FAULT TOLERANT AND HIGHLY AVAILABLE ENTITLEMENT SERVER

A Thesis Presented to the Faculty of the Graduate School
University of Missouri - Columbia

In Partial Fulfillment of the Requirements for the Degree
Master of Science

By

HARCHARAN SINGH

Dr. Gordon K Springer, Thesis Supervisor

July 2011

The undersigned, appointed by the Dean of the Graduate School, have examined the project entitled

FAULT TOLERANT AND HIGHLY AVAILABLE ENTITLEMENT SERVER

Presented by

Harcharan Singh

A candidate for the degree of

Master of Science

And hereby certify that in their opinion it is worthy of acceptance.

_____

Dr. Gordon Springer

_____

Dr. William Harrison

_____

Dr. Harry Tyrer

ACKNOWLEDGEMENTS

I would like to thank the following people who, in their own individual way, helped me throughout my studies and completion of my project.

First and foremost, I would like to thank my advisor, Gordon K Springer. I thank him for his expert tutelage, patience, guidance, and confidence entrusted upon me to complete the project, without which this project would not have been successful.

Secondly I would like to thank to my committee members for taking time from their busy schedules and contribute valuable insights.

Lastly I would like to thank to my friends and my family for their effective work and support in the collaboration. My honors and achievements are dedicated to all of these people.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

FAULT TOLERANT AND HIGHLY AVAILABLE ENTITLEMENT SERVER

Harcharan Singh
Dr. Gordon K. Springer, Project Advisor

ABSTRACT

Web accessible resources containing research related data and information have been important for researchers for the past 15 years. The sharing of information through the web leverages the research and development process. At present, the web is one of the most popular, fast, and convenient mediums for sharing resources. Originally, all resources made available on the web were accessible to anyone with a web browser. However, this has been found to be unsatisfactory for a variety of reasons and situations. Thus, recent efforts have been made to provide protection of users and resources using the robust capabilities of the web.

The current project is based, in part, on the use of Shibboleth to provide restricted access to resources via the web. Shibboleth consists of the Service Provider and an Identity Provider to authenticate access to the resources. These services and the incorporation of a separate Entitlement Server provide fine-grained access to protected resources. This project incorporates multiple Entitlement Servers to provide a robust authorization environment that can continue to operate in the event of server or network failures in the trusted environment.

The design proposed in this project decentralizes the authorization process by running multiple entitlement server applications in the network. All servers form a

logical group, which is used for the authorization process. The project outlines a procedure of interaction between a service provider and the group of entitlement servers for performing the authorization of users. Multiple entitlement servers in the network help in achieving a fault tolerant and highly available authorization process. All entitlement servers in the logical group stay synchronized. The authorization process can proceed when at least one entitlement server is present in the logical group.

Each of the entitlement servers present in the group maintains enough information about the users to make detailed authorization decisions. An information synchronization methodology is utilized such that each of the entitlement servers has consistent data. The scalable architecture of the authorization process allows the addition of an additional entitlement server to the group on the fly. Moreover, the entitlement server can move in the network from one physical machine to another without affecting the authorization process. The design also considers the security risk factors so that any communication message between two entities is encrypted to avoid disclosure of the messages. Similarly, the active servers in the network mutually authenticate each other to avoid having an imposter server attempting to break into the authorization process.

Chapter 1
INTRODUCTION

With today's advanced computing power, the Internet is meant to be more than just passing the bits from one machine to another. A group of people from various universities believe that the Internet could be used as a tool to catalyze the speed of research and development. Several institutions came together in 1996 and formed a network, named Internet2, which is comprised of members willing to share their most current and updated knowledge and research.

Internet2 is a not-for-profit organization, designed to leverage the sharing of high end resources among the groups with common interest. Internet2 consists of more than 200 institutions, over 40 corporate members and over 30 research and education network and connector organizations[1]. The Great Plains Network[2] (GPN) is one of the connectors in Internet2. It is formed by more than 20 leading universities in seven states in the Midwest to create a robust networking infrastructure to support research and development.

It is possible (not obligatory though) for subsets of institutions to form a Virtual Organization (VO). The members of the virtual organization consist of two or more organizations. The goal of a virtual organization is to influence the research by sharing research resources. A virtual organization provides its member institutions and people

1. http://www.internet2.edu
2. http://www.greatplains.net

with a secure, robust and inter-operable collaborative research environment.

In a virtual organization, where information is shared across institutional boundaries, the authentication and authorization process becomes imperative. Authentication is the process of determining the true identity of an entity. Authorization is the process of specifying the access privileges the entity has to resources. An entity is said to be “authorized” to use a resource, provided that the entity has appropriate access privileges to the resource. During the communication between two entities, authentication and authorization makes sure that the entities are what they claim to be and they have the right privileges.

Identity Management is a process that deals with identifying an individual in a system and ensures no two individuals have the same identity. For example, each student at the University of Missouri has a unique student number and an email identifier that can also serve as a login identifier to various systems at the University. All the information related to a student such as classes enrolled in, account information and other associated data is correlated with the student number provided by the university. The Identity Management system refers to a set of technologies that deal with the creation of identities, and the administration and termination of identities in the system. Each institution in the Great Plains Network maintains their own Identity Management System in order to identify the members of their respective institutions (faculty, staff, employees and students). The institution that provides a user with user credentials is known as the home institution of that user. The users wanting access to a

resource must present their identity data for validation and verification by a system prior to being granted access to the system. Examples of accessing a system include things like a student registering for a class or a professor accessing library resources online.

Individuals have attributes associated with their personal data such as a name, an email address, or how the user is affiliated with an organization (e.g. a student, faculty or staff member). As such Identity Providers keep track of individuals and the attributes associated with them. Entitlements connect authenticated users with the resources to be accessed to reliably authorize the resources usage. Entitlements are associated with resources to be shared and identify users who are eligible to use the resource. The Entitlement Servers provide a way for limiting access to resources by individuals. The resources and access attributes are beyond the scope of what identity provides and expected to be responsible.

A Service Provider (SP) manages a shared resource and prevents unauthorized access to the resource. Whenever a user needs to access some resource, the user sends the request to the service provider of the resource. There can be more than one resource protected by a service provider. A resource that is protected by the service provider is known as a restricted resource. When a user sends a request to the service provider to gain access to some restricted resource, the service provider performs an authorization check before allowing the user to access the resource.

Shibboleth [8] is a standards-based, open source software package. A university or other identity providers can use it for its own single sign-on purposes. But, Shibboleth utilizes its real power when it is used by large federations. Shibboleth allows an identity provider to release just the right information needed by a resource to know that the user is allowed to have access [8].

The procedure for making a request to gain access to a resource is as described. A user sends a request to a service provider of the resource requesting to gain access. If the user is not already authenticated, the Service Provider redirects the user to the user's home institution in order to become authenticated. After the authentication process completes, the service provider of the resource determines if the user is authorized. The service provider uses the entitlement server during the authorization process to check if the user has any needed entitlements or not. Depending on the access control policies of the resource, the user is either granted or denied access to the resource.

At present there is only one entitlement server in the network that serves authorization requests. One drawback of using a single entitlement server is that each service provider must be aware of the location where the entitlement server is available. Moreover, multiple entitlement servers can make the authorization process less prone to errors and available for providing service the vast majority of the time. Each entitlement server operates independently of the others. However, to maintain

consistency in decisions of authorization, all servers must use consistent data to perform entitlement verifications.

This project focuses on creating a fault tolerant entitlement server. The proposed design defines how the service provider communicates to the entitlement server and how the synchronization of the data between the entitlement servers takes place. This design is predicated on using the multicast networking protocol [7] to provide a means for entitlement servers to communicate among themselves without having to know where they exist in the network or even know beforehand how many there are. The multicast protocol offers an ability to form a logical group of the software applications, which intend to receive identical data. A software application can join the logical group using the multicast protocol. The packetized chunk of data sent to the logical group is delivered efficiently to all the software applications in the group [7]. In this project, all the entitlement servers join the multicast group to communicate among themselves and sometimes with the service provider. All the communication including request messages are encrypted to ensure that message exchange takes place between the legitimate parties only.

A service provider performs the entitlement checks of the users on a point-to-point connection with an entitlement server selected from the multicast group. The service provider communicates to the multicast group for selecting an entitlement server. Each entitlement server in the multicast group responds back with enough information that can be used for a point-to-point connection. The analogy is similar to

saying “hello” to a group of friends and getting a response from each friend. The service provider stores the connection information response from each of the entitlement servers for future reference.

A service provider selects any one of the entitlement servers and establishes a point-to-point connection. The service provider uses the point-to-point connection to create a timed session with the entitlement server. Timed sessions are used to avoid third parties from breaking into the conversation as described in the thesis by Ciordas [9]. In order to authorize users, the service provider utilizes this timed session.

If a service provider does not make any requests to the entitlement server for a predefined period of time, the session between the service provider and entitlement server expires. The service provider must renew an expired session with the same entitlement server in order to submit further authorization queries. In case the entitlement server, with whom the session was created previously, is not reachable, the service provider tries another entitlement server from the list of entitlement servers and tries to create a session with one of them. When the service provider does not have any entitlement servers in its list to create a session, the service provider contacts the multicast group again to collect data about the entitlement servers currently in the group. The service provider tries any one entitlement server again and creates a session with it. In case none of the entitlement servers is available in the multicast group due to software or hardware failure, the authorization process fails and access requested is denied.

Each entitlement server maintains data to perform the entitlement checks. To avoid false negative results of the entitlement checks, the design includes a synchronization process of the data maintained by the entitlement servers. The synchronization process verifies the integrity of the data at all the entitlement servers at regular intervals. The synchronization process executes at regular intervals, and synchronizes the databases using multicast communication. The synchronization messages that are passed among the entitlement servers in the group are also encrypted. A detailed explanation of the synchronization is provided in Chapter 5. All the messages that are exchanged between the entitlement servers are encrypted using the public /private key pairs. Then encryption ensures that only the intended applications (entitlement servers) can understand the messages that are passed in the network.

The next chapter describes the background that is required to understand the function of the authorization process that is performed using the redundant entitlement server.

Chapter 2
BACKGROUND

Chapter 2 includes a description of the various software components of the Shibboleth-enabled system that take part in the authentication and authorization of the users. This chapter presents a brief overview of the system that is required to understand the design and function of the project.

2.1 Shibboleth Definition

The web provides a user interface to access remote resources. All resources and materials available on the web were initially available to anyone with access to a web browser. This open access is no longer acceptable for a variety of reasons and situations. Part of these changes stem from efforts to protect users and resources using the robust capabilities of the web. The resources needing special privileges to be accessed are referred to as restricted resources.

There are various methodologies employed to restrict users from accessing resources. Providing a username-password pair is one of the most common methods. The users must enter their access credentials to access a restricted resource. The web user may have a separate username-password pair to access different resources. The Shibboleth mechanism provides an infrastructure to facilitate users having a single username-password pair to access multiple resources.

Shibboleth is an open source software package, and its goal is to provide a single sign-on capability using distributed federated identity standards. Users authenticate themselves with their organizational credentials, and the organization passes minimal identity information necessary to the restricted resource manager in order to enable an authorization decision. Shibboleth leverages the organization's identity and access management system, so that the individual's relationship with an institution determines access rights to services that are hosted both on and off campus [8]. The Shibboleth system includes two major software components: the Shibboleth Identity Provider (IdP) and the Shibboleth Service Provider (SP). Although, these two components are deployed separately both work together to provide secure access to web-based resources.

Shibboleth is based on the Security Assertion Markup Language (SAML) standard published by the Organization for the Advancement of Structured Information Standards (OASIS) [5] [2]. OASIS is a global consortium that produces the standards used for web services, e-commerce and the like [3].

2.2 Why Use Shibboleth-enabled Systems

Universities, corporate organizations, government agencies and the like are increasingly contributing to research and development, and want to share their resources with others over the web. For security purposes in the past, there were methods such as providing username- password pairs, digital certificates and Kerberos authentication that were used to make the access control decisions. As the number of resources over the web grows exponentially, the users need to keep track of multiple

credentials or certificates to gain access to multiple resources. The Shibboleth system addresses the issue of how vital it is for the users to keep track of multiple credentials or certificates to gain access to multiple resources. Therefore, Shibboleth provides a single sign-on methodology, in which a user needs to login using the credential provided by the user's home institution, and can use it to have access to resources within or across institutional boundaries.

2.3 Functioning of the Fine Grained-Authorization System

The Shibboleth enabled environment that is currently used at the University of Missouri is described in detail in this section. This implementation supports a fine grained authorization that is based on the entitlements of the user. Shibboleth makes use of two main components that are an Identity Provider and a Service Provider. Another important component taking part in authorization is an Entitlement Server (ES). The ES is used by the service provider to make authorization decision based on data that is not present at an Identity provider.

The Shibboleth Service Provider or simply, Service Provider (SP), is used to protect resources from un-authorized access in a Shibboleth environment. When a user chooses to utilize a service provider-protected resource, the user sends the request to the service provider for that resource. Before granting access to a resource, the service provider makes sure that the user who has requested the resource has the required privileges. Message exchange takes place between the service provider and the identity provider. The service provider also contacts an entitlement server to check entitlements

needed by the user. The service provider uses the entitlement server to make access decisions based on the data that may be independent of the institution. This data is maintained only by the entitlement server. Based on the data received from the identity provider and the entitlement server, the service provider makes a fine grained decision, whether or not the user should be granted access to the resource.

The Shibboleth Identity Provider or simply, Identity Provider (IdP), is integrated with the Identity Management System of the user's home institution. The Shibboleth code uses the Identity provider to authenticate users. Upon request, the Shibboleth IdP sends asserted attributes related to the users. The attributes that are released to the service provider can be controlled by the user, thus ensuring the user's level of privacy. A detailed description of the attributes and the user's privacy is provided in the thesis by Ciordas [9]. The institutions with users wishing to access a restricted service must run the Shibboleth IdP.

A service provider may have rules of access that cannot be made by an IdP. For example, name of the Virtual Organization (VO) of users. This type of information is beyond users' organization boundaries. Hence attributes are needed that an IdP does not manage, but are essential for a service provider to decide who to provide service to. The Entitlement Server (ES) is used to manage these attributes. An entitlement server helps a service provider to perform fine grained authorization. The service provider utilizes a client application known as an "entitlement client" to interact with the

entitlement server. A secure channel is created by the entitlement client between the service provider and the entitlement server to ensure information security.

The entitlement server uses a public/private key encryption [6] for mutual authentication with a service provider. The service provider encrypts the messages using its private key and entitlement server's public key which are provided during the deployment. Upon receiving, the entitlement server decrypts the messages using its private key and service provider's public key. If the decryption is successful that means the mutual handshake is successful. Figure 2.1 shows the pictorial representation of the mutual handshake. A mutual handshake between the service provider and the entitlement server takes place to confirm that the peer is what it claims to be. After the handshake is complete a symmetric encrypted key is shared in between the service provider and the entitlement server creating secure channel. The goal of the secure channel between the service provider and the entitlement server is to hide the communication messages from all other users. On the secure channel the service provider sends a query to the entitlement server to check if a user has a particular entitlement or not. The entitlement server looks up the information stored with it and sends "yes" or "no" back to the service provider.

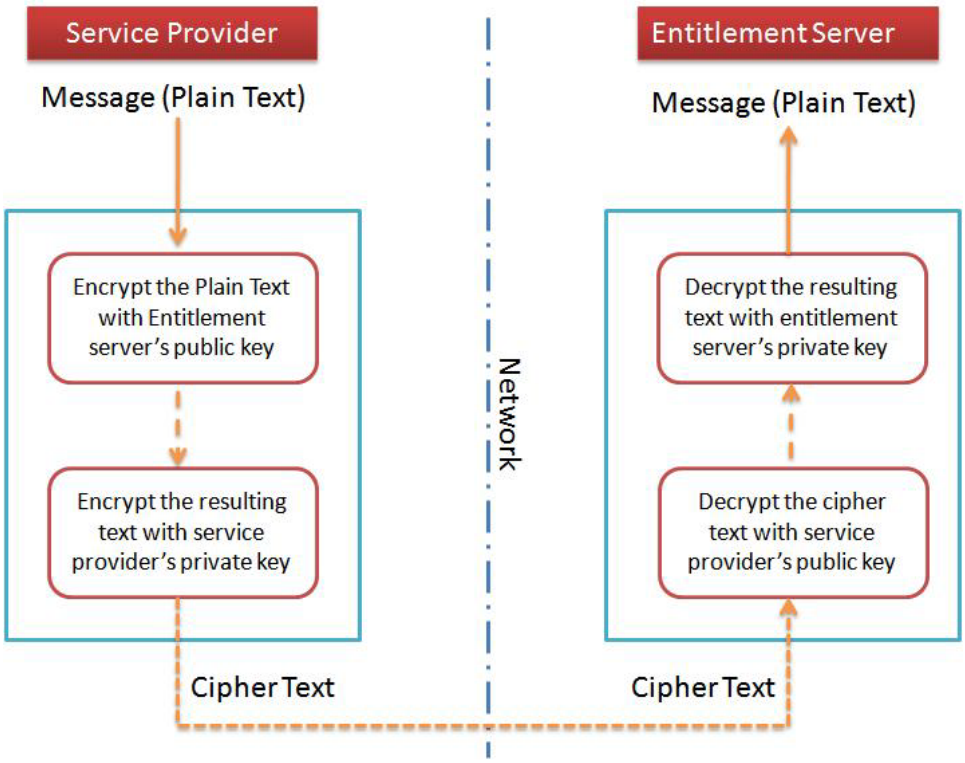


Figure 2.1: Public/Private Key Encryption

The secure channel created between a service provider and an entitlement server is timed. If there is no communication for a predefined time period, the channel is marked invalid for communication. In this case, the service provider has to re-create a secure channel for carrying out further communication with the entitlement server.

A pictorial representation of the Shibboleth authentication system currently deployed at the University of Missouri is shown in Figure 2.2.

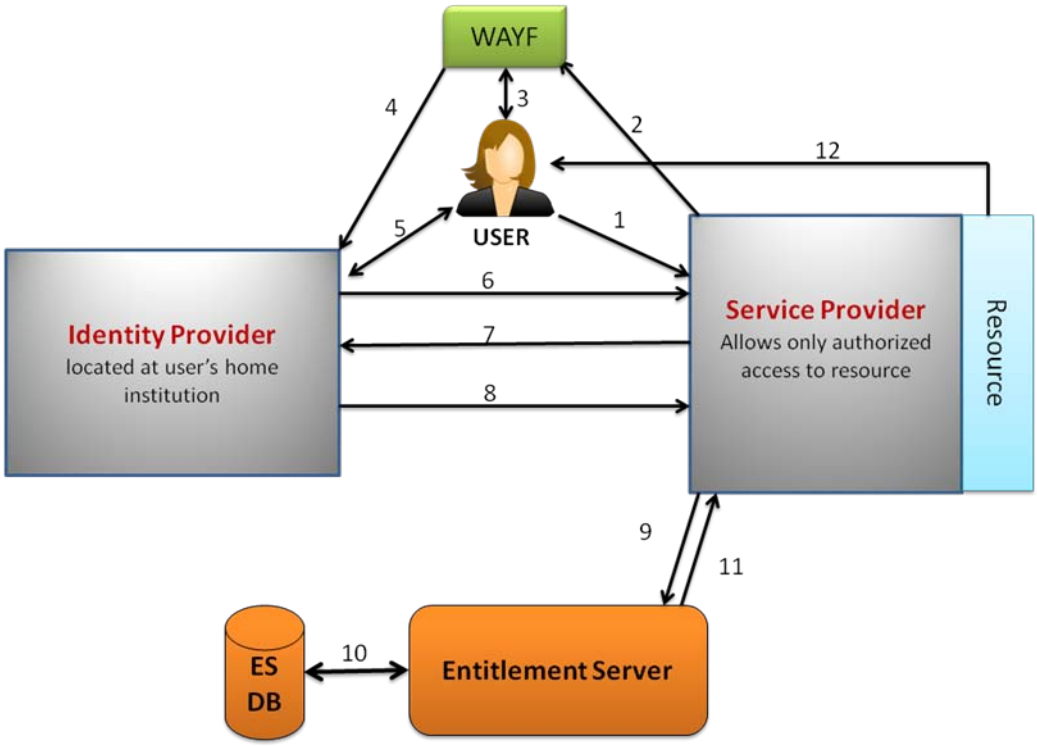


Figure 2.2: Shibboleth System with Single Entitlement Server

A description of the Shibboleth sign-on process is majorly adopted from the paper by Morgan et al [5], and includes the following steps: the players include the user, who wants to use a protected web resource; the resource provider web site, that has installed the Shibboleth SP software; and the user's home institution, which has installed the Shibboleth Identity Provider software.

The user navigates to the web resource using a web browser. The resource site is protected, and hence requires information about the user in order to decide whether or not access is permitted. The Shibboleth SP software redirects the browser to a "navigation" page (called a WAYF), which presents the user with a list of the institutions whose users may access the resource. The user selects his/her home institution, and the

browser is sent to the home institution's web site running the Shibboleth Identity Provider software. This site uses a web sign-on method chosen by the user's home institution. The user now sees the familiar login web page of its home institution. The user enters his/her username and password, and selects the login button. If successful the Shibboleth Identity Provider software sends the browser back to the original resource site and includes in the message, some security information proving that the user has signed in. The Shibboleth SP software on the resource site validates the security information and requests additional information (attributes) about the user by making a request to the home institution's Identity Provider service. The SP receives the user's attributes from the home organization's Identity Provider.

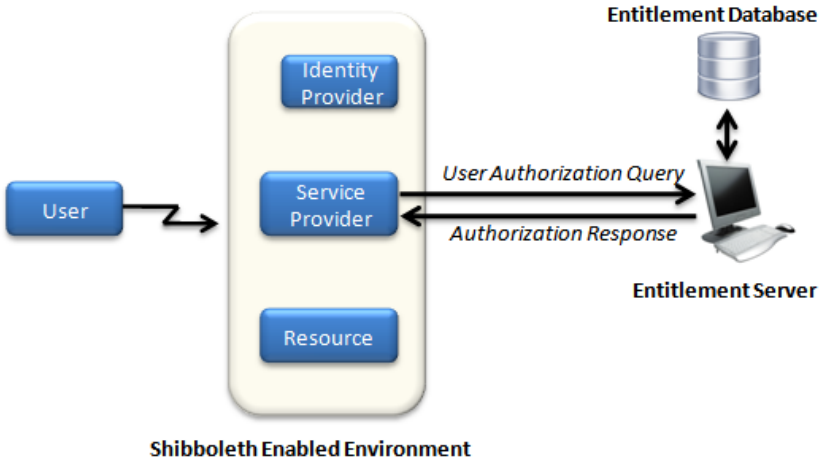
There are attributes that are tied to resources rather than users. These attributes are not in an IdP so cannot be verified by an IdP. Hence after receiving the attributes from the IdP the SP may request an entitlement server to check if the user has additional attributes (entitlements) or not. Upon receiving a yes or no response from the entitlement server, the SP checks its access policy to decide whether the user can access the resource or not. The detailed description of the process and the format of message passing between entities can be found in the thesis by Ciordas [9].

Chapter 3 discusses the overall design to achieve fault tolerance and a highly available authorization process using multiple entitlement servers.

Chapter 3
FAULT TOLERANCE AND SYNCHRONIZATION PROCESS DESIGN

This chapter describes the design of the authorization system. In the Shibboleth-enabled environment at the University of Missouri, just one entitlement server is involved in the authorization of the users (as shown in Figure 3.1). The purpose of the proposed design is to achieve fault tolerance, and a high availability of entitlement authorization services provided by an entitlement server. The service provider uses the entitlement client application to make the entitlement requests.

Providing fault tolerance comes from having multiple instances of the entitlement server operating independently. In the event that one instance of the entitlement server fails, due to hardware or software failure, other instances can take over for the failed server during the authorization process.



Figur 3.1: Shibboleth Environment using a Single Entitlement Server

To achieve a distributed authorization mechanism, the multicast networking protocol [7] is utilized. The multicast networking protocol enables an application to send identical data to multiple receivers with one message. The strategy used in multicast minimizes the duplication of data during the transmission from the source to multiple destinations. A detailed explanation of the multicast networking protocol's working is described in Appendix A.

Before the entitlement server starts authorizing users, the entitlement server is provided with the public key of the service provider(s) to decrypt the messages arriving from the service provider. The entitlement server becomes a part of a predefined multicast group. The multicast group is used to provide a medium for entitlement servers to communicate among themselves. Using multicast, the entitlement servers do not need to know where or how many other entitlement servers are running on the network. However, they can communicate simply by sending and receiving messages on the multicast group communication path. The layout of the entities is shown in Figure 3.2.

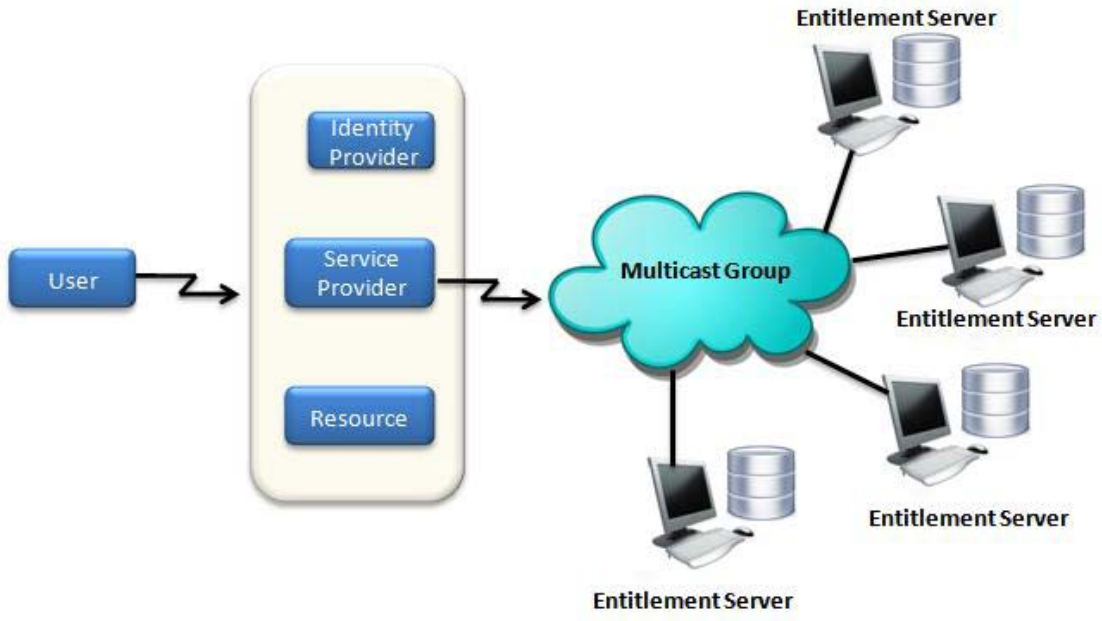


Figure 3.2: Shibboleth Environment Using Multiple Entitlement Servers

For authorization, a service provider first queries the multicast group to find out what entitlement servers are present in the group. The service provider stores the information (IP address and port number at which each entitlement server is listening) about the servers present in the group. The service provider then can choose any one of the entitlement servers from the multicast group to handle a request. The service provider mutually authenticates itself with the chosen entitlement server using the public and private key for that server. After a successful mutual authentication, a key is shared between the service provider and the entitlement server. This key is known as a symmetric key. The symmetric key is named so because both the sender and the receiver of the message have the same key. The same key is used for encryption at the sender's end and for decryption at the receiver's end. The symmetric key results in the establishment of a secure point-to-point channel between the service provider and the

entitlement server (Figure 3.3). A service provider uses this secure channel to make authorization queries. An example of a query made by a service provider to an entitlement server is “Does user A have X entitlement?” If the service provider is inactive for certain predefined time period (3 minutes), the symmetric key expires and no longer valid to make requests. The service provider again needs to mutually authenticate with the server using public/private key pairs and share a new symmetric key.

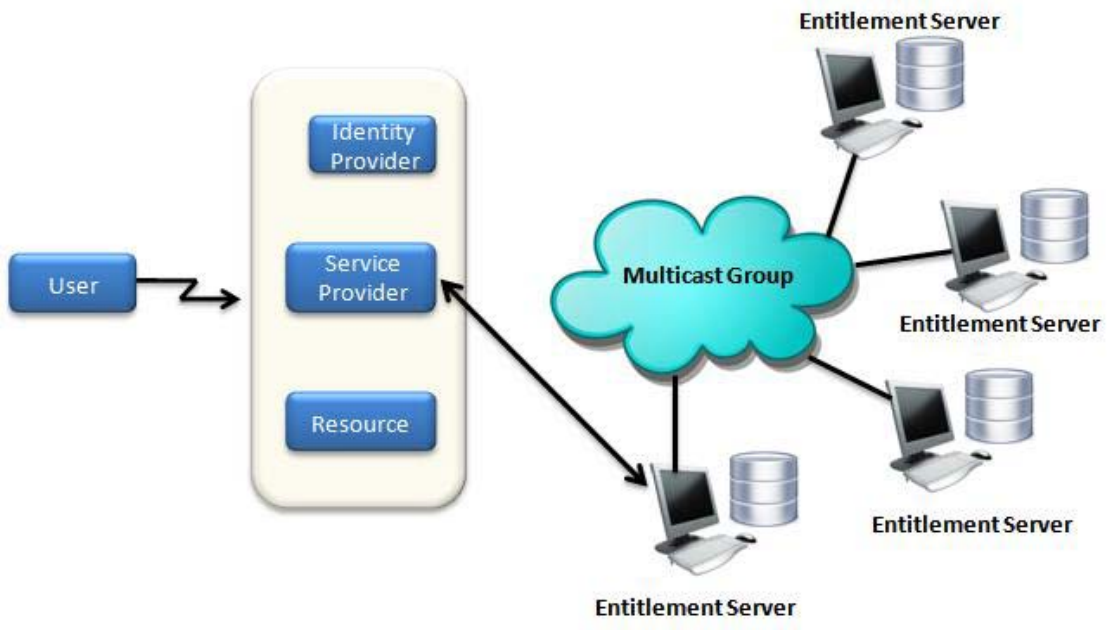


Figure 3.3: Secure Point-to-Point Channel with Entitlement Server

As soon as an authorization query is received from any service provider, the entitlement server first validates the session with that service provider. A session is the time period for which the shared symmetric key remains valid for encryption and

decryption. If a valid session exists, the entitlement server processes the query. The entitlement server searches for the requested user/entitlement being present in its entitlement database and sends the query response back to the service provider. The response is “yes” or “no”, based on whether the user has the requested entitlement or not.

In the case that an entitlement server, which has created a secure communication channel with the service provider, becomes unreachable, the service provider chooses any other entitlement server from the list of available servers, and sets up a secure channel with the newly chosen entitlement server (Figure 3.4). At this point the preceding authorization queries are sent to the new secure channel that has been created.

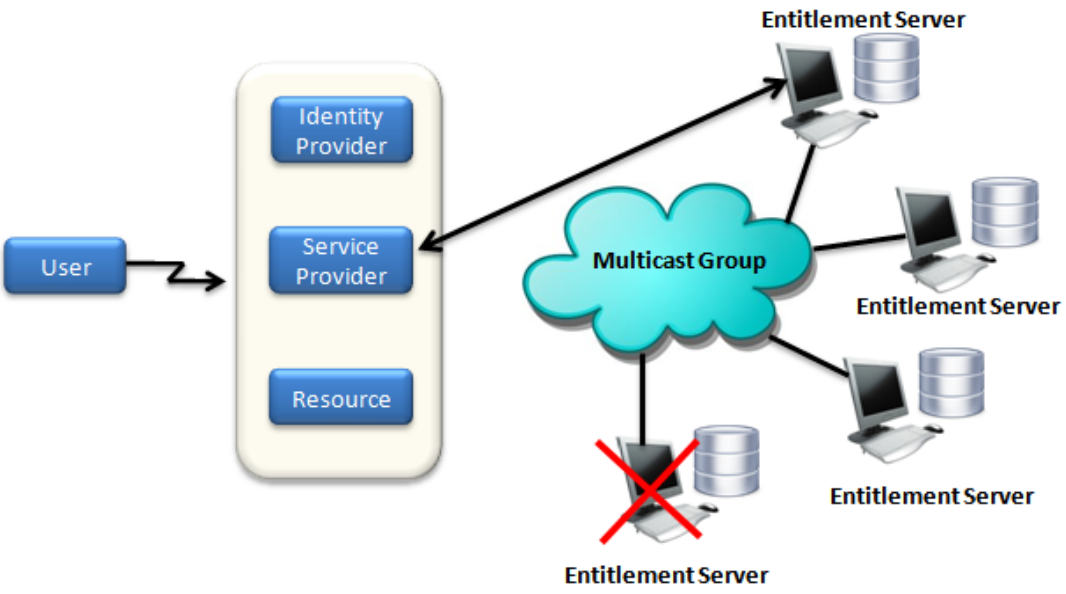


Figure 3.4: Secure Point-to-Point Channel with Different Entitlement Server

If no entitlement server responds to the request for a secure channel, the service provider queries the multicast group again to find any available entitlement servers at this time. The service provider stores the information of the available entitlement servers in the multicast group, proceeding as before to contact one of them. In case if no entitlement server is found in the multicast group, the user is not granted access to the resource and an appropriate error message is displayed on the web page. In the background, the message “NO_ES_SERVER_FOUND” is logged in the logging file. This makes easy to troubleshoot the issue.

An administrator of the entitlement database uses an entitlement client application to communicate with an entitlement server in order to modify the entitlements of the users. The modifications in the entitlement database are done point-to-point. The other entitlement servers in the network are not aware of the modifications made to the entitlement database, except the entitlement server that has the point-to-point communication connection with the entitlement client application. The entitlement server on which the database changes take place is responsible for informing the other entitlement servers in the multicast group about the entitlement changes. In order to notify the entitlement servers, of the changes in entitlements, the entitlement server forwards the entitlement change transaction to the multicast group. Using this methodology, all the entitlement servers in the multicast group become aware of any database changes on any entitlement server in the group. This procedure helps to maintain data consistency, and is encrypted to protect the data being passed in the transaction.

The entitlement modification transaction is forwarded to all entitlement servers in the multicast group using the multicast protocol. The multicast protocol employs the User Datagram Protocol (UDP) to transmit data. UDP does not guarantee the delivery of packets sent during the communication [7]. Therefore, when the entitlement server informs other entitlement servers in the multicast group about the database changes, there is a possibility of packet loss in the network that would prevent the message from reaching its destination. To ensure data consistency, the design of the fault tolerant system includes a database synchronization process.

Each entitlement server maintains a log of each modification done to its database. The log contains the complete transaction, and the time at which the transaction is applied to the entitlement database. These logs are the snapshot of the entitlement database. The logs are stored in the memory rather than on the disk to avoid extra overhead of reading from and writing data to the disk. As the memory is a limited resource, the design ensures that the logs are not kept in the memory forever, as accumulation of large number of logs can exhaust memory. Shortage of memory results in failure of the entitlement server. The logs are deleted from the memory after database synchronization is done using them.

The logs maintained with an Entitlement Server acts as a checkpoint of the database. The synchronization process does not synchronize the complete database each time but it only synchronizes the logs, if needed. The checkpoint in the database avoids bulky and unnecessary information transfer across the network. For example, at

a particular point of time an entitlement database, let's say, has 2000 entries. If the synchronization process checks the inconsistency based on the database, instead of transaction logs, then even one entry missing in database will cause at-least 2000 entries transferred across the network. Hence, the synchronization is done based on the transaction logs, but not on complete database.

The transaction logs contain the most recent 50 updates or all the updates done in past one hour, whichever result in larger number of logs. The synchronization process is run every fifteen minutes to synchronize the logs. The logs of an Entitlement Server are compared with the logs of all the other Entitlement Servers and a common list of logs is prepared without duplications. The process of synchronization based on the logs reduces the traffic in the network during synchronization.

The communication between the entitlement servers during the entitlement database synchronization is encrypted. The encryption is done using public/private key pair encryption technique [6]. In order to encrypt the synchronization messages, each entitlement server in the group is provided with two public/private key pairs. One pair of keys is used when an entitlement server acts as sender of multicast message and the other key is used when the same entitlement server act as receiver. These pairs of public/private key are generated and provided by the administrator. These key pairs are separate and independent from the public/private key pair shared among the entitlement server, and the service provider. During the synchronization process the sender encrypts the message using its private key and the receiver's public key. At the

receiving end the receiver decrypts the message using its private key and sender's public key. At the transmitting end, the encryption of data by sender's private key ensures that the actual sender has sent the data, as no other entity outside the multicast group have the private key of the sender. Encryption using public key of the receiver makes sure that only intended receivers can decrypt the data.

The next chapter explains in detail the modifications made to the entitlement client application to support redundant function of the entitlement server.

Chapter 4
ENTITLEMENT CLIENT APPLICATION

This chapter explains the entitlement client application's design, and the changes that have been made to the existing entitlement client application to incorporate the redundant entitlement servers in the network.

The entitlement client is an application used as an interface to communicate with the entitlement server. There are three types of operations that can be performed by the entitlement client application: SP_SETUP, SP_LOOKUP and SP_USE. Whenever a service provider wants to perform an entitlement check of a user, the service provider invokes the entitlement client application. An example of how the entitlement client application is invoked is shown in Figure 4.1. The entitlement client application validates the data provided during invocation, guaranteeing that data sent to an entitlement server is in the correct format. The three operations that can be performed by the entitlement client application are explained in detail in the following subsections.

For SP_SETUP	Invoke_client(**10** , **osprey.rnet.missouri.edu**@**missouri.edu**, **greatplains.net**)
	→ 10 designates that client wants to do the SP_SETUP → osprey.rnet.missouri.edu is the service provider's username → missouri.edu is the service provider's institution → greatplains.net is the service provider's virtual organization
For SP_LOOKUP	Invoke_client(**20**, **osprey.rnet.missouri.edu@missouri.edu**, **greatplains.net**, **springer@missouri.edu**, **greatplains.net**, **urn:mace:greatplains.net:repository**)
	→ 20 indicates that client wants to perform SP_LOOKUP → osprey.rnet.missouri.edu is the service provider's username → missouri.edu is the service provider's institution → greatplains.net is the service provider's virtual organization → Springer is the user whose entitlement is to be checked → missouri.edu is the institution of the user whose entitlement is to be checked → greatplains.net is the virtual organization of the user whose entitlement is to be checked → urn:mace:greatplains.net:repository is the entitlement to be checked if the user possesses

Figure 4.1: Example of Arguments Passed During the Invocation of the Client Application for SP_SETUP and SP_LOOKUP on osprey Server

4.1 SP_SETUP Operation

The SP_SETUP operation is executed by the entitlement client application on behalf of a service provider to obtain the connection information, such as the IP address and port number of all the entitlement servers available in the network. This operation uses the multicast protocol to query the group of servers. The entitlement client application uses the GNU Database Management (GDBM) to store the information (IP

address and port) about the entitlement servers available in the network [9]. GDBM database is an open source, flat file database system. GDBM uses a hashing technique for the fast searching of the keys. The data stored is in key/value pairs, with the schema of the database shown below in Figure 4.2. After determining all the entitlement servers in the network the entitlement client (on behalf of a service provider) selects any one of the entitlement servers and mutually authenticates with the service provider using the public/private key pair. After successful mutual authentication, a symmetric key (32 bytes) is shared establishing a secure channel between the entitlement client and the entitlement server.

Entitlement Server connection Information	Symmetric Key
128.206.116.66 9832	NOT_A_SYMM_KEY
128.206.116.91 3498	NOT_A_SYMM_KEY
128.206.116.93 2394	NOT_A_SYMM_KEY
128.206.116.112 8739	NOT_A_SYMM_KEY

Figure 4.2: Example DB Entries

There are two fields, as shown in Figure 4.2. The first field contains the entitlement server's connection information (IP address and port number). The server's connection information is the IP address and port number of the entitlement server where a request may be sent to make entitlement checks. The second field is the symmetric key which is shared with the entitlement server. Initially, the entitlement client stores a default string (NOT_A_SYMM_KEY) in the symmetric key field for all of the entitlement servers indicating that the symmetric key is not yet shared with any of

the entitlement servers. After the entitlement client application shares the symmetric key with a particular entitlement server in the database, the value of the shared symmetric key overwrites the default string for that particular entitlement server in the database. The flow chart in Figure 4.3 shows the working of the SP_SETUP operation.

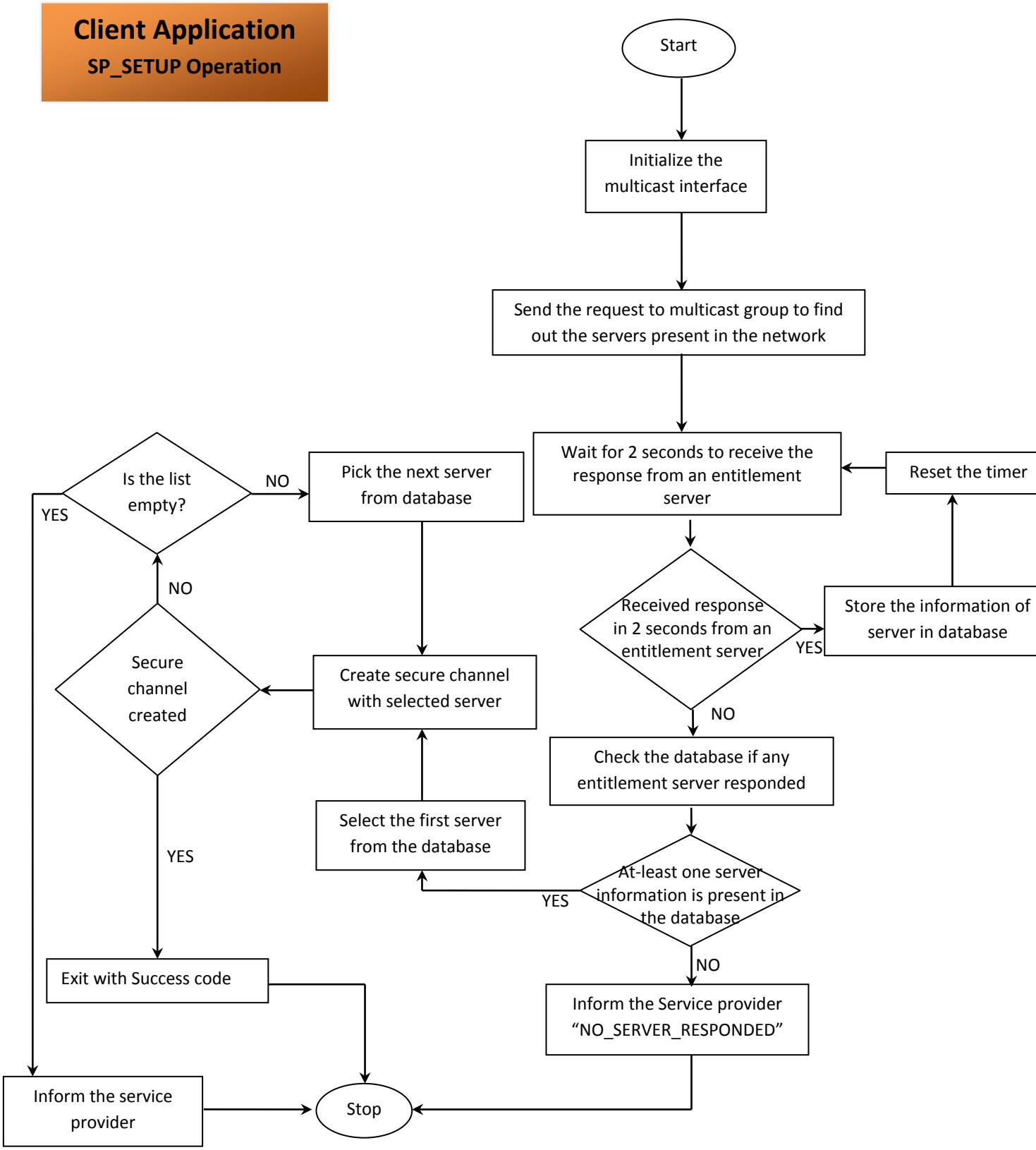


Figure 4.3: SP_SETUP Operation

Whenever a service provider invokes an entitlement client application with the operation type SP_SETUP, two functions are performed by the entitlement client application. First, the entitlement client application searches the available entitlement servers in the network, storing the information of all the entitlement servers present in the network in the database. Second, the entitlement client chooses any one entitlement server from the database and creates a secure channel for communication.

To look up the entitlement servers that are present in the network, the entitlement client initializes its multicast socket interface. After a successful initialization, the entitlement client sends the "CONNECTION_REQUEST" message to the multicast group. As soon as the connection request is received by the entitlement servers, the connection details (IP address and port) are sent back to the entitlement client application by the servers. When an entitlement client receives responses from the entitlement servers, the entitlement client application stores the connection information in its GDBM database. In order to make sure that all the entitlement servers have responded, the entitlement client application waits for an additional 2 seconds after the last response received from the entitlement servers. If none of the entitlement servers respond within the time frame, it is assumed that there are no more entitlement servers in the network. If no entitlement server is present in the multicast group, the authorization of the user fails and the user is denied access to the requested service. The user is presented with a web page indicating this failure.

If at least one of the entitlement servers in the network responds, the entitlement client application chooses one entitlement server. The client application

mutually authenticates the Service Provider and the chosen entitlement server using public and private keys [6]. Upon a successful mutual authentication, a secure channel (also known as a session) is created between the entitlement client (running on the Service Provider system) and the entitlement server. Creating a secure channel with an entitlement server completes the SP_SETUP operation. All requests for an entitlement check are performed on this secure channel using the SP_LOOKUP operation. The SP_LOOKUP process is described in the following section.

4.2 SP_LOOKUP Operation

The SP_LOOKUP operation is used to check if a user has a particular entitlement. Before granting access to a resource, the service provider checks if the user has the required entitlements to use the service. The Service Provider uses the entitlement client to perform an entitlement lookup of the user. The entitlement client selects a previously created secure channel created by the SP_SETUP operation. The request for the entitlement lookup is then sent over this secure channel. The response from the entitlement server (“yes” or “no”) is received by the entitlement client, and the service provider is informed accordingly. Based on the response from the entitlement server, the service provider makes the authorization decision.

The secure channel created between an entitlement client application and an entitlement server is timed. Before making the entitlement check request, the validity of the session between the entitlement client application and the entitlement server is investigated. If the entitlement client application discovers that the session created

previously has timed-out or expired, the entitlement client application renews the session and requests the entitlement check using the renewed session.

If an error occurs communicating with an entitlement server, the entitlement client application tries to create a secure channel with one of the other entitlement servers whose information is present in the database. All the unprocessed entitlement check requests are sent to this newly created connection. In the case that the entitlement client application is not able to create a session with any of the entitlement servers from the database, the service provider is informed of the circumstances and is asked to do SP_SETUP again. The entitlement request is sent again after a successful completion of SP_SETUP.

Using the multicast protocol facilitates the ability to relocate the entitlement servers in the network. In the event that all entitlement servers are moved from one machine to another, their connection information changes accordingly. If entitlement lookups fail, it may be because the entitlement client application has out-of-date information about the servers, and an SP_SETUP operation is performed to get the new connection information about the entitlement servers. Figure 4.4 shows the SP_LOOKUP operation flow chart performed by the entitlement client.

Client Application

SP_LOOKUP Operation

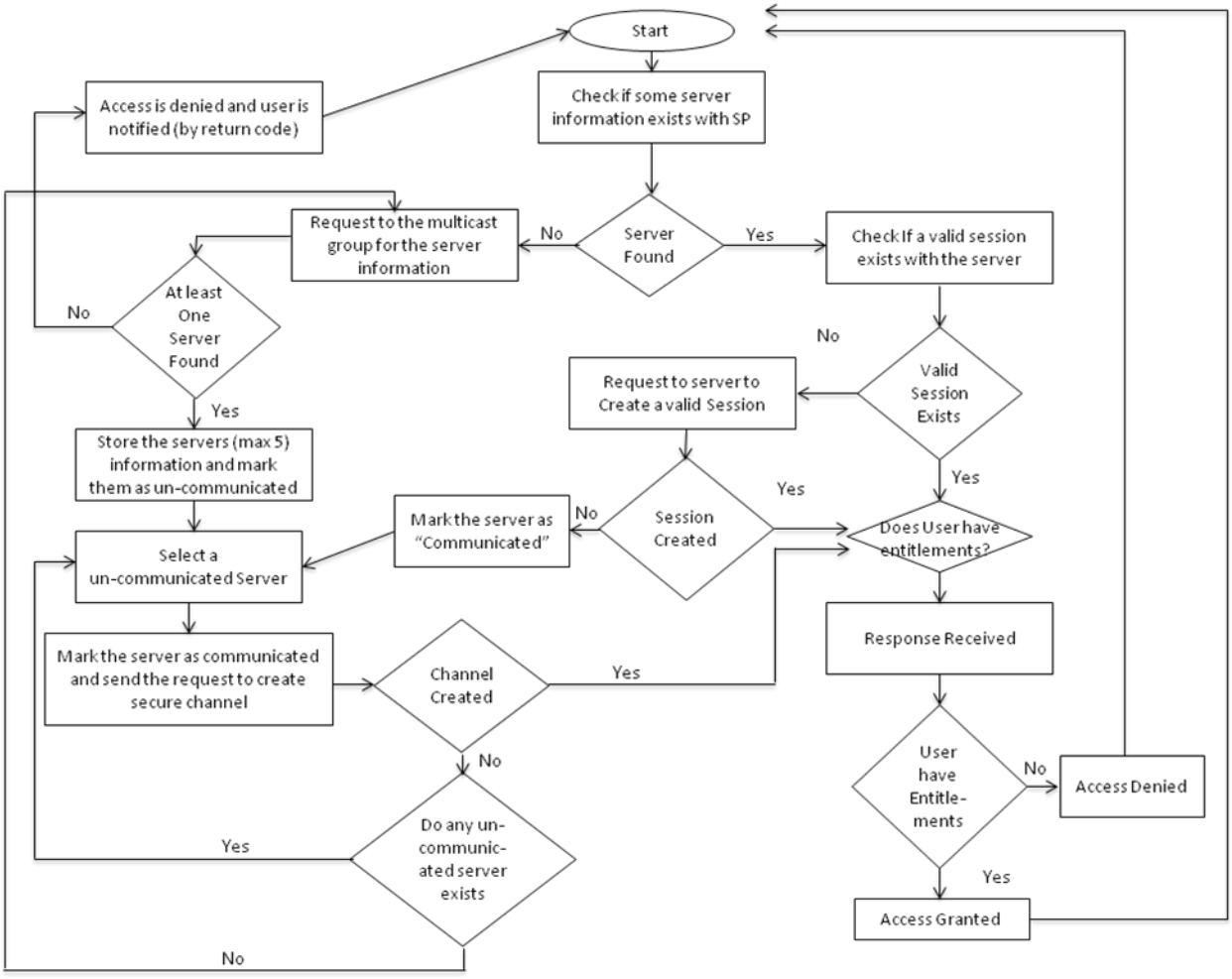


Figure 4.4: SP_LOOKUP Operation

4.3 SP_USE Operation

The administrator users execute the SP_USE operation to perform administrative tasks (add, delete or update) on the entitlement database. The administrator users are first authorized, because the administrator must have “root” or “admin” entitlement authority to make changes in the entitlement database. The authorization is done using the SP_LOOKUP operation, as explained in Section 4.2. The entitlement client selects the secure channel, renewing it if necessary and the administrative query is then sent over the selected/renewed secure channel.

If the secure channel is not renewable for some reason, the entitlement client creates a new secure channel with any entitlement server whose information is present in entitlement client’s database. If the information of all the entitlement servers is out-of-date, the SP_USE operation fails. In this case, the SP_SETUP operation is performed by client application to collect the latest connection information of the entitlement servers, followed by a SP_LOOKUP of the administrator user.

After validating the client application’s session, the administrator user’s session is validated. If the entitlement client discovers that the administrator’s session has expired the entitlement client performs SP_LOOKUP to renew the session of the administrator user. In the next step the manipulation query (add, delete or update) is sent to the entitlement server. After the query is executed the entitlement server returns the result of the query back to the entitlement client. A detailed explanation of the SP_USE function is given in the thesis by Ciordas [9].

Chapter 5
THE ENTITLEMENT SERVER APPLICATION

The entitlement server is designed to be fault tolerant, and highly available to its potential users. It communicates securely (over an encrypted channel) with the entitlement client. The basic method for performing an entitlement check is the same as that explained in the thesis by Ciordas [9]. Additional function is added in order to incorporate multicast communication and database synchronization among the entitlement servers. The sections of this chapter explain the modifications made to the existing entitlement server module to make it fault tolerant, and highly available.

5.1 Multi-threaded Module Architecture

The complete functioning of the entitlement server has been divided into various threads running parallel to each other. Threading is a concept used to run two or more instruction streams concurrently. The entitlement server starts its execution by creating network interfaces that can be used by its various threads. One of the major design changes in the entitlement server is that it has two interfaces (ports) open for communication with the external entities instead of just one. One interface is used to communicate with the entitlement client that requests entitlement checks, and the other interface is used to communicate with the other entitlement servers in the multicast group for initial database synchronization explained in Section 5.1.2. The other major design change in the entitlement server is that, when the request is received from the entitlement client, instead of creating a new process, the entitlement server creates a new thread. This modification in the design is done for two reasons: a process is

expensive in (a) setup time and (b) memory space, *and* Inter-Process Communication (IPC) between threads is much easier.

Apart from performing entitlement checks and doing an initial synchronization of the database, the entitlement server provides a function to process the requests made to the multicast group, and to synchronize the database. Both of these functions are placed in separate threads and explained in the following subsections. Figure 5.1 shows the entitlement server along with its multiple threads. The client handler thread and the initial database synchronization thread are created upon request while the multicast-message-handler thread, dbAdmin thread, dbAdmin-handler and database writer threads run concurrent to main thread of the entitlement server. The following section explains the functionalities of the thread that handles requests from the entitlement client.

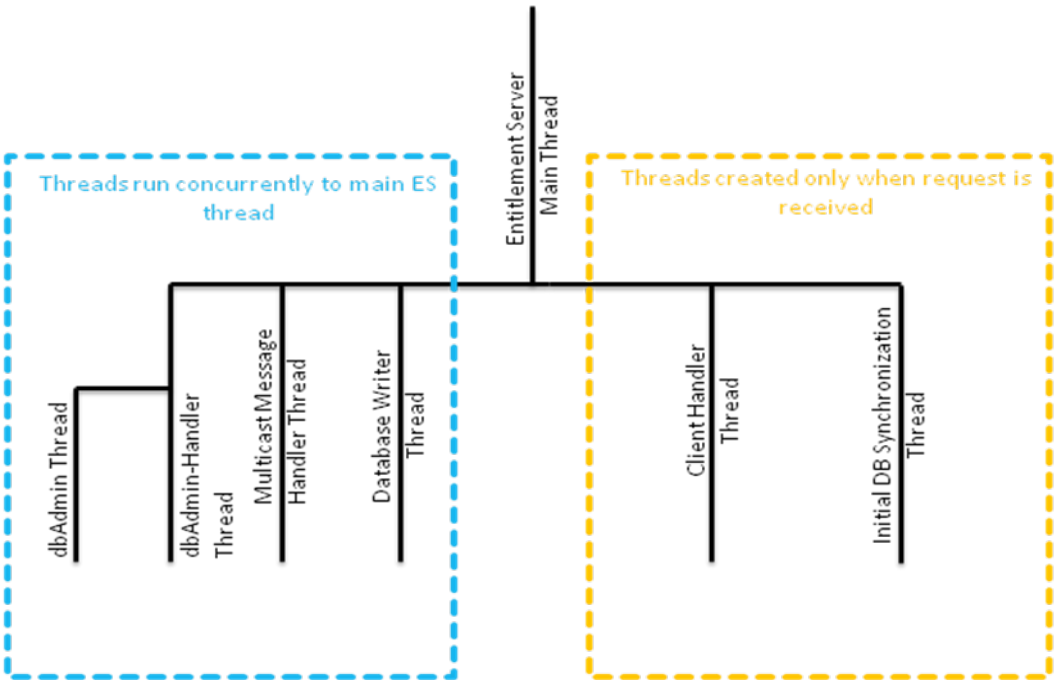


Figure 5.1: Multi-Threaded Entitlement Server

5.1.1 Client Handler Thread

The client handler thread is used to handle an entitlement check request from an entitlement client. As soon as the entitlement server detects that an entitlement client wants to communicate with it, the entitlement server creates an entitlement client handler thread (Figure 5.2). The requests from different entitlement clients result in the creation of different threads, and each of these threads is responsible for handling the entitlement clients individually. The client handler thread is created in such a way that when the entitlement client is done communicating with the entitlement server, the thread exits and the hardware and software resources utilized by the thread are reclaimed. After creating a thread for the entitlement client request, the entitlement server again starts listening for any new incoming connection. The client handler thread deals with the request from entitlement clients in parallel with the other activities of the entitlement server.

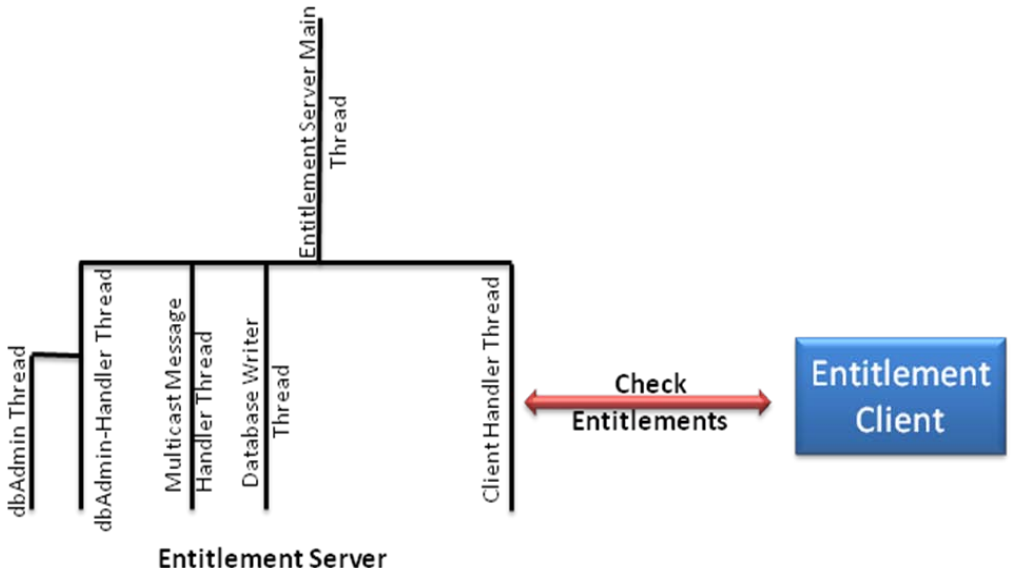


Figure 5.2: Entitlement Client and Entitlement Server Communication

The function of the client handler thread is similar to the working of the process as explained in the thesis by Ciordas [9]. An entitlement server can serve several entitlement clients by creating a separate client handler thread for each entitlement client sending a request. One alteration done to the existing design by Ciordas is that, the client handler thread does not write the transaction to the database directly. A separate dedicated thread, known as database writer thread is employed to write to database. This modification is done to avoid opening of database every time a request is received. The database writer thread opens the database only once and stores the identifier (known as file descriptor) with itself. So whenever a transaction is to be written to database, the client handler thread queues the transaction in the writer queue and sends a signal to database writer thread to perform a database write. The database write operations are done by the dedicated thread; the database read operations, however, are done by the client handler thread individually. The client handler thread uses the database file identifier, created by the database writer thread, to perform read operations. This approach also eliminated the use of mutual exclusion locks as the write to database is done using a dedicated write thread.

Apart from doing the entitlement checks, the client handler thread is also responsible for logging the update transactions along with the time stamp, since these logs are used to synchronize the database. The process of database synchronization is explained in Section 5.1.5. After logging a transaction, the client handler thread informs other entitlement servers about the modifications to the entitlement database. In order to inform all the entitlement servers, the client handler thread sends a transaction to

the multicast group. Therefore, the overall function of this thread is to validate user entitlements and notify other servers of any database changes.

5.1.2 Initial Database Synchronization Thread

When an entitlement server starts up, it synchronizes its database with the existing entitlement servers' database, before joining the authorization system. Figure 5.3 shows the thread communication and Figure 5.4 is the flow diagram of initial database synchronization. The starting entitlement server first checks whether it has any database with itself. In case the starting entitlement server have database, it first decides whether (self) database is synchronized with the existing database or not. For that purpose new entitlement server selects an existing entitlement server from the group and performs MD5 and database timestamp comparison. In case, the MD5's are same that means the databases are already synchronized so no need of synchronizing mechanism. If MD5's are different, then, the timestamps of the databases are compared. The database that is updated most recently gets the precedence. If the existing entitlement server is updated more recently than the new entitlement server then, the database transfer takes place from existing to new entitlement server. On the other hand, if new entitlement server is updated more recently than the existing entitlement server then, the new entitlement server sends its database to all the existing entitlement servers via multicast protocol.

In case, the new entitlement server does not have any database with itself, then it selects an existing entitlement server from the group and transfers complete database

to self. During the selection of an entitlement server from the multicast logical group, if there is no existing entitlement server present in the logical group, the new entitlement server opens self database if database is present with self otherwise it creates a fresh new database. After synchronizing database with the existing servers the new entitlement server joins the multicast group and starts serving the clients for authorization checks.

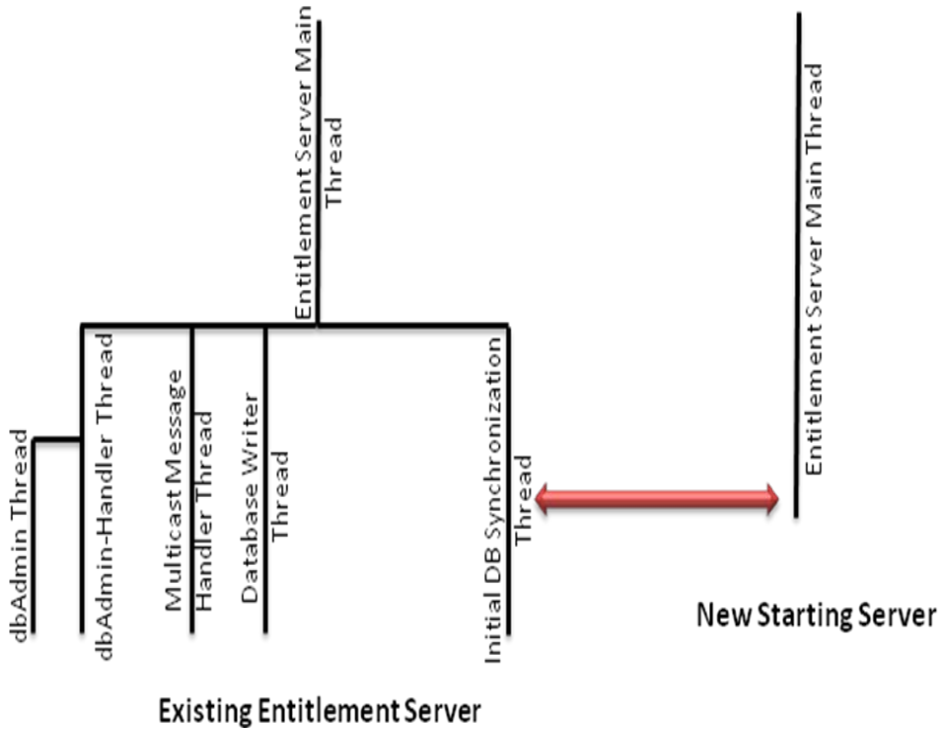


Figure 5.3: Initial Database Synchronization Thread

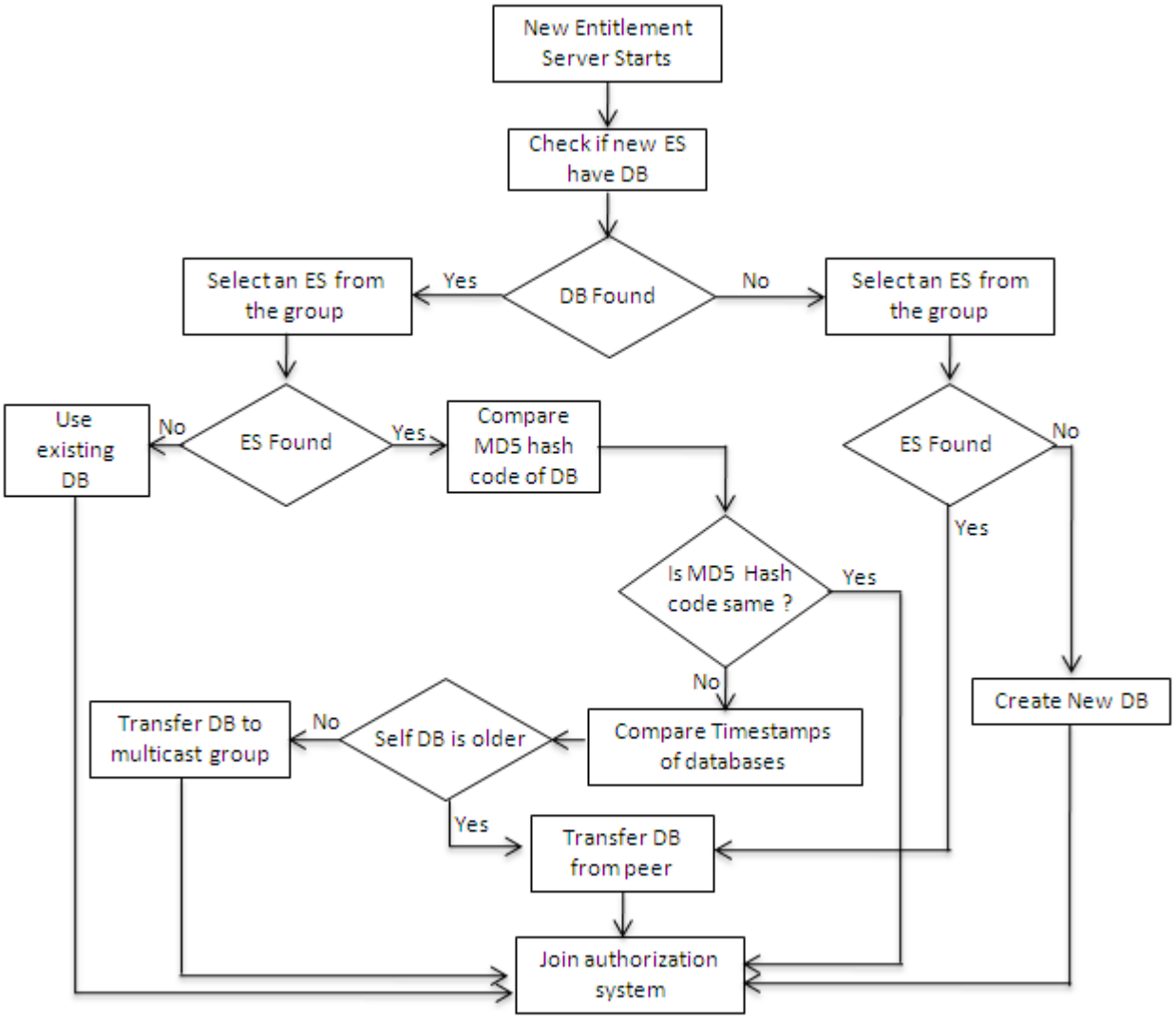


Figure 5.4: Flow Diagram of Initial Database Synchronization

5.1.3 Database Writer Thread

The database writer thread performs all the write operations in the entitlement server. At the startup, this thread opens all three databases (i.e. the entitlement database, the symmetric key database and the time database) and stores the identifiers with itself. Whenever the client handler thread has a write operation, it queues the transaction in a writer queue created by the main thread and sends a signal to the

database writer thread indicating that a transaction is ready to be written to database. The transaction has complete information that is required by the database writer thread to perform a write operation. As soon as the database writer thread receives a signal from client handler thread, the database writer thread performs the write operation in the appropriate database. Once the write operation is successful the database writer thread removes that transaction from the queue. Having a dedicated writer thread in the entitlement server has two main advantages. First, this thread avoids opening and closing of database files for every request that is received from entitlement clients. Second, as all the write operations are done in a single thread as opposed to multiple threads, mutual exclusion between the different database write operations is not utilized now.

5.1.4 DbAdmin Thread

The dbAdmin thread implements the database synchronization capability. The dbAdmin interacts with the multicast message handler thread (explained in Section 5.1.5) for carrying out the database synchronization. The dbAdmin first inspects whether the databases in the multicast group are out of sync or not. DbAdmin uses the MD5 hash code [6] sent by the entitlement servers upon request to find out if synchronization is required or not. DbAdmin uses the logs maintained with entitlements servers to synchronize the database. The reason, dbAdmin does database synchronization using logs instead of using actual database is given in next paragraph. To synchronize, dbAdmin first sends a request to multicast group (all entitlement servers) to send MD5 hash code of the logs maintained. After sending request dbAdmin

waits for two seconds to make sure that all the entitlement servers have responded. DbAdmin then compares all the hash codes received and makes a note of servers that have different hash code. In case all the MD5 hash codes are same the synchronization process is terminated as all the database are same. If hash codes differ with each other, dbAdmin makes a note of the entitlement servers that have un-equal hash code. To proceed with synchronization dbAdmin sends request to the noted entitlement server to send their logs on point-to-point connection. After receiving logs dbAdmin traverses all the logs and prepares a list of logs that is the union of all the logs without duplication. Then dbAdmin sends the new list created to the multicast group. All entitlement servers receive the combined list and apply the missing transaction(s). Each entitlement server determines missing transaction by comparing the list received on multicast with the list that is possessed by self. After applying the missing transactions by the entitlement servers, all the databases are synchronized.

The reason for using logs instead of actual database is to create a checkpoint on database. The checkpoint is created to minimize the data transfer across the network during synchronization process. At a given time the checkpoint has all the logs of past one hour or most recent fifty transactions whichever result in greater number of transactions in the checkpoint. The synchronization process runs every fifteen minutes and make sure that the data in the checkpoint is synchronized. Upon finding any discrepancies in the logs, the missed transactions are applied to the database to make it consistent.

The message exchange between the dbAdmin and the entitlement servers are encrypted using the public private key encryption [6]. Two pairs of the public/private key are distributed to all the entitlement servers in the multicast group. One pair is for the sender of the message, and other pair is for the receiver. The sender encrypts the message by the sender's private key, and again encrypts the encrypted message by the receiver public key after two level of encryption the message is sent to the receiver. When the encrypted message is first received, it is first decrypted by the private key of the receiver, then decrypted using the public key of the sender. Encrypting the message twice serves two purposes. First, encrypting the message by sender's private key ensures that message has been sent by the intended sender because only sender can have its private key. Second, encrypting the message by the receiver's public key ensures that the message can be decrypted only by the anticipated receiver, resulting in hiding the data from non-authorized applications. The keys that are shared among the entitlement servers of the multicast group are different than the keys shared among the service provider and the entitlement servers. Hence, the encrypted synchronization data passed between the servers is not visible to anyone outside the multicast group.

5.1.5 DbAdmin-Handler Thread

The dbAdmin Handler thread handles the fact that it disallows multiple instances of the dbAdmin to be active at a given time. The dbAdmin runs parallel with the entitlement server in a separate thread, and as many dbAdmins are present as the number of entitlement servers present in the multicast group. Due to the multiple

presence of dbAdmin in the network, the dbAdmin handler thread is responsible for avoiding multiple active dbAdmin entities in a particular multicast group. This thread regularly monitors the multicast group, ensuring that only one active dbAdmin is active at a given time. The procedure involved in selecting the active dbAdmin is explained in the next paragraph.

Initially, when an entitlement server becomes part of a particular multicast group, the dbAdmin handler thread checks if any active dbAdmin is already available in that multicast group. The thread takes charge of database synchronization if there is no other active dbAdmin available in the network. When there is a clash between two or more dbAdmin handler threads to take charge of the synchronization process, the thread running on a machine with the lowest IP address takes precedence in the synchronization process. This mechanism ensures that only one instance of dbAdmin is active at a given time in a particular multicast group.

5.1.6 Multicast-Message-Handler Thread

The multicast message handler thread processes all requests that are received through the multicast group. There are three types of requests that are received by the multicast group: the database update requests (sent by entitlement servers), the connection requests (sent by entitlement clients) and the database synchronization requests (sent by dbAdmin).

When an entitlement server wants to notify the entire group of entitlement servers that the entitlement database has been updated it sends database update

requests to the multicast message handler thread. The client handler thread (explained in Section 5.1.1) is used to notify the group about the update. An entitlement client requests the multicast message handler thread to send the connection information (IP address and port number), so that the entitlement server can be connected on a point-to-point connection. DbAdmin (explained in Section 5.1.3) sends the database synchronization messages to the multicast message handler thread.

The protocol explained in this section make it possible for the entitlement server to run on multiple systems and achieve a fault tolerant authorization system. The next chapter discusses accomplishments and conclusion.

Chapter 6
CONCLUSION

Fault tolerance is crucial to the entitlement server, considering the vast number of requests for entitlement checks for the users. The entitlement server is used to authorize users based on data that is not present in the Identity Provider. For example, the name of the virtual organizations to which a user belongs, the authorizations that are required to access resources are not stored in the Identity Provider. The entitlement server is used to store and manage this data. The main objective of this project is to devise a fault tolerant entitlement server.

This project implements a strategy that ensures that when one entitlement server fails, the entitlement client chooses another entitlement server from a group of entitlement servers and proceeds with the entitlement checks. The authorization process does not fail on the failure of an entitlement server. The fault tolerance is achieved by running multiple entitlement servers in the network independently yet logically connected using the multicast networking protocol.

The approach used in the project allows the addition of entitlement servers dynamically. That means no restart of the process is required to bring an additional entitlement server into the authorization process. At times when the number of requests for entitlement checks increases, more entitlement servers can be added to serve all the incoming requests. This provides a highly scalable environment for entitlement checks. Also, since the location of the entitlement server is determined on the fly, it is very easy to relocate an entitlement server from one physical machine to

another. It does not affect the authorization process. No changes need to be made to the entitlement client application. The dynamic selection of the entitlement server also makes it possible to remove an entitlement server from the network. In case the entitlement server is removed the entitlement client dynamically locates another entitlement server and continues to authorize the user. The complete process of routing requests to an entitlement server is transparent to the user. So the user does not notice any change in the front end when the entitlement server is added/removed/moved in the network.

The addition, removal or relocation of the entitlement servers, without impacting the entitlement client application, is facilitated using the multicast network technology. When an entitlement client application comes up it does not have any information about the location of any entitlement server. The very first authorization request triggers the client application to perform a lookup for an entitlement server. For the purpose of lookup of entitlement servers, the entitlement client sends a lookup request to the multicast group (the logical group of all entitlement servers in the network). In the process of looking up an entitlement server, the entitlement client sends a request to the multicast group instead of sending to a particular node/machine. This enables the process of addition, removal or relocation possible without impacting or making changes in the entitlement client application. The only requirement for the entitlement server to take part in the authorization process is that it should be a part of the multicast group. Hence it should be on a physical machine that is in a multicast-

enabled network. A multicast-enabled network means the routers of that network must support multicast communication.

Database synchronization among the entitlement servers is also done by using the multicast technology. Database synchronization is triggered by the update event of the database. The entitlement server on which a database update occurs is responsible for the synchronization. The synchronization packets are sent to the multicast group of the entitlement servers. Each server receives and updates its database. Along with this database synchronization process another module runs at regular intervals to check the integrity of all the databases. If some discrepancy is detected by the module, it performs the synchronization via multicast network technology.

While designing and implementing the functioning of the redundant authorization process, performance of the entitlement client and server is considered. The original version of the entitlement server created a process for each incoming request from an entitlement client, but the new version creates a thread for each entitlement client. The creation of a thread is less expensive than creation of a process hence it increases the performance of the entitlement server. However there is a drawback of creating multiple threads instead creating multiple processes. As, the threads that serve client are created in the same address space as the main thread of entitlement server, there is a possibility that the number of request increases enormously the entitlement server can run out of storage. One entitlement client sends one request at a time. For the entitlement server to run out of memory large number of entitlement client should send the request at same time. This scenario is also addressed

by the design. When an entitlement server runs out of storage it does not respond to the entitlement client that is requesting for authorization, in that case the entitlement client picks the next entitlement server from the group and proceeds with the authorization until if finds an entitlement server that can process the request. In case all the entitlement servers are out of storage a new entitlement server can be added dynamically to the group for undisrupted authorization service.

There can be few future enhancements to the system explained in this document. One of them is, load balancing of the authorization requests. According to the design of the proposed entitlement server, all the requests from a client are routed to the entitlement server with which the secure channel is established. This design can clog a single entitlement server with all the requests whereas other entitlement servers do not have any request to process. So a process/module can be developed to balance the load across the entire multicast group. As another enhancement, a monitoring system can be developed and deployed that does constant monitoring of the entire system. For example, it would be possible to monitor if all the entitlement servers are up and running, or, whether the requests are being missed due to any kind of failure. The proposed system can be migrated to support IPv6 communication.

The purpose of the project is to enhance the existing authorization process and make it fault tolerant and highly reliable. That involved the understanding of the existing authorization process. For completion of the project, research has been done on multicast network protocol and also study is done to develop an algorithm for the database synchronization. As the project has gone through all the steps of the software

development life cycle, so as a whole, the project was a great learning experience as an individual.

This project was presented before the members of the GPN in the annual meeting held in June 2010 at Kansas City (MO). This project was awarded an outstanding rating by a panel of judges and stood at first place. The entitlement server has now gone live on one of the servers at the University of Missouri - Columbia.

In conclusion, the redundant entitlement server is a step forward to make the existing authorization process more highly available. This project adds to the availability of the entitlement server by placing multiple entitlement servers on the network. The proposed designed also facilitates the portability of the entitlement server.

BIBLIOGRAPHY

1. Vassiliki Koutsonikola and Athena Vakali, LDAP: Framework,Practices, and Trends Aristotle University http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1336746 (January 2010)
2. R. O. Sinnott, J. Jiang, J. Watt, O. Ajayi. Shibboleth-based Access to and Usage of Grid Resources. GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing Publisher: IEEE Computer Society. September 2006
3. www.**oasis**-open.org/ (August 2010) - The official website of "Organization for the Advancement of Structured Information Standards" (OASIS). It is an non-profit international consortium that drives the adoption of open standards for global information security.
4. Samir Saklikar and Subir Saha Motorola. "*Next Steps for Security Assertion Markup Language (SAML)*". India Research Labs Bangalore, India
5. R.L. "Bob" Morgan, Scott Cantor, Steven Carmody, Walter Hoehn and Ken Klingenstein "*Federated Security: The shibboleth Approach"*. EDUCAUSE QUARTERLY November 2004
6. S.M. Bhaskar, S.I. Ahson "*Information Security: A Practical Approach*". Alpha Science International Ltd. Oxford, UK.
7. Makofske and Almeroth "*Multicast Sockets: Practical Guide for Programmers*" Morgan Kaurmann Publishers.

8. http://shibboleth.internet2.edu/ (February 2010) - Official website of Shibboleth "A Project of the internet2 Middleware Initiative"
9. Ionut Ciordas "*Fine-Grained Authorization In The Great Plains Network Virtual Organization*" University of Missouri-Columbia https://mospace.umsystem.edu/xmlui/handle/10355/4967 (June 2011)

Appendix A
Working of Multicast Protocol

The transmission of data from a source to a destination in a network can be done in many different ways. Depending on the protocol used, transmission can be categorized as broadcast, unicast and multicast transmission. Broadcast means to send to everyone. A broadcast packet sent by a single sender is received by all the receivers in the network. Unicast is the opposite of broadcast. In unicast communication only two parties take part in the communication. There is only one sender and one receiver in unicast. This is also known as point-to-point communication. A telephone conversation between two individuals is a perfect example of unicast communication. Multicast is somewhere in between broadcast and unicast. There are multiple receivers but not all. Various applications of multicast communication can include online chat groups, audio/video streaming and video-conferencing.

The multicast protocol provides the application programmer with the ability to send a message once yet it is delivered to potentially many users. The multicast protocol uses an efficient strategy to deliver the message to a group of users. Instead of sending N separate but identical packets to each of N receivers, one multicast packet is sent that reaches all the N receivers. A special range of Internet Protocol addresses is used to create logical multicast groups of the receivers. In IPv4 the range of multicast is 224.0.0.0 through 239.255.255.255. The receivers, who intend to receive multicast packets of a particular interest group, join that group. The source sends the packet to the multicast group and all the receivers that joined that group receive the packet.

Multicast communication relies on additional service of the network known as the “multicast forwarding tree” [7]. The concept is that the sender is placed at the root of the tree and receivers are placed on the leaves of the tree. The packet starts from the source (root) and make its way to all the leaves (receivers). When a branch is detected incoming multicast packet is copied and sent to each outgoing branch (Figure B1 and B2). This strategy ensures the delivery of packetized information to multiple receivers. For the deployment of multicast, the forwarding tree must be created. The network must have the required software and hardware to support multicast. Routers and switches in the network must be configured to allow multicast communication.

Based on the sender, multicast can be any-source multicast (ASM) or source-specific multicast (SSM). In any-source multicast any user in the network can send the multicast message to the group. In source-specific multicast, the packets that are delivered to receiver are those originating from a specific source requested by the receiver. Based on the level used to implement multicast, multicast can be network-layer multicast or application-layer multicast. In network-layer multicast routers actively take part in the delivery of packets. Routers make the copies of the packet as needed and forward to multicast receivers. In the application-layer multicast, end systems communicate using an overlay structure. Intermediate routers need not support multicast. Multicast function is achieved at the application level.

The design of the project described here uses any-source multicast and network layer multicast. In the project design any source can send data and any receiver can join and receive data. The project has implemented public/private key encryption to

communicate on multicast. As a result only the authentic participants can understand the data exchange over multicast. The authentic participants are provided with a pair of public/private keys. The sender is provided with its private key and receiver's public key. The receiver is provided with its private key and the sender's public key. While sending, the sender encrypts the message two times, first using its private key and then with receiver's public key. On the receiver end the message is decrypted with receiver's public key and then sender's private key.

In the project all the entitlement servers are provided with the two pairs of public/private keys they are sender's private key, sender's public key, receiver's private key and receiver's public key. During the synchronization process dbAdmin is considered as the sender and the entitlement servers are considered as the receivers of the multicast messages. The dbAdmin uses its private key and then entitlement server's public key to encrypt the message. After encryption the message is sent to the multicast group. All the entitlement servers receive the message from the multicast group. All of them decrypt the message using their private key and dbAdmin's public key. After decryption is successful the entitlement server's take the appropriate steps to fulfill the request received on multicast group. Even though an unauthentic application has joined the group it will not have access to any message passed in the multicast network. Likewise, unauthentic application can send a message but none of entitlement servers accept it since the unauthentic application does not have the private key of the dbAdmin. Hence the communication is secure among entitlement servers during synchronization.

In addition to providing the public/private key pairs for synchronization all the entitlement servers are also supplied with a separate public/private key pair to communicate with the entitlement client (on behalf of the service provider). These pairs of keys are supplied to facilitate the sharing to the symmetric key among the service provider and the entitlement server. The entitlement client, while sending messages to the entitlement servers, encrypts the messages with the service provider's private key and then again encrypts the message with entitlement server's public key. On the other end of the communication the entitlement servers decrypt the messages using its private key and then by the service provider's public key. The entitlement server takes appropriate action and shares the symmetric key. There after the communication is done using that symmetric key. As per the discussion in the previous paragraph the interaction between the entitlement client and entitlement servers is not visible to any unauthentic application.

In the current implementation the entitlement servers uses "232.78.3.1" multicast IP address to form the logical group and uses "9623" port for receiving the messages.

Figure B1 shows the example network. Routers in the network support multicast communication. Shaded hosts represent the hosts joined a multicast group. Router D may or may not be supporting multicast. Figure A2 represents the forwarding tree of the network in Figure A1.

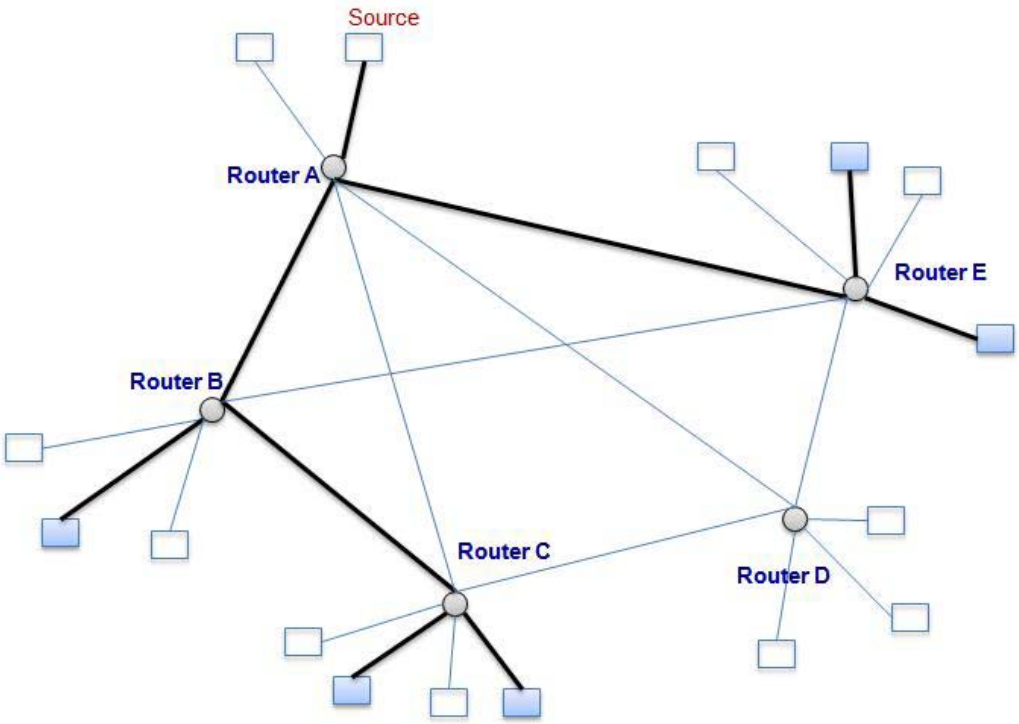


Figure A1: Multicast Network

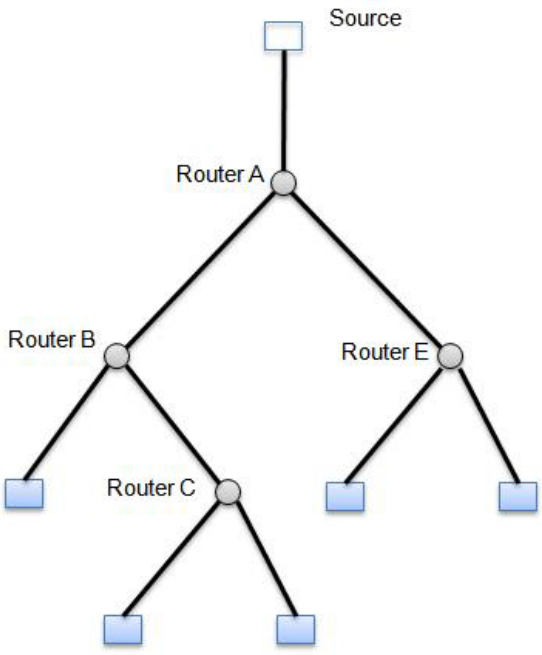


Figure A2: Forwarding Tree