Active Network Approach to the Design Of Secure Online Auction Systems

by

Basem Shihada

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
March 2001

© Copyright by Basem Shihada, 2001

DALHOUSIE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "Active Network Approach to the Design of Secure Online Auction Systems" by Basem Shihada in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated _______________

Supervisor: Dr. Sampalli Srinivas _______________________________

Readers: Dr. Nur Zincir-Heywood _______________________________
(Department Member)

Mr. John Statton _______________________________
(Innovatia, Aliant Inc. – External)

Dr. Dawn Jutla _______________________________
(St. Mary's University – External)

DALHOUSIE UNIVERSITY

DATE: April 04, 2001

AUTHOR: Basem Shihada

TITLE: Active Network Approach to the Design Of Secure Online Auction Systems

DEPARTMENT: Computer Science

DEGREE: Master of Computer Science **CONVOCATION:** May **YEAR:** 2001

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individual or institutions.

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in this thesis (other than brief excerpts requiring only properly acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

DEDICATED TO

MY PARENTS

TABLE OF CONTENTS

LIST OF TABLES vii
LIST OF FIGURES viii
ABSTRACT ix
GLOSSARY x
ACKNOWLEDGEMENT xiii
SPECIAL THANKS xiv
Chapter 1 INTRODUCTION 1
1.1 Online Auction Systems 1
1.2 Active Networks 2
1.3 Objective of the Thesis 3
1.4 Outline of the Thesis 4
Chapter 2 AUCTION SYSTEMS 5
2.1 Traditional Auction Systems 5
2.2 Online Auction Systems 8
Chapter 3 ACTIVE NETWORK TECHNOLOGY 14
3.1 Active Network Architectures 14
3.2 Dalhousie University SAVE test bed 17
Chapter 4 DESIGN APPROACH 19
Chapter 5 IMPLEMENTATION DETAILS 25
5.1 Internal Service Implementation Approach 25
5.2 Security Implementation Approach 34
5.2.1 Authentication 34
5.2.2 Authorization 38
5.2.3 Encryption 39
Chapter 6 ONLINE AUCTION SYSTEM OPERATION 41
6.1 Auction Active Routers 42
6.2 Auction Server 42
6.3 Auction Client 44

Chapter 7 EXPERIMENTAL RESULTS 47
Chapter 8 CONCLUSION AND FUTURE WORK 51
8.1 Contribution of the Thesis 51
8.2 Features of the Design 52
8.3 Moving from Prototype to the Real World System 53
8.4 Future Work 53
Appendix A PLAN 57
Appendix B Cryptix 3.2 Toolkit 62
Appendix C System Detail Operation 68
Appendix D Experimental Numiraical Values 83
REFERENCES 88

LIST OF TABLES

Table 2.1 A summary of the basic characteristics of auctions 6

Table D.1 Total Client Transaction Time (Delay). 83

Table D.2 Total Client Encryption Time. 84

Table D.3 %CPU & Memory Usage 85

LIST OF FIGURES

Figure 2.1 Basis online auction system 10
Figure 2.2 Screen snapshots of traditional online auction system platform: server operations 11
Figure 2.3 Screen snapshots of traditional online auction system platform: client operations 12
Figure 3.1 Dalhousie SAVE testbed configurations 18
Figure 4.1 Active node middleware structure 21
Figure 4.2 Active online auction system network overview 21
Figure 4.3 Distributed Database architecture overview 22
Figure 5.1 SAVE testbed network topology 25
Figure 5.2 Set Data list service 27
Figure 5.3 Get winner list service 27
Figure 5.4 Get Data list service 28
Figure 5.5 Get Winner list service 29
Figure 5.6 Get New Price service 30
Figure 5.7 Update Data service (normal mode) 31
Figure 5.8 Update Data service (Encryption mode) 32
Figure 5.9 Update Winner service 33
Figure 5.10 JAVA stub for PLAN security certificate 34
Figure 5.11 Active Node output as a result of receiving the first message 35
Figure 5.12 The Application output as a result of receiving the second message 35
Figure 5.13 Active Node output as a result of receiving the third message 36
Figure 5.14 JAVA Representation of the PLAN Certificate 37
Figure 5.15 Security policy (QCM) 38
Figure 5.16 Encryption steps 39
Figure 6.1 Overview of secure online auction system 41
Figure 6.2 Screen snapshots of Active online auction system platform: Server operations 43
Figure 6.3 Active node when receiving the items from the auction server 44
Figure 6.4 Screen snapshots of Active online auction system platform: Client operations 45
Figure 6.5 Active router operations when receiving bid from the client 46
Figure 6.6 Active router operations when receiving an encrypted bid 46
Figure 7.1 Total Transaction time 48
Figure 7.2 Total encryption time 49
Figure 7.3 % CPU usage 49
Figure 7.4 % Memory usage 50
Figure 8.1 Extension of the implementation using a hierarchical auction structure 55

ABSTRACT

Online auction systems require high-speed bid transmission, large bandwidth and maximum bid security. Most current implementations of online auction systems perform all the auction operations on the auction server without any support from the network. This results in a large number of bid collisions, thus reducing the performance. Furthermore, many current implementations do not provide security measures such as client authentication and authorization, bid validation and bid data encryption.

Active Networking is a new networking paradigm that inserts intelligence within the network by offering dynamic programming capability to network routers as well as to packets traveling in the network. The result is a more flexible and powerful network that can be used to speed up the deployment of applications.

This thesis presents a novel approach to the design of secure online auction systems using active networks. The goal of this design is to develop efficient and portable quality of service mechanisms to perform a variety of auction system tasks in a secure manner. The thesis discusses the system design model and its implementation on an active network test bed. The utilization of active network technology accomplishes several enhancements such as reduction in number of bid collisions, better server utilization, distribution of the security process, adjustment of the server bandwidth and enabling of distributed monitoring the auction rules. Performance results from experiments on the test bed validate many of the advantages of the proposed approach.

GLOSSARY

ABone— Active Network Backbone; virtual test bed for the active networks research program funded by DARPA ITO. It is composed of a set of computer systems configured into virtual active networks. These systems execute under operating systems built specifically for active networking.

Active Network— A network that allows the intermediate routers to perform computations on data packets.

Active Node— Is a daemon program that acts as a physical router.

Authentication— Validation of the sender in a communication.

Authorization— The eligibility to access the services depending upon the level of privilege.

ANTS—Active Network Toolkit. A distributed program that runs over many active nodes.

Confidentiality— How to prevent snooping of information by those who are not authorized to access it.

Cryptix— Is a cleanroom implementation of Sun's Java Cryptography Extensions (JCE) version 1.1. In addition to that it contains the Cryptix provider which delivers a wide range of algorithms and support for PGP 2.x. with new effort going into the next generation based on the Sun JCE 1.2 specification. Available at http://www.cryptix.org/

DAN—Distributed Code Caching for Active Networks. Type of active network implementations. The services are ordered in a linked list.

Diffie-Hellman key exchange protocol— Protocol used to securely generate a same secret (private) key on different hosts.

Digital Signature— A process that changes the order of the message bytes in the memory based on a certain chosen key.

Encryption— A process that scrambles a message so that an unauthorized user can not read it.

Event-based Application— An application, which are directed by an input event that can occur at any time.

IPSec— A model of Virtual Private Network (VPN) data security. IETF's new security protocol for layer 3 tunneling.

NetScript— A dynamic programmable active network using a language called NetScript.

Ocaml— A functional programming language. Packet Language for Active Networks (PLAN) uses this language for its services.

PLAN—Packet Language for Active Networks. A new language for programs that form the packets of a programmable network.

PLAND— Is a daemon program that acts as a physical router and as an active node.

PLAN Bridge— An application written in JAVA; runs on test bed, to enable remote connections from any application outside the test bed to communicate with the internal PLAN routers.

PLAN Tuple—PLAN data type. Data structure similar to list, with a difference that it holds group of different data types, such as, string, integer, char, and byte array together.

QCM—Query Certificate Manager. Security policy used to identify the eligibility of accessing the active node services depends on the pre-defined level of privilege.

QoS—Quality of Service. It is a measure of performance for a transmission system that reflects its transmission quality and service availability.

SAVE—Secure Active VPN Environment. It is the test bed setup at the Dalhousie University faculty of Computer Science to investigate the performance of the Active VPNs.

Synchronization— Multiple instances of an application maintaining the same information state on the server.

SwitchWare—An Active Network architecture developed by University of Pennsylvania. This architecture is based on two levels of hierarchy, low-level packet switching and high-level packet services.

VoIP—Voice over IP. The ability to carry normal telephony-style voice over an IP-based internet with POTS-like functionality, reliability, and voice quality.

VPN—Virtual Private Network. Is defined as customer connectivity deployed on a shared infrastructure with the same policies as a private network.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to my supervisor Dr. Sampalli Srinivas for numerous reviews of my work and his insightful and helpful comments, support and encouragement during every stage of my graduate studies. I would also like to thank my readers Dr. Nur Zincir Heywood, faculty of CS Dalhousie University, Mr. Johan Stratton, Innovatia, Aliant Inc., and Dr. Dawn Jutla, St. Mary's University, for kindly offering to read my thesis and insightful discussion, Thanks also to my SAVE colleagues for their help with obtaining numerous papers and documents relating to active networks and for their insightful discussions. Last but not least, my sincere thanks to my father Prof. Dr. Abdel-fattah Shihada and my mother Suhailah Abu-Saymeh for their love, support, and motivation. It is to them that I dedicate this thesis.

SPECIAL THANKS

The online auction system designed in this thesis was implemented on the SAVE (Secure Active VPN Environment) test bed.

The SAVE project at Dalhousie University, under principal investigation Dr. S. Srinivas, is funded by the Canadian Institute for Telecommunications Research (CITR), under the major project "Resource Management in High Speed Networks". I would like to express my sincere thanks to CITR and its industry affiliates Aliant Inc., Bell Canada, Bell Mobility, General Data Comm., and Nortel Networks for the support.

Chapter 1

INTRODUCTION

Auctioning is perhaps one of the most popular systems of trade in the world of commerce. The origins of auctioning dates back to ancient Rome, circa 193 A.D. A Roman senator, as legend goes, bid on the entire Roman empire. He eventually bid the equivalent of what would amount to $14 million today in order to secure control of the empire from the then emperor. Auctions became the rage of Rome. Emperor Gaius Caesar Germanicus, a.k.a. Caligula, held auctions on a frequent basis. He recognized the people's desire to bid on items, and possibly receive a good bargain. After this, auctions expanded to other areas. Spreading and gathering worldwide fame, auctions snowballed into what they have become today. The government, companies and businesses hold frequent auctions on a variety of items, including confiscated, repossessed, and surplus items [1].

1.1 Online Auction Systems

The Internet and the World Wide Web have energized the already fast-moving world of commuting and created previously unthinkable opportunities for communication between computer users. The Internet has successfully introduced many new services such as Voice over IP (VoIP) telephony, videophony, video conferencing, information retrieval from multimedia documents, and many computer-supported cooperative activities.

With the advent and the rapid growth of the Internet, online auction systems have gained practical importance in computer networking and electronic commerce. The idea that computer networking can support strategic business objectives could probably decrease the demands placed on servers and allow for more distributed data management. As electronic commerce activities have proliferated, interest has increased in online auction systems especially those that provide high performance, reliability, and security. Indeed, online auctions have played a vital role in expanding current business transactions to much higher levels by allowing a larger number of potential customers and companies to interact in a shorter time with lower costs.

Online auction systems have been able to increase their marketing ability through the use of Internet supports. A technology of an online auction system allows markets to access bidder information dealing with who buy, how they buy, when they buy, and what they buy. If the information shows that there is enough demand for a particular product, special services must be implemented to develop the bidding operations. These services must be implemented in real time, that is, as soon as data is entered in to a service unit, the results must be processed fast enough to be used immediately.

Although considerable work has been done to enhance the overall performance of online auction systems, they are still largely dependant on the organization of the average network bandwidth latency, the reduction of the management overhead, and the minimization of synchronization. The drawback of the current auction systems is the result of performing all the auction operations on the auction server without any support from the network. This results in a large number of bid collisions, thus reducing the performance. Furthermore, many current implementations do not provide strong security mechanisms with high system performance such as client authentication and authorization, bid validation and bid data encryption.

1.2 Active Networks

Active networking is a new networking paradigm that inserts intelligence within the network by offering dynamic processing and programming capability to network nodes and packets traveling in the network [2]. Routers in active networks are able to perform computations up to the application layer. The result is a more flexible and powerful network that can be used to speed up the deployment of new applications. Applications that can benefit from active networks include network management, congestion control, multicasting and caching.

Several architectures have been researched for deploying active networks [3]. Three kinds of active network architectures have emerged: *active packets*, in which the computations are limited to the packets traveling in the network; *active nodes*, in which the computational power comes from the nodes only; and the *hybrid version* which combines both active packets and active nodes approaches. According to [6], the hybrid

architecture appears to be the most promising one. Among the different active network frameworks that have been proposed, PLAN (Packet Language for Active Networks) has emerged as an important hybrid architecture [3,4]. PLAN combines security, flexibility, efficiency and provides an execution engine for developing active network applications.

The current traditional online auction system relies on a general policy of "fast, secure, and reliable" to accomplish more of its missions in the required time with reduced network resources. Many aspects of the growing mission rely on the availability of advanced computing capability and high-speed network availability. However, an alternative approach to achieve the required auction missions with existing networking resources is the direct implementation of the active network concept, in which an application executes a certain program over the network node (server, router). This active node will work as an intermediate service on the data packets that flow over the network. Based on this approach, an active network concept should develop or provide QOS guarantee, reliable features, and a security framework.

1.3 Objective of the thesis

The primary objective of this thesis is to present a novel approach to the design of a secure online auction system using active networks. The goal of this design is to develop efficient and portable quality of service mechanisms to perform a variety of auction system tasks in a secure manner. The thesis discusses the system design model and its implementation on an active network test bed. The utilization of active network technology accomplishes several enhancements such as reduction in number of bid collisions, better server utilization, distribution of the security process, adjustment of the server bandwidth and enabling of distributed monitoring the auction rules. Performance results from experiments on the test bed validate many of the advantages of the proposed approach.

The work in this thesis has been done under the auspices of the Secure Active VPN Environment (SAVE) project in active networks at Dalhousie University. The SAVE project is part of a major project in Resource Management in High Speed Networks

funded by the Canadian Institute for Telecommunications Research (CITR), and is collaborated by five universities and five telecommunication companies across Canada.

1.4 Outline of the thesis

Chapter 2 surveys existing traditional auction systems, with a summary of the basic characteristics of each system. It also addresses the parameters required and the issues involved in the design of online auction systems. Some implementation snapshots have been taken to show the steps of the auction system. Chapter 3 covers a survey of the existing of active networking technology platforms, operations, types, implementations, advantages, and disadvantages. Chapter 4 focuses on the active network design of the secure online auction system, and describes how a regular auction system can be converted to an active system. Chapter 5 specifically illustrates the internal active node and explains the implemented services. It addresses and explains the used security levels, such as, authentication, authorization and encryption. Chapter 6 mainly shows a demonstration of the implementation and illustrates the steps of running this system. Some snapshots of some pre-runs operations have been provided. Chapter 7 describes the experimental results of the system overall performance, the performance reflects the levels of analysis that shows the total transaction time, encryption time, and the router CPU/Memory usability during different stages of the system. Chapter 8 summarizes the achieved goals of this approach and some suggestions for future work.

Chapter 2
AUCTION SYSTEMS

This chapter surveys multiple existing traditional auction systems, with a summary of the basic characteristics of each. It also addresses the importance of the parameters required and the issues involved in the design of online auction systems. Some implementation snapshots have been taken to show the steps in the operation of online auction systems.

2.1 Traditional Auction Systems

There are many different types of auctions in common use [5]. Table 1 summarizes a number of traditional auction types. Auction systems are classified into two levels of hierarchy. The first level of hierarchy depends on whether the system is single or double-sided. The second level depends on whether they are sealed-bid or open-outcry. The English, Dutch, and Vickrey auctions are considered single-side auction types. On the other hand, the Call market and the Continuous Double Auction are considered as double side auctions [6].

The following define the terms used in the above paragraph. In s*ingle sided* auction, bidders could be buyers or sellers. *Double sided* auction admits multiple buyers and sellers at once. In a s*ealed-bid* auction the bidder, after observing his/her item, submits a closed sealed-bid. The highest two bids are allowed to continue. In *open-outcry* a public auction system used for futures trading on the floors of futures exchanges where communication is by way of shouting and hand signs between traders. Stock exchanges used to run an open outcry method of trading, but for the most part are now fully computerized [5,6].

English (open-outcry) auctions are probably the most common. Users start with the highest price they are willing to pay for an item, and the bidding session continues until the auction duration is complete or no more bids are received. The product is sold to the highest bidder at their bid price. English auctions allow the seller to have a pre-chosen price below which the item will not be sold [6,7]. All traders prefer the confidentiality in these types of auctions. Usually the competition is high in the English Auction, because the bidders are carried away with enthusiasm [7]. As a simple example, suppose that a

seller has 10 items for sale at $100 each. If 5 bidders bid $150 for each, then the earliest 3 bids are accepted and get to buy the items for $150 each.

Vickrey (second-price) auction, like the English auction, is used to sell a single item; the only difference is that the highest bidder obtains the item at the price offered by the second highest bidder [7]. This is a good psychological format because bidders have the ability to bid what they think the item is approximately worth without any worry about what others will bid [7].

Type	Rules
English, Ascending price	The seller starts with the lowest price he/she wants for the item. Bidding increases until either no more bids are placed or the auction session is over. The winning bidder is the one with the highest bid. The bidder may re-assess evaluation during auction.
Dutch, Descending price	Seller starts the session with the highest price possible for the item. The bid is lowered continuously until the session is over or the bid price match the seller needs.
First-price	The bidders are not known to each other and bids submitted in written form. The highest bid is the winner and pays the exact amount he bid.
Vickrey, Second-price	The bidders are not known to each other and bids submitted in written form. The highest bid is the winner and pays the second highest amount bid.
Open-outcry auction	A public auction system used for futures trading on the floors of futures exchanges where communication is by way of shouting and hand signs between traders.
Sealed-bid auction	Bidder, after observing his/her item, submits a closed sealed-bid. The two highest bids are allowed to continue; for all other the auction is over.

Table 2.1 Summary of the basic characteristics of each auction.

Again considering the previous example, where 5 bidders bid $150 for each item and the next bid price was $120. Depending on the auction rules the earliest 3 bids are accepted and get to buy the items for the price of $120.

Dutch auction is another special type of auction, which provides the ability for a seller to sell a number of identical items. The seller specifies the minimum price and the exact number of items that are available at that price. The bidders bid above the seller's price for the number of items that they want to buy. At the end of the auction session, the highest bidders earn the right to purchase those items at the minimum valid bid [5,7].

Consider the following example, a seller has 10 items to sell at $100 each. 10 people bid $100 for one item each, the earliest 8 Bidders will be awarded the item since the bid amounts are the same and the earlier bids are accepted.

Dutch auctions have a basic rule: if a person bids higher than all others it will be guaranteed the item. The rest of items will be sold to the earliest bidder who bids the second largest price. In some special cases, if a sufficient number of bidders bid above the starting bid, the final price of the item will be increased. If less than 8 bidders bid on the items then only that number of items will be sold at $100. The lowest bid may not get any item [7].

First bid (First-price) auction is an important auction, which gives the buyer the opportunity to evaluate the price and the number of the items that the buyer wants to buy. Each buyer is allowed to give one bid. The product is sold to the highest bid price. None of the buyers know the other bids.

In *Double auction* both the buyers and the sellers maintain a list of prices. At the auction session, both lists are searched to find the closest price between the buyers and the sellers; this price will be the price of the item.

As a second example, suppose that four sellers offer to sell one item at prices of $100, $200, $300, and $400, and 4 buyers offer to buy the item at prices of $400, $300, $250, and $50. Will agree to buy the items in a price ranged between $200 and $250 [6,7].

Due to the fact that the majority of the current online auction systems are using English auction rules, and also this auction type supports the breadth of auction styles, the English auction was chosen for implementation in this thesis, the implementation can be easily modified to support auction types by changing the services in the source code.

2.2 Online Auction Systems

Online auction systems have been able to increase their marketing prowess through the use of Internet supports. A technology of an online auction system allows markets to access bidder information dealing with who buy, how they buy, when they buy, and what they buy in fast and secure manner.

There are a variety of potential needs for building an online auction system, for example, the need for more people to securely interact with the auction companies regardless of the geographical distances. The following are the basic processes required in an online auction system [8]:

1. *Identification and Registration for both seller and buyer:* This is one of the most important steps, which deals with security implication such as authentication for both parties, exchanging cryptographic keys, some product profiles, assigning priority limits, and deadlines [8].
2. *Identifying the auction events:* The auction event involves of identifying the auction type, auction rules, product information, names, quantity, quality, and initial prices.
3. *Scheduling and Advertising:* Scheduling times for auction sessions, session starting and closing, pre-registration process, pre-login process, and required advertising for the available products.
4. *Bidding:* Addresses the bidding rules, then start collecting bids and finally apply the bid controlling rules to validate the collected bids.
5. *Closing the Auction:* There are two ways for closing the session; either by pre-defined time or by observing no more bids being placed. The best bidding value is selected.
6. *Trade Settlement:* This step involves of identifying the required payments, ways of products transfers, guarantee rules, testing, leasing, and financing.

For any auction system to be online it must identify each of these parameters [8]:

1. *Type of auction.* e.g. English, Dutch, First price bid, Second price bid, double auction.

2. *Anonymity.* e.g. what information is revealed during the auction and after the auction closed.
3. *Auction Rules.* e.g. opening and closing, timing, changing from type to other, withdrawing form the session, and minimum number of bids.
4. *Payment Rules.* e.g. methods of payments, number of product sources, and way of exchange the products with the money.
5. *Registration on bid.* e.g. bid amount, bid speed, bid technique, risk of bid lose or bid delay [8].

Online auction systems are categorized under *event-based applications* [7,8], which are directed by an input event that can occur at any time. Event-based applications are much more open-ended than other applications, and within them, any participant can interact with the application at any time, resulting in an input event. The design of communication mechanisms for event-based applications is relatively more complex and difficult to implement than the basic applications. More importantly, event-based applications, typically require much larger bandwidth, and generally suffer from the problem of synchronization. Synchronization refers to how multiple application instances running on different machines maintain the same application state information (each participant might run separate instance of the application).

Figure 2.1 shows the basic hierarchy process of having an online auction system based on event-driven Client/Server application. The auction server contains the auction functions, security functions, and clients information. The client application consists of auction functions and items information. The communication between the client and the server is done using sockets. Many traditional online auction systems such as ubid.com [31], ebay.com [32], Auctionfirst.com [33], and AucBid.com [34] are using the basic Client/Server design for their auction implementation. During the auction session the buyers, the sellers and the auction rules are the active functions on the server. Therefore the auction process session (registration, authentication, authorization and session process) is done on the server.

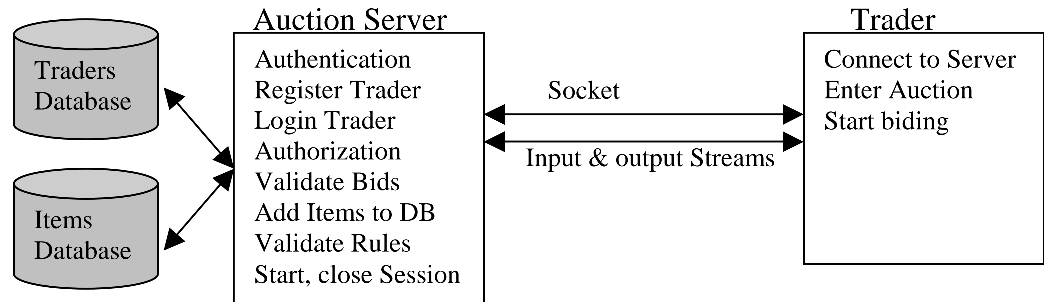


Fig. 2.1 The basic online auction system

Figure 2.2 shows snapshots of an online auction system platform, which shows the auction server side functionality. The server starts with the ability of storing the auction items, the items are stored based on their pin code (unique for each individual item type), item name, item description and item initial price also must be identified. The server controls the auction session timing buy placing the item auction session date and time. Figure 2.3 shows snapshots of an online auction system platform, which reflects the client side functionality. The client connects to the server with three different options. Register new user, login, and logout. A successful client registration and login leads to a successful access the available auction sessions. A session status indicator gives the client an indication of the auction session activity.

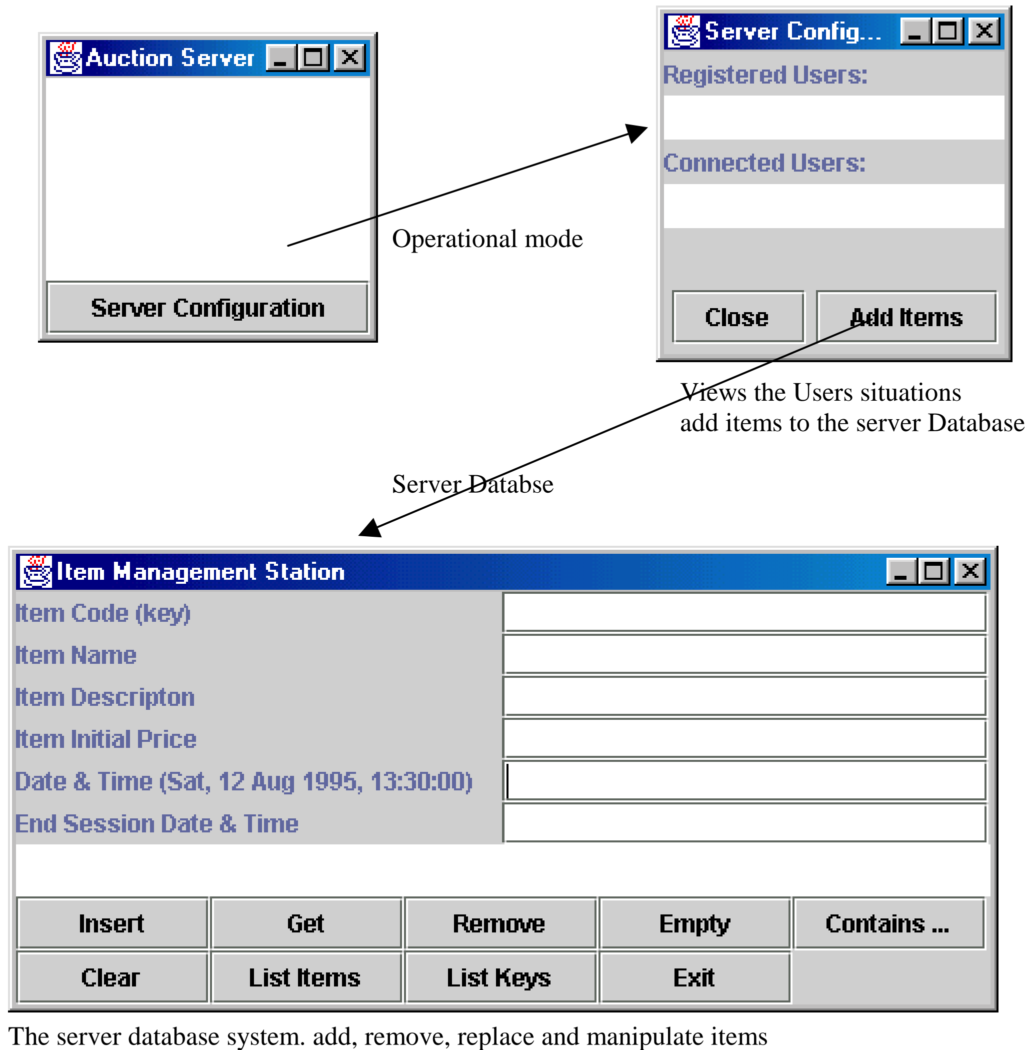
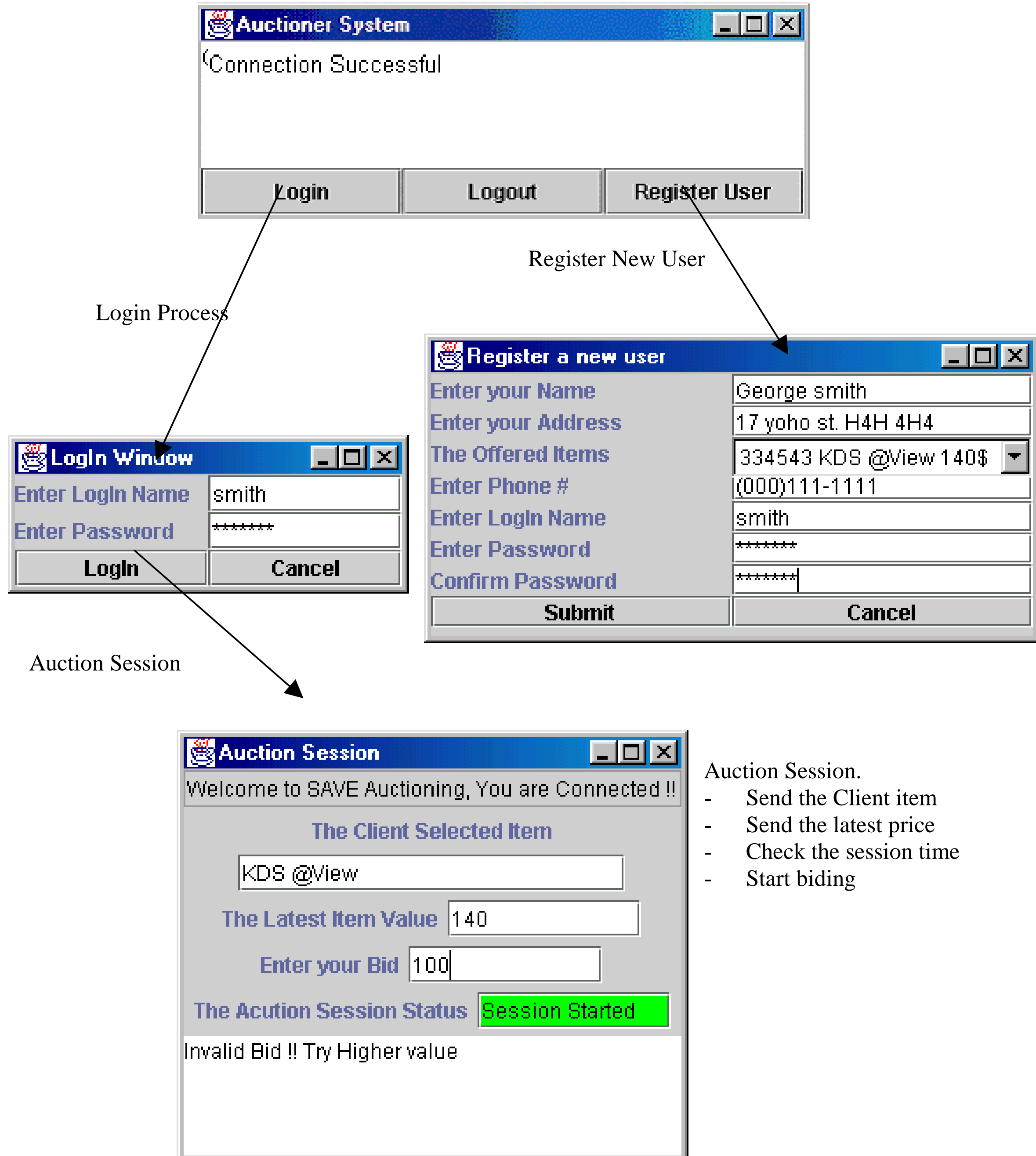


Fig. 2.2 Screen snapshots of traditional online auction system platform: Server Operations

The client side



Auction Session.
- Send the Client item
- Send the latest price
- Check the session time
- Start biding

Fig 2.3 Screen snapshots of traditional online auction system platform: Client Operations

In order for current online auction companies to increase their server performance, they must address some important issues such as increasing the bandwidth, expanding the server to cover more geographical area and distributing some of the server tasks and load. In addition to the above-mentioned issues, the companies should also consider the online auction drawbacks such as the following [8]:

1. *Server Overloading:* Most of the current implementations of online auction system are maintained all the auction functions on the server. This will cause the server to be overloaded and sometime crashed.

2. *Bid Collisions:* Due to the fact that the server and only the server will accept and validate all the coming bids, bids might collide, so some bids will not be able to be processed.

3. *Delays:* Auctions can take a week to finish, and then there are often further delays until the item is shipped to you.

4. *Security:* How to make sure that the received data is from the valid sender. How to protect the information from those who are not authorized to access it. And make sure that every application is eligible to access the server depends on the level of privilege.

5. *Others:* Other drawbacks such as Fraud Risk, Payment and Disappointment [8].

Trying to eliminate the drawbacks of the current online auction systems is the main motivation of this thesis. Using active network technology for online auction systems helps developing an efficient and high portable quality of service mechanisms to perform a variety of auction system tasks in a secure manner.

Chapter 3
ACTIVE NETWORK TECHNOLOGY

An active network is a network where the intermediate routers can perform computations on the passing packets. In addition, users can increase the network intelligence level by injecting their programs to the network to call some services; these services will perform the required operations over the packets. Active networks have different types of architectures. Each has its own characteristics to perform a certain level of performance or security [9]. This chapter covers a survey of the existing active networking technology platforms, operations, types, implementations, advantages, and disadvantages.

3.1 Active Network Architectures

The following is a survey of some different active network architectures currently proposed. This survey will help to identify an appropriate architecture for secure online auction systems. The main idea behind the concept of active network came from the activity of the packets itself. Predefined functions are stored on the active nodes, the programmed packets can determine which service is to be called to do the required computation over the packet data.

An Architecture for Active Network [9]. Proposed at Georgia Institute of Technology, users can control the functionality of the active node in such a way they can chose what service is needed to accomplish their tasks. This approach is problematic to use with auction systems, because most of the Internet clients do not have the required knowledge of active network, therefore, it is difficult to give the client the choice of choosing between the provided services by the active node.

DAN (Distributed Code Caching for Active Networks) [9]**.** Proposed at Washington University, in DAN all services are ordered in a linked list, each function calls the next. The DAN architecture implements security by filling a database with all the required information, public keys, and signatures about the agents and clients. Therefore security is reduced to the available database information about the auction clients. In online auctions it is difficult to keep track of this information, because the number of clients is unpredictable and often changes.

ANTS (Active Network Toolkit) [9,10]**.** Implemented at University of Washington, in ANTS the network can be seen as a distributed program running over many active nodes. ANTS need to be executed in a restricted network environment that implies a limitation to the shared resources. ANTS is not suitable architecture for online auction systems, because the server (holds the whole information, rules, managements, items and timing) cannot operate in a distributed mode. Also, auction systems need an open environment system, which provides clients an access to any shared resources for example, item prices.

NetScript. Proposed at University of Columbia, provides a dynamically programmable network using a language called NetScript. This language enables the network users to control the process of packets inside the network over the active nodes. NetScript views the Internet as virtual engines for executing the code. NetScript views the network as a single programmable abstraction rather than a heterogeneous collection of programmable intermediate nodes. Since auction systems require a distributed and heterogeneous environment, NetScript may not be a suitable chose [9,10].

SwitchWare (PLAN). Implemented by both UPenn and Bellcore. PLAN (Packet Language for Active Network) is used in the SwitchWare architecture, and works as a control switch over the received packets from the network [10,11]. As such, any PLAN program is interpreted over the active node by the routers, and the execution of the program results in the packet being sent to another destination. The design requirements for implementing an online active auction system are flexibility, safety and security. The SwitchWare environment addresses the above-mentioned requirements [10,11]. Flexibility in PLAN is given in two levels of hierarchy, the packet switching level and the service level. The PLAN language controls the packet switching level; PLAN also provides some network configuration, diagnostics and distributed communication among the routers. At a higher-level (service level) PLAN is able to call some services (usually written in Ocaml or C) to perform the required actions on the received packets [11,12].

Safety (reduce the mistakes or un-intended behavior of the auction session) and security (privacy, integrity, authentication and availability) are important key features of

PLAN. PLAN is designed to be pointer-safe; concurrently executing programs cannot interfere with one another, and provide (on-demand) packet-by-packet authentication. Most security services are implemented using a higher level programming language functional language called OCaml [11,12].

Active auction system should have the highest level of performance to reduce latency on the clients. PLAN programs are made interpretation simple and lightweight, thus common tasks can be done quickly [11,12]. PLAN programs execute remotely, and terminated safely. As a result of a PLAN program execution, a service implements a task over the data attached with the program.

In this thesis the SwitchWare architecture has been selected, because the project has a programmable element performing switching functions [11,12]. The Switch has input and output ports controlled by a software. This software can be modified depending on the application request. SwitchWare architecture helps the online auction system to work in high performance levels. PLAN- Programming Language for Active Network is the networking programming language used in my active auction system implementation architecture. PLAN can reach an acceptable performance level when the PLAN code call other programming services to perform the required process over the data. PLAN has special characteristics that can be used to address the security and safety issues, major benefit of performance, and flexibility in configuring and diagnosing the distributed computing applications [11,12].

The current implementations of auction system do not provide strong security mechanisms with high system performance. We can take the advantage of active router to perform a scalable distribution of the required items over the network depending on the client's interest. Each active node will be programmed to give information only about the clients selected items and nothing about other items. This network level security is provided by the SwitchWare architecture through the PLAN authentication and authorization process. On the other hand, a real increase of the auction server bandwidth will be achieved, which will increase the performance speed for the whole auction system. Appendix A provides detailed advantages and disadvantages of PLAN.

Using PLAN, it will be possible to filter the incoming packets that have the client's bids with the item prices over the active node. The server periodically updates the

intermediate active routers with the latest item prices, and then make the node responsible of rejecting or accepting the client bids. The server is no longer doing the bid validation process, which is expected to free up the server workload.

Generally, the distributed architecture of active networks provides a good solution to reserving the communication bandwidth and better optimizing the auction distributed management performance.

3.2 Dalhousie University SAVE test bed

Our research in active networks at Dalhousie focuses on building a security architecture and investigating the performance of applications over active Virtual Private Networks (VPN). We are considering two interesting research avenues: (a) how we can use active networks to deploy VPNs; and (b) how we can use VPNs to deploy effective active network applications. Design and deployment of VPN on demand, network management and secure multicasting are some of the sub-projects that are currently being researched by our group. The SAVE group have adopted PLAN as our platform for developing active network applications. Appendix C provides complete steps for PLAN installation. The work of this thesis belongs to the second category of research, namely, deploying effective and secure active network applications.

The work that led to the results presented in this thesis has been done on the test bed set up in our lab for these experiments and research. Figure 3.1 shows the layout of the test bed. It currently has five systems: three active PIII 450 128MB nodes running Debian Linux and two multimedia workstations running Windows NT4.

Active network execution engines PLAN, ANTS and NetScript are available on the three active nodes. VPN tunnels can be created using IPSec Free S/WAN and PPTP (Point-to-Point Tunneling Protocol). AnetD provides connection to the ABone and Internet access is via the Dalhousie network. Plans are underway for new equipment additions. In order to provide a controlled environment for our research and experiments, we have the capability of running the equipment on a local (traffic free if we so choose) network.

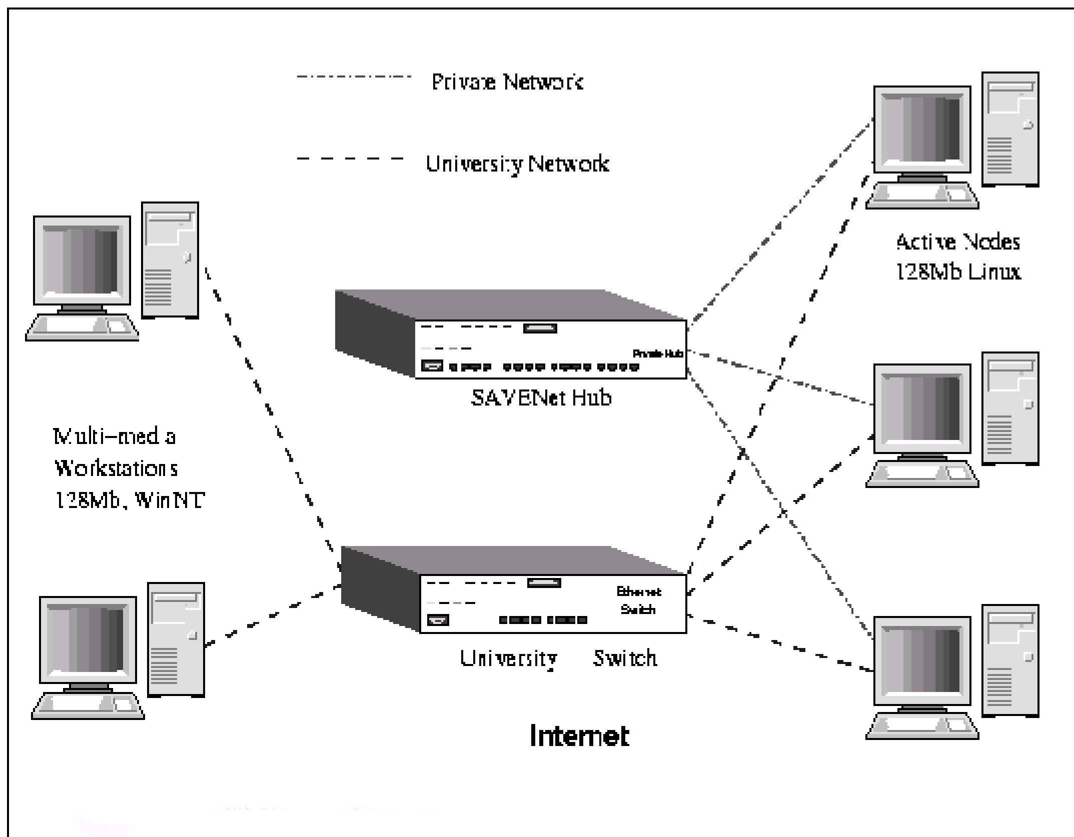


Fig. 3.1: Dalhousie University SAVE test bed configuration

This thesis presents the design of online auction system using active network technology implemented on the SAVE test bed using SwitchWare architecture and PLAN as an active network packet programming language and C as a service level programming language.

Chapter 4
DESIGN APPROACH

This chapter presents the design of the online auction system using active networks. A description of the design is illustrated using the SwitchWare project implemented by both JAVA (application level programming language) and PLAN (switch level active network programming language) [11,12]. This design approach is based on an *n* number of auction users on *k* different online active routers each with *p* number of different items. This design requires developing an efficient and highly portable secure mechanism to perform various client needs. Efficiency is a performance measure to indicate the effective utilization of bandwidth within a network. Portability refers to how easily the auction system can be implemented independently from any low-level primitives like network and operating system environments. Security refers to authentication, authorization, encryption and non-repudiation. For example, times of great demand on the server, the server can become bottlenecked from both the client high transactions and the congested packets, which need a retransmission overhead [12]. The load on the routers will occur especially on the routers that are near the server. At this time, the latency occurs on the client side.

This active auction system is designed based on the concept of the client/server approach with stream sockets. The system consists of an auction server application that allows multiple clients to connect to the server and begin the auction session. As the server receives each client connection, an instance of the auctioneer is created to process the client in a separate thread of execution. This enables the client to participate in the auction session independently. The server maintains the client information and the auction session information, so that it can determine if the client information and bids are valid. Each client application maintains its own version of the auction interface on which the state of the auction session is displayed. The clients can only place one unique value in the bidding field.

The following provides an insight of the top-level application Client/Server side of the active auction system. When the active auction server starts, it begins accepting client connections. Then, a new client will be added to the connected client table. The server

starts checking whether this client is an already registered or a new client. If it is a new client a new process will start to collect the client information. If it is an already registered client the auctioneer creates a process to negotiate the authentication and authorization process through the input and output streams. The client method controls the information that is sent to the active routers and the information that is received from the server. The server acknowledges the client connection, identifies the client, identifies the selected items, and updates information about the client and the auction session. A valid bid, valid user, and valid auction session time are all methods used to ensure the correct implementation of auction rules.

As mentioned earlier, an online auction system is an event-based application. This type of application suffers from a synchronization problem. In this design the methods that are responsible for validating the client bids are all kept synchronized, to prevent more than one client from modifying the state information of the item price simultaneously. The Server/Client application implements multi-threading task operations so that a separate thread is used to continually read messages that are sent from the server to the client.

Figures 4.1 & 4.2 illustrate the active network approach [13,14]. Basically, the purpose of the active node is to work as computationally fast validations applied to all bid packets. Only the bids that pass through the validation update the item price.

Figure 4.3 illustrates a design of a database distribution mechanism; using this mechanism it is possible to reach an acceptable network usability and speed. The auction database server distributes the items over the active nodes. The auction system can be seen as a collection of active nodes. Each node holds the auction rules, auction services, and part of the database. This part is dynamically located over the nodes. The items can be added, removed or changed depending on the use of the clients.

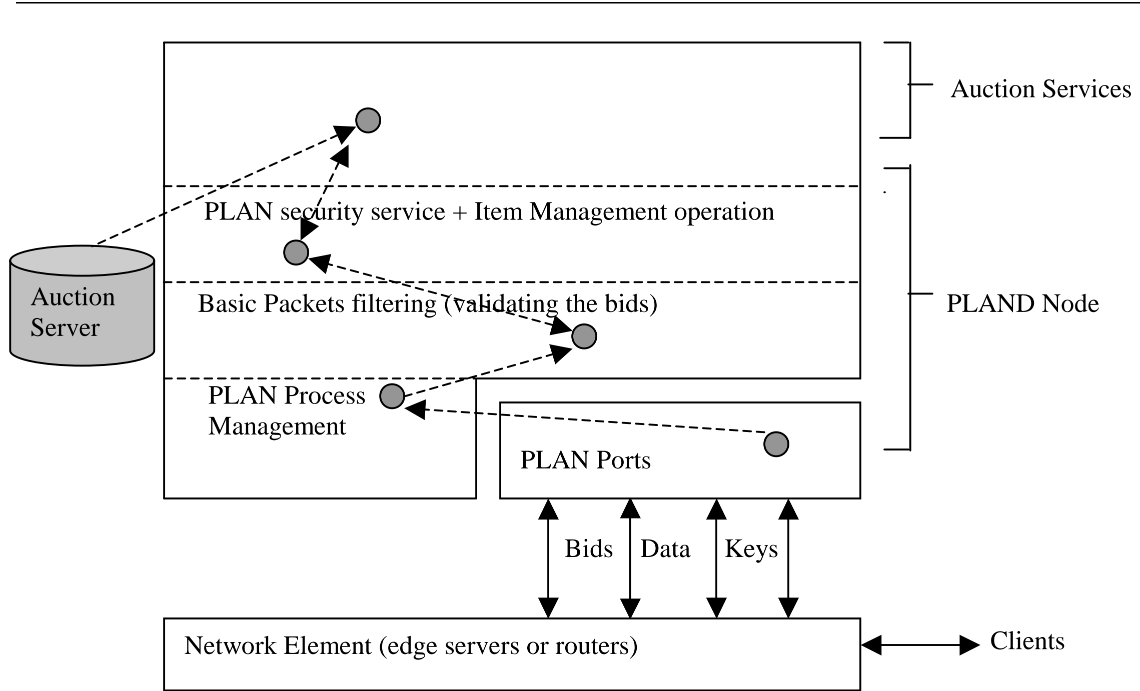


Fig. 4.1 Active node middleware structure.

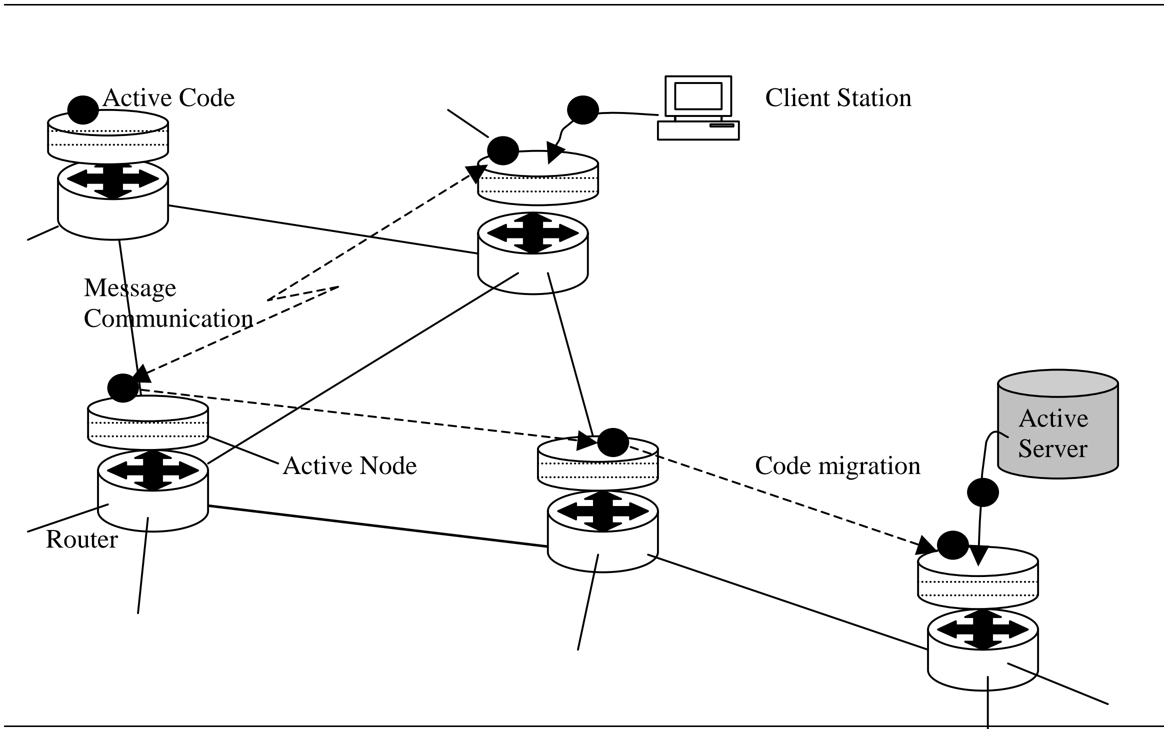


Fig. 4.2: Active online auction system network overview.

The advantages of using a dynamic distributed database are:

1. *Location independent:* After studying the redundant requests from a certain location it could give a view of what items should be stored on the active node close to that location.
2. *Network performance:* The distribution of the database will support a variety of disparate communication networks.
3. *Locality:* All operations at a given time on a given item are controlled on that item only on that active node.

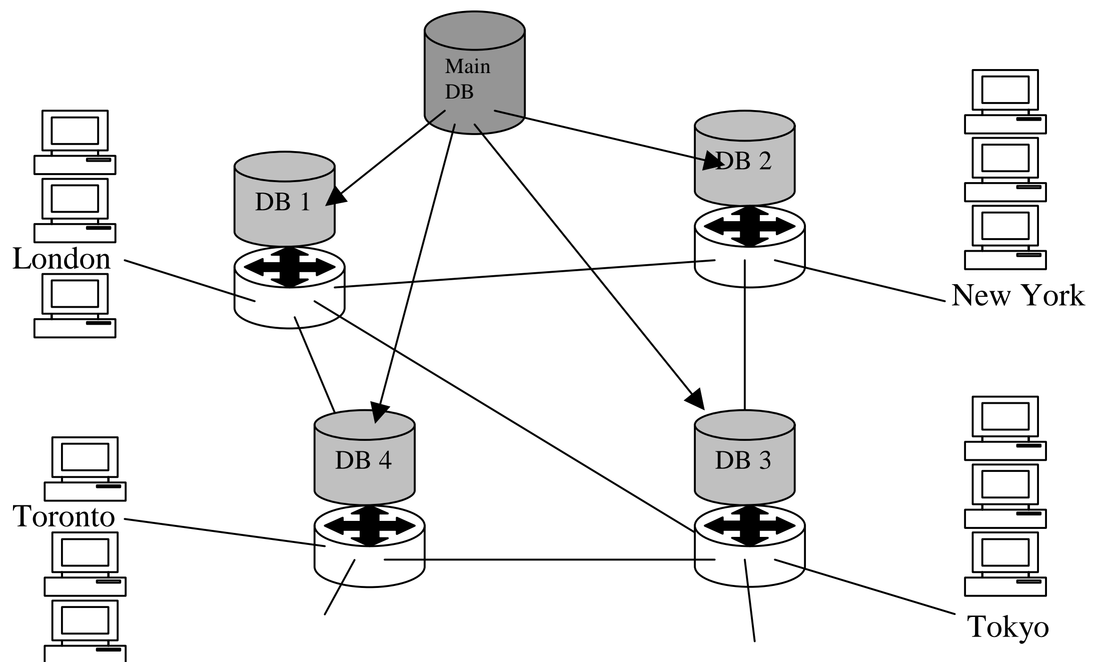


Fig. 4.3: Distributed Database architecture overview

Furthermore, if each client wishes to bid on several items, there will be no need for overloading one router with the whole database or a large part of the database. For the sake of the prototype implementation, a hash table based database system was used. Usually, this technique will enable multiple connections with the in-between network devices. In the auction applications this matter will not be a big concern, because the client will have his/her connections for a short time, which will be the bidding time.

Management functionality is added for both performance and security reasons. For example, if a client entered an auction session for a certain item, there would be no need at all for the server to provide this client with any information about the other ongoing auction sessions or other items. Similarly, the client cannot violate the auction rules or parameters, because the auction rules are all pre-programmed and cannot be changed unless the system administrator wants to. As soon as the auction session is over the client is disconnected from the active node.

The high level Client/Server application communicates with the active network using PLAN ports. PLAN ports listens for connections from both the active router and the host application [11,12]. All packets sent from the host application to the active router will be called active packets. Messages created from within the active node are service dependent, so its purpose is to do the required filtering over the active packets.

Here is step by step summary procedure for the active online auction system design operation:

1. The PLAN active nodes start with the ability to communicate with external host applications via PLAN ports. PLAN ports are a modified version of a UDP/IP stream socket.
2. The active auction server application starts and injects a PLAN program to call remote services to exchange the database items, retrieve winner lists and other auction requirements.
3. The active auction client application starts and connects to the PLAN active node, then an authentication and authorization procedures starts.
4. During a pre-defined auction session time, the client can access the available items on the PLAN active node.
5. The server application can at any time request the winner list, the database list or client information from the PLAN active node.
6. By the time the auction session ends, the clients are forbidden from accessing the PLAN active node, by disabling the connection.

Security (authentication and authorization) is applied using two levels of hierarchy. First, the host application, which will inject PLAN programs to the network, must authenticate itself to the active node using public keys. Second, the client uses the shared

secret key to either digitally sign or encrypt the bidding data. Furthermore, for the client host application to be connected to the server, it first must go through some active nodes. Here is the step by step procedure for application to node authentication process:

1. The client host application asks the active node for an authentication.
2. The active node injects a PLAN program to call a remote service to exchange the keys and identify a selected key exchange protocol.
3. When the active node receives the request, it first verifies the signature with the key to make sure that the certificate is still valid (the certificate has exchange keys, public key, client host application keys, and both addresses).
4. The active node sends its public key to the client and stores the key sent from the client host application.
5. The client receives an active node signature and does the same process that the active node has done.
6. The active node receives the approved message from the client host application and repeats this action with each incoming packets [11,12].

Chapter 5
IMPLEMENTATION DETAILS

This chapter describes the internal PLAN active node and explains the implemented services. It also addresses and explains the used security levels, such as, authentication, authorization and encryption. Sample services code implementations have been provided.

5.1 Internal Services Implementation Model

PLAN auction routers are designed to handle the auction tasks individually. All results will be returned back to the server and some will be shared with clients. Communication between the routers and clients is accomplished via PLAN programs [15]. Figure 5.1 shows the network topology on the SAVE test bed. Four independent PLAND routers are running on each host. PLAND is a daemon program that acts as a physical router and as an active node.

I have chosen four routers to give a realistic view of the auction system implementation. PLAN Bridge is implemented to enable non-active applications to communicate with the PLAN active routers from remote hosts.

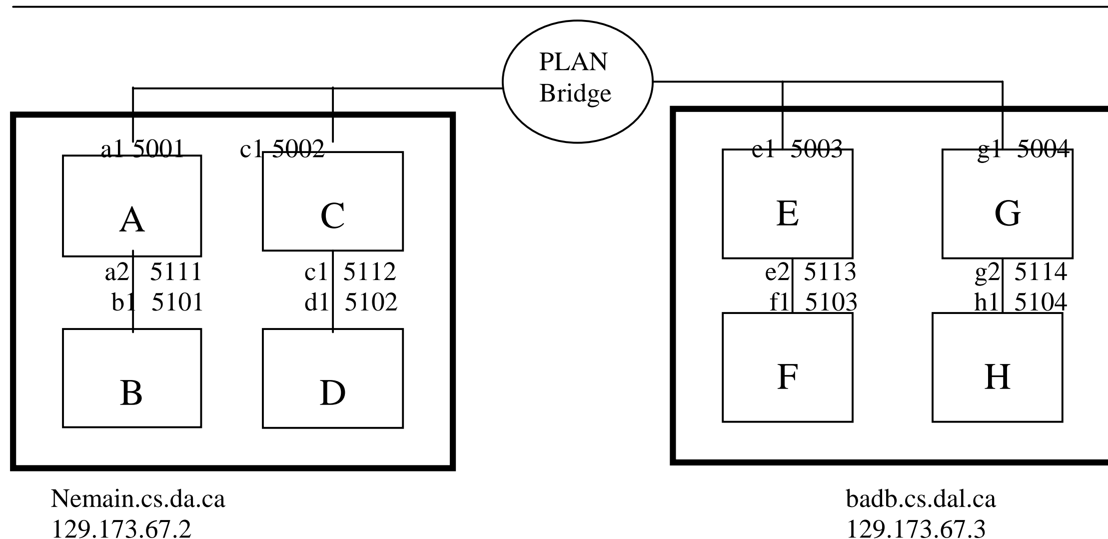


Fig. 5.1: SAVE test bed network topology.

To create this topology, each individual node must maintain a network interface file to identify the internal connection between the routers. e.g. node A has an interface file as shown below.

```
2
a1 ip nemain:5001 nemain:5002, badb:5003
a2 ip nemain:5111 nemain:5101
```

The above interface file describes node A with two internal interfaces, a1 identifies node A connection ports with its neighbor node C and node E on badb.cs.dal.ca. a2 identifies the second interface of node A connected to node B. The rest of the active nodes communicates with each other using similar interface files [15,16].

In addition to the core services available to PLAN, additional services have been implemented to support the online auction system tasks. The following is an overview of how these services are designed and implemented.

The services implemented in this design use the technical guidelines given in PLAN documentation [15,16,17,18,19,20,21].

Store Data to the active node: PLAN_UNIT setDataList (PLAN_LIST items)

Figure 5.2 describes how this service is designed. The auction server invokes this service as soon as the database is ready to be distributed and stored over the active nodes. A PLAN program is injected by the application (server) to deliver the item list to the active node. When SetDataList service is first invoked the active node checks if the list is not empty, then it identifies its size and allocates the required memory. Finally, the item list is stored as a global list located on the active node. However, if an empty list is received the size will be set to –1 and the memory allocation will be NULL. This will raise an exception.

Initialize Winner List on active node: PLAN_UNIT setWinner ( )

This service is invoked from the setDataList service. Figure 5.3 illustrates how a new winner list can be created. This list will have the item pin code and the user who bid the highest for this item.

Get the Database from active node: PLAN_LIST getDataList ( )

This service is only called from the auction server application. As shown in Figure 5.4, at the time the service is called, it will return back the data list from all active nodes. The server checks the latest prices of items at different auction sessions. This service is required to apply the auction rules, check the item codes, validate the latest prices and make sure that the items have not been changed for any reason.

```
PLAN_UNIT setDataList (PLAN_LIST items)
{
if (items.size > 0)   {
   itemsData.size = items.size;
   itemsData.pValues = (PLAN_VALUE *) calloc(items.size,sizeof(PLAN_VALUE));

     if (items.pValues != NULL)  {
    for(I = 0; I < itemsData.size; I++)  {
       itemsData.pValues[I].tag = STRING_TAG;
       itemsData.pValues[I].val.szVal = items.pValues[I].val.szVal ;   }
    setWinner();
   }
   else  {
     itemsData.size = -1;
     winner.size = -1;  }
  }
 else {
   printf("received an empty data list from the server !!");
   itemsData.size = 0;
   winner.size = 0;
   itemsData.pValues = NULL;
   winner.pValues = NULL;  }
}
```

Fig. 5.2 SetDataList Service

```
PLAN_UNIT setWinner()
{
   int i;
   winner.size = maxClients ;
   winner.pValues = (PLAN_VALUE *) calloc(200,sizeof(PLAN_VALUE));

   for (i =0; i< winner.size; i++)  {
        winner.pValues[i].tag = STRING_TAG;
        winner.pValues[i].val.szVal = "Empty"; }
 }
```

Fig. 5.3 setWinner Service

```
PLAN_LIST getDataList ( )
   {
     PLAN_LIST result;
     if (itemsData.size > 0)  {
        result.size = itemsData.size;
        result.pValues = (PLAN_VALUE *) calloc(itemsData.size,sizeof(PLAN_VALUE));
        if (result.pValues != NULL)  {
           printf("Get the data list from the planD\n");
           for (I=0;I<result.size;I++)  {
            result.pValues[I].tag = STRING_TAG;
            result.pValues[I].val.szVal = itemsData.pValues[I].val.szVal; }
           return result;
        }
        else  {
            printf("ItemsData is Empty \n");
            result.size = 0;
            result.pValues = NULL;
            return result;  }
      }
    return result;
   }
```

Fig. 5.4: getDataList Service

Get Winners from active node: PLAN_LIST getWinner ( )

This service is only called from the auction server application. Figure 5.5 shows if this service is called it will return a PLAN list. This list will contain the winner names and their items. The server either displays or saves this list for any future contacts with clients. This service is used to list the items, the winners, know what items are interesting to the clients and on which active node they are located.

```
PLAN_LIST getWinner ()
    {
      PLAN_LIST result;
      if (winner.size > 0) {
        result.size = winner.size;
        result.pValues = (PLAN_VALUE *) calloc(winner.size,sizeof(PLAN_VALUE));
        if (result.pValues != NULL)   {
          printf("Get the Winner list from the planD \n");
          for (I=0;I<winner.size;I++)  {
           result.pValues[I].tag = STRING_TAG;
           result.pValues[I].val.szVal = winner.pValues[I].val.szVal;  }
          return result;
        }
        else  {
            printf("Winner is Empty \n");
            result.size = 0;
            result.pValues = NULL;
            return result;  }
       }
     return result;
    }
```

Fig. 5.5: getWinner Service

Get Latest Item Prices from the active node:

PLAN_STRING getNewPrice (PLAN_STRING itemCode)

This service is only called from the client application. Using any valid item code, the client can get the latest price of that item. Figure 5.6 shows the process of this service. If this service is called with a valid item code, it will return the item latest price. The item price will be displayed on the clients GUI. This service is used to item the items and keep the client updated with latest item price. This service has been designed to be a client on demand operation; this will reduce the number of operations on the active node.

```
PLAN_STRING getNewPrice (PLAN_STRING itemCode)
{
  PLAN_STRING result;
   while (i < itemsData.size)    {
        if(strcmp(itemsData.pValues[i].val.szVal , itemCode)==0)    {
           printf ("Get Item New Price\n");
           result = itemsData.pValues[i+1].val.szVal;
           break;     }
      i= i+2;     }
       return result;   }
```

Fig. 5.6: getNewPrice Service

Update the Item Price on the active node (non-secure version):

```
PLAN_BOOL updateData (PLAN_STRING code, PLAN_STRING name, PLAN_INT val )
```

This service is called from the Client application. Figure 5.7 illustrates the operation of this service. Using a valid item code, the client can bid on any item he/she likes. The non-secure version of this service takes three arguments, the item code, the user name (the name is very important to keep the active node intelligent enough to know the last person made the bid), and the bid value. The service checks whether this item exists on the active node or not. If it exists and the received bid is higher than the current value of the item, the price is changed and true is returned to the user. Otherwise false is returned. To prevent bid race conditions, all coming bids are pushed into a queue; this queue will keep the bids safe from accessing the updateData service simultaneously. The client will either receive true to indicate the bid is valid and the item is already updated. Otherwise the client receives false to indicate a bid rejection (the bid was less than the current price of the item). This service is considered the bottleneck of the system. In other words, this service will be the called the most number of times. The accepted bid will cause the updateWinner( ) service to be called with two arguments, the item code and the client name. This service will be explained later in this chapter.

```
PLAN_BOOL updateData (PLAN_STRING code, PLAN_STRING name, PLAN_INT val)
 {
   PLAN_BOOL result= PLAN_FALSE;
   int i = 0;
      while ( i < itemsData.size)    {
     if (strcmp (itemsData.pValues[i].val.szVal , code ) == 0 )    {
       printf("The Item Found\n");
       if( atoi(itemsData.pValues[i+1].val.szVal) < val )     {
         printf ("Update Item :%s", code);
         printf ("With new Price%d ", val);
         result = PLAN_TRUE;
         updateWinner(code,name);    // call to update the winner
         break;     }
       else    {
        printf ("Bid is less than the current price \n ");
        result = PLAN_FALSE;        }
      }
     else    {
       printf("The received Item not Match the item Code\n") ;    }
     i = i+2;     }
 return result;
 }
```

Fig. 5.7: updateData Service (normal mode)

Update the Item Price on the active node (secure version):

```
PLAN_BLOB updateData (PLAN_BLOB  data )
```

In the secure mode updateData service was changed to accept encrypted data as a byte array. Figure5.8 illustrate the new changes. The parameters have been changed to a PLAN byte array. The received data (ciphered data) representing the user name, item code and user bid value is in an encrypted form. The data will be decrypted and the service will go through the same process as the non-secure updateData service version.

```
  PLAN_BOOL updateData (PLAN_BLOB cipher)
{
  PLAN_BOOL result= PLAN_FALSE;
  FILE *cfptr, *cipptr;
  char code[10];
  char name[20];
  int  val;

printf("Received CipherText with Value size=%d, bytes=%s\n", cipher.size,
cipher.bytes);
execlp("PLANEncGate/java SAADecrypt",NULL);
if ((cipptr = fopen ("PLANEncGate/DES3CIPH.out","w+")) == NULL)
   fprintf(cipptr,"%s",cipher.bytes);
   fclose(cipptr);
if ((cfptr = fopen ("PLANEncGate/bid.dat","r")) == NULL)
   printf("File could not be opend. \n");
    fscanf(cfptr,"%s%s%d",code,name,&val);
    printf("%s",code);
    printf("%s",name);
    printf("%d",val);
  fclose(cfptr);
```

Fig. 5.8: updateData service (Encryption mode)

Update the Item winner List on the active node:

PLAN_UNIT updatewinner (PLAN_STRING stritem, PLAN_STRING strname)

This service is an internal service called from the updateData service. Figure 5.9 illustrates this service with two arguments, the item code, and the user whose bid has been accepted. This service is very important to keep the active node intelligent to know the last person making the bid, which specifies who should buy this item. This service first checks whether this item exists on this active node or not, then checks whether this item is exists in the winner list or no. If the winner name exists, it will be changed, otherwise it will add both the item and the winner to the list. This service will never face a race condition, because it will be activated as soon as a new valid item is accepted from the updateData service. As we mentioned before the winner list will be called from the auction server to see the latest item winner. See Appendix A for more low-level PLAN implementations details, and more in PLAN grammar, and PLAN service documentations.

```
PLAN_UNIT updateWinner(PLAN_STRING stritem, PLAN_STRING strname)
    {
      while (i < winner.size)     {
          if(strcmp(winner.pValues[i].val.szVal , stritem)==0)          {
            printf ("Update Item%s%s%s ",stritem ," with new winner ",strname);
            winner.pValues[i].tag = STRING_TAG;
            winner.pValues[i].val.szVal = stritem;
            winner.pValues[i+1].tag = STRING_TAG;
            winner.pValues[i+1].val.szVal = strname;
            break;          }
          else if (strcmp(winner.pValues[i].val.szVal ,"Empty")==0)          {
            printf ("Update Item%s%s%s ",stritem ," with new winner ",strname);
            winner.pValues[i].tag = STRING_TAG;
            winner.pValues[i].val.szVal = stritem;
            winner.pValues[i+1].tag = STRING_TAG;
            winner.pValues[i+1].val.szVal = strname;
            break;          }
       i = i+2;    }      }
```

Fig. 5.9 updateWinner Service

The general scenario for communication between the application (server or client) written in java (using java-PLAN stubs) with PLAND starts from creating an active packet, which will contain a PLAN program. These active packets get passed through the PLAND ports to the active node, and then the program gets evaluated. As soon as an extra functionality of the PLAN program is needed, a resident service is called and converts the PLAN data type to C data type (using PLAN-C bridge) for evaluation.

5.2 Security Implementation Model:

When connected to the Internet, the system will be connecting to many unknown networks and their users. Three issues must be addressed. First, how to make sure that the received data is from the valid sender (authentication). Second, the eligibility to access the services depends on the level of privilege (authorization). Third, how to protect the information from those who are not authorized to access it (confidentiality).

5.2.1 Authentication:

Authentication is applied using two levels of hierarchy. First, the host application injecting PLAN programs must authenticate itself to the active node. Second, a shared secret key is created to either digitally sign or encrypt the transmitted data. To accomplish the above, Diffie-Hellman key exchange protocol [22] was implemented. For the host application (implemented in JAVA) to authenticate itself to the active node, it must maintain an initial certificate, which includes the signers public key, a cookie, the time at which the certificate is not valid before, the time at which the certificate is not valid after, the senders Exchange ID, the receiver's address (active node) and the sender's address (Application host, port). Figure 5.10 illustrates the JAVA implementation of the certificate. Once the first certificate is generated, a PLAN program is injected to call the DhmessageOne ('a, blob) service. The service takes two arguments, the generated certificate and its digital signature. The digital signature (in string form) looks like "R: <digits>, S: <digits>". Both the public and private keys are used to generate the digital signature [22,23].

```
/* The Plan Certificate  */
   byte[ ] signer ;              /* DSA (Digital Signature Algorithm) Signer */
   byte[ ]   cookie ;            /* Random data to mix it up a bit */
   long notValidBefore ;      /* Expiration date */
   long notValidAfter ;        /* Expiration date */
   int  exchange_id            /* Exchange ID generated first at sender */
   Value.Host  node_host  /* PLAN host address */
   InitAddress application_address       /* Application Address  host & port*/
```

Fig. 5.10: JAVA stub for PLAN security certificate

The active node receives the first message, and verifies the signature of the certificate with the certificate signer's public key. Then, it makes sure that the certificate is still valid (current time > notValidBefore and current time < notValidAfter). The active node builds a list and saves an entry for the application. Figure 5.11 illustrates the active node output after it receives the first message from the application [23]. The active node stores the received exchange ID and calculates its own exchange ID. All messages transferred between the application and the active nodes are of type PLAN Tuple [22,23,24].

```
Authproto: [ DHmessageOne ] with cert (<blob:566>, <blob:39>, 972591217,
972592217, 0, 129.173.67.2:5001, (154.5.35.117:5000, <port:154.5.35.117:5000,7>))
Storing entry with key <Application: ((154.5.35.117:5000), <port:154.5.35.117:5000,7>)
```

Fig. 5.11 Active Node output as a result of receiving the first message

Figure 5.12 illustrates the active node secret key generation. PLAN generates a response certificate (PLAN Tuple) and its signature. The certificate includes the exchanged ID's, the public value, the application public key and both addresses. [23,24].

```
Authproto: [ DHmessageTwo ] with cert (<blob:566>, <blob:39>, 972577600, 9725782
00, 0, 0, <blob:308>, <blob:566>, (129.173.67.2:5001, <port:129.173.67.2:5111,1>),
129.173.67.2:5001)
Calculated shared secret
|403797324765797599377814131191646911047575745858487720
9641074638524781322812378316096590892694462738187451155071487728287030
4623067677283580289113352614247251161063602437442626270414901537567714
1246633839467151281671207741166551434659260758840984048511672076679110
30483579546662130750729273554872715321136225|
sSPI=0
rSPI=0
```

Fig. 5.12 The Application output as a result of receiving the second message

Based on the pre-defined (p and g) global values, the application starts computing the shared secret key, verifies the signature of the received certificate, looks up the exchanged ID and verifies the certificate validity.

Figure 5.13 illustrates the result of a successful processing of message two sent by the active node. The application host sends the certificate back to the active node and invokes

the last service DHmessageThree [23,24]. The active node receives the final message and repeats the actions taken by the application for the previous message. The exchange process is successfully completed [22,23,24].

```
Authproto: [ DHmessageThree ] with cert (<blob:566>, <blob:39>, 972577600, 9725782
00, 0, 0, <blob:308>, <blob:566>, (129.173.67.2:5001, <port:129.173.67.2:5111,1>),
129.173.67.2:5001)
Calculated shared secret
|21257744116579759937781413119164691104757574585848 7720
```

Fig. 5.13: Active Node output as a result of receiving the third message

Figure5.14 shows the JAVA stub implementation for certificate conversion from JAVA data type to a PLAN data type.

```
/* Convert a certificate to a PLAN value.  The PLAN value is a tuple of *
 * which there are two kinds: one for the initial request and one       *
 * for the remaining messages.                                           */

    Vector vecCert = new Vector();
    Vector   senderAddr = new Vector();   // Application Address is VTuple [Host X port]

   // just for the first connect request to the node

vecCert.addElement(new Value.Blob(signer));
vecCert.addElement(new Value.Blob(cookie));
vecCert.addElement(new Value.PString(String.valueOf(notValidBefore)));
vecCert.addElement(new Value.PString(String.valueOf(notValidAfter)));

// User exchange IDall exchange_id( ) to calculate the XID
vecCert.addElement(new Value.Int(exchange_id( )));

// get the current host X port and change it to plan datatype
vecCert.addElement(new Value.Host(activehost));

// The application is host x port
try{
    senderAddr.addElement(new Value.Host(new
                    Activehost.IPv4UDP(InetAddress.getLocalHost(),5000)));
    senderAddr.addElement(new Value.Port(new
                    Activehost.IPv4UDP(InetAddress.getLocalHost(),5000),randInt));
   }catch (Exception e){ }

// add the peerAddr Vector -> VTuple to vecCert
vecCert.addElement(new Value.VTuple(senderAddr));

// convert to VTuple Plan DataType
valTuple = new Value.VTuple(vecCert);
```

Fig. 5.14: JAVA Representation of PLAN Certificate

5.2.2 Authorization:

Auction systems have two interactive parts, the auction server and the clients. Each part has its own active services to interact with. The eligibility to access the services depends on the level of privilege. The design considers every application that tries to access the active node as un-trusted application. In other words, the active node must keep a map of the principal users such as client applications and server applications.

Figure 5.15 illustrates the security policy for the Query Certificate Manager (QCM) [25,26]. The implementation uses QCM with few changes [27]. The changes consist of adding the auction services that are used by the server (*setDataList*, *getDataList*, *setWinner*, *getWinner*, *addItem*, *removeItem*) and the services that are accessed by the client (*updateData*, *updateWinner* use, *getNewPrice*) [24,25].

The authorization process is activated as soon as a host application (Client/Server) tries to access the active node services. The active node receives the application public key and checks with QCM. If the key is in the sets of keys, the active node checks for eligibility to using the required services [27]. If the service is defined under this public key, then it will be accessed. Otherwise, a service not present exception will be thrown.

```
online
{
  enhancements = {
     Principal(<PublicKey="P:<digits>,Q:<digits>,ALPHA<digits>Y:<digits>)
    };
    enhancements0 = {“setDataList”,”getDataList”,”setWinner”,”getWinner”};
    acl = {
        ( enhancements, enhancements0, {} )
    };
  enhancements1  = {
     Principal(<PublicKey="P:<digits>,Q:<digits>,ALPHA<digits>Y:<digits>)
    };
    enhancements2 = {“updateData”,”updateWinner”,”getNewPrice”};
    acl = {
        ( enhancements1 , enhancements2, {} )
    };
}
```

Fig. 5.15: Security policy (QCM)

5.2.3 Encryption:

Secure online auction systems must provide secure facilities (e.g. encryption) to protect exchanged data between the host application and the active node.

An online auction system should protect the bid unit (item PIN code, user login name and the bid value). In this design, an open-source encryption scheme, Cryptix toolkit, [28,29] is used to encrypted/decrypt the bid units between the host application and the active node. Cryptix is a cleanroom implementation of Sun's Java Cryptography Extensions (JCE) version 1.1. In addition to that it contains the Cryptix Provider, which delivers a wide range of algorithms and support for PGP 2.x. with new effort going into the next generation based on the Sun JCE 1.2 specification.

Furthermore, the encryption scheme is implemented with features to prevent replay attacks (bi-directional data encryption is provided). At the client side, the Cryptix toolkit is capable of the following:

- Generates a triple-DES key,
- Read the bid data (the item code, login name, & bid value).
- Encrypt the client bid using the electronic code book (ECB) mode, see Appendix B.

And at the router side, Cryptix toolkit is capable of the following:

- Decrypt the received cipher text and write the plaintext data to the C service to continue the bid update process.

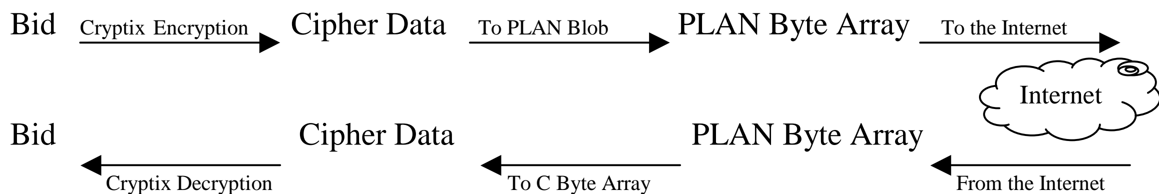


Fig. 5.16: Encryption steps

Since the size of the sent and received data is small, encryption and decryption processing is fast. It takes on average between 19 to 23 milliseconds to complete the encryption operation. On the other hand, it takes on average between 21 to 25 milliseconds to perform the decryption operation [28,29]. Figure 5.16 illustrates the encryption process between the host application and the active node [28]. The figure

shows that bids will remain private and secure even if the auction system became more widespread. Refer to appendix B for detailed explanations of Criptix Library.

This implementation provides two different levels for bid protection. The auction system administrator can select data protection using either digital signature or data encryption, depends on the level of system performance and security needed by the administrator. Please refer to appendix C for a detailed system implementation.

Chapter 6
ONLINE AUCTION SYSTEM OPERATION

This chapter provides the required steps to run the online auction system. Furthermore, it illustrates how it operates under different conditions. The implemented auction system consists of three applications: the auction active router(s), the server application and the client application. Please refer to appendix C for a detailed system activation process.

Figure 6.1 shows the diagram overview of the setup. The active database server starts with the auction items, which will be distributed over the PLAN active routers. The clients connect to the active routers and start the auction bidding process.

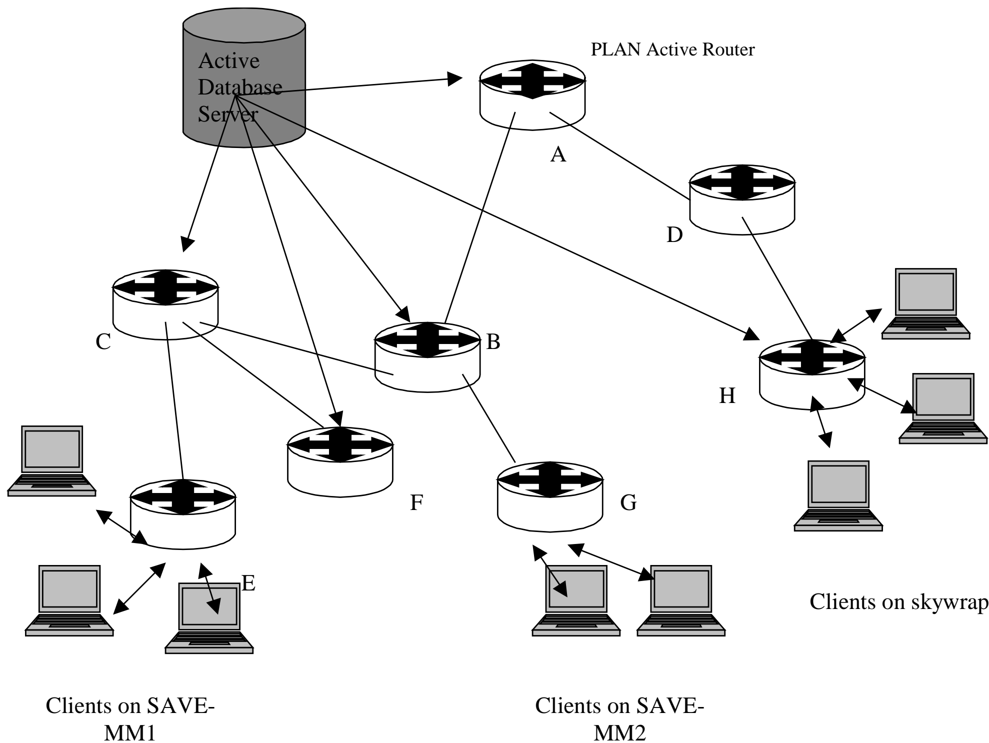


Fig. 6.1: Overview of secure online auction system

6.1 Active Auction Routers

The test bed topology consists of 8 active routers. The active routers must be activated on the test bed hosts (`nemain.cs.dal.ca` and `badb.cs.dal.ca`). The following commands will activate the routers:

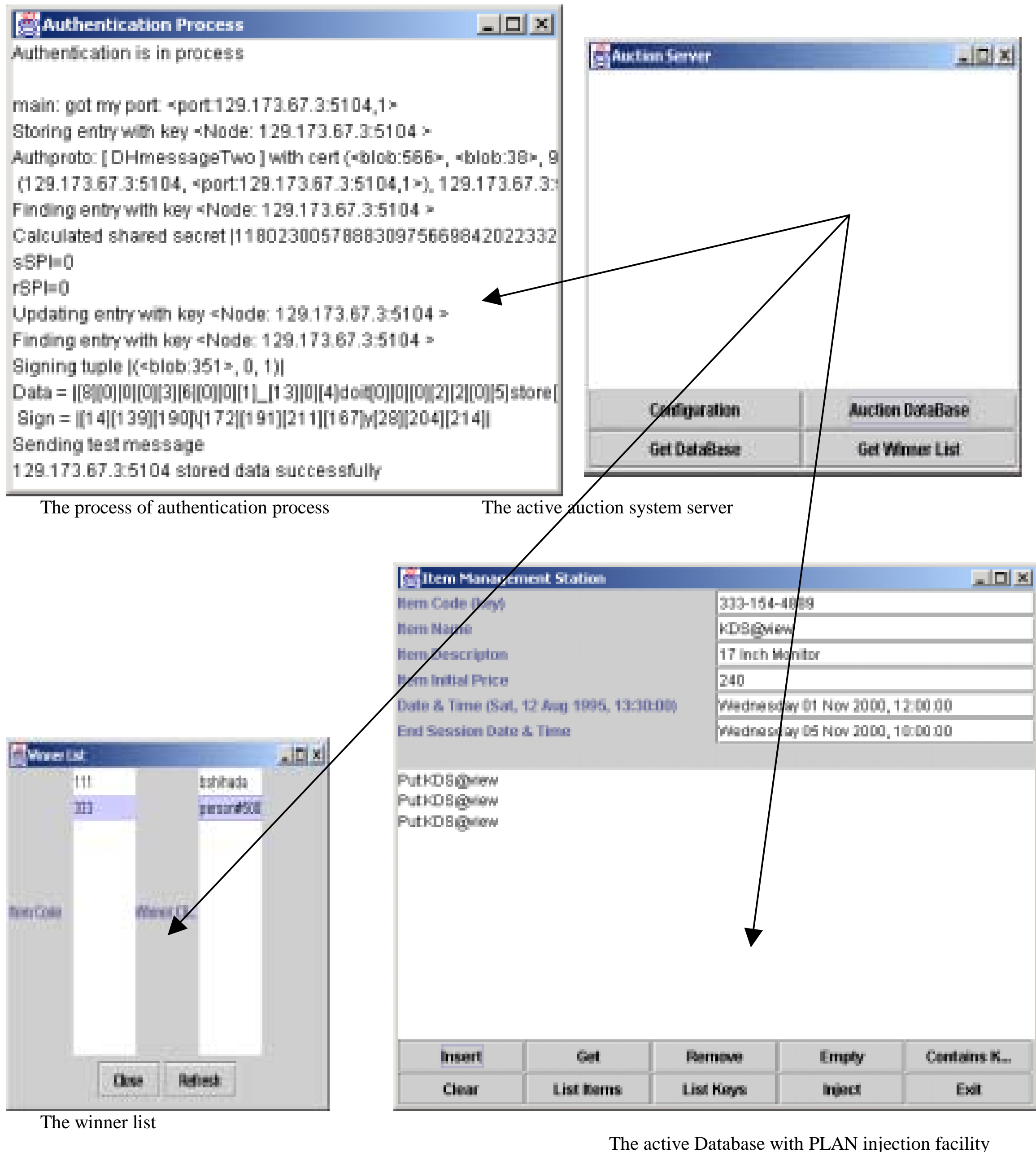
```
pland -ip 5001 -hf hostfile  -policy expand.qcm a      pland -ip 5003 -hf hostfile  -policy expand.qcm e
pland -ip 5101 -hf hostfile  -policy expand.qcm b      pland -ip 5103 -hf hostfile  -policy expand.qcm f
pland -ip 5002 -hf hostfile  -policy expand.qcm c      pland -ip 5004 -hf hostfile  -policy expand.qcm g
pland -ip 5102 -hf hostfile  -policy expand.qcm d      pland -ip 5104 -hf hostfile  -policy expand.qcm h
```

The above shows each individual active router starting by assigning it a port number and an interface file. The policy parameter has been used to indicate that the active router will be able to use a security policy in this case named expand.qcm. This QCM related policy provides each active router with rules indicating the level of service-access privilege for host applications. By starting the active routers, the network topology is up and running. The network topology can be maximized or minimized depending on the geographical disparity of the clients.

6.2 The Auction Server

As indicated previously, the communication between the host applications and the active routes is accomplished using PLAN programs via PLAN Ports. The first operation requires the auction server to compile the required PLAN programs for all auction transactions. Then the auction server starts the authentication process with each and every active node. Figure 6.2 illustrates the server authentication process with one active node.

The auction server GUI is changed to handle various functions (*get database*, *get winner list* and *add/remove items*). The auction server is directly connected with an inter-server database.



The process of authentication process

The active auction system server

The winner list

The active Database with PLAN injection facility

Fig. 6.2: Screen snapshots of Active online auction system platform: Server operations

This database will be distributed to each active router. The database consist of *insert, get, remove, empty, list items, clear, list keys,* and *inject functions*. The auction items are injected to the active routers using the inject button.

As shown in Figure 6.3 the output of the active router process after receiving a copy of the injected database. The active router saves the database, initializes the winner list and

exits the service. For clarification purposes, the active router is programmed to print interactive messages at each stage it reaches. An item list is created and the items are stored as codes followed by the item price.

According to Figure 6.2, the auction server has the capability of viewing the list of winners. This list is designed to show the item code and the latest user name that bid the highest price on the item. The server can retrieve the winner list from the active router as many times as it want.

```
Entered setDataList function
Initalize the list for the items
Received item #0is:333-154Received item #1is:240Received item #2is:78674Received
item #3is:240
Copy the items list to the service DONE!
Initalize the list for the Winners
Exit setDataList function

Entered setDataList function
Initalize the list for the items
Received item #0is:333-154-4889Received item #1is:240Received item
#2is:99689Received item #3is:240
Copy the items list to the service DONE!
Initalize the  list for the Winners
Exit setDataList function
```

Fig. 6.3: Active node when receiving the items from the auction server.

6.3The Auction Client

The auction client application starts and then compiles PLAN programs that are required to accomplish the client transactions. Figure 6.4 illustrates the authentication process between the client and the active router. The result of the authentication process appears in a separate screen. The auction client GUI has extended functionality such as, *get latest item price* and *send bid* to the active router. Clients have the opportunity to bid on any item they want. The correct port connection to the active router and a valid item PIN code are required to start bidding. On the other hand, the client is capable of knowing the latest price of existing bid items during the auction session.

At the moment when an active router receives bids from a client, it checks whether this item is still available. If true, the router checks the bid price if it is greater than the

current price. If true, the value of the item will be updated with the received value. As a result the winner list will be updated with the client login name. Figure 6.5 illustrates multiple cases of active router situations during the stages of the bid process. Figure 6.6 illustrates the active router stage process when receives an encrypted bid from client.

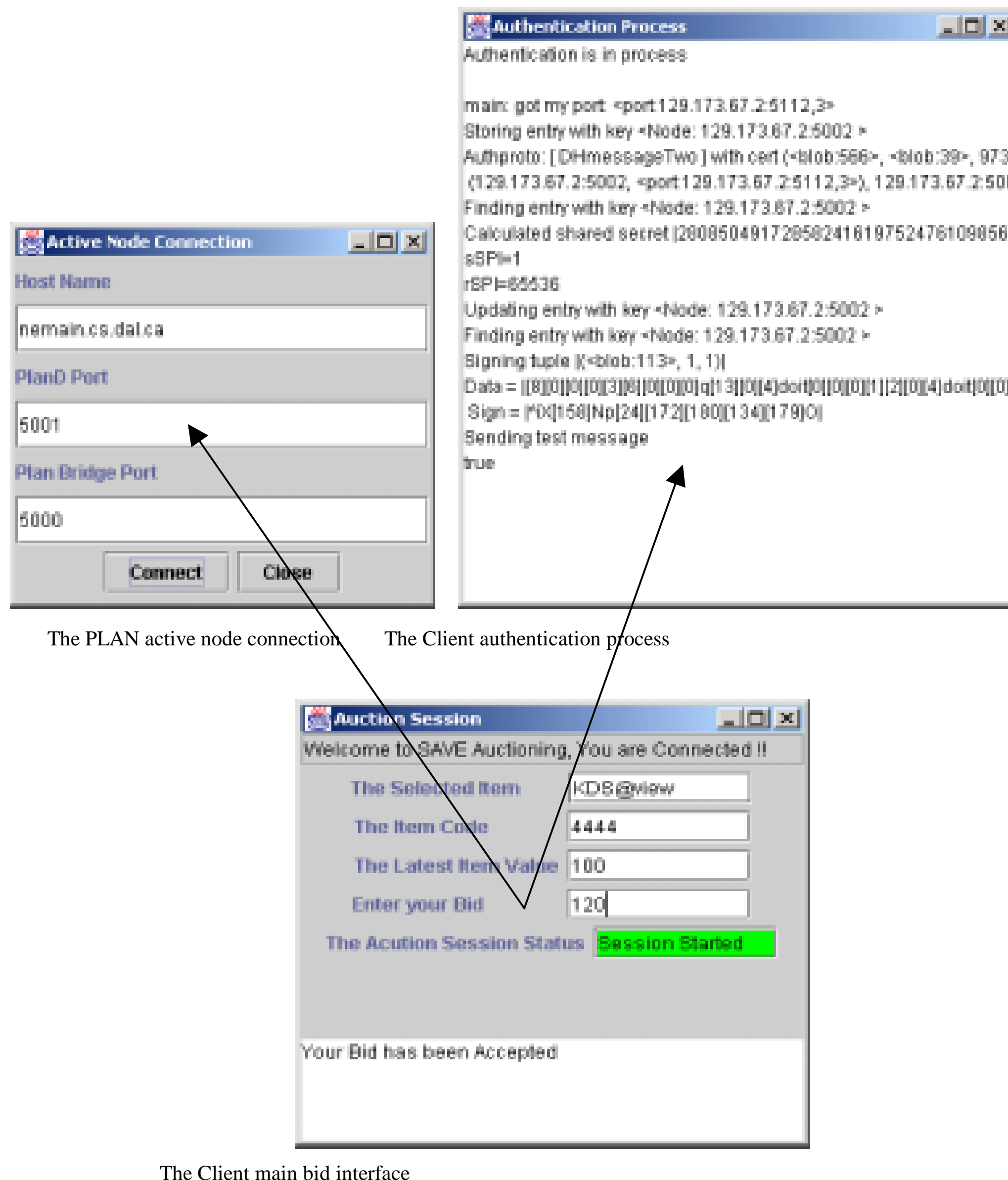


Fig. 6.4: Screen snapshots of Active online auction system platform: Client operations

```
Entered updateData function
4444
100
The received Item not Match the item Code
Exit the updateData function

Entered updateData function
4444
100
The Item Found
Update Item :4444With new Price120
Update Item4444 with new winner person#500
New Winner
Exit the updateData function

Entered updateData function
4444
120
The Item Found
Bid is less than the current price
 Exit the updateData function
```

Fig. 6.5: Active router operations when receiving bids (normal mode)

```
Received CipherText with Value size=24, bytes=£‘`Ï  J.#6–,p©®Q£Ëq|0Ç

Un-Decrypted Text is
TWRT4245shihada120

Start updateData Process
Item Code TWRT4245
Current Value 100

The Item Found
Update Item :TWRT4245 With new Price120
No cipher text is in the queue !!

Exit the updateData function
```

Fig. 6.6 Active router operations when receiving bids (encrypted mode)

Chapter 7

EXPERIMENTAL RESULTS

In this section representative results are presented to highlight the application performance. The performance of this design is a reflection of the overall system transaction time, bid latency, bid collision, routers usage (CPU, Memory) and the data encryption time. Several random item PIN codes, item prices, user names, session periods are used to track the probability of system errors as will as bottleneck situations. The following explains some experiment parameters:

1. *Number of Transactions:* An average of 58 different simultaneous client transactions have been performed. And a total of 320 iterations received by the PLAN active routers.
2. *Number of Experiments:* A total of 48 different experiments have been performed over different time periods of day and night to get different network loads.
3. *Size of the Database:* The auction items were provided from a file, with a total of 113 item codes (PINs) distributed over the PLAN active routers. Therefore, each router receives on average 14 different items.
4. *Auction session time period:* The auction sessions ranged between one to three hours, which is relatively considered a short time period. The reason being is to enable more items iteration and more auction sessions testing.

Table D.1, D.2, and D.3 shows the numerical value resulted from the system experimental process. See Appendix D for the tables.

Figure 7.1 graphs the numerical values, which resulted from the total client transaction time. The total time is the summation of the bid encryption time, the network routing time, and the in-router data processing. It shows that the total bidding time is almost similar and ranges between 490 to 530 milliseconds.

Figure 7.2 illustrates the client bid encryption time. Using different sizes of item codes and user names its shown that the encryption time ranges between 10 to 20 milliseconds. This test gives an indication of how fast the Cryptix 3.2 toolkit encryption engine is over the system hardware. The graph peaks have been caused due to the internal unknown functionality of Cryptix toolkit.

Figure 7.3 illustrates how much CPU usage the router used through different auction conditions (e.g. at one stage the active router has received 12 different bids at the same time. The CPU usage has reached 4%).

Figure 7.4 illustrates how much memory usage the active router used during several auction session periods. It can be seen that the active router used an average of 3% from total memory (64MB).

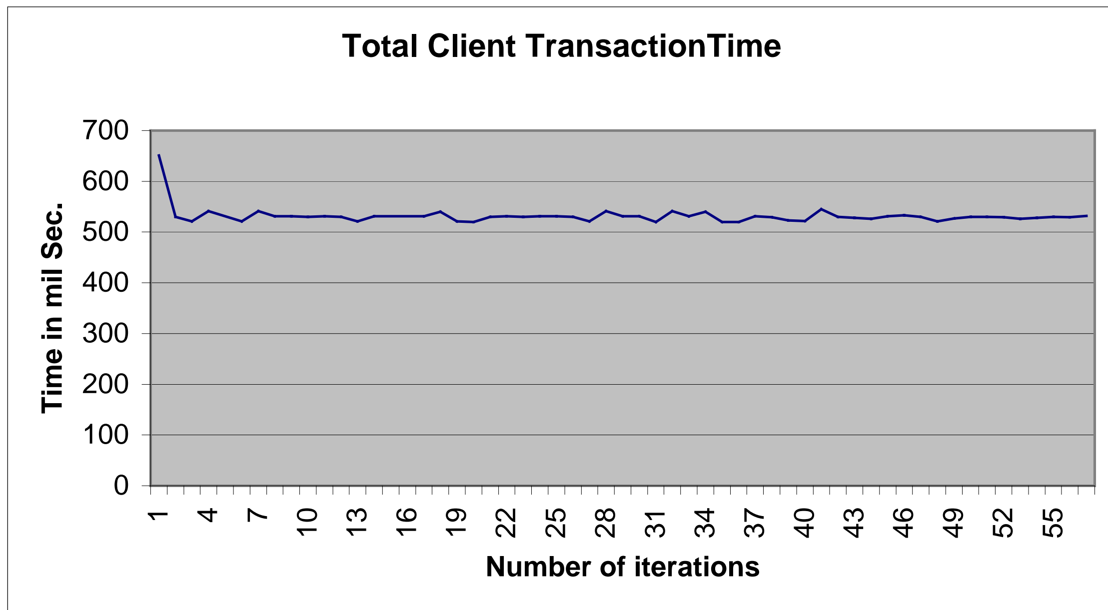


Fig. 7.1: Total transaction time

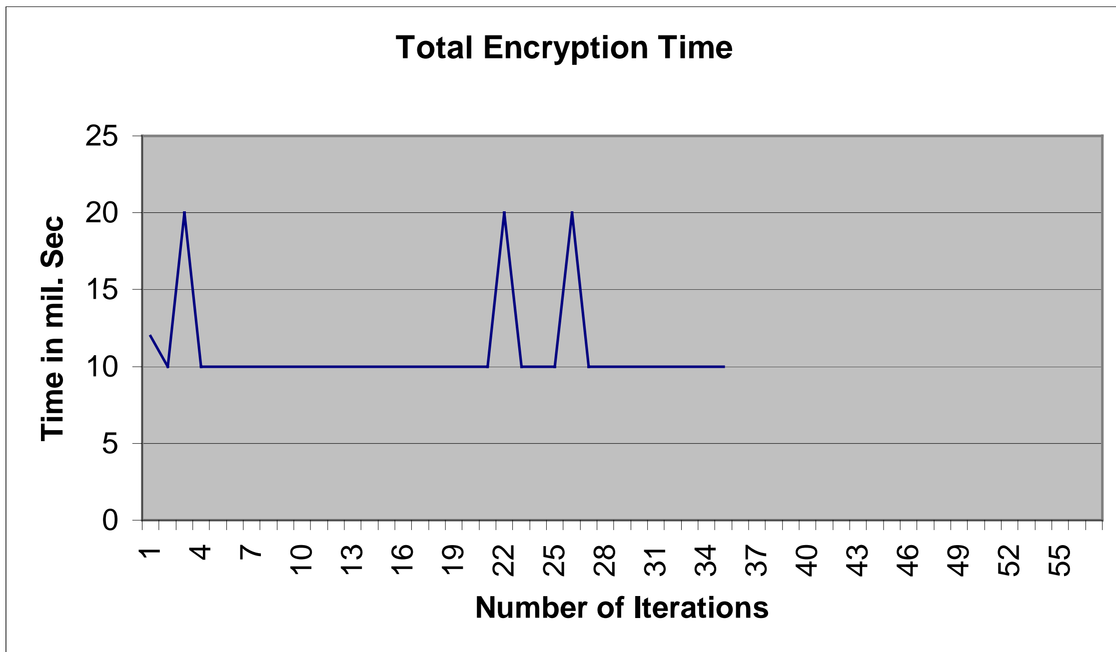


Fig. 7.2: Encryption time

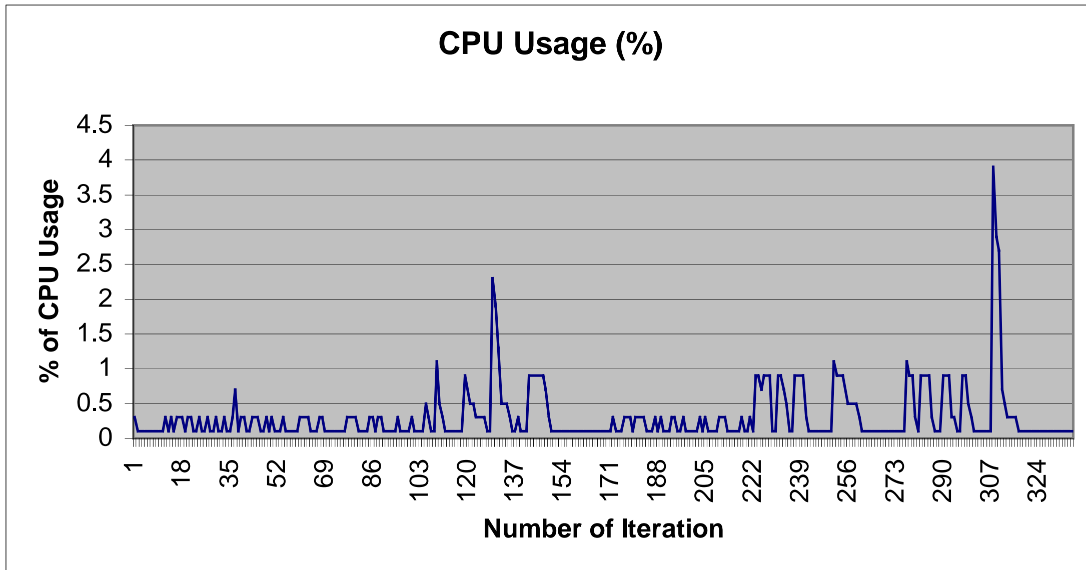


Fig. 7.3: CPU usage

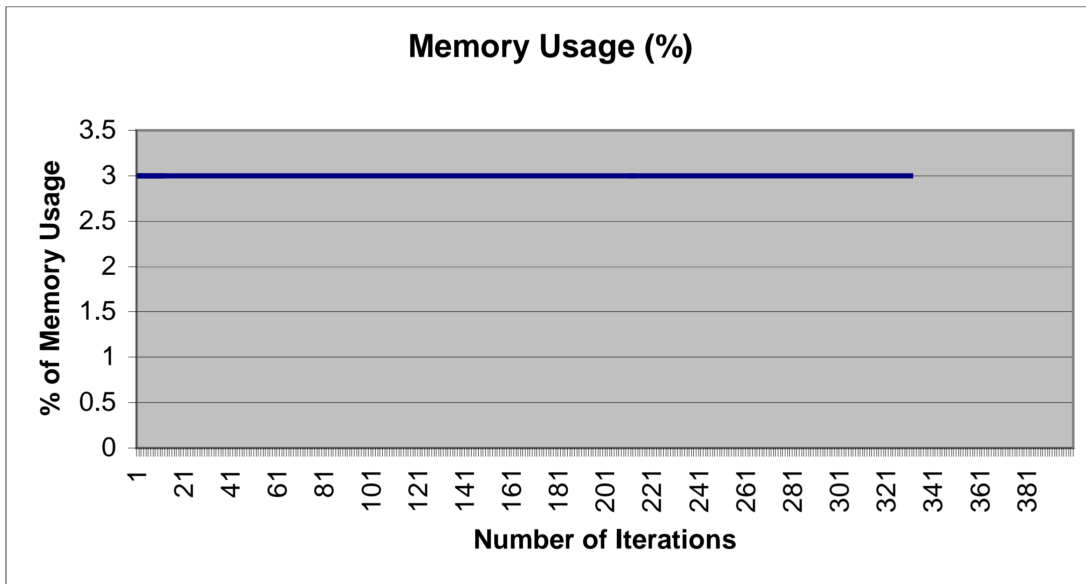


Fig. 7.4: Memory Usage

The results described in this chapter lead us to conclude that active network can be inherited and used to improve the auction application performance. The results shows that even with high demand on the PLAN active router, it still can perform the auction tasks with low CPU and Memory usage. The client total delay time is stable and kept minimum (around 500 milliseconds) under different time and network circumstances.

Chapter 8
CONCLUSION & FUTURE WORK

Active network technology may well become an important technology in the field of computer networking. Programmability of network devices (i.e. routers) gives the network more intelligence and provides the network application the flexibility needed for mobile activity (i.e. code migration). Generally, the distributed architecture of active networks provides a good solution to reserving the communication bandwidth and better optimizing the auction distributed management performance.

This thesis presented an active network approach to the design of secure online auction system. The complete process of validating the incoming bids on the active routers before reaching the server frees bandwidth that can be utilized by the server to do other tasks and reduce the server amount of computation. Also illustrates the security issues facing this approach. The implementation uses PLAN as a glue language between the active router services. PLAN was designed to provide flexibility, reliability, security and functionality [9,10,11].

8.1 Contribution of the Thesis

The primary contribution of this thesis is the design and implementation of a secure online auction system using active networks. The goal of active networks is to increase the set of design options available in distributed systems. The flexibility of a programmable infrastructure introduces considerable work for network control and management. The following goals were achieved in the design:

1. *Evaluation of a distributed application over Active Networks:* Evaluate the performance of online auction system (distributed application) over active networks from the theoretical and practical perspectives.
2. *Distributed Servicing:* Give the active node the responsibility to work as an intermediate multi-operational unit. This unit will take care of security, node operations and online auction session requirements.
3. *Distributed Security:* Adopt new distributed security mechanisms that are applied over active network. Authentication process is done between each application and

the active node independently. Each application has its own private key exchanged independently and saved on the active node. The data, which transferred between the application and the active node, gets encrypted to ensure data confidentiality.

4. *Quality of Service:* Providing a distributed application, with a network support will reduce the packet congestion, enable higher bandwidth and free up the server.

8.2 Features of the Design

The implemented auction system has provided several advantages to the online auction systems; the following is a description of these features:

1. *Reduced bid collisions:* The design provides a distributed pre-validation process that allows multiple bidders to send their bids at the same time (bids not necessary to have the same value).
2. *Expanding the reach of the server:* Using active routers, the control process of the server can cover a wider geographical area. The process of validating the trader bids is started and terminated inside the network using active routers. This expands the server reach allowing better server utilization.
3. *Distributed Security processes:* If a trader connects to the active node, the process of authentication and authorization will be activated in a distributed manner. Authentication makes sure that a secure private key is exchanged between the PLAN active router and the client. The process of authentication leads to use the private key to either digitally sign or encrypt the data. The authorization process gives the PLAN active router the ability to become a firewall to pre-identify the trader transactions. A QCM policy is added to control the privilege level of service access.
4. *Server bandwidth adjustment:* The active router checks and validates the bid reducing the server workload. Bids arriving below the actual price of the item and un-authorized traders are filtered within the active node.
5. *Monitoring the Auction Rules:* The active router takes care of who wants to withdraw from the auction session, checks the auction session time, checks the

trader status and other auction rules. This allows the server to take care of the other tasks like monitoring the bidding agents (area of extension to electronic auction).

8.3 Moving from Prototype to Real World System

The prototype implemented in this thesis has the potential to be transformed into a commercial product. The current Internet architecture dose not support active nodes. There are two ways in which businesses can utilize the ideas proposed in this thesis. First, router usage can be rented from service provider, which will allow businesses to deploy active network software on the routers. Second, a community of active network nodes that has already been deployed like the Active Network Backbone (ABone) [] can be used by the businesses. This way the end-users will not see any costs of the implementation.

The prototype is robust in the event of failures. For example, if a connection between the client and the router breaks down or if the router crashes, all the pre-done operations can be retrieved because they are saved on the permanent storage each time an operation is performed.

The prototype is robust in the event of security threats. The system implements authentication of the sender (using certificates), implements authorization to access the service depending on the level of privilege, and data encryption to ensure that the data is confidential.

However, additional work needs to be done to study the full effects of failures and bandwidth caused because of the complexity of the network environment.

8.4 Future Work

Active networks provided the discussed online auction system with a distributed structure to all the active routers. This allows bidding session to start on each individual router. For a future considerations the current structure can be expanded to function in a hieratical configuration, as Figure 8.1 illustrate, the network becomes a compound of multi-layers routers; each runs an active router with different assigned services. The main auction server will distribute the items database to the edge routers of the network. Then,

the biding sessions start. The clients will connect to the routers that are geographically located close to them to minimize the biding time delay. Therefore, every router will have the same item database and the same start value for each item. The item values will be different by the end of the bidding session.

Once the auction server wants to collect the list of the winners, routes at level B collect the list from routers at level C. While traversing up to the top level, level A routers filter the winner list before passing it to the auction main server. Using this approach we can maintain an almost inside the network processing of the auction application. Providing substantial reduction of overhead processing by the main auction server or any individual router.

However, the auction system suffers from a database problem known as data redundancy, because each router of level C will hold the same item code.

A simple user management protocol could be introduced to manage and track clients interested over any item. Incorporating the management protocol in active routers will be responsible for moving items from one router to another depending on the clients demand.

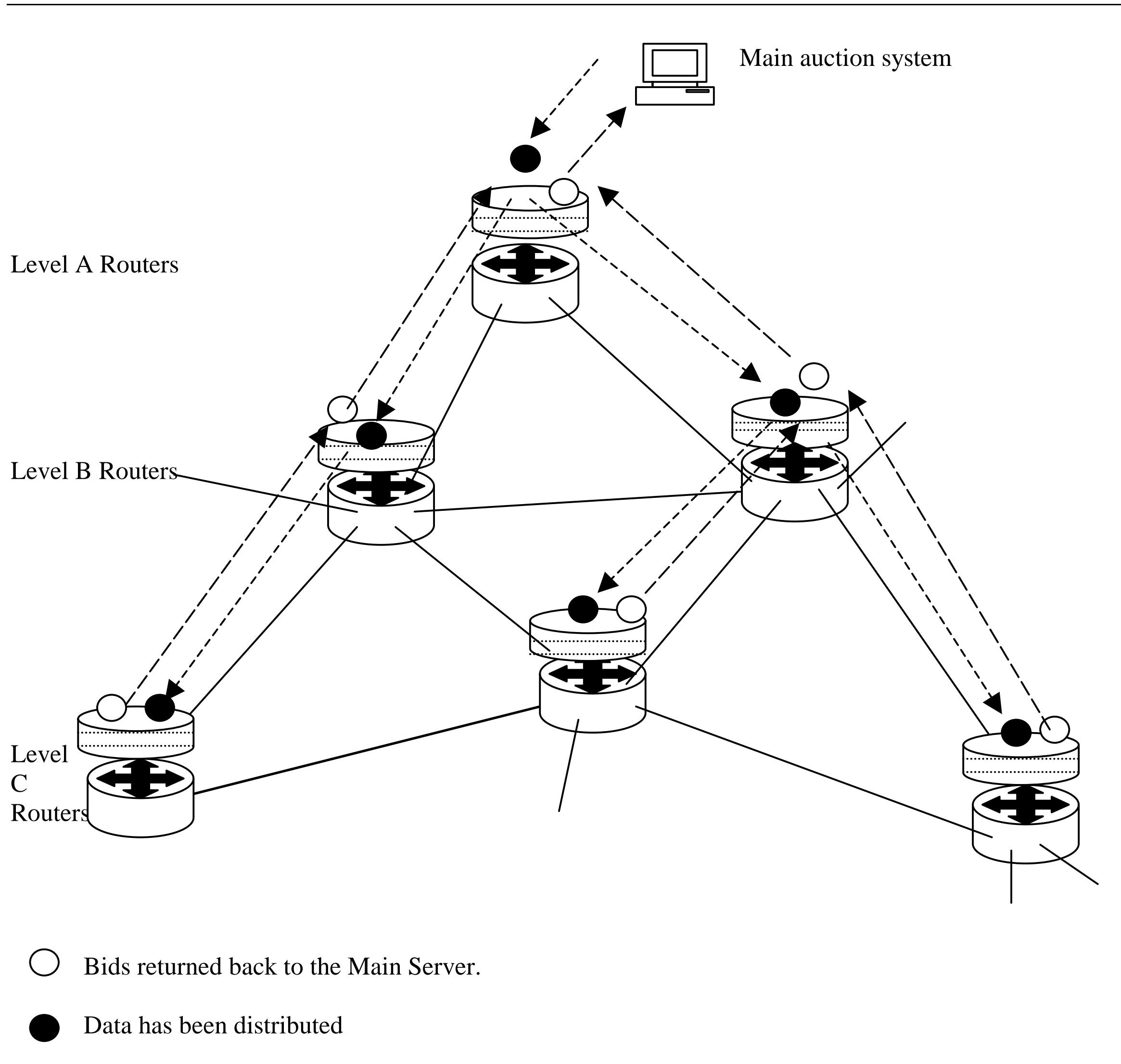


Fig. 8.1: Extension of the implementation using a hierarchical auction structure.

On the other hand, other extensions can take place to change the auction rules to include other types of auctions. For example, *Dutch auction*, requires the PLAN active router to maintain a service to start a descending counter to reflect the item price.

First-Price auction, requires from PLAN active router to maintain a service to keep track of the first bid received from the client and then filter the coming bids to get the highest item price. *Vickrey (Second-Price) auction*, requires a service to keep track the coming bids and filter them to get the second highest bid, to be the item price.

As mentioned in chapter 1, the goal of this thesis was to develop efficient and highly portable secure online auction system to perform various client needs. I hope that the

results were encouraging and the experimental results are reflects the improving performance of the application and reduces the total server overloading. Therefore, it can be seen that combining the active network technology with online auction systems gives the auction systems better performance.

Appendix A—PLAN

Definition:

“PLAN (Packet Language for Active Networks) is a new language for programs that form the packets of a programmable network. These programs replace the packet headers (which can be viewed as very rudimentary programs) used in current networks.

PLAN programs are lightweight and of restricted functionality. Allowing PLAN code to call node resident service routines written in other, more powerful languages mitigates these limitations. PLAN programming environment applied on an IP-free internetwork. PLAN is based on the simply typed lambda calculus and provides a restricted set of primitives and datatypes. PLAN defines a special construct called a chunk used to describe the remote execution of PLAN programs on other nodes.

Primitive operations on chunks are used to provide basic data transport in the network and to support layering of protocols. Remote execution can make debugging difficult, so PLAN provides strong static guarantees to the programmer, such as type safety. A more novel property aimed at protecting network availability is a guarantee that PLAN programs use a bounded amount of network resources”[11].

Features:

- Simple typed lamda calculus based on scripting language, with the support of a set of primitives and datatypes.
- Strong and static typed and dynamic check to provide efficient and safe data transport.
- Flexibility, PLAN moves away from a world with a fixed set of operations, and into one where node resident services can be easily combined on-the-fly.
- Safety and Security, safety means reducing the risk of mistakes or unintended behavior, and by security means the usual concept of protecting privacy, integrity. This means that PLAN programs are pointer safe, and concurrently executing programs cannot interfere with one another.

- Usability, PLAN programs execute remotely so all PLAN programs are statically typeable and are guaranteed to terminate, as long as they only call service routines that terminates.
- Error handling support, PLAN has a try-catch error exception handling and number of resources the packet can archive before it destroyed.
- Support two remote executions. First, OnRemote, which enable the PLAN program to be executed over the destination host. Second, OnNeighbour, which enable the PLAN program to be executed on a host, this host must be explicitly identified as neighbor.
- Use of Data chunks. A chunk is a combination of three components, the PLAN program code, an entry point function and a list of data.
- PLAN programs are passed throw the PLAN ports, so PLAN programs are replacing the IP header and payload.

Advantages:

- A resource-bounded parameter is used to make the execution of PLAN program safe. So PLAN programs guaranteed to terminate locally and globally. That gives a direct view that it is not possible for any PLAN program to be executed forever.
- Two level architecture. PLAN programs are passed within the network, the services will be called as long as a request of more process over the data is required.
- PLAN packets are un-interpreted at any active node.
- Chunks encapsulation is used to permit protocol adaptation synchronization.
- PLANnet is a version of PLAN, which has been already used in active networks. PLANnet is a standard suite of internet-like services which has two levels of programmability, all packets contains programs written in PLAN and the router functionality is extended depending on the type of the application implemented written in Ocaml or C [21].

Disadvantages:

- Data caching has not yet been implemented, so lots of data redundancy.
- Services must be written in Ocamle, which is an ML-like syntax functional programming. The C-Bridge has been implemented to enable a C to Ocamle data conversions [21].
- Non-GUI has been added to improve the level of programming.

PLAN Grammar:

```
program            ::= def-list
def-list           ::= def | def def-list
def                ::= fundef | exndef | valdef
fundef             ::= fun var ( paramlist ) : type-expr = expr
                        | fun var ( ) : type-expr = expr
param              ::= var : type-expr
paramlist          ::= param | param paramlist
exndef             ::= exception var
valdef             ::= val var : type-expr = expr
type-expr          ::= tuple-type-list
tuple-type-list    ::= nontuple-type-list * tuple-type-list
nontuple-type-list ::= nonlist-type-exp | nonlist-type-exp list
nonlist-type-exp   ::= base-type | ( type-expr )
base-type          ::= unit | int | char | string | bool | host | port | key | blob |
                        exn | dev | chunk
expr               ::= value
                        | op-expr
                        | if expr then expr else expr
                        | raise var
                        | try expr handle id =? expr
                        | let def-list in expr end
                        | ( expr-list )
arg-list           ::= expr | expr , arg-list
expr-list          ::= expr | expr ; expr-list
value              ::= var | true | false | () | [] | [ expr-list ]
                        | int-literal | char-literal | string-literal
                        | ( arg-list )
                        | |var |( arg-list )
                        | |var |( )
op-expr            ::= id ( ) | id ( arg-list )
                        | unary-op expr
                        | expr binary-op expr
```

```
                        | nary-op-expr
unary-op        ::= ~ | not | hd | tl | fst | snd | # int-literal | noti
                        | explode | implode | ord | chr
binary-op       ::= / | % | * | + | - | and | or | < | <= | > | >= | = | <> | :: | "
                        | << | >> | xori | andi | ori
nary-op-expr        ::= OnRemote ( expr , expr , expr , expr )
                        | OnNeighbor ( expr , expr , expr )
                        | RetransOnRemote ( expr , expr , expr , expr , expr , expr )
                        | foldr ( expr , expr , expr )
                        | foldl ( expr , expr , expr )
int-literal     ::= digit | nonzero-digit digit-string
digit-string    ::= digit | digit digit-string
nonzero-digit       ::= [1 - 9]
digit       ::= [0 - 9]
char-literal    ::= ' character '
character       ::= ~[', \] | \\ | \n | \t | \b | \r | \' | \"
string-literal      ::= "" | " strchar-list "
strchar-list    ::= strchar | strchar strchar-list
strchar         ::= ~[", \] | \\ | \n | \t | \b | \r | \' | \"
id      ::= var | var . id
var         ::= varstartchar | varstartchar varchar-list
varstartchar        ::= [ a - z, A - Z ]
varchar     ::= [ a - z, A - Z, 0 - 9, ]
varchar-list    ::= varchar | varchar varchar-list
```

Secure PLAN:

Models

Two major threats to PLAN active networking system

The public resources of the system: CPU, memory, and network

The packets themselves and the information stored on routers.

For example,

1. Denial-of-Service. Because of the greater expressibility of active network programs (compared to protocol packet headers), there is greater potential for the misuse of the system's public resources, thus denying service to other programs. For general programs, the public resources should be fairly apportioned, while those with more privilege could gain additional latitude.

2. Data Protection. Programs should be protected from interference by other programs. In particular, one program should not be able to read or write data private to another program without authorization, either while the packet program is in transit or when it is running. This property implies program isolation.
3. Spoofing. As mentioned, we would like to allot greater privilege to some packets, such as those associated with the node administrator. Therefore, it is important that these packets be properly authenticated, and that no impersonation attacks be possible [22].

Architecture

The architecture is based on partition the problem of defending against these attacks into the PLAN level and the service level, using different mechanisms at each level. At the PLAN level, security is obtained via functional restriction: the nature of the PLAN language and the "core services" made available to all PLAN programs prevent attacks, particularly denial-of-service and protection attacks, from being formulated.

At the service level, we make use of an authorization system to govern access to "unsafe services" such services (e.g., network management) are necessary for the operation of the active node, but should not be made available to general users. Our architecture associates with each principal (user) a set of service routines and policies that are allowed at his/her level of privilege. The policies are enforced and the routines are made available after the user is successfully authorized [23].

Appendix B— Cryptix3.2 Toolkit

Definition:

Cryptix 3 is a cleanroom implementation of Sun's Java Cryptography Extensions (JCE) version 1.1. In addition to that it contains the Cryptix Provider which delivers a wide range of algorithms and support for PGP 2.x. Cryptix 3 runs on JDK 1.1, JDK 1.2 (Java 2) and JDK 1.3 [25].

Package list:

cryptix
cryptix.provider
cryptix.provider.cipher
cryptix.provider.elgamal
cryptix.provider.key
cryptix.provider.mac
cryptix.provider.md
cryptix.provider.mode
cryptix.provider.padding
cryptix.provider.rsa
cryptix.test
cryptix.tools
cryptix.util.checksum
cryptix.util.core
cryptix.util.gui
cryptix.util.io
cryptix.util.math
cryptix.util.mime
cryptix.util.test
java.lang
java.security
java.security.interfaces
netscape.security

Cryptix speed:

Speed is a block cipher with variable key size, data block size and number of rounds (in the style of RC5).

These parameters are set as follows:

- The key size is taken from the encoded form of the key passed to *initEncrypt* or *initDecrypt*. It can be any even number of bytes from 6 to 32 (6, 8, 10, ... 32).
- The length of the data block defaults to the value of the parameter "Alg.blockSize.SPEED", or 8 bytes (c.f. IDEA and DES) if that parameter is not found. It can be set by calling *setBlockSize*(), and can be 8, 16, or 32 bytes.
- The number of rounds defaults to the value of the parameter "Alg.rounds.SPEED", or 64 (which gives “adequate” security) if that parameter is not found, it can be set by calling *setRounds*(), and can be any number from 32 upwards, divisible by 4 (32, 36, ...).

Cryptix DESKeyGenerator

A key generator for Triple DES with 3 independent DES keys.

A total of 24 bytes are generated, with a parity bit as the LSB of each byte (i.e. there are 2^168 possible keys). The keys are encoded in the order in which they are used for encryption. A Triple DES key is considered weak if any of its constituent keys are weak, or if two or more of those keys are equal, ignoring parity.

Cryptix Encryption Modes:

CBC

cryptix.provider.mode.FeedbackMode

Implements a block cipher in CBC mode. The block size is the same as that of the underlying cipher.

CFB

Extends cryptix.provider.mode.FeedbackMode

Implements a byte-oriented stream cipher using n-bit CFB with an n-bit-sized block cipher.

The full block size of the supplied cipher is used for the Cipher Feedback Mode. The bytes supplied are processed and returned immediately.

CFB_PGP

Extends CFB

Use of this feedback mode is deprecated. It's used for compatibility only!

This class implements PGP's (i.e. Zimmerman's) non-standard CFB mode. (For the standard method. It replaces cryptix.pgp.CFB in version 2.2, and cryptix.pgp.PGP_CFB in version Cryptix 2.2.0a.

The differences between this and standard CFB are that:

- The IV passed to the underlying CFB class is always zero. An additional IV should be included as the first block of the input stream (when encrypting), and this additional IV should be unique.

- Before each encryption or decryption operation (i.e. each call to update or crypt), the CFB shift register is encrypted, regardless of the current position within a block.

For an unusual output of a cipher depends on the exact boundaries between data passed to each encryption/decryption call. Normally, the lengths of data passed to each call do not

matter as long as they make up the correct input when concatenated together - but this class is an exception.

As a result, using this mode with *CipherInputStream* and *CipherOutputStream* may produce unexpected output, and is not recommended.

OFB

Extends cryptix.provider.mode.FeedbackMode

Implements a byte-oriented stream cipher using n-bit OFB with an n-bit-sized block cipher.

The full block size of the supplied cipher is used for the Output Feedback Mode. The bytes supplied are processed and returned immediately.

PCBC

Extends cryptix.provider.mode.FeedbackMode

Implements a block cipher in PCBC mode. The block size is the same as that of the underlying cipher.

[28]

Example Encryption code used

Key generation:

```
//generate key for file encryption
 try{
   random = new SecureRandom();
   keygen = KeyGenerator.getInstance("DES-EDE3");
   keygen.initialize(random);
   key = keygen.generateKey();
   rkey = (RawKey) key;
   yval = rkey.getEncoded();
   Bkey = new BigInteger(yval);
   w = cryptix.util.core.BI.dumpString(Bkey);
     }
  catch(Exception ex)
  {
   System.out.println("Error in generating the key");
  }
```

Define Encryption technique:

```
// define the encryption mode and initialize it

Cipher des=Cipher.getInstance("DES-EDE3/ECB/PKCS#5","Cryptix");
des.initEncrypt(key)
```

Encrypt the Data

```
ciphertext = des.crypt(bt);
```

Example Decryption code used

Define the Decryption method:

```
// the decryption method
Cipher des=Cipher.getInstance("DES-EDE3/ECB/PKCS#5","Cryptix");
```

Initialize the Decryption key

```
des.initDecrypt(key);
```

Decrypt the data

```
decrypted = des.crypt(bt);
```

Sample output from using Cryptix 3.2

```
The Encryption Key = Multi-Precision Integer 102 bits long...
     sign: Positive
 magnitude: 333333207361796D 6120323230
```

This is a copy of the input Data:

333 shihada 220

```
The number of bytes in the plaintext input file is = 16
The new number of bytes in the plaintext input file is = 16
 ciphertext.length = 16
Ciphertext = Multi-Precision Integer 127 bits long...
     sign: Positive
 magnitude: 43301E860A29C1A8 EB08EB520866E108
```

The cipher data is

C0-†
)Á¨ë   ëR   fá

[29].

Appendix C— System Detail Operation

<u>Install PLAN:</u>

From an Internet connected computer go to Packet Language for Active Networks PLAN web site.

http://www.cis.upenn.edu/~switchware/PLAN/

Source Installation

PLAN was built using a number of publicly available packages. They are:

a. OCaml 3.00, an ML dialect
   See The Caml page for general information, or grab the distribution. Files here are in a number of formats (tarred/gzipped, i386 RH 6.1 RPM, etc.). Note that the PLAN software may not work with versions of OCaml other than 3.0.

b. CamlP4, a preprocessor of Caml, version 3.00
   General information or the distribution (tar.gz format).

c. OCaml patch to enable Ethernet access on Linux machines (optional)

You must first install all of these packages. Please follow the installation instructions given at each of the sites. To use the OCaml patch which enables Ethernet access, first download the OCaml distribution. Then, in the ocaml-3.00 directory, enter the command

```
patch -p1 < ocaml-patch-3.0
```

then proceed with the installation of OCaml (see the INSTALL file in that directory). Note that this patch will only work on Linux machines.

Next, unpack all of the PLAN source. For a UNIX platform, you should have obtained PLAN-ocaml-3.21-src.tar.Z. This can be unpacked simply by doing

```
uncompress PLAN-ocaml-3.21-src.tar.Z
tar -xvf PLAN-ocaml-3.21-src.tar
```

Note that these operations should be performed in the directory that you would like the source to be unpacked. This shall hereafter be referred to as the “top level directory”. This will create one subdirectory “plan-3.21”, hereafter referred to as the plan directory.

Due to problems building Camlp4, PLAN cannot currently be installed on a Windows platform. We hope to rectify this situation in the near future.

Building

Simply type make from within the plan directory. This will build the standard executables, PLAND and inject. To build all of the executables (including test executables), do make all instead. All executables are put in the bin directory. Additionally, there are some scripts in that directory that make use of these executables. All created exectuables consist of OCaml bytecode; see below for instructions to build native code versions.

The default version of the Makefile assumes a couple of things:

- The OCaml distribution is available from your path (i.e., the names of ocamlc, etc. are not fully-quantified in the Makefile).
- OCaml 3.00 has been compiled to use user-level, and not POSIX, threads. In other words, when you built OCaml, you initially called configure *without* the -with-pthread option. How to build with POSIX threads is described in more detail below. Without POSIX threads, PLAN cannot be compiled to native code (but will compile fine to bytecode).

This will build a version of PLAN that runs only on top of IP, and is "single-threaded" (that is, only one PLAN program executes at a time; others must wait for it to complete). Other configurations are possible, explained below.

Compilation options

PLAN can be built with different options depending on your needs. PLAN may be customized in two ways. First, there are a series of variables at the top of the Makefile:

```
# WINDOWS = true            ### compile under Windows (not supported yet)
# SUNOS = true              ### compile under SunOS
# USE_QCMGUI = true         ### compile with QCM GUI support
# USE_ANEP = true           ### use ANEP packet formats
# USE_POSIX_THREADS = true  ### OCaml compiled to use POSIX threads
USE_QCM = true              ### Use QCM as the node policy manager
# USE_KEYNOTE = true        ### Use Keynote as the node policy manager
```

```
# USE_PENTIUM_CC = true      ### Use pentium cycle counter for timing
USE_C_CRYPTO = true          ### Use C version of SHA1 code
```

By default, QCM is used as the PLAN security policy manager (compiling without QCM causes all authorization requests to be accepted), with crypto routines written in C. Uncomment any of the other variables to enable those options. For example, if you compiled OCaml to use POSIX threads (by providing -with-pthread to configure) then you would uncomment this option. Note that this also allows PLAN to be compiled to native code by enabling some additional targets.

Another method of customizing how PLAN is built is to modify the compilation flags in the Makefile variable CAMLP4FLAGS. The comments in the makefile have a description of the options you may use:

```
# Currently supported flags:
# UNIXNET : compile support for Ethernet access (Linux w/ modified Ocaml only)
#         this option is REQUIRED if you are building the loader and
#         have the patch installed.
# PLANPORT_TCP : should use TCP as the underlying socket mechanism
#              for plan ports (rather than UNIX domain sockets)
# PLAN_UNSAFE : eliminates some checks assuming programmer knows
#             what he's doing (yes)
# MULTI_EVAL : forks a thread for each PLAN program so that more
#            than one program can be executed at once
# QUEUE : use a queue between linklayer and network layer (rather than upcalls)
# VERBOSE : shows all messages coming in and going out
# DEBUG : adds extra print messages
# RIP_DEBUG : shows all RIP processing
# FLOW_DEBUG : shows all flow-based routing processing
# TIMERS_ON : prints timing information for various parts of the system
#
# these options will be turned on automatically based on other flags
#
# PTHREAD : should be turned on when POSIX threads are used
```

QCM : turned on when using QCM as the policy manager

ANEP : turned on when using ANEP packet format

USE_CYCLE_COUNTER : turned on when using the pentium cycle counter

C_CRYPTO : turned on when using C version of crypto code

The last five mentioned options will be added automatically by selecting one of the Makefile variables mentioned earlier (such as USE_POSIX_THREADS), so you should not set them yourself.

The default options are:

CAMLP4FLAGS = DPLAN_UNSAFE -DAUTH_DEBUG -DPLANPORT_TCP

If, for example, you wanted to make use of Ethernet (assuming you have applied the OCaml patch) and wanted to do queue-based, rather than upcall-based, processing (not recommended), you would specify:

CAMLP4FLAGS = DPLAN_UNSAFE -DUNIXNET -DPLANPORT_TCP -DQUEUE

Using Ethernet

To use Ethernet, you must be running on a Linux platform and you must have the patched version of the OCaml 3.00 distribution. Ethernet availability is based on the SOCK_PACKET socket type used for direct Ethernet access from user-level, and OCaml was modified to reveal this socket. When you've built plan, you will have to run pland and inject as the superuser for Ethernet access to work properly.

Compiling to Native Code

To compile to native code, you must be use OCaml built to use POSIX threads (by adding with-pthread as an option to configure), and set the Makefile variable USE_POSIX_THREADS in the PLAN Makefile. Then you may build the targets standalone_nat and inject_nat which will place executables pland_nat and inject_nat in the bin directory. These should operate just like their bytecode versions.

PLAN Components

The PLAN software consists of a number of different programs to allow you to set up your own Active Network. PLAN software can run either directly on top of Ethernet (given a special version of OCaml), or on top of IP. There are essentially three types of applications:

Active Router

This is a daemon that runs on your machine, processing packets received from the active network or users. There are two possible daemons that may be used:

- **pland**

  This is the standard PLAN daemon that uses a fixed (non-loadable) service base. It may also be compiled to native code for performance improvement.

- **plan_loader**

  Same as **pland** but with the additional ability to dynamically load new services accessible by PLAN in the form of OCaml bytecodes. This has slightly worse performance and may not be run as native code.

Host Applications

these are end-user applications that inject active packets into the virtual active network, and receive any output that results. The applications in the distribution are

- **inject**

  This program allows you to inject an arbitrary active packet into the active network and prints any results. Programs in the rout_tests directory form useful examples.

- **server**

  Demonstrates the use of PLAN ports. See [17] for more on PLAN ports.

- **authapp**

  Demonstrates the use of security features new to PLAN 3.2. See [22] for more on PLAN security.

<u>Utilities</u>

These are helpful tools for developing PLAN

- **plan** (*read-eval-print PLAN*)

  This program allows you to test non-network PLAN code by interactively typing in expressions to be evaluated and seeing their printed results. This is useful for

verifying that code correctly parses, that expressions behave as you expect them too, etc. Examples in the interp_tests directory may be run in this mode.

- **parsetest**
  This program parses the given PLAN program and pretty-prints it. Is useful for detecting parse errors.

- **typetest**
  This program performs the full frontend functionality on the given PLAN program: lexing, parsing, internalizing, type inference, conversion to wire format.

- **plan_keygen**
  This program may be used to generate DSA keys for use by the PLAN security infrastructure. See [22].

- **Experiments**
  There are a number of programs used to generate the experimental measurements presented [16].

All executable applications are stored in the bin directory. All can be built by doing make *name* where *name* is listed above. Thus, to build the typetest executable, you would do make typetest.

Documents

If you are unfamiliar with PLAN, you should begin with the tutorial. Versions of these documents are also available on the PLAN homepage.

Supported Architectures

This version of the PLAN interpreter has been tested on

- i586 platforms running
  - Redhat Linux 4.1, 4.2, 5.0, 5.1, 5.2, 6.0, and 6.1. Kernel versions 2.0.28-2.0.36 have been tested; best performance is acheived using Linux kernel 2.0.30 (2.0.33 seems particularly bad). The newer kernel versions also

seem OK, in particular 2.2.12. POSIX and non-POSIX threads have been used.

- Sun SPARC's running
    - SunOS 5.5.1-5.7, POSIX and non-POSIX threads.
- IBM Workstations running
    - AIX 4.3.1.0

Unless otherwise stated, all software and documents at this site are:
© Copyright 1997,1998,1999 The SwitchWare Project
Maintained by: PLAN-maint@www.cis.upenn.edu

<u>Auction system Server Programs:</u>

AuctionServer.java

This class represents the main Multithreaded Auction Server.

Methods:

- AuctionServer is the main construction at this method all variables are initialized on the Server.
- Auctioner threads are created and initialized.
- Registered and connected hash tables are initialized.
- Start the process of Aucthentication.
- Start the PLAN ports with the both routers nemain and badb.
- Start compiling the PLAN program files. From the Server side its only needed to parse items data file only
- Initialize the GUI.
- Control thread that allows continuous update to the items of the connection with the pland
- Return the database items from the 4 pland's nodes
- Registers a new User on the Server.
- Logs a User in the network.
- Remove the user from the connected users hashtable.
- Returns the user interested items and the price .
- Update the list of items in the server.
- Return the updated list of items in the server.
- Sets the itemtable on the server these items will be sent to the pland as list or vector.

How to Start the Server:

From any (windows/linux) based operating systems type the following:

```
java AuctionServer  Host_name  application_port  Max_clients
```

Auctioner.java

This class is used to manage each client (auctioneer) as a separate thread.

Methods:

- Auctioner is the main class constructor to initialize the variables.
- The main run function for each client as thread.
- Broadcast the updated item value to all clients.

How to Start the Auctioner:

Auctioner starts when the server is active and a client start a connection to the server.
ItemHashTable.java

This class represents the items database on the server.

Methods:
- Constructs the GUI on the server.
- Insert a valid item to the table.
- Get the item information back.
- Remove item from the table.
- Empty all the table.
- Inject the data to the PLAN routers.
- List items by key, which is the item code.

How to start ItemHashTable:

From the GUI of the Server, select a button called DataBase, then the item hash table will be activated and ready to do the database operations.

User.java

This class represents a user account on the network.

Methods:
- Multiple user constructions, depending on the user situation.
- Return/set the user name.
- Return/set the user password.
- Return/set the user Address.
- Identify the user connection situation (login/logout).

Item.java

This class represents the item data on the network.

Methods:
- Return/set item name.
- Return/set item code.
- Return/set item description.
- Set the latest item price.
- Return the latest item price.
- Get/Set the bidding session date and time.

Bid.java

This class represents the Bidding.

Methods:
- Constructs the Bid
- Return/set client bid value.
- Return the client user name, who made the bid.

Node.java

This class used to specify the current Active Node.

Methods:
- Identify the connected Host.
- Identify the used ports.
- Constructs the GUI for the above processes.

AddRemove.java

This class is used to add or remove items from the PLAN active database system.

Methods:
- Constructs the GUI.
- Do the process of addition.
- Do the process of deletion.

DatabaseView.java

This class is used to view the database, which already stored on the PLAN routers. This operation is connected with getdatabaselist service on the PLAN active router.

Methods:
- Constructs the GUI as two separated columns, one for the item code and the other for item price.
- Refresh, used to keep track of the latest data from the PLAN router.

WinnerClient.java

This class is used to view the winner users on the PLAN router. This operation is connected with getwinnerlist service on the PLAN active router.

Methods:

- Construct the GUI, as two columns one for the item code and the other is for the client user name.
- Refresh, used to keep track of the latest winners from the PLAN router

HostServer.java

This class represents the main server situation, list of connected users, list of registered users and refreshes the GUI.

Methods:
- Construct the GUI.
- List the registered users.
- List the connected users.
- Refresh the operations.

Auction System Client Programs:

Client.java

This class used to manage each client/server thread for bidding.

Methods:
- Initiate the applet graphical interface.
- Make a connection to server and get associated streams... start separate thread to allow this applet to continually update its output.
- The main run method to control thread that allows continuous update of the text area display.
- Process the message sent from the server or PLAN router.
- Get the latest item price.
- The User Bids will be evaluated in the Plan Interpreter.. so we need to do the following steps:
    - Change the Bid -> byteArray Blob in the Plan Data Type by Value.java
    - Prepare the Vector which will hold the parameters
    - Prepare the plan program -> string
    - Call the Inject.java
    - From Inject we will pass the bid in the deliver service
- Get the auctioner latest bit.
- Always check when to start the bidding session.
- Always check when to the session is over.
- Receive data will be called by the InjectD program to save the data received from the Plan interpreter.

HostComp.java

This class represents the main client computer host GUI.

Methods:
- Class constructor, used to construct all the data variables.
- Identification of the bidding process:
  - To set your Connection with the Active Node press Node Button
    NOTE: You must already kneed what Node holds your Item
  - To leave the Session press Logout Button
  - To Enter the SAVE Auction System press Login Button
    NOTE: To Enter the SAVE Auction System you must have Registered yourself before
  - To register yourself as New User press Register User Button
- Auction performed, will perform the actions from the user GUI.
- Set/Return the computer user will be activated from the HostRegs to set the user for this computer, this user will be used in the Client.java program
- Set the connections.

How to Start the Client computer:

From any (windows/linux) based operating systems type the following:
    java HostComp host_name application_Port

HostReg.java

This class represents the main registration user information GUI

Methods:
- Construct the GUI.
- Compo box to give the user a chance to change between available items.

HostLogin.java

This class represents the main Login GUI

Methods:
- Construct the GUI.
- Ask the user name and password.
- Display the resultant actions.

<u>Routed PLAN Programs:</u>

PlanProcessFile.java

This class used to manage each PLAN router actions.

Methods:
- Construct & initialize both the Plan Port & the Active host.
- Parse the PLAN programs.
- Send PLAN programs throw the Plan Ports.
- Get the results of the client bids (true/false).
- Get the result of the Server operations (winner list, database list).
- Get the current PLAN port.
- Print anything return back from PLAN routers (exceptions).

<u>Data Security (Encryption):</u>

SAAEncrypt.java

This class used to encrypt the data going from the client to the PLAN Node and vise versa.
Note:

Java Program 2 for Triple-DES Encryption by J. Orlin Grabbe
http://www.aci.net/kalliste/javacrypt_index.htm
I have used this technique to provide a secure data encryption to my
Auction system..
This program has been modified by B. Shihada in the 07 of Nov 2000
The modification is done to fit my purposes of Data Encryption

Methods:
- Initialize the data variables.
- Generate the key for file encryption. Using DH message exchange.
- Select the method of encryption.
- Do the data Encryption.

SAADecrypt.java

This class used to decrypt the data coming from the PLAN router, and going from the client.
Note:

Java Program 2 for Triple-DES Encryption by J. Orlin Grabbe
http://www.aci.net/kalliste/javacrypt_index.htm
I have used this technique to provide a secure data encryption to my
Auction system..
This program has been modified by B. Shihada in the 07 of Nov 2000
The modification is done to fit my purposes of Data Encryption

Methods:
- Initialize the data variables.
- Initialize the used key, which has been created from the DH message Exchange.
- Select the decryption method.
- Do the data Decryption.

DhkeyAgreement2.java

This program executes the Diffie-Hellman key agreement protocol
between 2 parties: PLAN router & client system.
By default, preconfigured parameters (1024-bit prime modulus and base

generator used by SKIP) are used.

Note:
Copyright 1997, 1998, 1999, 2000 by Sun Microsystems, Inc.,
901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
All rights reserved.

PLAN Programs:

add.plan

```
fun add(code:string,pri:string) :unit = deliver(getImplicitPort(),addItem(code,pri))
```

remove.plan

```
fun remove(code:string) :unit = deliver(getImplicitPort(),removeItem(code))
```

getItemsData.plan

```
fun getItems() :unit = (deliver(getImplicitPort(),getDataList()))
```

getNewPrice.plan

```
fun newPrice (icode:string) :unit = (deliver(getImplicitPort(),getNewPrice(icode)))
```

getWinner.plan

```
fun getWinnerList() :unit = (deliver(getImplicitPort(),getWinner()))
```

sendBidData.plan

```
fun sendBid (cipher:blob,seal:blob) :unit =
                    (deliver(getImplicitPort(),updateData(cipher,seal)))
```

sendItemsData.plan

```
fun sendItems (item:'a list) :unit = (deliver(getImplicitPort(),setDataList(item)))
```

Appendix D—EXPERIMENTAL TABLES

Number of Iterations	SAVE-MM1 System	SAVE-MM2 System
1	571	651
2	531	530
3	530	521
4	531	541
5	652	531
6	531	521
7	531	541
8	521	531
9	530	531
10	531	530
11	530	531
12	530	530
13	540	521
14	531	531
15	521	531
16	531	531
17	530	531
18	531	540
19	521	521
20	531	520
21	641	530
22	540	531
23	530	530
24	521	531
25	521	531
26	531	530
27	531	521
28	531	541
29	531	531
30	531	531
31	531	520
32	521	541
33	531	531
34	530	540
35	521	520
36	531	520
37	531	531
38	621	529
39	531	523
40	541	522
41	521	545
42	521	530
43	530	528
44	531	526
45	531	531
46	530	533
47	531	530
48	530	521
49	551	527

50	531	530
51	521	530
52	520	529
53	540	526
54	531	528
55	540	530

Table D.1 Total Client Transaction Time (Delay).

Number of Iterations	SAVE-MM1 System	SAVE-MM2 System
1	10	10
2	20	10
3	10	10
4	10	50
5	10	10
6	10	10
7	10	20
8	10	10
9	10	10
10	10	20
11	10	20
12	10	10
13	10	10
14	10	10
15	10	10
16	10	10
17	10	10
18	10	10
19	10	20
20	10	10
21	20	10
22	10	10
23	10	10
24	10	10
25	20	10
26	10	10
27	10	10
28	10	10
29	10	10
30	10	10
31	10	10
32	10	10
33	10	10
34	10	10
35	12	11
36	10	10
37	20	10
38	10	10
39	10	50
40	10	10
41	10	10
42	10	20
43	10	10

44	10	10
45	10	20
46	10	20
47	10	10
48	10	10
49	10	10
50	10	10
51	10	10
52	10	10
53	10	10
54	10	20
55	10	10

Table D.2 Total Client Encryption Time.

shihada 10270 0.1 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10271 0.1 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10272 0.1 3.0 4016 1936 pts/2 S 18:32 0:00 pland -ip 5002 -hf hostfile c
shihada 10273 0.1 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10274 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10275 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10276 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10277 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10278 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10279 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10280 0.0 3.0 4016 1936 pts/2 S 18:32 0:00 pland -ip 5002 -hf hostfile c
shihada 10281 0.0 3.0 4016 1936 pts/2 S 18:32 0:00 pland -ip 5002 -hf hostfile c
shihada 10282 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10283 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10284 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10285 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10286 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10287 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10288 0.0 3.0 4016 1936 pts/2 S 18:32 0:00 pland -ip 5002 -hf hostfile c
shihada 10289 0.0 3.0 4016 1936 pts/2 S 18:32 0:00 pland -ip 5002 -hf hostfile c
shihada 10290 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10291 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10292 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10293 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10294 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10295 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10296 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10297 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10298 0.0 3.0 3972 1904 pts/2 S 18:32 0:00 pland -ip 5101 -hf hostfile b
shihada 10299 0.0 3.0 4008 1928 pts/2 S 18:32 0:00 pland -ip 5001 -hf hostfile a
shihada 10300 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d
shihada 10301 0.0 3.0 3980 1908 pts/2 S 18:32 0:00 pland -ip 5102 -hf hostfile d

shihada	10302	0.0	3.0	4008	1928	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10303	0.0	3.0	3972	1904	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10305	0.0	3.0	3980	1908	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10306	0.0	3.0	4008	1928	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10307	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10308	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10309	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10310	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10311	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10313	0.0	3.0	4016	1936	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10388	0.0	3.0	4016	1936	pts/2	S	18:34	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10389	0.0	3.0	4036	1940	pts/2	S	18:34	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10270	0.1	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10271	0.1	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10272	0.1	3.0	4036	1944	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10273	0.1	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10274	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10275	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10276	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10277	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10278	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10279	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10280	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10281	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10282	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10283	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10284	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10285	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10286	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10287	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10288	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10289	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland	-ip	5002	-hf	hostfile c
shihada	10290	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10291	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10292	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10293	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10294	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10295	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10296	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10297	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10298	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland	-ip	5101	-hf	hostfile b
shihada	10299	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a
shihada	10300	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10301	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland	-ip	5102	-hf	hostfile d
shihada	10302	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland	-ip	5001	-hf	hostfile a

shihada	10303	0.0	3.0	3992	1912	pts/2	S	18:32	0:00	pland -ip 5101 -hf hostfile b
shihada	10305	0.0	3.0	4000	1916	pts/2	S	18:32	0:00	pland -ip 5102 -hf hostfile d
shihada	10306	0.0	3.0	4036	1940	pts/2	S	18:32	0:00	pland -ip 5001 -hf hostfile a
shihada	10307	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10308	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10309	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10310	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10311	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10313	0.0	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10389	0.0	3.0	4036	1940	pts/2	S	18:34	0:00	pland -ip 5001 -hf hostfile a
shihada	10393	0.0	3.0	4036	1940	pts/2	S	18:34	0:00	pland -ip 5001 -hf hostfile a
shihada	10395	0.0	3.0	3992	1912	pts/2	S	18:34	0:00	pland -ip 5101 -hf hostfile b
shihada	10396	0.0	3.0	4036	1944	pts/2	S	18:34	0:00	pland -ip 5002 -hf hostfile c
shihada	10397	0.0	3.0	4000	1916	pts/2	S	18:34	0:00	pland -ip 5102 -hf hostfile d
shihada	10270	0.1	3.0	4036	1940	pts/2	S	18:32	0:00	pland -ip 5001 -hf hostfile a
shihada	10271	0.1	3.0	4016	1920	pts/2	S	18:32	0:00	pland -ip 5101 -hf hostfile b
shihada	10272	0.1	3.0	4036	1944	pts/2	S	18:32	0:00	pland -ip 5002 -hf hostfile c
shihada	10273	0.1	3.0	4016	1920	pts/2	S	18:32	0:00	pland -ip 5102 -hf hostfile d

Table D.3 %CPU & Memory Usage

REFERENCES

[1] Auction History
http://www.dickeranddicker.com/dicker/auction/history.cfm

[2] S. Munir, “Active Networks” - A Survey, 1997

[3] D. Tennenhouse and D. Wetherall, “Towards an Active Network Architecture,” *Networks and Systems Group*, MIT (1996)

[4] M. Hicks, J. Moore, D. Alexander, P. kakkar, C. Cunter, and S. Nettles, “The PLAN System for Building Active Networks,” CIS, *University of Pennsylvania*.

[5] Auction Types
http://www.agorics.com/~agorics/auctions/auction2.html

[6] P. Wurman and M. Wellam, “A Parameterization of the Auction Design Space,” CS North Carolina State University.

[7] Regular Auction Types. http://www.magtawaran.com/help/auctiontypes.cfm Feb. 2001.

[8] M. Kumar and S. Feldman. “Internet Auctions,” T.J. Watson Research Center.

[9] K. Psounis, “Active Networks: Applications, Security, Safety, and Architectures,” Stanford University. *IEEE Commu.* 1999

[10] S. Bhattacharjee, K. Calvert and E. Zegura, “An Architecture for Active Networking,” *Networking and Telecommunications Group*, MIT. IFIP 1996,

[11] M. Hicks, P. Kakkar, J. Moore, C. Gunter, and S. Nettles “PLAN: A Packet Language for Active Networks,”, *University of Pennsylvania. DARPA*.

[12] M. Hicks, P. Kakkar, J. Moore, C. Gunter and S. Nettles “Network Programming Using PLAN,” *University of Pennsylvania. DARPA*.

[13] R. Kawamura and R. Stadler, “Active Distributed Management for IP Networks,” *IEEE Trans. Commun.* 2000.

[14] U. Legedza, D. Wetherall, and J. Guttag, “Improving the Performance of Distributed Applications using Active Networks,” MIT 1998.

[15] M. Hicks, “PlanD: The PLAN Active Router,” v. 3.1. *DARPA*. 2000

[16] M. Hicks and P. Kakkar ,"The PLAN Tutorial," v. 3.2 . *DARPA*.

[17] M. Hicks, J. Moore, and P. Kakkar "PLAN Programmer's Guide," v. 3.2. *DARPA*. 2000

[18] M. Hicks, "PLAN Service Programmer's Guide," v. 3.2. *DARPA*. 2000

[19] T. Moore, "PLAN Grammar," v. 2.2. *DARPA*.

[20] J. Moore, M. Hicks, and S. Nettles "Chunks in PLAN: Language Support for Programs as Packets,"Dept. CIS, *University of Pennsylvania*. 1999.

[21] C. Labonte, and S. Srinivas, "New Mechanisms for Extending PLAN Functionality in Active Networks,", *International working Conf. On Active Networks* (IWAN2000), Tokyo, October 2000.

[22] M. Hicks and A. Keromytis "A Secure PLAN,", University of Pennsylvania. *DARPA*.

[23] M. Hicks, "PLAN Security Guide," v.3.2 *DARPA*.2000

[24] D. Alexander, W. Arbaugh, and A. Keromytis. "Safety and Security of Programmable Networks Infrastructures," *University of Pennsylvania* 1998

[25] D. Alexander, W. Arbaugh, A. Keromytis and J. smith. "Security in Active Networks," *University of Pennsylvania*. 1999

[26] A. Gunter, and T. jim, "Policy-Directed Certificate Retrieval (DRAFT)," *Univ. of Pennsylvania*, 1998

[27] T. Jim, "QCM Reference Manual," *University of Pennsylvania*, 1998.

[28] Cryptix 3.2 data encryption library works over java http://www.cryptix.org/products/cryptix31/index.html

[29] Java Program 2 for Triple-DES Encryption by J. Orlin Grabbe

http://www.aci.net/kalliste/javacrypt_index.

[30] Active Network Backbone (ABone)

http://www.isi.edu/abone/

[31] ebay.com at http://www.ebay.com

[32] auctionfrist.com at http://www.auctionfirst.com

[33] auction bid at http://www.aucbid.com