 LNCS, Vol 493/1991, 74-89, DOI 10.1007/3-540-53982-4_5

[43.] CERT, 1996a. "UDP Port Denial-of-Service Attack," Advisory CA-96.01, Computer Response Team, Pittsburg, PA.

[44.] Cervasato, I., Meadows, C., Pavlovic, D. (2005), An encapsulated authentication logic for reasoning about key distribution. In, CSFW-18, IEEE Computer Society, Los Alamitos.

[45.] Cervesato, I., Jaggard, A., Scedrov, A., Tsay, J.K., Walstad, C. (2005) ,Breaking and fixing publickey kerberos (Technical report).

[46.] Clark, D., Lambert, M. and Zhang, L. (1987) "NETBLT: A High Throughput Transport Protocol ." Proc. of SIGCOMM '87, pp. 353-359.

[47.] Clark, D., Sollins, L., Wroclwski, J.,Katabi, D., Kulik,J., Yang, X., (2003) New Arch, Future Generation Internet Architecture, Technical Report, DoD – ITO.

[48.] Colt Security Consulting CIO Cloud Survey May (2011).

[49.] Comon, H., Dauchet,M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M., Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2008.

[50.] Contejean, E., Paskevich, A., Urbain, X., Courtieu ,P., Pons, O., and Forest,J., PAT, an approach for certified automated termination proofs. In ACM SIGPLAN Work. on Partial evaluation and program manipulation, pages 63-72. ACM, 2010.

[51.] Cyberterrorism (2000), Testimony before the Special Oversight Panel on Terrorism Committee on Armed Services U.S. House of Representatives (testimony of Denning, D. E.).

[52.] Datta, A. Derek, A., Mitchell, J.C., Pavlovic, D. (2005),A derivation system and compositional logic for security protocols. Journal of Computer Security 13, 423–482.

[53.] Datta, A. Derek, A., Mitchell, J.C., Warinschi, B. (2006),Computationally sound compositional logic for key exchange protocols. In, Proceedings of 19th IEEE Computer Security Foundations Workshop, pp. 321–334. IEEE, Los Alamitos 324,

[54.] Datta, A., Derek, A., Mitchell, J.C., Roy, A. (2007), Protocol Composition Logic (PCL). Electronic.Notes Theory. Computer. Sci. 172, 311–358.

[55.] Datta, A., Derek, A., Mitchell, J.C., Shmatikov, V., Turuani, M. (2005), Probabilistic polynomial time semantics for a protocol security logic. In, Caires, L., Italiano, G.F., Monteiro, L.,Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 16–29. Springer, Heidelberg.

[56.] Davis, F. D. (1986). A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results. Boston, MIT. PhD thesis.

[57.] Davis, F. D. (1989). "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology." MIS Quarterly 10(3): 318-340.

[58.] Davis, F. D., Bagozzi, R. and Warshaw, P. (1989), "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models." Management Science 35(8): 982-1003.

[59.] Dierks, T., and Allen , C. (1999), "The TLS Protocol Version 1.0", RFC 2246.

[60.] Dierks, T., Rescorla, E. (2006), The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346.

[61.] Duke, M., Braden, R., Eddy, W., Blanton, E. (2006): A Roadmap for Transmission Control Protocol (TCP), RFC 4614, IETF, September 2006.

[62.] Dunigan, T., Mathis, M. and Tierney, B. (2002), " A TCP Tuning Daemon", in Proc of IEEE SuperComputing 2002.

[63.] Durand, I., and J., Meseguer, A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. pages 69- 85. Springer, 2010.

[64.] Durand, I., Autowrite: A tool for term rewrite systems and tree automata. Electronic Notes in Th. Comp. Sci., 124(2):29-49, 2005.

[65.] Durgin, N., Mitchell, J.C., Pavlovic, D. (2001), A compositional logic for protocol correctness. In Proceedings of 14th IEEE Computer Security Foundations Workshop, pp. 241–255. IEEE, Los Alamitos.

[66.] Durgin, N., Mitchell, J.C., Pavlovic, D. (2003), A compositional logic for proving security properties of protocols. Journal of Computer Security 11, 677–721.

[67.] F´abrega ,F.J.T., Herzog, J.C.,Guttman, J.D. (1998),Strand spaces, Why is a security protocol correct? In, Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, CA, pp. 160–171. IEEE Computer Society Press, Los Alamitos.

[68.] Falby, N., Fulp, J.,Clark, P., Cote, R., Irvine, C., Dinolt, G., Levin, T., Rose, M., and Shifflett, D. (2004), "Information assurance capacity building, A case study," Presented in Proc. 2004 IEEE Workshop on Information Assurance, U.S. Military Academy, June, 2004, 31-36.

[69.] Feng, W. and Tinnakornsrisuphap, P. (2000): The failure of TCP in high-performance computational grids. SC '00, Dallas, TX, Nov. 4 - 10, 2000.

[70.] Feuillade, G., Genet, T., and Viet Triem, T., Reachability analysis over term rewriting systems. 33(3):341-383, 2004.

[71.] Fiedrich, O.,"On the connections between reqriting and formal language theory " REWRITING TECHNIQUES AND APPS,11th International Conf RTA 2000, Nowrich UK, July 2000 LNCS 1631/1999, 672, 2000.

[72.] Fishbein, M. and Ajzen, I. (1975), Belief, Attitude, Intention, and Behavior: An Introduction to Theory and Research. Reading, Addison-Wesley.

[73.] Floyd, S. and Fall, K. (2009) : Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Transactions on Networking, 7(4): 458-472, 1999.

[74.] Ford, B. (2007), Structured Streams: a New Transport Abstraction, ACM SIGCOMM, August 27-31, 2007, Kyoto, Japan.

[75.] Furedi, F. (2002), Culture of Fear. London: Publisher Continuum.

[76.] Garland, D. (2008), On the concept of moral panic. Crime, Media, Culture, 4(1), 9–30.

[77.] Gefen, D. and Straub, D. (2000), "The Relative Importance of Perceived Ease of Use in IS Adoption: A Study of E-Commerce Adoption." Journal of the Association for Information Systems 1.

[78.] Giesl, J., Schneider-Kamp, P., and Thiemann, R., AProVE 1.2: Automatic termination proofs in the dependency pair framework. In Intl Joint Conf. on Automated Reasoning, pages 281-286. Springer, 2006.

[79.] Globus XIO:unix.globus.org/toolkit/docs/3.2/xio/index.html;accessed on November 1, 2010.

[80.] Gorodetsky, V., Skormin, V. and Popyack, L. (Eds.) (2001), Information Assurance in Computer Networks, Methods, Models, and Architecture for Network Security, St. Petersburg, Springer.

[81.] Grossman, R.L., Gu, Y., Hanley, D., Hong, X., Lillethun, D., Levera, J., Mambretti, J., Mazzucco, M., and Weinberger, J. (2002), " Experimental Studies Using Photonic Data Services at IGrid 2002." FGCS, 2003.

[82.] Gu,Y., Grossman, R., (2007) UDT, UDP-based Data Transfer for High-Speed Wide Area Networks. Computer Networks (Elsevier). Volume 51, Issue 7, 2007.

[83.] H. I. , (2008) for Information Technology, H. U. of Technology, et al. Infrastructure for HIP.

[84.] Hacker, T., Athey, B., and Noble, B.(2002), "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area-Network", in Proc of IPDPS 2002.

[85.] Hamill, J., Deckro, R., and Kloeber, J., (2005), "Evaluating information assurance strategies," in Decision Support Systems, Vol. 39, Issue 3 (May 2005), 463- 484.

[86.] Harrison, D. (2004) ,RPI NS2 Graphing and Statistics Package,http,//networks.ecse.rpi.edu/~harrisod/graph.html; accessed on October 2, 2010.

[87.] Hasebe, K., Okada, M. (2004), Non-monotonic properties for proving correctness in a framework of compositional logic. In, Foundations of Computer Security Workshop, pp. 97–113.

[88.] Heffernan, A. (1998), RFC 2385 "Protection of BGP Sessions via the TCP MD5 Signature Option", August 1998.

[89.] Herbert, T. (2006), Linux TCP/IP Networking for Embedded Systems (Networking), Second Edition. Charles River Media, November 2006.

[90.] Herzog, J. (2004), Computational Soundness for Standard Assumptions of Formal Cryptography. PhD thesis, MIT.

[91.] http://venturebeat.com/2011/02/01/how-secure-is-the-smart-grid/ Accessed October 21, 2011.

[92.] http://www.cio.gov/documents.Federal-Cloud-Computing-Strategy.pdf ;accessed October 16,2011.

[93.] http://www.pikeresearch.com/research/smart-grid-cyber-security Accessed October 21, 2011.

[94.] ISO 27001/2 International Standards Organizations 2009-2010, ISO.

[95.] Jacquemard, F., Decidable approximations of term rewriting systems. In Rewriting Techniques and Applications, pages 362-376. Springer, 1996.

[96.] Jokela, P., Moskowitz,R., and Nikander, P. (2008), Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP). RFC 5202, IETF, April 2008.

[97.] Jouannaud, J.P.and Kirchner, C., Solving equations in abstract algebras: a rule-based survey of unification. In Computational Logic: Essays in Honor of Alan Robinson, chapter 8, pages 257-321. The MIT-Press, 1991.

[98.] Joubert, P., King, R., Neves, R., Russinovich, M. and Tracey, J., (2001)., Highperformance memory-based web servers, Kernel and user-space performance. USENIX '01, Boston, Massachusetts, June 2001.

[99.] Jray, W., (2000), "Generic Security Service API Version 2 ,C-bindings", RFC 2744.

[100.] Katabi, D., Hardley, M. and Rohrs, C. (2002): Internet congestion control for future high bandwidth-delay product environments. ACM SIGCOMM '02, Pittsburgh, PA, Aug. 19 - 23, 2002.

[101.] Kent, S., Atkinson, R.,(1998) "Security Architecture for the Internet Protocol", RFC 2401.

[102.] Kirchner, C., Kirchner, H. and de Oliveira, A., Analysis of rewrite based\ access control policies. Electronic Notes in Th. Comp. Sci., 234:55-75, 2009.

[103.] Kohler, E., Handley, M., and Floyd, S. (2006): Designing DCCP: Congestion Control Without Reliability, Proceedings of SIGCOMM, September 2006.

[104.] Korp, M., Sternagel, C., Zankl H. and Middeldorp, A., Tyrolean termination tool 2. In R. Treinen, editor, Rewriting Techniques and Applications, volume 5595 of Lecture Notes in Computer Science, pages 295-304. Springer Berlin / Heidelberg, 2009.

[105.] Laganier, J., Eggert,L. (2008),Host Identity Protocol (HIP) Rendezvous Extension. RFC 5204, IETF.

[106.] Laganier, J., Koponen, T., and Eggert, L., (2008) Host Identity Protocol (HIP) Registration Extension. RFC 5203, IETF.

[107.] Leech, M. (2003), "Key Management Considerations for the TCP MD5 Signature Option," RFC-3562, Informational, July 2003.

[108.] Leon-Garcia, A., Widjaja, I. (2000), Communication Networks, McGraw Hill,2000.

[109.] Linn, J., (2000) "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[110.] Linn, J., (1996), "The Kerberos Version 5 GSS-API Mechanism", IETF, RFC 1964, June 1996.

[111.] Mathis, M, Mahdavi, J., Floyd, S., and Romanow, A., (1996)., TCP selective acknowledgment options. IETF RFC 2018, April 1996.

[112.] Mazzucco, M., Ananthanarayan, A., Grossman, R., Levera, J. and Bhagavantha Rao, G. (2002): Merging multiple data streams on common keys over high performance networks. SC '02, Baltimore, MD, Nov. 16 - 22, 2002.

[113.] Meadows, C. (1994), A model of computation for the NRL protocol analyzer. In, Proceedings of 7th IEEE Computer Security Foundations Workshop, pp. 84–89. IEEE, Los Alamitos.

[114.] Melnikov, A., Zeilenga, K., (2006)., Simple Authentication and Security Layer (SASL) IETF, RFC 4422, June 2006.

[115.] Menezes, A.J., Oorschot van , P.C., and Vanstone, S.A. (1997),Handbook of Applied Cryptography, CRC Press, 1997.

[116.] Micciancio, D., Warinschi,B. (2004), Soundness of formal encryption in the presence of active adversaries. In, Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg.

[117.] Mitchell, J.C, Roy, A., Rowe, P. and Scedrov, A. 2008. Analysis of EAP-GPSK authentication protocol. In Proceedings of the 6th international conference on Applied cryptography and network security (ACNS'08), Steven M. Bellovin, Angelos Keromytis, Rosario Gennaro, and Moti Yung (Eds.). Springer-Verlag, Berlin, Heidelberg, 309-327, 2008.

[118.] Moskowitz, R., and Nikander, P. (2006), RFC 4423, Host identity protocol (HIP) architecture, May 2006.

[119.] Moskowitz, R., Nikander, P., Jokela, P., and Henderson, T. (2008), Host Identity Protocol. RFC 5201, IETF, April 2008.

[120.] National Digital Economy Strategy (2010). Department of Broadband, Communications and the Digital Economy.

[121.] Neuman, C., Yu, T., Hartman, S., Raeburn, K. (1996), Kerberos Network Authentication Service (V5), IETF, RFC 1964, 1996.

[122.] NIST SP 800-37 (2004), Guide for the Security Certification and Accreditation of Federal Information Systems.

[123.] NS2. http,//isi.edu/nsna/ns; accessed on October 2, 2008.

[124.] Paulson, L.C. (1998), The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128.

[125.] Postel, J. (1974), A Graph Model Analysis of Computer Communications Protocols. University of California, Computer Science Department, PhD Thesis, 1974.

[126.] PSU Evaluation Methods for Internet Security Technology (EMIST) , 2004, http,//emist.ist.psu.edu; accessed on December 1, 2009.

[127.] Rabin, M. (1979), "Digitized signatures and public-key functions as intractable as Factorization," MIT/LCS Technical Report, TR-212.

[128.] Rescorla, E., Modadugu, N. (2006), "Datagram Transport Layer Security" RFC 4347, IETF, April 2006.

[129.] Rivest, R.L., Shamir, A., and Adleman L.M.(1978) ,"A method for obtaining digital signature and publickey cryptosystems," Communication of ACM, 21, (1978),120-126.

[130.] Roy, A. , Datta, A., Derek,A.(2007), Mitchell, J.C., Inductive trace properties for computational security. WITS.

[131.] Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A. (2000), Modelling and Analysis of Security Protocols. Addison-Wesley Publishing Co., Reading.

[132.] Schwartz, M. (1996) ,Broadband Integrated Networks, Prentice Hall.

[133.] Serjantov, A., Sewell, P., and Wansbrough, K.(2001), The UDP Calculus: Rigorous Semantics for real networking. Proc TACS 2001: Fourth International Symposium on Theoretical Aspects of Computer Software, Tohoku University, Sendai, Oct. 2001.

[134.] Smart Grid, Smart City Project http://www.ret.gov.au/energy/energy_progams/smartgrid/Pages/default.aspx;acce ssed on November 2, 2011.

[135.] Stevens, W.R.(1994), TCP/IP Illustrated Vol.1: The Protocols, 1994.

[136.] Stewart, R. (2007),(Editor), Stream Control Transmission Protocol, RFC 4960.

[137.] Stiemerling, M., Quittek, J., and Eggert, L. (2008), NAT and Firewall Traversal Issues of Host Identity Protocol (HIP) Communication. RFC 5207, IETF, April 2008.

[138.] Stoica, I., Adkins, D.,Zhuang, S., Shenker, S., Surana, S. (2002), Internet Indirection Infrastructure, Presented at Proc. ACM SIGCOMM, August 2002.

[139.] Szalay, A., Gray, J., Thakar,A., Kuntz, P., Malik,T., Raddick, J., Stoughton. C., Vandenberg,J. (2002),: The SDSS SkyServer - Public access to the Sloan digital sky server data. ACM SIGMOD 2002.

[140.] Teraflow Testbed, http://www.teraflowtestbed.net.

[141.] Tison, S., " Tree Automata and Term reqrite systems Sophie Tison, LNCS 1833, Bachmair, L. Editor REWRITING TECHNIQUES AND APPS 11th International Conf RTA 2000, Nowrich UK, July 2000 176, 2000.

[142.] Touch, J. (2007), "Defending TCP Against Spoofing Attacks," RFC-4953, Informational, Jul. 2007.

[143.] UDT source release. http://udt.sourceforge.net

[144.] United Nations Children's Fund, Climate Change and Children, (2010).

[145.] US Government, Federal Cloud Computing Strategy (February 2010).

[146.] Wang, X., Yu, H. (2005), "How to break MD5 and other hash functions," Proc. IACR Eurocrypt 2005, Denmark, pp.19-35.

[147.] Warinschi, B. (2003), A computational analysis of the Needham-Schroeder(-Lowe) protocol. In, Proceedings of 16th Computer Science Foundation Workshop, pp. 248–262. ACM Press, New York.

[148.] Williams, N.(2009), "Clarifications and Extensions to th eGeneric Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.

[149.] Xu, L., Harfoush, K., and Rhee, I. (2004): Binary increase congestion control for fast long-distance networks. IEEE Infocom '04, Hongkong, China, Mar. 2004.

[150.] Yung, C.M. (2005) (eds.) ICALP. Analysis of EAP-GPSK Authentication Protocol LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg, 325, 2005.

[151.] Zhang, M., Karp, B., Floyd, S., and Peterson,L.(2003): RR-TCP: A reordering-robust TCP with DSACK. Proc. the Eleventh IEEE International Conference on Networking Protocols (ICNP 2003), Atlanta, GA, November 2003.

[152.] Zhang, Y., Yan, E. and Dao, S.K. (1998): A measurement of TCP over long-delay network. The 6th International Conference on Telecommunication Systems, Modeling and Analysis, Nashville, TN, March 1998.

Recommended Readings:

1. K.H. Rosen. Discrete Mathematics and its Applications, McGraw Hill 1995.
2. J.L. Gersting. Mathematical Structures for Computer Science, Freeman 1993.
3. J.K. Truss. Discrete Mathematics for Computer Science, Addison-Wesley 1991
4. R. Johnsonbaugh, Discrete Mathematics, 5th ed. Prentice Hall 2000.
5. C. Schumacher, Fundamental Notions of Abstract Mathematics, Addison-Wesley, 2001
6. M. Sipser, Introduction to Theory of Computing, 2nd ed,Thomson Technology, 2006
7. F. Baader and T. Nipkow. Term rewriting and all that. C.U.Press, LNCS. 2007.
8. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2011

# Appendices

## Appendix A

### Mathematical Notations

We briefly introduce some of the mathematical notations and the graphical symbols that we will use throughout this work:

| | |
|---|---|
| $\forall$ - | Universal quantifier for all the proposition is true for all possible values in the universe of discourse |
| $\exists$ - | Existential quantifier exists the proposition is true for some value(s) in the universe of discourse |
| $\rightarrow$ - | Given the statements p and q, an implication is a statement that is false when p is true and q is false, and true otherwise |
| $\leftrightarrow$ - | A biconditional statement is true whenever the truth value is the same for both p and q and false otherwise |
| $\neg$ - | 'NOT:'Negation - a method of assigning the opposite truth value to the statement |
| $\in$ and $\notin$ - | Denote set membership and non membership |

Set Union

$$A \cup B = \{x : x \in A \vee x \in B\}$$

Set Intersection

$$A \cap B = \{x : x \in A \wedge x \in B\}$$

Difference

$$A - B = \{x : x \in A \wedge x \notin B\}$$

Symmetric Difference

$$A \triangle B = (A - B) \cup (B - A)$$

Basic Quantifiers

| Statement | True when... | False when... |
|---|---|---|
| $\forall x\ P(x)$ | $P(x)$ is true for every $x$ | There is an $x$ for which $P(x)$ is false |
| $\exists x\ P(x)$ | There is an $x$ for which $P(x)$ is true | $P(x)$ is false for every $x$ |

Mixed Quantifiers

| Statement | True when... | False when... |
|---|---|---|
| $\forall x \forall y\ P(x,y)$ | $P(x,y)$ is true for every pair $x,y$ | There is at least one *pair* $x,y$ for which $P(x,y)$ is false |
| $\forall x \exists y\ P(x,y)$ | For every x, there is a $y$ for which $P(x,y)$ is true | There is an $x$ for which $P(x,y)$ is false for every $y$ |
| $\exists x \forall y\ P(x,y)$ | There is an $x$ for which $P(x,y)$ is true for every $y$ | For every $x$, there is a $y$ for which $P(x,y)$ is false |
| $\exists x \exists y P(x,y)$ | There is at least one pair $x,y$ for which $P(x,y)$ is true | $P(x,y)$ is false for every pair $x,y$ |

Negation Truth

| Statement | True when... | False when... |
|---|---|---|
| $\forall x \forall y\, P(x,y)$ | $P(x,y)$ is true for every pair $x,y$ | There is at least one *pair* $x,y$ for which $P(x,y)$ is false |
| $\forall x \exists y\, P(x,y)$ | For every x, there is a $y$ for which $P(x,y)$ is true | There is an $x$ for which $P(x,y)$ is false for every $y$ |
| $\exists x \forall y\, P(x,y)$ | There is an $x$ for which $P(x,y)$ is true for every $y$ | For every $x$, there is a $y$ for which $P(x,y)$ is false |
| ∃x∃y$P(x,y)$ | There is at least one pair $x,y$ for which $P(x,y)$ is true | $P(x,y)$ is false for every pair $x,y$ |

## JAVA Codes

| | |
|---|---|
| Program Name | : ProjectUDT.java, |
| Required Supplementary Program | : GenerateBarGraph.java |
| Required file | : UDTFile.txt |
| Author | : Danilo V. Bernardo |

---

**Code : ProjectUDT.java**

```
// ProjectUDT.java

/***********************************************
 *  Program Name: ProjectUDT.java
 *  Required Program: GenerateBarGraph.java
 *  Required file   : UDTFile.txt
 *
     This program is created to  aid  my

    Research on UDT

 *
 *
 *  @author: Dan Bernardo
 *  @version Last modified Nov 2, 2011
 *

************************************************/

import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.*;
import javax.swing.*;
import javax.imageio.*;
import javax.swing.border.Border;




// Details used

/************************************************
 * import java.awt.Container;
 * import java.awt.Graphics;
 * import java.awt.Graphics2D;
 * import java.awt.Rectangle;
 * import java.awt.Dimension;
 * import java.awt.Robot;
 * import java.awt.event.ActionEvent;
 * import java.awt.event.ActionListener;
 * import java.awt.event.WindowAdapter;
 * import java.awt.event.WindowEvent;
 * import java.awt.geom.Path2D;
 * import java.awt.image.BufferedImage;
 * import java.io.BufferedReader;
 * import java.io.File;
 * import java.io.FileReader;
 * import java.util.ArrayList;
 * import java.util.StringTokenizer;
 * import javax.imageio.ImageIO;
 * import javax.swing.JFrame;
 * import javax.swing.JMenu;
 * import javax.swing.JMenuBar;
 * import javax.swing.JMenuItem;
 * import javax.swing.JOptionPane;
 * import javax.swing.JPanel;
 * import javax.swing.JScrollPane;
 * import javax.swing.JTextArea;

************************************************/


Public class ProjectUDT extends JFrame implements ActionListener{

     /** JLabel object to show the help information about the program
     */

```

```
        private ImageIcon icon;
        private JFrame jf;
        private JLabel programInfoLabel, iconLabel ;
        private JMenuBar menuBar;
        private JMenu menu;
        private JMenuItem fileMenuItem;
        private JMenuItem exitMenuItem;
        private JMenu helpMenu;
        private JMenuItem helpMenuItem;
        private Panel
panel;

    /** Main
     */

     public static void main(String[] args){
          new ProjectUDT() ;
     }



/**
        *  Constuctor to initialize and display
layout
        *  Initializes and defines screen size, frame,
panel

*/

     public ProjectUDT(){

                // Label
Project

                jf = new JFrame("Project UDT - DBernardo 2011");
                programInfoLabel =new JLabel();
                ImageIcon icon= new
ImageIcon("image.gif");
                iconLabel = new JLabel("", icon,
JLabel.CENTER);

                Panel panel = new Panel();
                panel.add(iconLabel);
                jf.add(panel, BorderLayout.CENTER);

                // Menu Display



           menuBar = new JMenuBar() ;
           menu = new JMenu("Menu");



           menuBar.add(menu) ;

           fileMenuItem = new JMenuItem("Process File -input text file
           only");
           fileMenuItem.addActionListener(this) ;

           exitMenuItem = new JMenuItem("Exit");
           exitMenuItem.addActionListener(this) ;

           menu.add(fileMenuItem) ;
           menu.add(exitMenuItem) ;

          helpMenu = new JMenu("Help");
          helpMenuItem = new JMenuItem("Info about the program");
          helpMenuItem.addActionListener(this) ;
          helpMenu.add(helpMenuItem) ;
          menuBar.add(helpMenu) ;

               // Frame location and size

          jf.setJMenuBar(menuBar);
               jf.setSize(500,400);
               jf.setLocation(jf.getToolkit().getScreenSize().width/2-
jf.getWidth()/2,jf.getToolkit().getScreenSize().height/2-jf.getHeight()/2);

               jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

jf.setVisible(true);

     }


/**
        *  Method actionPerformed calling event
        *  Initializes and defines screen size, frame,
panel

*
        *  @param   actionEvent

*
     */

     public void actionPerformed(ActionEvent actionEvent){

     JMenuItem source = (JMenuItem)(actionEvent.getSource());
          if (source.getText().equalsIgnoreCase("exit")) {
                  System.exit(0) ;
          } else if (source.getText().equalsIgnoreCase("Process File -input text file
          only")){
                  processFile("Enter the File Name of Text
                  File") ;

          } else if (source.getText().equalsIgnoreCase("Info about the
          program")){

showHelp();
          }

     }


/**
        *  Method showHelp
        *  Opens a window with basic information

*

*
     */




       public void showHelp(){

```

```
                final JFrame helpFrame = new JFrame(" Program Help")
;
                JPanel box = new JPanel();
                JButton exitButton = new
JButton("Close") ;
                exitButton.addActionListener(new
ActionListener(){
                                public void
actionPerformed(ActionEvent e){


helpFrame.setVisible(false) ;

}






}) ;

                box.setLayout(new BoxLayout(box, BoxLayout.PAGE_AXIS));
                box.add(new JLabel("Thank you"));
                JPanel pane = new JPanel(new
BorderLayout());
                pane.add(box, BorderLayout.PAGE_START);
                pane.add(exitButton, BorderLayout.PAGE_END);
                Border padding =
BorderFactory.createEmptyBorder(150,150,150,150);
                helpFrame.setLocation(helpFrame.getToolkit().getScreenSize().width/2-
helpFrame.getWidth()/4,helpFrame.getToolkit(). +
                .getScreenSize().height/4-
helpFrame.getHeight()/4);

helpFrame.setSize(500,400);
           pane.setBorder(padding) ;
                helpFrame.add(pane) ;
                helpFrame.setVisible(true)
;

}




/**
        *  Method setFramePosition
        *  Initializes and defines frame size and
location

*
        *  @param   frame

*

*/

     public void setFramePosition(JFrame frame){

        Toolkit toolKit = Toolkit.getDefaultToolkit();
        Dimension screenSize = toolKit.getScreenSize();
        int width = (int) screenSize.getWidth();
        int height = (int)
screenSize.getHeight();
```

```
        height = height - (height / 3);
        width = width - (width / 3);
        int centerWidth=(int)(screenSize.getWidth()-
width)/2;
        int centerHeight=(int)(screenSize.getHeight()-height)/2;
        frame.setSize(width, height);
        frame.setLocation(centerWidth, centerHeight);
     }


/**
        *  Method getFileName
        *  Gets I/O error standard message

*
        *  @param   message

*

*/

     public String getFileName(String message){

           String fileName = (String)JOptionPane.showInputDialog(

this,

message,
                "File
Name",
                JOptionPane.PLAIN_MESSAGE,

null,

null,

"");
           return fileName ;
     }


/**
        *  Method generateBarChart

*
        *  Calls GenerateBarGraph to create graph
        *  Saves file to an image
file

*
        *  @param   ArrayList <Double> dataValues - for the
values
        *  @param   ArrayList <String> dataNames  - for the label where the
values taken

*/

     public void generateBarChart( ArrayList<Double> dataValues, ArrayList<String>
     dataNames){

           final JFrame frame = new JFrame() ;
           setFramePosition(frame) ;
           frame.getContentPane().add(new GenerateBarGraph(dataValues, dataNames, "Trend of Terabyte UDT High Speed Data
           Transfer"));

```

```
              JPanel objJPanel = new JPanel() ;
                 objJPanel.setLayout(new BorderLayout())
;
              JButton proceedButton = new JButton("Save Graph to GIF format")
     ;
              proceedButton.addActionListener(

                         new
ActionListener(){

                                    public void
           actionPerformed(ActionEvent event){

frame.setVisible(true);

                                         //
captures screen


try {
                                    BufferedImage screencapture = new Robot().createScreenCapture(new
                              Rectangle(Toolkit.getDefaultToolkit().getScreenSize()) );
                                              String
name = "graph.gif";
                                              File f
= new File(name);

ImageIO.write(screencapture, "gif", f);

}

                                         catch
(Exception e) {
                                                // TODO Auto-
generated catch block

e.printStackTrace();

}
JOptionPane.showMessageDialog(null, "Completed!", "Graph Options", JOptionPane.PLAIN_MESSAGE);

frame.setVisible(false); //

}

}

) ;

            JButton cancelButton = new JButton("Cancel") ;
            cancelButton.addActionListener(
                         New
ActionListener(){
                                public void
actionPerformed(ActionEvent event){

frame.setVisible(false) ;

}

}

) ;

            objJPanel.add(proceedButton, BorderLayout.CENTER) ;
```

```
              objJPanel.add(cancelButton, BorderLayout.EAST) ;
              frame.add(objJPanel, BorderLayout.SOUTH) ;
           frame.setVisible(true) ;
     }


/**
        *  Method showFileContent

*
        *  Opens and displays file and entries

*
        *  @param   dataFile

*

*/


     public void showFileContent(File dataFile){

           final JFrame frame = new JFrame() ;
           setFramePosition(frame) ;

           final ArrayList<String> dataEntities = new
           ArrayList<String>() ;
           final ArrayList<Double> dataValues = new ArrayList<Double>();


             frame.setLayout( new BorderLayout()) ;
       JTextArea fileContentArea = new JTextArea() ;
             fileContentArea.setEditable(false) ;
             JScrollPane scrollPane = new
     JScrollPane(fileContentArea);
             frame.add(scrollPane, BorderLayout.CENTER) ;


Try {
     FileReader objFileReader = new FileReader(dataFile) ;
     BufferedReader objBufferedReader = new BufferedReader(objFileReader) ;

     String dataLine = objBufferedReader.readLine() ;
     int index = 0 ;
     int lineNum = 0 ;
                   while( dataLine !=
                   null){
                        if (lineNum != 0){

                             // optional - console
                             System.out.println(dataLine);

                             String []values =
                             dataLine.split("\t") ;
                             if (values.length >
                             2){
                                  dataValues.add(Double.parseDouble(values[0]))
                                  ;
                                  dataValues.add(Double.parseDouble(values[1]))
                                  ;
                                  dataEntities.add("RTT") ;
                                  dataEntities.add("SendRate")
                                  ;
                             }

                        }
                        fileContentArea.append(dataLine + "\n")
                        ;
                        dataLine = objBufferedReader.readLine()
                        ;
                        lineNum ++ ;
                   }
                   objBufferedReader.close() ;
                   objFileReader.close() ;
           } catch (FileNotFoundException e) {
                   e.printStackTrace();
           } catch (IOException ioe) {
                   ioe.printStackTrace() ;
           }


        JPanel objJPanel = new JPanel() ;
        objJPanel.setLayout(new BorderLayout()) ;
        JButton proceedButton = new JButton("Get the Graph") ;
        proceedButton.addActionListener(
           new ActionListener(){
                   public void actionPerformed(ActionEvent event){
                        frame.setVisible(false)
                        ;
                        generateBarChart(dataValues, dataEntities)
                        ;
                   }
           }

) ;

        JButton cancelButton = new JButton("Cancel") ;
        cancelButton.addActionListener(
           new ActionListener(){


                   public void actionPerformed(ActionEvent event){
                        frame.setVisible(false)
                        ;



                   }
           }
     )
     ;

        objJPanel.add(proceedButton, BorderLayout.CENTER) ;
        objJPanel.add(cancelButton, BorderLayout.EAST) ;

        frame.add(objJPanel, BorderLayout.SOUTH) ;

        frame.setVisible(true)
;
     }



/**
        *  Method processFile

*
        *  Gets and validates
file
```

```
*
        *  @param   message

*

*/


     public void processFile(String message){

           String fileName = getFileName(message) ;

           if ((fileName != null) && (fileName.length() >
           0)) {
                   File dataFile = new File("./" +
                   fileName);
                  boolean exists =
           dataFile.exists();

               if (exists)
           {
                 showFileContent(dataFile) ;

               }
           else{
                   processFile("File Does not exists\nEnter the correct
                   name: ") ;
               }
                   return;
           } else if ( fileName != null && fileName.length() ==
           0){
                   processFile("File Does not exists\nEnter the correct
                   name: ") ;
           }
     }
}
```

## Code : GenerateBarGraph.java

```
// GenerateBarGraph.java

/***********************************************
 *  Program Name: GenerateBarGraph.java
 *  Main Program: ProjectUDT.java
 *
 *  This program is created to aid  my research
on  UDT



 *
 *
 *
 *
 *  @author: Dan Bernardo
 *  @version Last modified Nov 2, 2011
 *

***********************************************/
```

```
import java.awt.Color;

import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.util.ArrayList;
import javax.swing.JPanel;

public class GenerateBarGraph extends JPanel{

        // Declare variables, arrays

         private ArrayList<Double> dataValues;
         private ArrayList<String> dataNames;
         private String graphTitle;
         public GenerateBarGraph(ArrayList<Double> dataValues, ArrayList<String> dataNames, String graphTitle){

             this.dataNames  = dataNames;
             this.dataValues = dataValues;
             this.graphTitle = graphTitle;
         }


/**
      *  Method painComponent

*
      *  Get values and plot

*
      *  @param graphics

*/


         public void paintComponent(Graphics graphics) {

             super.paintComponent(graphics);


            // set
bar_width

             int bar_width = 50;
             Font objeFont_title = new Font("Book Antiqua", Font.BOLD, 15);
             Font font_label = new Font("Book Antiqua", Font.PLAIN, 10);
             FontMetrics fontMetrics_title = graphics.getFontMetrics(objeFont_title);
             FontMetrics fontMetrics_label = graphics.getFontMetrics(font_label);

             Dimension objDimension = getSize();
             int frameWidth = objDimension.width;
             int frameHeight = objDimension.height;

             if (dataValues == null || dataValues.size() == 0)
               return;

```

```
                double minDataValue = 0;
                  double maxDataValue = 0;

                  for (int i = 0; i < dataValues.size();  i++) {
                    if (minDataValue > dataValues.get(i) )
                        minDataValue = dataValues.get(i);
                    if (maxDataValue < dataValues.get(i))
                        maxDataValue = dataValues.get(i);
                  }


                  // Compute and validate values

                  int titleWidth = fontMetrics_title.stringWidth(graphTitle);
                  int q = fontMetrics_title.getAscent();
                  int p = (frameWidth - titleWidth) / 2;
                  graphics.setFont(objeFont_title);
                  graphics.drawString(graphTitle, p, q);

                  int title_height = fontMetrics_title.getHeight();
                  int label_height = fontMetrics_label.getHeight();

                  if (maxDataValue == minDataValue)



                  return;

                  double graphScale = (frameHeight - title_height - label_height) / (maxDataValue -    minDataValue);
                  q = frameHeight - fontMetrics_label.getDescent();

                  graphics.setFont(font_label);

                  for (int j = 0; j < dataValues.size(); j++) {

                    int valueP = j * bar_width + 1;
                    /*
                    if (j%2==0){
                                     valueP +=50;
                    }*/

                    // optional -System.out.println("valueP: " + valueP);
                    int valueQ = title_height;
                    int height = (int) (dataValues.get(j) * graphScale);

                    if (dataValues.get(j) >= 0)
                      valueQ += (int) ((maxDataValue - dataValues.get(j)) * graphScale);
                    else {
                      valueQ += (int) (maxDataValue * graphScale);
                      height = -height;
                    }

                  // Identifies trend by colors

                    if (j % 2 == 0) {
                                     graphics.setColor(Color.BLUE);  //RTT
                    } else {
                                     graphics.setColor(Color.GREEN);  //SendRate
                    }

                  // Plot
      values

                    graphics.fillRect(valueP, valueQ, bar_width - 2, height);
                    graphics.setColor(Color.black);
                    graphics.drawRect(valueP, valueQ, bar_width - 2, height);

                    // Label values

                    int labelWidth = fontMetrics_label.stringWidth(dataNames.get(j));
                    p = j * bar_width + (bar_width - labelWidth) / 2;
                    graphics.drawString(dataNames.get(j), p, q);
                  }

              }

      }
```

## UDT Codes

```
// Simulation program for UDT ( UIC)

//lac.uic.edu //

// Description: the program is used to simulate UDT on NS-2 //

#ifndef __NS_UDT_H__

#define __NS_UDT_H__


#include "agent.h"

#include "packet.h"


const int MAX_LOSS_LEN = 300;


struct hdr_udt

{

   int flag_;

   int seqno_;

   int type_;

   int losslen_;

   int ackseq_;

   int ack_;

   int recv_;

   int rtt_;

   int bandwidth_;

   int loss_[MAX_LOSS_LEN];
};

  static int off_udt_;
  inline static int& offset() { return off_udt_; }
  inline static hdr_udt* access(Packet* p) {return (hdr_udt*) p->access(off_udt_);}
  int& flag() {return flag_;}
  int& seqno() {return seqno_;}
  int& type() {return type_;}
  int& losslen() {return losslen_;}
  int& ackseq() {return ackseq_;}
  int& ack() {return ack_;}
  int& lrecv() {return recv_;}
  int& rtt() {return rtt_;}
  int& bandwidth() {return bandwidth_;};
  int* loss() {return loss_;}
};

class UdtAgent;

class SndTimer: public TimerHandler
{
public:
  SndTimer(UdtAgent *a) : TimerHandler() { a_ = a; }

protected:
  virtual void expire(Event *e);
  UdtAgent *a_;
};
```

```
class SynTimer: public TimerHandler
{
public:
  SynTimer(UdtAgent *a) : TimerHandler() { a_ = a; }

protected:
  virtual void expire(Event *e);
  UdtAgent *a_;
};

class AckTimer: public TimerHandler
{
public:
  AckTimer(UdtAgent *a) : TimerHandler() { a_ = a; }

protected:
  virtual void expire(Event *e);
  UdtAgent *a_;
};

class NakTimer: public TimerHandler
{
public:
  NakTimer(UdtAgent *a) : TimerHandler() { a_ = a; }
```

```
protected:
   virtual void expire(Event *e);
   UdtAgent *a_;
};


class ExpTimer: public TimerHandler
{
public:
   ExpTimer(UdtAgent *a) : TimerHandler() { a_ = a; }


protected:
   virtual void expire(Event *e);
   UdtAgent *a_;
};
class LossList
{
protected:
   const bool greaterthan(const int& seqno1, const int& seqno2) const;
   const bool lessthan(const int& seqno1, const int& seqno2) const;
   const bool notlessthan(const int& seqno1, const int& seqno2) const;
   const bool notgreaterthan(const int& seqno1, const int& seqno2) const;


   const int getLength(const int& seqno1, const int& seqno2) const;


   const int incSeqNo(const int& seqno) const;
   const int decSeqNo(const int& seqno) const;
```

```
protected:
   int seq_no_th_;                  // threshold for comparing seq. no.
   int max_seq_no_;                 // maximum permitted seq. no.
};
////////////////////////////////////////////////////////////////////////////////
class SndLossList: public LossList
{
public:
   SndLossList(const int& size, const int& th, const int& max);
   ~SndLossList();

   int insert(const int& seqno1, const int& seqno2);
   void remove(const int& seqno);
   int getLossLength();
   int getLostSeq();
private:
   int* data1_;                     // sequence number starts
   int* data2_;                     // seqnence number ends
   int* next_;                      // next node in the list
   int head_;                       // first node
   int length_;                     // loss length
   int size_;                       // size of the static array
   int last_insert_pos_;            // position of last insert node
};
```

```
////////////////////////////////////////////////////////////////////////////////

class RcvLossList: public LossList

{

public:

   RcvLossList(const int& size, const int& th, const int& max);

   ~RcvLossList();


   void insert(const int& seqno1, const int& seqno2);

   bool remove(const int& seqno);

   int getLossLength() const;

   int getFirstLostSeq() const;

   void getLossArray(int* array, int* len, const int& limit, const double& interval);


private:

   int* data1_;                    // sequence number starts

   int* data2_;                    // sequence number ends

   double* last_feedback_time_;            // last feedback time of the node

   int* count_;                    // report counter

   int* next_;                     // next node in the list

   int* prior_;                    // prior node in the list;


   int head_;                      // first node in the list

   int tail_;                      // last node in the list;

   int length_;                    // loss length

   int size_;                      // size of the static array

};
```

```
class AckWindow
{
public:
  AckWindow();
  ~AckWindow();

  void store(const int& seq, const int& ack);
  double acknowledge(const int& seq, int& ack);

private:
  int* ack_seqno_;
  int* ack_;
  double* ts_;

  const int size_;

  int head_;
  int tail_;
};
class TimeWindow
{
public:
  TimeWindow();
  ~TimeWindow();
```

```
   int getbandwidth() const;

   int getpktspeed() const;

   bool getdelaytrend() const;

   void pktarrival();

   void ack2arrival(const double& rtt);

   void probe1arrival();

   void probe2arrival();

private:

   const int size_;

   double* pkt_window_;

   int pkt_window_ptr_;

   double* rtt_window_;

   double* pct_window_;

   double* pdt_window_;

   int rtt_window_ptr_;

   double* probe_window_;

   int probe_window_ptr_;

   double last_arr_time_;

   double probe_time_;

   double curr_arr_time_;

   bool first_round_;

};


class UdtAgent: public Agent

{

friend SndTimer;
friend SynTimer;
friend AckTimer;
friend NakTimer;
friend ExpTimer;
public:
  UdtAgent();
  ~UdtAgent();
  int command(int argc, const char*const* argv);
  virtual void recv(Packet*, Handler*);
  virtual void sendmsg(int nbytes, const char *flags = 0);
protected:
  SndTimer snd_timer_;
  SynTimer syn_timer_;
  AckTimer ack_timer_;
  NakTimer nak_timer_;
  ExpTimer exp_timer_;

  double syn_interval_;
  double ack_interval_;
  double nak_interval_;
  double exp_interval_;
  int mtu_;
  int max_flow_window_;
  int flow_window_size_;
  SndLossList* snd_loss_list_;
  RcvLossList* rcv_loss_list_;
```

```
double snd_interval_;

int bandwidth_;

int nak_count_;

int dec_count_;

volatile int snd_last_ack_;

int local_send_;

int local_loss_;

int local_ack_;

volatile int snd_curr_seqno_;

int curr_max_seqno_;

int dec_random_;

int avg_nak_num_;

double loss_rate_limit_;

double loss_rate_;

AckWindow ack_window_;

TimeWindow time_window_;

double rtt_;

double rcv_interval_;

int rcv_last_ack_;

double rcv_last_ack_time_;

int rcv_last_ack2_;

int ack_seqno_;

volatile int rcv_curr_seqno_;

int local_recv_;

int last_dec_seq_;
```

```
   double last_delay_time_;

   double last_dec_int_;

   bool slow_start_;

   bool freeze_;

   bool firstloss_;

protected:

   void rateControl();

   void flowControl();

   void sendCtrl(int pkttype, int lparam = 0, int* rparam = NULL);

   void sendData();

   void timeOut();

};

#endif
```

**Packet.cpp**

```
/*****************************************************************

This file contains the implementation of UDT packet handling modules.

A UDT packet is a 2-dimension vector of packet header and data.

*****************************************************************/

//////////////////////////////////////////////////////////////////////////////

//    0                   1                   2                   3

//    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

//   |                        Packet Header                          |

//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

//   |                                                               |

//   ~              Data / Control Information Field                 ~

//   |                                                               |

//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

//

//    0                   1                   2                   3

//    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

//   |0|                        Sequence Number                      |

//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

//   |ff |o|                     Message Number                      |
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//  |                        Time Stamp                        |
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//
//  bit 0:
//     0: Data Packet
//     1: Control Packet
//  bit ff:
//     11: solo message packet
//     10: first packet of a message
//     01: last packet of a message
//  bit o:
//     0: in order delivery not required
//     1: in order delivery required
//
//   0                   1                   2                   3
//   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//  |1|            Type             |            Reserved          |
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//  |                       Additional Info                       |
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
//  |                        Time Stamp                        |
//  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//
//  bit 1-15:
```

```
//
//
//      0: Protocol Connection Handshake
//          Add. Info:    Undefined
//          Control Info: Handshake information (see CHandShake)
//      1: Keep-alive
//          Add. Info:    Undefined
//          Control Info: None
//      2: Acknowledgement (ACK)
//          Add. Info:    The ACK sequence number
//          Control Info: The sequence number to which (but not include)
//          all the previous packets have beed received
//          Optional:     RTT
//                        RTT Variance
//                        advertised flow window size (number of packets)
//                        estimated bandwidth (number of packets per //second)
//      3: Negative Acknowledgement (NAK)
//          Add. Info:    Undefined
//          Control Info: Loss list (see loss list coding below)
//      4: Congestion Warning
//          Add. Info:    Undefined
//          Control Info: None
//      5: Shutdown
//          Add. Info:    Undefined
//          Control Info: None
//      6: Acknowledgement of Acknowledement (ACK-square)
//          Add. Info:    The ACK sequence number
//          Control Info: None
//      7: Message Drop Request
//              Add. Info:    Message ID
//              Control Info: first sequence number of the message
//                            last seqeunce number of the message
//      65535: Explained by bits 16 - 31
//
//   bit 16 - 31:
//  This space is used for future expansion or user defined control packets.
//
//    0                   1                   2                   3
//    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//   |1|                 Sequence Number a (first)                   |
//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//   |0|                 Sequence Number b (last)                    |
//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//   |0|                 Sequence Number (single)                    |
//   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//
//   Loss List Field Coding:
// For any consecutive lost sequence numbers that the difference /between
//      the last and first is more than 1, only record the first (a) and the
//      the last (b) sequence numbers in the loss list field, and modify the
//      the first bit of a to 1.
//      For any single loss or consecutive loss less than 2 packets, use
//      the original sequence numbers in the field.
```

```
#include "packet.h"

const int CPacket::m_iPktHdrSize = 12;

// Set up the aliases in the constructure

CPacket::CPacket():

m_iSeqNo((int32_t&)(m_nHeader[0])),

m_iMsgNo((int32_t&)(m_nHeader[1])),

m_iTimeStamp((int32_t&)(m_nHeader[2])),

m_pcData((char*&)(m_PacketVector[1].iov_base))

{

   m_PacketVector[0].iov_base = (char *)m_nHeader;

   m_PacketVector[0].iov_len = CPacket::m_iPktHdrSize;

}

CPacket::~CPacket()

{

}

int CPacket::getLength() const

{

   return m_PacketVector[1].iov_len;

}

void CPacket::setLength(const int& len)

{

   m_PacketVector[1].iov_len = len;

}
```

```
void CPacket::pack(const int& pkttype, void* lparam, void* rparam, const int& size)
{
   // Set (bit-0 = 1) and (bit-1~15 = type)
   m_nHeader[0] = 0x80000000 | (pkttype << 16);
   // Set additional information and control information field
   switch (pkttype)
   {
   case 2: //0010 - Acknowledgement (ACK)
      // ACK packet seq. no.
      if (NULL != lparam)
         m_nHeader[1] = *(int32_t *)lparam;
      // data ACK seq. no.
      // optional: RTT (microsends), RTT variance (microseconds) advertised flow window size
(packets), and estimated link capacity (packets per second)
      m_PacketVector[1].iov_base = (char *)rparam;
      m_PacketVector[1].iov_len = size;
      break;
   case 6: //0110 - Acknowledgement of Acknowledgement (ACK-2)
      // ACK packet seq. no.
      m_nHeader[1] = *(int32_t *)lparam;
      // control info field should be none
      // but "writev" does not allow this
      m_PacketVector[1].iov_base = (char *)&__pad; //NULL;
      m_PacketVector[1].iov_len = 4; //0;
      break;
```

```
case 3: //0011 - Loss Report (NAK)

  // loss list

  m_PacketVector[1].iov_base = (char *)rparam;

  m_PacketVector[1].iov_len = size;

  break;

case 4: //0100 - Congestion Warning

  // control info field should be none

  // but "writev" does not allow this

  m_PacketVector[1].iov_base = (char *)&__pad; //NULL;

  m_PacketVector[1].iov_len = 4; //0

  break;

case 1: //0001 - Keep-alive

  // control info field should be none

  // but "writev" does not allow this

  m_PacketVector[1].iov_base = (char *)&__pad; //NULL;

  m_PacketVector[1].iov_len = 4; //0

  break;

case 0: //0000 - Handshake

  // control info filed is handshake info

  m_PacketVector[1].iov_base = (char *)rparam;

  m_PacketVector[1].iov_len = size; //sizeof(CHandShake);

  break;

case 5: //0101 - Shutdown

  // control info field should be none

  // but "writev" does not allow this

  m_PacketVector[1].iov_base = (char *)&__pad; //NULL;
```

```
    m_PacketVector[1].iov_len = 4; //0
    break;
  case 7: //0111 - Message Drop Request
    // msg id
    m_nHeader[1] = *(int32_t *)lparam;
    //first seq no, last seq no
    m_PacketVector[1].iov_base = (char *)rparam;
    m_PacketVector[1].iov_len = size;
    break;
  case 65535: //0x7FFF - Reserved for user defined control packets
    // for extended control packet
    // "lparam" contains the extneded type information for bit 4 - 15
    // "rparam" is the control information
    m_nHeader[0] |= (*(int32_t *)lparam) << 16;
    if (NULL != rparam)
    {
      m_PacketVector[1].iov_base = (char *)rparam;
      m_PacketVector[1].iov_len = size;
    }
    else
    {
      m_PacketVector[1].iov_base = (char *)&__pad;
      m_PacketVector[1].iov_len = 4;
    }
```

```
      break;
   default:
      break;
   }
}
iovec* CPacket::getPacketVector()
{
   return m_PacketVector;
}
int CPacket::getFlag() const
{
   // read bit 0
   return m_nHeader[0] >> 31;
}
int CPacket::getType() const
{
   // read bit 1~15
   return (m_nHeader[0] >> 16) & 0x00007FFF;
}
int CPacket::getExtendedType() const
{
   // read bit 16~31
   return m_nHeader[0] & 0x0000FFFF;
}
```

```
int32_t CPacket::getAckSeqNo() const
{
  // read additional information field
  return m_nHeader[1];
}
int CPacket::getMsgBoundary() const
{
  // read [1] bit 0~1
  return m_nHeader[1] >> 30;
}
bool CPacket::getMsgOrderFlag() const
{
  // read [1] bit 2
  return (1 == ((m_nHeader[1] >> 29) & 1));
}
int32_t CPacket::getMsgSeq() const
{
  // read [1] bit 3~31
  return m_nHeader[1] & 0x1FFFFFFF;
}
```

**Sendfile.cpp**

```
#ifndef __WIN32
#include <cstdlib>
#endif
#include <fstream>
```

```
#include <iostream>
#include <udt.h>
using namespace std;
int main(int argc, char* argv[])
{
  //usage: sendfile [server_port]
  if ((2 < argc) || ((2 == argc) && (0 == atoi(argv[1]))))
  {
    cout << "usage: sendfile [server_port]" << endl;
    return 0;
  }
  UDTSOCKET serv = UDT::socket(AF_INET, SOCK_STREAM, 0);
#ifdef WIN32
  int mss = 1052;
  UDT::setsockopt(serv, 0, UDT_MSS, &mss, sizeof(int));
#endif
  short port = 9000;
  if (2 == argc)
    port = atoi(argv[1]);
  sockaddr_in my_addr;
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(port);
  my_addr.sin_addr.s_addr = INADDR_ANY;
  memset(&(my_addr.sin_zero), '\0', 8);
  if (UDT::ERROR == UDT::bind(serv, (sockaddr*)&my_addr, sizeof(my_addr)))

  {

    cout << "bind: " << UDT::getlasterror().getErrorMessage() << endl;

    return 0;

  }

  cout << "server is ready at port: " << port << endl;

  UDT::listen(serv, 1);

  sockaddr_in their_addr;

  int namelen = sizeof(their_addr);

  UDTSOCKET fhandle;

  if (UDT::INVALID_SOCK == (fhandle = UDT::accept(serv, (sockaddr*)&their_addr,
&namelen)))

  {

    cout << "accept: " << UDT::getlasterror().getErrorMessage() << endl;

    return 0;

  }

  UDT::close(serv);

  // aquiring file name information from client

  char file[1024];

  int len;

  if (UDT::ERROR == UDT::recv(fhandle, (char*)&len, sizeof(int), 0))

  {

    cout << "recv: " << UDT::getlasterror().getErrorMessage() << endl;

    return 0;

  }
```

```
if (UDT::ERROR == UDT::recv(fhandle, file, len, 0))
{
   cout << "recv: " << UDT::getlasterror().getErrorMessage() << endl;
   return 0;
}
file[len] = '\0';
// open the file
ifstream ifs(file, ios::in | ios::binary);
ifs.seekg(0, ios::end);
int64_t size = ifs.tellg();
ifs.seekg(0, ios::beg);
// send file size information
if (UDT::ERROR == UDT::send(fhandle, (char*)&size, sizeof(int64_t), 0))
{
   cout << "send: " << UDT::getlasterror().getErrorMessage() << endl;
   return 0;
}
UDT::TRACEINFO trace;
UDT::perfmon(fhandle, &trace);
// send the file
if (UDT::ERROR == UDT::sendfile(fhandle, ifs, 0, size))
{
   cout << "sendfile: " << UDT::getlasterror().getErrorMessage() << endl;
   return 0;
}
```

```
  UDT::perfmon(fhandle, &trace);
   cout << "speed = " << trace.mbpsSendRate << endl;
   UDT::close(fhandle);
   ifs.close();
   return 1;
}
```

**Recvfile.cpp**

```
#ifndef WIN32
#include <arpa/inet.h>
#endif
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <udt.h>
using namespace std;
int main(int argc, char* argv[])
{
   if ((argc != 5) || (0 == atoi(argv[2])))
   {
      cout << "usage: recvfile server_ip server_port remote_filename local_filename" << endl;
      return 0;
   }
   UDTSOCKET fhandle = UDT::socket(AF_INET, SOCK_STREAM, 0);
   sockaddr_in serv_addr;
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(short(atoi(argv[2])));
```

```
#ifndef WIN32
  if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0)
#else
  if (INADDR_NONE == (serv_addr.sin_addr.s_addr = inet_addr(argv[1])))
#endif
  {
    cout << "incorrect network address.\n";
    return 0;
  }
  memset(&(serv_addr.sin_zero), '\0', 8);
  if (UDT::ERROR == UDT::connect(fhandle, (sockaddr*)&serv_addr, sizeof(serv_addr)))
  {
    cout << "connect: " << UDT::getlasterror().getErrorMessage() << endl;
    return 0;
  }
  // send name information of the requested file
  int len = strlen(argv[3]);
  if (UDT::ERROR == UDT::send(fhandle, (char*)&len, sizeof(int), 0))
  {
    cout << "send: " << UDT::getlasterror().getErrorMessage() << endl;
    return 0;
  }
  if (UDT::ERROR == UDT::send(fhandle, argv[3], len, 0))
  {
    cout << "send: " << UDT::getlasterror().getErrorMessage() << endl;
    return 0;
```

```
   }

   // get size information
   int64_t size;
   if (UDT::ERROR == UDT::recv(fhandle, (char*)&size, sizeof(int64_t), 0))
   {
      cout << "send: " << UDT::getlasterror().getErrorMessage() << endl;
      return 0;
   }

   // receive the file
   ofstream ofs(argv[4], ios::out | ios::binary | ios::trunc);
   int64_t recvsize;
   if (UDT::ERROR == (recvsize = UDT::recvfile(fhandle, ofs, 0, size)))
   {
      cout << "recvfile: " << UDT::getlasterror().getErrorMessage() << endl;
      return 0;
   }

   UDT::close(fhandle);
   ofs.close();
   return 1;
}
```

cc.h

```
/************************************************************

DISCLAIMER: The algorithms implemented using UDT/CCC in this file may be modified. These
modifications may NOT necessarily reflect the view of the algorithms' original authors.

***************/

#ifndef WIN32

  #include <sys/time.h>

  #include <time.h>

#endif



#include <cmath>

#include <vector>

#include <algorithm>

#include <window.h>

#include <ccc.h>

#include <udt.h>

using namespace std;

/*********************************************************************

TCP congestion control

Reference:

M. Allman, V. Paxson, W. Stevens (consultant), TCP Congestion Control, RFC  2581, April 1999.

Note:

This base TCP control class can be used to derive new TCP variants, including those
implemented in this file: HighSpeed, Scalable, BiC, Vegas, and FAST.

*********************************************************************/
```

```
class CTCP: public CCC
{
public:
   void init()
   {
      m_bSlowStart = true;
      m_issthresh = 83333;
      m_dPktSndPeriod = 0.0;
      m_dCWndSize = 2.0;
      setACKInterval(2);
      setRTO(1000000);
   }
   virtual void onACK(const int& ack)
   {
      if (ack == m_iLastACK)
      {
         if (3 == ++ m_iDupACKCount)
            DupACKAction();
         else if (m_iDupACKCount > 3)
            m_dCWndSize += 1.0;
         else
            ACKAction();
      }
      else
      {
         if (m_iDupACKCount >= 3)
```

```
        m_dCWndSize = m_issthresh;
        m_iLastACK = ack;
        m_iDupACKCount = 1;
        ACKAction();
      }
   }
   virtual void onTimeout()
   {
      m_issthresh = getPerfInfo()->pktFlightSize / 2;
      if (m_issthresh < 2)
      m_issthresh = 2;
      m_bSlowStart = true;
      m_dCWndSize = 2.0;
   }
protected:
   virtual void ACKAction()
   {
      if (m_bSlowStart)
      {
        m_dCWndSize += 1.0;
        if (m_dCWndSize >= m_issthresh)
        m_bSlowStart = false;
      }
      else
        m_dCWndSize += 1.0/m_dCWndSize;
   }
```

```
  virtual void DupACKAction()
  {
    m_bSlowStart = false;
    m_issthresh = getPerfInfo()->pktFlightSize / 2;
    if (m_issthresh < 2)
      m_issthresh = 2;
      m_dCWndSize = m_issthresh + 3;
  }
protected:
  int m_issthresh;
  bool m_bSlowStart;
  int m_iDupACKCount;
  int m_iLastACK;
};
/*****************************************************************
Scalable TCP congestion control
Reference:
Tom Kelly, Scalable TCP: Improving Performance in Highspeed Wide Area  Networks, Computer
Communication Review, Vol. 33 No. 2 - April 2003
*****************************************************************/
class CScalableTCP: public CTCP
{
protected:
  virtual void ACKAction()
  {
    if (m_dCWndSize <= 38.0)
      CTCP::ACKAction();
    else
    {
      if (m_bSlowStart)
        m_dCWndSize += 1.0;
      else
        m_dCWndSize += 0.01;
    }
    if (m_dCWndSize > m_iMaxCWndSize)
      m_dCWndSize = m_iMaxCWndSize;
  }
  virtual void DupACKAction()
  {
    if (m_dCWndSize <= 38.0)
      m_dCWndSize *= 0.5;
    else
      m_dCWndSize *= 0.875;
    if (m_dCWndSize < m_iMinCWndSize)
      m_dCWndSize = m_iMinCWndSize;
  }
private:
  static const int m_iMinCWndSize = 16;
  static const int m_iMaxCWndSize = 100000;
  static const int m_iCWndThresh = 38;
};
```

```
/*****************************************************************
HighSpeed TCP congestion control
Reference:
Sally Floyd, HighSpeed TCP for Large Congestion Windows, RFC 3649,
Experimental, December 2003
*****************************************************************/
class CHSTCP: public CTCP
{
public:
  virtual void ACKAction()
  {
    m_dCWndSize += a(m_dCWndSize)/m_dCWndSize;
  }
  virtual void DupACKAction()
  {
    m_dCWndSize -= m_dCWndSize*b(m_dCWndSize);
  }
private:
  double a(double w)
  {
    return (w * w * 2.0 * b(w)) / ((2.0 - b(w)) * pow(w, 1.2) * 12.8);
  }
  double b(double w)
  {
    return (0.1 - 0.5) * (log(w) - log(38.)) / (log(83000.) - log(38.)) + 0.5;
```

```
   }


private:

   static const int m_iHighWnd = 83000;

   static const int m_iLowWnd = 38;

};

/*****************************************************************

BiC TCP congestion control

Reference:

Lisong Xu, Khaled Harfoush, and Injong Rhee, "Binary Increase Congestion  Control for Fast
Long-Distance Networks", INFOCOM 2004.

*****************************************************************/

class CBiCTCP: public CTCP

{

public:

   CBiCTCP()

   {

      m_dMaxWin = m_iDefaultMaxWin;

      m_dMinWin = m_dCWndSize;

      m_dTargetWin = (m_dMaxWin + m_dMinWin) / 2;



      m_dSSCWnd = 1.0;

      m_dSSTargetWin = m_dCWndSize + 1.0;

   }

protected:

   virtual void ACKAction()
```

```
{
  if (m_dCWndSize < m_iLowWindow)
  {
    m_dCWndSize += 1/m_dCWndSize;
    return;
  }
  if (!m_bSlowStart)
  {
    if (m_dTargetWin - m_dCWndSize < m_iSMax)
      m_dCWndSize += (m_dTargetWin - m_dCWndSize)/m_dCWndSize;
    else
      m_dCWndSize += m_iSMax/m_dCWndSize;
    if (m_dMaxWin > m_dCWndSize)
    {
      m_dMinWin = m_dCWndSize;
      m_dTargetWin = (m_dMaxWin + m_dMinWin) / 2;
    }
    else
    {
      m_bSlowStart = true;
      m_dSSCWnd = 1.0;
      m_dSSTargetWin = m_dCWndSize + 1.0;
      m_dMaxWin = m_iDefaultMaxWin;
    }
  }
  else
  {
    m_dCWndSize += m_dSSCWnd/m_dCWndSize;
    if(m_dCWndSize >= m_dSSTargetWin)
    {
      m_dSSCWnd *= 2;
      m_dSSTargetWin = m_dCWndSize + m_dSSCWnd;
    }
    if(m_dSSCWnd >= m_iSMax)
      m_bSlowStart = false;
  }
}

virtual void DupACKAction()
{
  if (m_dCWndSize <= m_iLowWindow)
    m_dCWndSize *= 0.5;
  else
  {
    m_dPreMax = m_dMaxWin;
    m_dMaxWin = m_dCWndSize;
    m_dCWndSize *= 0.875;
    m_dMinWin = m_dCWndSize;

    if (m_dPreMax > m_dMaxWin)
    {
      m_dMaxWin = (m_dMaxWin + m_dMinWin) / 2;
```

```
            m_dTargetWin = (m_dMaxWin + m_dMinWin) / 2;
         }
      }
   }
private:
   static const int m_iLowWindow = 38;
   static const int m_iSMax = 32;
   static const int m_iSMin = 1;
   static const int m_iDefaultMaxWin = 1 << 29;
   double m_dMaxWin;
   double m_dMinWin;
   double m_dPreMax;
   double m_dTargetWin;
   double m_dSSCWnd;
   double m_dSSTargetWin;
};
/*****************************************************************
TCP Westwood
reference:
http://www.cs.ucla.edu/NRL/hpi/tcpw/
*****************************************************************/
class CWestwood: public CTCP
{
public:
   CWestwood(): m_dBWE(1), m_dLastBWE(1), m_dBWESample(1), m_dLastBWESample(1)
   {
      gettimeofday(&m_LastACKTime, 0);
   }
   virtual void onACK(const int& ack)
   {
      timeval currtime;
      gettimeofday(&currtime, 0);
m_dBWESample = double(ack - m_iLastACK) / double((currtime.tv_sec –
m_LastACKTime.tv_sec) * 1000.0 + (currtime.tv_usec - m_LastACKTime.tv_usec) / 1000.0);
m_dBWE = 19.0/21.0 * m_dLastBWE + 1.0/21.0 * (m_dBWESample + m_dLastBWESample);

      m_LastACKTime = currtime;
      m_dLastBWE = m_dBWE;
      m_dLastBWESample = m_dBWESample;

      if (ack == m_iLastACK)
      {
         if (3 == ++ m_iDupACKCount)
         {
            m_bSlowStart = false;
            m_issthresh = int(ceil(getPerfInfo()->msRTT * m_dBWE));
            if (m_issthresh < 2)
            m_issthresh = 2;
            m_dCWndSize = m_issthresh + 3;
         }
         else if (m_iDupACKCount > 3)
            m_dCWndSize += 1.0;
         else
            ACKAction();
      }
      else
      {
         if (m_iDupACKCount >= 3)
            m_dCWndSize = m_issthresh;
            m_iLastACK = ack;
            m_iDupACKCount = 1;
         ACKAction();
      }
   }
   virtual void onTimeout()
   {
      m_issthresh = int(ceil(getPerfInfo()->msRTT * m_dBWE));
      if (m_issthresh < 2)
      m_issthresh = 2;
      m_bSlowStart = true;
      m_dCWndSize = 2.0;
   };
private:
   double m_dBWE, m_dLastBWE;
   double m_dBWESample, m_dLastBWESample;
   timeval m_LastACKTime;
};
```

```
/*****************************************************************

TCP Vegas

Reference:

L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection
and avoidance. In Proceedings of the SIGCOMM '94 Symposium (Aug. 1994) pages 24-35.

Note:

This class can be used to derive new delay based approaches, e.g., FAST.

*****************************************************************************/

class CVegas: public CTCP

{

public:

   CVegas()

   {

      m_iSSRound = 1;

      m_iRTT = 1000000;

      m_iBaseRTT = 1000000;

      gettimeofday(&m_LastCCTime, 0);

      m_iPktSent = 0;

      m_pAckWindow = new CACKWindow(100000);

   }
   ~CVegas()
   {
      delete m_pAckWindow;
   }
   virtual void onACK(const int& seq)
   {
      double expected, actual, diff; //kbps
      int rtt = m_pAckWindow->acknowledge(seq, const_cast<int&>(seq));
      if (rtt > 0)
          m_iRTT = (m_iRTT * 15 + rtt) >> 4;
          timeval currtime;
          gettimeofday(&currtime, 0);
          if ((currtime.tv_sec - m_LastCCTime.tv_sec) * 1000000 + (currtime.tv_usec -
            m_LastCCTime.tv_usec) < m_iRTT)
         return;
      expected = m_dCWndSize * 1000.0 / m_iBaseRTT;
actual = m_iPktSent / ((currtime.tv_sec - m_LastCCTime.tv_sec) * 1000.0 + (currtime.tv_usec -
m_LastCCTime.tv_usec) / 1000.0);
      diff = expected - actual;
      if (m_bSlowStart)
      {
         if (diff < gamma)
            m_bSlowStart = false;
         if (m_iSSRound & 1)
            m_dCWndSize *= 2;
         m_iSSRound ++;
```

```
    }
    else
    {
      if (diff < alpha)
        m_dCWndSize += 1.0;
      else if (diff > beta)
        m_dCWndSize -= 1.0;
    }
    gettimeofday(&m_LastCCTime, 0);
    m_iPktSent = 0;
    if (m_iBaseRTT > m_iRTT)
      m_iBaseRTT = m_iRTT;
  }
  virtual void onPktSent(const CPacket* pkt)
  {
    m_pAckWindow->store(pkt->m_iSeqNo, pkt->m_iSeqNo);
    m_iPktSent ++;
  }
  virtual void onTimeout()
  {
  }
protected:
  int m_iSSRound;
  int m_iRTT;
  int m_iBaseRTT;
  timeval m_LastCCTime;
```

```
   int m_iPktSent;

   static const int alpha = 30; //kbps

   static const int beta = 60;  //kbps

   static const int gamma = 30; //kbps

   CACKWindow* m_pAckWindow;

};

/*****************************************************************************

FAST TCP

Reference:

1. C. Jin, D. X. Wei and S. H. Low, "FAST TCP: motivation, architecture, algorithms,
performance", IEEE Infocom, March 2004

2. C. Jin, D. X. Wei and S. H. Low, FAST TCP for High-Speed Long-Distance Networks, Internet
Draft, draft-jwl-tcp-fast-01.txt,

   http://netlab.caltech.edu/pub/papers/draft-jwl-tcp-fast-01.txt

Note:

   Precision of RTT measurement may make great difference in the throughput

*****************************************************************************/

class CFAST: public CVegas

{

public:

   CFAST()

   {

      m_dOldWin = m_dCWndSize;

      m_iNumACK = 100000;

   }
   virtual void onACK(const int& ack)
   {
      if (ack == m_iLastACK)
      {
         if (3 == ++ m_iDupACKCount)
         {
            m_dCWndSize *= 0.875;
            return;
         }
      }
      else
      {
         if (m_iDupACKCount >= 3)
         {
//          m_dCWndSize = m_issthresh;
//          return;
         }
         m_iLastACK = ack;
         m_iDupACKCount = 1;
      }
      if (0 == (++ m_iACKCount % m_iNumACK))
         m_dCWndSize += m_iIncDec;
      int rtt = m_pAckWindow->acknowledge(ack, const_cast<int&>(ack));
      if (rtt > 0)
         m_iRTT = (m_iRTT * 7 + rtt) >> 3;
      timeval currtime;

      gettimeofday(&currtime, 0);

if ((currtime.tv_sec - m_LastCCTime.tv_sec) * 1000000 + (currtime.tv_usec
m_LastCCTime.tv_usec) < 2 * m_iRTT)

        return;

  m_dNewWin = 0.5 * (m_dOldWin + (double(m_iBaseRTT) / m_iRTT) * m_dCWndSize + alpha);

      if (m_dNewWin > 2.0 * m_dCWndSize)

        m_dNewWin = 2.0 * m_dCWndSize;

         m_iNumACK = int(ceil(fabs(m_dCWndSize / (m_dNewWin - m_dCWndSize)) / 2.0));

      if (m_dNewWin > m_dCWndSize)

         m_iIncDec = 1;

      else

         m_iIncDec = -1;

         m_dOldWin = m_dCWndSize;

         gettimeofday(&m_LastCCTime, 0);

          m_iPktSent = 0;

      if (m_iBaseRTT > m_iRTT)

          m_iBaseRTT = m_iRTT;

   }

private:

   static const int alpha = 200;

   double m_dOldWin;

   double m_dNewWin;

   int m_iNumACK;

   int m_iIncDec;

   int m_iACKCount;
```

```
};

/*****************************************************************************

Reliable UDP Blast

Note:

The class demostrates the simplest control mechanism. The sending rate can be set at any time
by using setRate().

*****************************************************************************/

class CUDPBlast: public CCC

{

public:

   CUDPBlast()

   {

      m_dPktSndPeriod = 1000000;

      m_dCWndSize = 83333.0;

   }

public:

   void setRate(int mbps)

   {

      m_dPktSndPeriod = (m_iSMSS * 8.0) / mbps;

   }

protected:

  static const int m_iSMSS = 1500;

};
```

```
/*****************************************************************************

Group Transport Protocol

Reference:

Ryan X. Wu, and Andrew Chien, "GTP: Group Transport Protocol for Lambda-GRIDs", in
Proceedings of the 4th IEEE/ACM International Symposium on

Cluster Computing and the GRID (CCGRID), April 2004

Note: This is a demotration showing how to use UDT/CCC to implement group-based control
mechanisms, such GTP and CM.

*****************************************************************************/

struct gtpcomp;

class CGTP: public CCC

{

friend struct gtpcomp;

public:

  virtual void init()

  {

    m_dRequestRate = 1;

    m_llLastRecvPkt = 0;

    gettimeofday(&m_LastGCTime, 0);

    m_GTPSet.insert(this);

    rateAlloc();

  }

  virtual void close()

  {

    m_GTPSet.erase(this);

    rateAlloc();
  }

  virtual void onPktReceived()

  {

    timeval currtime;

    gettimeofday(&currtime, 0);

int interval = (currtime.tv_sec - m_LastGCTime.tv_sec) * 1000000 + currtime.tv_usec -
m_LastGCTime.tv_usec;

    if (interval < 2 * m_iRTT)

      return;

    const UDT::TRACEINFO* info = getPerfInfo();

    double realrate, lossrate = 0;

    realrate = (info->pktRecvTotal - m_llLastRecvPkt) * 1500 * 8.0 / interval;

    if (info->pktRecvTotal != m_llLastRecvPkt)

lossrate = double(info->pktRcvLossTotal - m_iLastRcvLoss) / (info->pktRecvTotal -
m_llLastRecvPkt);

    if (0 == lossrate)

      m_dRequestRate *= 1.02;

    else if (lossrate * 0.5 < 0.125)

      m_dRequestRate *= (1 - lossrate * 0.5);

    else

      m_dRequestRate *= 0.875;

    if (m_dRequestRate > m_dTargetRate)

      m_dRequestRate = m_dTargetRate;

      requestRate(int(m_dRequestRate));

    m_llLastRecvPkt = info->pktRecvTotal;

    m_iLastRcvLoss = info->pktRcvLossTotal;

    m_LastGCTime = currtime;
    m_iRTT = int(info->msRTT * 1000);
  }
  virtual void processCustomPkt(CPacket* pkt)
  {
    if (m_iGTPPktType != pkt->getExtendedType())
      return;
    m_dPktSndPeriod = (1500 * 8.0) / *(int *)(pkt->m_pcData);
  }
public:
  void setBandwidth(const double& mbps)
  {
    m_dBandwidth = mbps;
  }
private:
  void rateAlloc();
  void requestRate(int mbps)
  {
    CPacket pkt;
    pkt.pack(0x111, const_cast<void*>((void*)&m_iGTPPktType), &mbps, sizeof(int));
    sendCustomMsg(pkt);
  }
private:
  double m_dTargetRate;
  double m_dBandwidth;
  double m_dRequestRate;
```

```
   timeval m_LastGCTime;
   int64_t m_llLastRecvPkt;
   int m_iLastRcvLoss;
   int m_iRTT;
private:
   static set<CGTP*> m_GTPSet;
   static const int m_iGTPPktType = 0xFFF;
};
set<CGTP*> CGTP::m_GTPSet;
struct gtpcomp
{
  bool operator()(const CGTP* g1, const CGTP* g2) const
  {
    return g1->m_dBandwidth < g2->m_dBandwidth;
  }
};
void CGTP::rateAlloc()
{
   if (0 == m_GTPSet.size())
      return;
   vector<CGTP*> GTPVec;
   copy(m_GTPSet.begin(), m_GTPSet.end(), GTPVec.begin());
   sort(GTPVec.begin(), GTPVec.end(), gtpcomp());
   int N = GTPVec.size();
   int n = 0;
   vector<CGTP*>::iterator i = GTPVec.begin();
  double availbw = (*(i + N - 1))->m_dBandwidth;

  double fairshare = availbw / N;

  while ((n < N) && ((*i)->m_dBandwidth < fairshare))

  {

    (*i)->m_dTargetRate = (*i)->m_dBandwidth;

    availbw -= (*i)->m_dTargetRate;

    fairshare = availbw / (N - n);

    ++ n;

    ++ i;

  }

  for (; i != GTPVec.end(); ++ i)

    (*i)->m_dTargetRate = fairshare;

}

/*****************************************************************************

Protocol using reliable control channel

Note:

The feedback method using sendCustomMsg() as shown in CGTP sends data using unreliable
channel. If some protocol nees reliable channel to  transfer control message, a seperate TCP
connection can be sarted.

The CReliableChannel class below can be used to derive such protocols.

*****************************************************************************/

class CReliableChannel: public CCC

{

public:

  int startTCPServer(sockaddr* addr)

  {

    if (-1 == (m_TCPSocket = socket(AF_INET, SOCK_STREAM, 0)))
    return -1;
  if (-1 == (bind(m_TCPSocket, addr, sizeof(sockaddr_in))))
    return -1;
  if (-1 == (listen(m_TCPSocket, 10)))
    return -1;
  if (-1 == (m_TCPSocket = accept(m_TCPSocket, NULL, NULL)))
    return -1;
  #ifndef WIN32
    pthread_create(&m_TCPThread, NULL, TCPProcessing, this);
  #endif
  return 0;
}
int startTCPClient(sockaddr* addr)
{
  if (-1 == (m_TCPSocket = socket(AF_INET, SOCK_STREAM, 0)))
    return -1;
  if (-1 == (connect(m_TCPSocket, addr, sizeof(sockaddr_in))))
    return -1;
  #ifndef WIN32
    pthread_create(&m_TCPThread, NULL, TCPProcessing, this);
  #endif
  return 0;
}
int sendReliableMsg(const char* data, const int& size)
{
  return send(m_TCPSocket, data, size, 0);
  }
protected:
  virtual void processRealiableMsg()
  {
    char data[1500];
    while (true)
    {
      recv(m_TCPSocket, data, 1500, 0);
      //process data
    }
  }
protected:
  int m_TCPSocket;
  pthread_t m_TCPThread;
private:
  static void* TCPProcessing(void* self)
  {
                        ((CReliableChannel*)self)->processRealiableMsg();
    return NULL;
  }
};
```

# Appendix B

```
C:\test>appclient
usage: appclient server_ip server_port

C:\test>appclient 172.22.42.57 9000
```

```
                                  CWn   PktSndPeriod(us                 RecvNA
SendRate(Mb/s)    RTT(ms)         d     )                  RecvACK      K

      0.367358        100         32                  1          1           0
       3.00126     12.832        307            3872.83         28          12
       1.88694      0.734        194            4536.45         29           2
       2.07238      0.268        161               2192         29           3
       2.70141      0.281        144            3336.36         34          12
       2.26593      0.267        143             3775.9         16           2
       2.33046      0.246        136               1927         30           4
       2.69334      0.266        132               3803         32          14
       2.75781      0.286        133            2255.17         34          12
       2.79011      0.276        130               2725         35          16
       2.36041      0.311        123               3222         18          10
       2.99447      0.308        126               3178         34          17
       2.39502      0.246        139            3963.83         34           9
       2.77402       0.27        128            2847.37         32          14
       2.68526      0.291        132               3919         22          12
       2.97555      0.309        132               2771         33          15
       2.89493        0.3        134               2788         34          19
       3.01587      0.296        122               3378         34          20
       2.94331      0.292        137            2868.09         34          13
       3.00723      0.308        127               3301         34          21
       2.74223      0.321        128            2250.42         26          17
       2.86267        0.3        135            2923.23         35          11
       2.92716      0.321        135               2063         34          15
       3.00783      0.331        132               2539         35          16
       2.44334      0.288        162            3603.11         36           9
       2.83848      0.296        129            3458.56         33          15
       2.70136      0.315        129               3411         29          12
       2.53208        0.3        141            3041.21         26          10
       2.83847      0.301        131            2828.63         34          15
       3.12876      0.343        136            2920.17         34          15
       3.16103      0.328        129               2628         34          18
       2.83572      0.308        126               3747         35          15
       2.80085      0.298        129            3164.47         34          16
       2.80619        0.3        142               3505         33          15
       2.88691      0.315        121               3702         22          11
       2.71752      0.331        128            3083.68         30          15
       2.76569      0.261        123            3570.48         34          19
       2.97577      0.281        128            3214.58         36          18
       2.78203      0.275        124             3225.5         33          19
```

| | | | | | |
|---|---|---|---|---|---|
| 2.52399 | 0.309 | 127 | 3665 | 36 | 14 |
| 2.53201 | 0.309 | 124 | 3861.05 | 35 | 14 |
| 2.62078 | 0.332 | 125 | 3569 | 21 | 12 |
| 2.31433 | 0.305 | 115 | 3818 | 30 | 15 |
| 2.59655 | 0.313 | 115 | 3734.01 | 33 | 17 |
| 2.3382 | 0.307 | 122 | 3969 | 34 | 15 |
| 2.88724 | 0.348 | 122 | 2974.28 | 36 | 20 |
| 2.45949 | 0.291 | 120 | 4605.45 | 34 | 16 |
| 2.33848 | 0.272 | 119 | 1946 | 35 | 13 |
| 2.73368 | 0.338 | 116 | 3469 | 22 | 17 |
| 2.51592 | 0.302 | 122 | 3591.25 | 34 | 16 |
| 2.32238 | 0.319 | 127 | 3684.68 | 39 | 14 |
| 2.66108 | 0.293 | 117 | 3743 | 34 | 16 |
| 2.55376 | 0.315 | 122 | 3509 | 35 | 14 |
| 1.92913 | 0.281 | 116 | 4653.46 | 32 | 4 |
| 1.85466 | 0.296 | 121 | 4749 | 31 | 5 |
| 2.6853 | 0.34 | 122 | 2486 | 33 | 17 |
| 2.20949 | 0.323 | 130 | 4084 | 20 | 12 |
| 2.23368 | 0.295 | 129 | 4266 | 30 | 10 |
| 2.31433 | 0.299 | 124 | 3082.31 | 18 | 7 |
| 2.78202 | 0.318 | 121 | 3256 | 33 | 18 |
| 2.47561 | 0.27 | 132 | 2763.42 | 32 | 9 |
| 2.4675 | 0.285 | 134 | 3056.21 | 33 | 11 |
| 2.62079 | 0.302 | 128 | 4355 | 31 | 15 |
| 2.85465 | 0.312 | 131 | 3478.09 | 144 | 76 |
| 2.96672 | 0.278 | 128 | 2211 | 32 | 18 |
| 3.09735 | 0.345 | 125 | 2536 | 36 | 18 |
| 2.83041 | 0.392 | 132 | 3481 | 35 | 17 |
| 2.80587 | 0.343 | 124 | 2360 | 34 | 15 |
| 3.01623 | 0.322 | 135 | 2618 | 21 | 12 |
| 2.18529 | 0.295 | 127 | 3487 | 28 | 12 |
| 2.52399 | 0.284 | 128 | 3512 | 32 | 11 |
| 2.86266 | 0.302 | 128 | 4036 | 31 | 18 |
| 2.65303 | 0.287 | 136 | 3696.37 | 31 | 14 |
| 2.79381 | 0.275 | 130 | 4045.78 | 109 | 39 |
| 2.69334 | 0.317 | 122 | 3197.24 | 38 | 13 |
| 3.29 | 0.327 | 130 | 2740 | 34 | 19 |
| 2.67722 | 0.282 | 125 | 3848 | 33 | 14 |
| 2.98359 | 0.303 | 149 | 1867 | 34 | 15 |
| 3.2659 | 0.479 | 142 | 3136 | 37 | 20 |
| 2.71752 | 0.293 | 134 | 3482.44 | 34 | 14 |
| 2.12077 | 0.375 | 132 | 3743 | 18 | 10 |
| 2.91081 | 0.339 | 127 | 3883 | 34 | 15 |
| 2.79033 | 0.299 | 146 | 1944 | 34 | 13 |
| 3.18523 | 0.376 | 129 | 3048 | 37 | 20 |
| 2.87067 | 0.32 | 126 | 2652.88 | 35 | 13 |
| 2.79014 | 0.293 | 135 | 1948 | 34 | 14 |
| 3.00768 | 0.688 | 200 | 3258.32 | 35 | 19 |
| 3.33051 | 0.346 | 131 | 2648 | 35 | 19 |
| 2.80621 | 0.314 | 131 | 3250 | 35 | 13 |
| 3.11265 | 0.35 | 129 | 2685 | 35 | 13 |
| 2.33045 | 0.31 | 137 | 3336 | 16 | 9 |

| | | | | | |
|---|---|---|---|---|---|
| 3.12013 | 0.329 | 133 | 2547 | 35 | 15 |
| 2.52189 | 0.322 | 138 | 3946.66 | 19 | 9 |
| 2.98666 | 0.416 | 140 | 3687.37 | 36 | 18 |
| 2.60461 | 0.291 | 135 | 3903.13 | 36 | 13 |
| 2.45142 | 0.265 | 135 | 3672 | 33 | 7 |
| 2.79008 | 0.302 | 127 | 2798.33 | 34 | 15 |
| 2.95137 | 0.3 | 124 | 3552.46 | 34 | 18 |
| 2.85437 | 0.549 | 136 | 2328 | 40 | 12 |
| 3.05644 | 0.302 | 127 | 3246.43 | 34 | 18 |
| 3.08039 | 0.28 | 128 | 3120.24 | 27 | 16 |