Carnegie Mellon University
Information Networking Institute

Internet Billing Server
Prototype Scope Document
INI Technical Report 1993-1

October 14, 1993

1 Introduction

The purpose of this document is to summarize the work involved in the design and development of the Internet Billing Server[1] prototype. Detailed documentation is available for the requirements, design, and communication and security components of the Internet Billing Server. In addition, some key issues that must be considered prior to developing a Billing Server product are covered in two separate research documents. Section 6 provides a listing of the relevant documentation.

The first two sections of this document provide a high level overview of the Internet Billing Server project. Section 3 discusses the implementation phase of the project and may require a greater technical background. Potential areas for future work are covered in section 4.

1.1 Why Have a Centralized Internet Billing Server?

There exists a number of commercial and private entities that would like to provide services over the Internet, but are constrained from doing so because of the costs and complexity associated with billing. Also, users of information services are faced with the problem of having to subscribe and receive bills for each individual service.

The Internet Billing Server (IBS) is designed as a centralized entity that can provide billing and advertisement (directory) services to companies, hereafter called service providers, wishing to provide information services over the Internet. Users of information services, hereafter called end users, benefit by receiving a single consolidated bill and having access to a directory of available information services.

1.2 General Requirements of a Billing Server

This section gives an overview of the requirements of a Billing Server. Refer to the Internet Billing Server Requirements Document for the detailed set of requirements used for developing the prototype.

1. Internet Billing Server prototype discussed in this document was designed and developed by ten graduate students to meet the partial fulfillment of the requirements for Masters degree in Information Networking at the Information Networking Institute, Carnegie Mellon University. The graduate students involved in this project were Eric Bodner, Risa Carlos, Bill Evans, Cesar Garcia, Sui Ha, James Harris, Weseley Reece, Roy Russo, Atsushi Sekino and Dwight Walker. These students graduated in 1993 and they belonged to the fourth batch of students graduating from the Information Networking Institute.

The requirements of the IBS fall into six categories:

(1) Authentication and Communication.

When making a transaction at a store, identification is relatively easy. A driver's license with a photograph or a credit card with a person's signature will usually suffice as identification. For electronic transactions, an equivalent method is needed for identifying the parties to a transaction. In addition, secure communications must be guaranteed between end users, service providers, and the IBS.

(2) Account Management.

The IBS must maintain information about each service provider and end user, including financial, billing, access control and contact information. The IBS should support hierarchical account structures for both service providers and end users that match organizational structures.

(3) Billing and Statement Generation.

The IBS must be able to generate statements for end users and service providers summarizing usage information. In addition, the IBS must be capable of accepting payments from end users and making payments to service providers, and applying them to the appropriate account balances. End users require a mechanism for requesting adjustments to their account when a charge is deemed incorrect.

(4) Accounting.

All monetary transactions that the IBS performs must be recorded. Records are kept of all payments made to service providers by the Billing Server and all payments made to the IBS by end users. The IBS must be able to generate financial reports based on this information.

(5) Transaction Handling.

The IBS must ensure that an end user is authorized to use a selected service. During the course of the transaction, the IBS must guarantee payment to service providers for approved end user service usage.

(6) Directory Services.

The IBS must provide the equivalent of “white pages” and “yellow pages” to provide listings and advertisements of the services provided through the IBS.

1.3 Goal of Internet Billing Server Project

The MS3 Billing Server project produced a comprehensive set of requirements. The goal of the Internet project was to demonstrate the technical feasibility of an Internet Billing Server and to provide a modular design that could be incorporated into a Billing Server product.

Technical feasibility was to be shown by interfacing a real service provider with the Billing Server and demonstrating service transactions. The Mercury System[2], a library database search application, was chosen as the service provider. Using the full set of requirements produced by the MS3 group as a starting point, the Internet Billing Server project team selected and revised a subset of the requirements for use in the prototype.

2 Internet Billing Server Design

This section summarizes some of the key issues and decisions in the design of the Billing Server prototype. Refer to the Internet Billing Server Design Document for a complete description of the design.

2.1 Design Goals

The Internet Billing Server group pursued the following goals while designing the IBS:

(1) Extensibility.

Although the design is based on a reduced set of requirements, it should be easily expanded without a major revision of the design.

(2) Simplicity of the interface between a service provider's server and the Billing Server.

2. The Mercury System; Carnegie Mellon University, Computing Services.

Service providers should not have to make major changes to their servers in order to utilize the Billing Server.

(3) Security.

The transaction protocols must protect both the end user and the service provider. The service provider must be guaranteed payment for rendered services, while the end user must be guaranteed the prices agreed to in the limit negotiation.

(4) Minimal communications with the Billing Server during a service transaction.

The protocols should be designed to minimize the bottleneck at the Billing Server.

2.2 Design Methodology

The Object Modeling Technique (OMT) created by Rumbaugh et. al. [3] was used for designing the Billing Server prototype. An object-oriented design methodology was desirable because of its extensibility. Objects have clearly defined interfaces that allow for easy reuse in future designs. The OMT methodology leads to a well-documented and easily understandable design.

The OMT methodology consists of developing an object model, dynamic model, and functional model of the system. For the purpose of the prototype design, the object and dynamic models were found to provide enough level of detail to continue into the software development phase of the prototype. Therefore, functional modeling was not included in the formal design process.

In order to create the object model, the requirements were examined to identify object classes, their attributes, and the associations among classes. The dynamic model consisted of the creation of event traces, showing the interactions between classes, and state diagrams, modeling the behavior of the key classes. The dynamic model produced the required set of operations for each class. These operations were then added to the object model. The resulting design greatly simplified implementation of the prototype.

The developers of this methodology also created an object-oriented CASE tool, OMTool, that was used in both the design and implementation phases of the project. OMTool supported the

3. Object-Oriented Modeling and Design; Rumbaugh, Blaha, Premerlani, Eddy and Lorensen; Prentice Hall, 1991

development of the object model, but provided little assistance for the dynamic model. The CASE tool generates source code in the C++ programming language, defining every class and providing stubs for each operation within the class. Software developers must then insert the code that implements the functionality of the operations. OMTool also proved to be a valuable tool for producing the data dictionary within the design document.

C++, the object-oriented extension to the C programming language, was chosen for implementing the prototype. The members of the project team were all familiar with the C language. In addition, C++ is compatible with the selected design methodology and CASE tool. An object-oriented programming language tends to produce reusable code due to its well-defined interfaces.

2.3 Design Decisions

This section discusses some of the key design decisions in the Billing Server prototype.

2.3.1 Billing Model

Only the debit billing model is included in the prototype design. End users and service providers pay subscription charges in the billing cycle prior to using the Billing Server. Additionally, end users deposit money at the Billing Server to be used for accessing information services in advance of the usage. Since payments are required in advance, the IBS does not incur the risk of bad debt. While a Billing Server product will likely need to support historical billing (i.e. payment at end of billing cycle for utilized services) and credit card payments, the debit model is sufficient to demonstrate the feasibility of a Billing Server.

2.3.2 Account Hierarchy Model

The IBS supports account hierarchies that mimic organizational structures. An organization can be both a provider and user of information services. The IBS allows an organization to have multiple service provider and multiple end user account hierarchies. Alternatively, an organization may be as small as a single account, in the case of an individual wishing to access information services.

Each organization must specify an account administrator who has the sole ability to create accounts

and modify the hierarchy of the organization. Within an account hierarchy, higher level accounts possess higher authority than child accounts. Each account has an account owner who represents the contact for the account and has certain authority over the account and its subaccounts. The owner of an end user account is called the end user, while the owner of a service provider account is called the service contact.

For the model used in the prototype, all billing in an account hierarchy is aggregated and presented to the root account. Also, the entire account balance is kept at the root account. For end user accounts, the account administrator specifies for each subaccount, the maximum amount of the root's account balance that can be spent in a particular billing cycle. This model provides the benefit of producing a single aggregated statement per hierarchy, while allowing easy tracking of expenditures for each account.

Since the organization and its account administrator are ultimately responsible for its accounts, approval by the IBS account administrator is performed for the entire organization rather than for each individual account. Included in the approval of an organization is the maximum number of accounts that may be opened. The account administrator may request increases to the account number limit through a modification request, which also must be approved by the IBS account administrator.

2.3.3 Accounts Receivable Model

The accounts receivable model defines the process of receiving deposits from end users and service providers and applying them to the proper accounts. The Billing Server prototype is designed with a realistic accounts receivable model. When a deposit is received, it is first verified that it was submitted with a valid root account number. If the deposit is for a valid root account, it is placed in the *pending deposit* database table. If the deposit clears, it is applied to the appropriate root account. A check that bounces is recorded in the *bounced deposit* database table for auditing purposes. If bounced checks continue to be a problem, the IBS can establish appropriate policies.

A deposit that cannot be traced to a proper root account is stored in the *nonapplied deposit* database table. When the IBS financial administrator determines the appropriate root account, the deposit is moved to the *pending deposit or bounced deposit* table, or applied directly to the account, depending on the current state of the receivable.

2.3.4 Adjustment Process

End users having a dispute with a charge on their statement may submit an adjustment request to the IBS, specifying the particular service item in error and the dollar amount to be adjusted. The adjustment request is placed in the *pending adjustment* table until reviewed by the IBS financial administrator. Adjustments that are approved are moved to the *adjustment* table for auditing. Also, the service provider and end user account balances are updated to reflect the adjustment.

2.3.5 Account States

An account can be in one of three states. When the account state is active, it may enter into service transactions. Subscription charges must be paid for active accounts. A deactive or closed account is not permitted to enter into service transactions. Accounts in the deactive or closed state do not incur subscription charges. The difference between deactivation and closing is that deactive accounts can be reactivated, while closing an account is permanent.

An account can be deactivated or closed by the account owner, the owner of a higher level account within the hierarchy, the account administrator of the organization, or an IBS administrator. Deactivated accounts can only be reactivated by someone having equal or higher authority than the deactivation agent. A service provider wishing to close an account must first place the account in a deactive state for two billing cycles in order to allow processing of adjustment requests for services rendered through the account.

2.3.6 Treatment

In a debit billing model, where service usage is paid in advance, treatment is only necessary for failure to pay the advanced subscription fee. Failure to pay the subscription fee moves the account into the deactive state. An account deactivated due to treatment can only be reactivated by submitting a request to the IBS administration. An account remaining deactive due to treatment is closed following two billing cycles.

2.4 Billing Server System Architecture

The object model divides into four subsystems:

(1) Database System.

The database system consists of a number of database tables for maintaining all of the information required to operate the Billing Server. Each database table has a corresponding object class that manages access to the table. The other subsystems must all interact with the database subsystem.

(2) Transaction System.

The transaction subsystem processes requests from end users, service contacts, account administrators, and IBS administrators. A typical transaction request generally involves validation of the request, authorization of the requestor, and manipulation of database tables. Some examples of transaction requests include submitting an open organization request, approval of an open organization, adjustment requests, and modification of account information.

(3) Billing System.

The billing subsystem generates statements for end users and service providers every billing cycle, summarizing service usage and subscription charges. It also generates payments to service providers for services rendered to end users during the billing cycle.

(4) Audit and Report System.

The reporting subsystem is responsible for producing various accounting reports for auditors and Billing Server administrators. It is both an online facility for retrieval of small sets of payment, deposit, and adjustment records and a system that supports summary type reports over all account records.

2.5 Administration and Service Servers

The decision was made to break the Billing Server into two separate servers. One server handles the administrative tasks such as opening accounts and generating statements, while the other processes requests resulting from sessions between end users and service providers.

The service server receives calls for limit negotiation, limit renegotiation, invoice submission, and transaction aborts. Limit negotiation is the protocol between end users and service providers which

ensures that both parties have agreed to the same price lists and to the user specified price cap. Limit renegotiation is the protocol for allowing the current service session to continue with the addition of items to the price list in the agreement, or with the setting of a higher price cap. Details on the service protocols are given in the Internet Billing Server Design Document and in section 3.2.1 of this document.

This break up of tasks into two separate servers should provide greater throughput. For example, the service server can continue to process service requests, while the administrative server is blocked on a system resource or a database access.

3 Summary of Prototype Implementation

In this section, the key components implemented in the Internet Billing Server prototype will be summarized. Greater detail on the system architecture, process flows, and programming logic are given in the Internet Billing Server Design Document.

3.1 Cross Realm Authentication and Security

The Internet Billing Server was implemented to support cross realm authentication. The final demonstration included a Billing Server realm and a service provider realm. Authentication involves validating the identity of one party to another over a communications network. Authentication was provided by the Kerberos Authentication System[4].

Cross realm authentication involves allowing external organizations the option of maintaining their own authentication servers. These trusted, external authentication servers will grant tickets for the ticket granting server in the Billing Server realm. This will allow organizations the ability to provide their own access control and also eliminate the need for the user to log in multiple times. Without cross realm authentication, the user would have to log into the organization's authentication server and again to the Billing Server authentication server.

The prototype provides a user interface for the account administrator of the organization to open a cross realm entry in the Kerberos database. This involves specifying a realm name and a password

4. Project Athena Technical Plan Section E2.1, Kerberos Authentication and Authorization System by S.P. Miller, B. C. Neuman, J.I. Schiller and J.H. Saltzer, M.I.T. Project Athena, Cambridge, Massachusetts.

which is used to initialize an entry into the Kerberos database in the form <krbtgt.USER_REALM_NAME@BILLING_SERVER_REALM>. This entry is mirrored by an entry in the user's Kerberos database, with the instance and realm names reversed, for the Billing Server realm, using the same password. With this entry in the Kerberos database, the ticket granting server in the user's realm can issue a ticket for the ticket granting server in the Billing Server realm. This is transparent to the user and eliminates the need for multiple logins.

The Internet Billing Server team installed version 4.0 of Kerberos onto DEC workstations. Several modifications were necessary in order to integrate Kerberos with the Billing Server and to provide for communications using the remote procedure call (RPC) facility of DCE. The modifications provided the following functionality:

The Kerberos server can accept ticket requests from socket connections or from DCE connections. This allows Kerberos users to use DCE communication code to contact the Kerberos server. It is expected that DCE communications code will be more portable than socket code in the future. The prototype was implemented using the DCE remote procedure call communications system.

Kerberos grants tickets to authenticate users to service providers prior to the client requesting service. The Kerberos ticket granting server has been modified to contact the Billing Server access control interface prior to issuing the ticket, which allows communications between the client and the service provider. Both positive and negative access control lists are maintained for service providers and end users by the Billing Server and are used to determine if the ticket should be granted.

The administration server is involved in transactions with both end users and service providers. Examples include account openings and modifications. Kerberos is used to authenticate both end users and service providers to this administration server. The Kerberos ticket and authenticator make each remote procedure call authenticated. In addition, the Kerberos id is used by the administration server as a unique identifier of the person requesting the transaction.

Using DCE, a Kerberos database management server has been created as a slave to the Billing Server to allow it to add new users, delete users and to change passwords directly.

3.2 Service Provider/Billing Server Interface Issues

The Internet Billing Server prototype involved the demonstration of a service provider interfacing with the Billing Server. The Mercury Library Services System developed at Carnegie Mellon University was chosen to serve as a typical service provider which would use a Billing Server to charge for the database services it provides. In order to support the Mercury System, which currently does not charge for its services, a pricing server was built. The pricing server, as illustrated in Figure 1, serves as a generic pricing and invoicing module that a service provider can use if they decide not to build their own. Because the pricing and invoicing functions could involve a great amount of customization, this functionality is best handled by each service provider. Since Mercury does not have a pricing module, the Internet Billing Server team built the pricing server to provide these functions. The Internet Billing Server is designed to accept the calls for limit negotiation, limit renegotiation, invoice submission and abort transactions directly from the service provider. This interface will be important in the future when service providers take over responsibility for the pricing and invoicing functions.

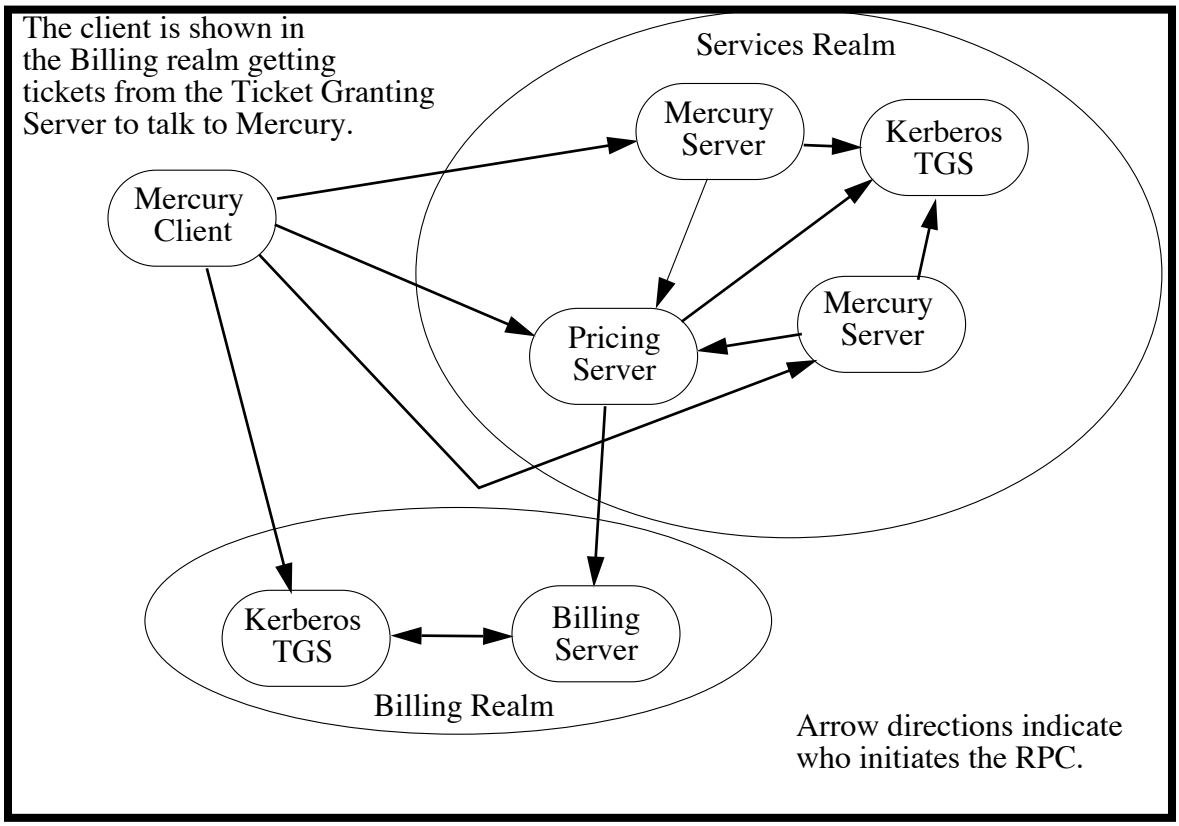

Figure 1: Server Interfaces Needed for Mercury to Communicate with the Billing Server

3.2.1 Pricing Server Interface

The following represent the generic set of services supported by the pricing server:

Requests from Mercury clients for a list of service prices.

The presentation of a price list and the setting of a price cap occurs the first time the client attempts to use a service item that has a usage charge. The prototype supports both free and charged items being offered by the Mercury System. This process of an end user accepting the price list and setting a price cap is limit negotiation and this information is stored at the Billing Server as the agreement between the end user and the service provider.

The Internet Billing Server also supports limit renegotiation. This is the process when an agreement can be updated to include additional service items which were not selected as

part of the original agreement, or when a price cap must be increased. The pricing server can detect both of these cases and send the necessary messages to the Mercury servers or Mercury clients to cause the initiation of the limit renegotiation process.

The submission of an agreement to the Billing Server during limit negotiation and limit renegotiation.

The Billing Server must approve the agreement and store a copy in order to guarantee payment to the service provider and to guarantee to the end user that the prices on the invoice are the same as on the original agreement made between the end user and service provider.

Requests from the Mercury Servers to check the client's available funds and to update the invoice.

The pricing server supports a call from the Mercury Servers to check to see if the next increment of service will cause the invoice total to exceed the price cap. The pricing server has a local copy of the agreement which was approved by the Billing Server. This agreement contains the price list of items, and the price cap which the client selected in the limit negotiation.

The Mercury Server calls the pricing server with the session identifier, the service item identifier, the quantity to be used, and a Kerberos identifier to identify itself to the pricing server. If the specified quantity of the service item will cause the invoice total to exceed the price cap in the agreement, or the specified service item is not in the agreement, the pricing server will respond with the corresponding message code. The Mercury Servers can use this code to cause the prompting of a limit renegotiation session with the client. Limit renegotiation is thus a way to increase the price cap or to include a wider range of service items in an agreement while a transaction is open.

The pricing server also supports a call to update the invoice at the pricing server for a given quantity of service. The pricing server maintains a living invoice which reflects the current amount of usage in the session. Normally the update invoice call is made after the service increment has been provided, while the check funds available call is made prior to delivering the service. The pricing server also supports a single call to update invoice which verifies that the funds are available prior to updating the invoice.

In the implementation with the Mercury Service, it was found that there will be times when a failure in providing the service item necessitates credit to the invoice. For example, in the printing of an abstract the invoice is updated when the job is sent to the printer. If a failure is detected for the printing job, a credit is sent to the pricing server.

Requests from the Mercury Servers to submit the invoice to the Billing Server or to abort the transaction.

The pricing server will package the service provider's version of the invoice and send it to the Billing Server when the Mercury Servers send a session complete signal. The pricing server will send a return message indicating whether or not the Billing Server accepted the invoice. The invoice can be rejected by the Billing Server due to an item being priced differently on the invoice than that found in the Billing Server's internal copy of the agreement, or due to a price cap being exceeded. In addition, the transaction can be simply aborted by the Mercury Servers.

3.2.2 Additional Pricing Server Issues

The Internet Billing Server implementation with the Mercury system included the following additional functional points:

Mercury was implemented in a different Kerberos realm from the Billing Server. The 'SERVICES' realm was set up for Mercury, while the Billing Server was in the 'BILLING' realm. Cross realm authentication was demonstrated in two ways. The first method involved the clients authenticating themselves to the Billing Server authentication server and obtaining a ticket for the 'SERVICES' realm's ticket granting server from the ticket granting server in the 'BILLING' realm. The second is the reverse case, where the client is in the 'SERVICES' realm and obtains a ticket for the 'BILLING' realm's ticket granting server from the 'SERVICES' ticket granting server. As mentioned in section 3.1, this is accomplished by initializing the two realm's Kerberos databases with an entry of the form <krbtgt.CROSS_REALM_NAME@HOST_REALM_NAME>.

The first case was demonstrated as a client performing a login to the Billing Server and then simply running the Mercury client code without an additional login necessary for Mercury. The second case allowed a login to the 'SERVICES' Kerberos authentication

server without the need for a login to the Billing Server in order to run its client code.

An objective in the design of the interfaces with the Mercury System was to provide for the concurrent existence of both paid and free services. For example, the process of limit negotiation is not presented to the client until an attempt to access a paid service has occurred. This flexibility allows Mercury to introduce portions of its services as paid services in an incremental manner.

The pricing server is designed to allow the price list files to change at any time. The prices agreed to by the client and service provider are kept in the agreement for that session and will not be affected by a price change during the transaction. The exception to this is when limit renegotiation occurs. In this case, the new price list presented to the client will reflect the current price list file. This is necessary since each accepted limit renegotiation generates an additional eight hour authorization number. A client could theoretically enter into a long transaction and go through limit renegotiation indefinitely to avoid increases in prices. The invoice generated in this case could have two invoice lines for the same service item. The first at the price prior to limit renegotiation, and the second after limit renegotiation.

The Internet Billing Server prototype with the Mercury System was designed to demonstrate as much functionality as possible for a typical information services provider. Given the constraints imposed by Mercury, the following are the types of charges that the team was able to demonstrate with the Mercury System:

Session charges. These represent a flat charge for connecting to a given database.

Custom charges for accessing different databases. The databases could be grouped under a generic service item identifier or have a unique service item identifier. These charges are usage based and involve charging for the number of records returned on a given query.

Printing charges. The databases of the current version of Mercury connected to, allowed the display and printing of abstracts. The pricing server module supported charging for the printing of these abstracts. Because printing involves sending the data to a print spooler which subsequently sends the data to the printer, there are several possible breakdown points involved in the process of printing. Therefore the design included the ability to recognize a print request that fails. These failed print jobs result in a credit on the invoice.

3.3 Client/Billing Server Transactions

The Internet Billing Server supports a wide range of account services for the end user and service provider clients. These administrative transactions are summarized in section 2.4. In this section, the Internet Billing Server client code will be summarized along with the key issues involved in these interfaces.

3.3.1 User Interfaces

The Internet Billing Server prototype included the following list of client executables. All of the executables perform a remote procedure call (RPC) to the Billing Server to validate the authority of the individual executing the code. For example, an account owner would not be able to run the account administrator's executable. This validation is performed using the user's Kerberos identifier contained in the Kerberos ticket which is sent as a parameter in the RPC. The graphical user interfaces were developed using Motif.

ibsclient -- Executable designed for the end user and service provider account owners

The following functionality is supported in this interface:

- review of account information
- review and modification of a descendant's account information in a given account hierarchy
- searching Directory Services,
- requesting adjustments
- deactivation, reactivation and closing of the account, or a descendant's account

aa -- Executable designed for the organization's account administrator

Individuals having an account with the Billing Server would serve as their own account administrator. The following functionality is supported in this interface:

- opening end user and service provider accounts
- review and modification of account information in any account hierarchy of the organization
- review and modification of organizational information
- search, addition, modification and deletion of directory services entries for offered services

- opening new realms to allow cross realm authentication with the Billing Server realm
- deactivation, reactivation and closing of an organization's accounts

ibsaa -- Executable designed for the Billing Server Account Administrator

These individuals are responsible for reviewing requests to open an organization with the Billing Server, requests to reactivate accounts, and the supervision of the structure of Directory Services. The following functionality is supported in this interface:

- approval or denial of requests to open new organizations
- approval or denial of modifications to organizational records
- deactivation, reactivation and closing of accounts
- approval or denial of requests to reactivate an account
- addition, modification and deletion of directory service entries and directory service categories

ibsfa -- Executable designed for the Billing Server Financial Account Administrator

These individuals are responsible for accounts receivable, accounts payable, auditing and review of financial reports produced by the Billing Server and the generation of statements for end users and service providers. Details on the debit model used in the Internet Billing Server prototype, and on the way that deposits, payments, and adjustments are processed are given in section 2.3. Greater detail can be found in the Internet Billing Server Design Document. The following functionality is supported in this interface:

- submission of received funds into the pending deposits, or the non-applied deposits database table
- retrieval and update of records in the non-applied deposits database table
- retrieval of records from the pending deposits table for movement to either the applied or bounced deposits table
- review of account records
- deactivation or reactivation of accounts (this applies to treatment of delinquent accounts)
- approval or denial of requests for adjustment on invoice items
- review of deposit, payment and adjustment records
- generation of statements for end users or for service providers
- generation of reports on deposits, adjustments, payments, bounced deposits, end user

usage for each service provider, and on accounts which have been deactivated due to treatment.

ibsopenorg -- Code designed to be obtained by new users of the Billing Server

This interface uses a special guest authentication scheme to allow the opening of an organization and the setting up of an account administrator. Once the organization is approved by the Billing Server Account Administrator, the account administrator can proceed to set up end user or service provider accounts and a new Kerberos realm if desired.

3.3.2 Directory Services

The implementation of Directory Services involves providing both a “yellow pages” and “white pages” type interface to end users, service providers, and account administrators. The “yellow pages” interface supports looking up service providers listed under different categories. The list of categories is displayed to the client and selection of a given category results in a list of service providers. Individual service provider descriptions and advertisements can then be obtained for a selected service provider.

The “white pages” interface supports the look up of individual service provider descriptions and advertisements given the service provider’s kerberos identifier. This portion of the design requires extensions in order to make it more meaningful to end users. These extensions are described in section 4.

3.4 Statement Generation

The Internet Billing Server generates end user and service provider statements whenever the Billing Server Financial Administrator initiates the process. Typically billing cycles are set up and the statements are generated once every billing cycle. The Internet Billing Server supports the generation of end user statements and service provider statements at different instantiations, but does not support breaking up the accounts further into billing groups.

The information contained in end user statements is as follows:

Itemized listing of the invoices for each account in a given hierarchy. Each sub-account’s usage

is itemized, including listing each invoice item charged to each account.

Total deposits and adjustments received for the account hierarchy during the current billing cycle.

Summary of the account hierarchy usage, charges, and the current financial state of the account hierarchy.

The information contained in service provider statements is as follows:

Itemized listing of each sub-account's end user usage in the current billing cycle.

Itemized listing of each sub-account's Billing Server charges in the current billing cycle. These include the percentage charged by the Billing Server on each transaction between an end user and the service provider. In addition, each service provider account may enter advertisements into Directory Services which causes a monthly Directory Services fee to be assessed. The Internet Billing Server design gives service providers the option of being listed in the "yellow pages" portion of Directory Services or not having an entry.

Summary of the account hierarchy total end user usage, total Billing Server charges, total adjustments, and the current financial state of the account hierarchy.

Indication of the amount the Billing Server owes to the service provider, and the date when the payment was processed.

3.5 Communications

The Internet Billing Server prototype uses the remote procedure tools in the Distributed Computing Environment (DCE) software developed for the Digital platform. Version 1.0 from Digital Equipment was installed on three Digital workstations. In order to demonstrate two Kerberos servers, the two servers which comprise the Internet Billing Server, the pricing server and two Mercury servers, the three machines had to be set up as both DCE servers and DCE clients. For example, the Kerberos server in the billing realm is both a client and a server. The Kerberos server makes requests of the Billing Server to see if a ticket can be issued for a service, given the requestor's Kerberos identifier. This is the access control functionality provided by the Billing Server. In this case, the Kerberos server is a client to the Billing Server. In other cases, clients make

requests of the Kerberos server to get a ticket to communicate with the Billing Server. In this way, the Kerberos server is acting as a server.

The DCE client components must be a version of DCE that can interoperate with the server components that are used in the Internet Billing Server prototype, which is DEC's version 1.0 for Ultrix. The Digital DCE client is the only version of DCE currently available that is capable of being used with the DCE servers used in the prototype. For this reason, and due to the scope of the prototype, communication between different types of platforms was not demonstrated in the Internet Billing Server prototype.

3.5.1 Integration of Kerberos and DCE

The Internet Billing Server prototype required the integration of the Kerberos server with the DCE remote procedure facility. In an effort to standardize the work, and make the programming consistent across all RPCs, the Internet Billing Server team set up a generic set of remote procedure call templates to be used under varying security requirements.

Authentication.

> This RPC uses the most basic level of security. It involves the client sending an authentication ticket to the Billing Server in the RPC parameters. The authentication ticket RPC verifies the identity of the client so that the IBS can check if the client is authorized to use the services of the Billing Server. The prototype does not use this level of security.

Mutual Authentication and Checksumming.

> This RPC includes authentication from the client to the server, as well as authentication from the server to the client. In addition to the authentication ticket, the client sends a number in encrypted form. The server decrypts the number, adds one to the number, re-encrypts the number, and returns the encrypted number back to the client. When the RPC call returns to the client, the client can verify that it is the server who sent the message by reading the encrypted number. Also, all parameters are checksummed to verify their integrity.

Mutual Authentication and Privacy.

This RPC includes the functionality of mutual authentication and the encryption of all the parameters between client and server. This RPC represents the most secure form of RPC and is used when the information that must be transferred is private. It is utilized extensively in the service transactions between the Mercury client and the pricing server and then from the pricing server to the Billing Server. Price lists and price caps have to be encrypted, and in the case of the client's version of the agreement, doubly encrypted. This encryption is done in order to ensure the integrity of the agreement. Invoices, and passwords are also encrypted.

3.5.2 Abstraction Communication Layer Built Over DCE

In addition to this work, the Internet Billing Server team developed an abstraction layer above the DCE RPC communications package. The abstraction layer was built using overloaded functions, redefined operators, templates and special infrastructure classes generated in the C++ programming language. In order to support a variety of needs, where some parameters could be encrypted and others sent in the clear, every RPC is given four character strings as parameters. Two of these parameters go from the client to server, with one being encrypted and the other sent as plain text. Likewise, the other two parameters pass information from the server back to the client, one of which is in encrypted form.

The benefits to this approach are as follows:

A single remote procedure call is used for all the types of requests between the client and the Billing Server. This allows the addition of new types of requests without having to write new RPC calls for each new request. The programmers simply pass a function name as a parameter to identify the target of the call.

The communication code is isolated in a single module which can be more easily tested and bullet proofed over time. A single change can be made, versus having to change the code in every RPC.

A simpler programming interface is provided for the application programmers. This greatly increased the programming productivity in the Internet Billing Server Project.

The additional communications layer greatly reduces the complexity of communication in the

Internet Billing Server prototype. This complexity is caused mostly by implementing Kerberos authentication and encryption on top of DCE. For example, an integer passed into a DCE RPC will be interpreted correctly at a server, even if the byte ordering or the size of the integer type differs on the client and server machines. However, by encrypting this integer outside of DCE and passing it into an RPC call, there is no automatic translation into the proper byte ordering and size at the server. Encryption produces a character array which contains no indication of type. The implemented communications layer avoids this problem by converting all values into a formatted character stream prior to the RPC call and converting the stream back into the individual values following the call.

This single RPC communications box is used between the client code and the administrative server portion of the Internet Billing Server. This includes communications between the interfaces described in section 3.3.1, and the administration server. For the communications between the Mercury clients, Mercury servers, pricing server and the Billing Server as described in section 3.2.1, separate RPC calls were written for each function. This was a result of Mercury being coded in C versus C++. Many of the tools used to develop the layer of abstraction are not as easily implemented in C as in C++. In addition, the number of calls between these entities numbers less than ten and can be easily managed as separate calls.

For these separate service RPCs, the prototype does not handle type conversion. This represents a problem in a cross-platform environment. A future version of DCE will have encryption and authentication built in, so that these conversion problems will be eliminated. An alternative solution is to convert all parameters into a formatted character stream prior to encryption, as was done for the administrative RPC. The implementation of conversion procedures for all types of variables and structures into character streams is more complex in C language, and therefore was not done due to time constraints.

4 Future Work

The Internet Billing Server prototype is quite extensive and includes a great deal of functionality. This is described in detail in the requirements and design documents. In order to take the Billing Server to the next level and on to a production version, the following issues need to be considered. These include items which were not implemented due to being outside the scope of the Internet Billing Server project, or due to time constraints. Many of these issues are covered in greater depth

in two additional research documents produced by the Internet Billing Server team. Issues of scalability and reliability as it pertains to the implementation of a Billing Server are discussed in one document. Cross-platform issues and communications are discussed in the second research document.

A variety of payment methods needs to be supported.

The Internet Billing Server supports the debit model as described in section 2.3.1. In order to attract a wider base of customers, charge cards, purchase orders, and other variations of a historical billing model will need to be supported. This will mean a greater financial risk for the Billing Server organization. This financial risk could be offset by charging an appropriate interest on delinquent funds.

The technical issues to be addressed in this area are how to best support the exchange of credit card numbers and forms of electronic money in an unsecure environment.

Billing groups need to be implemented.

The ability to generate statements for a selected sub-group of the Billing Server's clients is needed. This provides greater scalability and flexibility. Statement generation can be spread out over a specified period, with each billing group having their statements generated in a particular slice of the specified period. This also spreads out the load on the database system, since the statement generation process locks up the account databases when traversing the account hierarchies. It is also suggested that the Billing Server be further split into a financial server, besides the administrative and service servers, as implemented in the Internet Billing Server design. The financial server would support the statement generation process and the accounts receivable and accounts payable processing. In essence, the financial server would support the Billing Server Financial Administrator's interface as described in section 3.3.1.

The "white pages" portion of Directory Services needs to be redesigned.

The current implementation is described in section 3.3.2. A more meaningful implementation would be to allow clients to scroll through the names of service providers presented in an alphabetical format. The client could select a range of the alphabet and narrow in on the service name that is desired. After selecting the service name, the service

provider's full description and advertisement would be displayed.

The pricing server should support the transmission of invoices to the end user at the end of the transaction.

In interviews with library services personnel, a high demand for customized pricing and control over client interfaces was found. For these reasons, it is believed that the pricing server should be built by the service provider, with the Billing Server presenting a standard interface to perform limit negotiation, limit renegotiation and invoice submission.

Since it is believed that the pricing and invoicing functions are the responsibility of the service provider, the Internet Billing Server project did not implement invoicing between the pricing server and the end user. In the current system, the end user receives the account's invoices when the billing statements are generated. Ideally, end users should not have to wait until the end of the billing cycle to receive a copy of the invoice.

Memory scalability issues need to be analyzed.

The current design allows service providers to send the entire price list for its services over to the Billing Server for each session in which it is currently involved. In this case, the receiver of the RPC must allocate memory for the input arguments. If a vast number of service providers exist which have large price lists, then the Billing Server can potentially be hit at the same time with requests for limit negotiations, which its memory resources may be unable to handle.

In addition, there are some cases where the contents of a database table are brought into main memory and sent as a return argument for an RPC. For example, the entire pending deposit table is returned to the Billing Server financial administrator following a request for pending deposits. This problem has been solved for the retrieval of pending organization requests by using a unique identifier to reposition within the database table across multiple RPC calls. This solution allows a database table to be traversed in an ordered manner, returning either one record at a time or a set of records at a time. It should be relatively easy to implement this solution for other tables requiring ordered access.

Multi-threading at the server and other performance enhancing mechanisms should be addressed.

The Internet Billing Server uses an Ingres database to store information at the Billing Server. After some analysis, it was found that Ingres does not function in a multi-threaded environment. Since DCE creates a separate thread for each additional RPC received at the server, this deficiency is a problem. To solve this problem, the Internet Billing Server team set a variable in DCE that limits the number of threads to one. This is an inefficient solution, since each request must be processed to completion prior to beginning the next request. Multi-threading would yield better performance, especially when processing requests that utilize different parts of the database. A possible solution is to move to a database that supports multiple threads such as Informix or Oracle.

Closer analysis of security holes.

Version 4.0 of Kerberos does not account for replay attacks that happen within the timestamp period. This would allow a hacker to copy an authentication ticket and reuse it within the timestamp period, pretending to be the person the authentication ticket was copied from. To correct this problem, Kerberos client code would have to be modified to cache the authentication ticket so that they can not be reused. This would require modifications to Kerberos which we did not feel were essential for the prototype since these modifications have already been made to Kerberos version 5.0.

The client code which encrypts the price lists during limit negotiation, and limit renegotiation is provided by the service provider. Since the key is in a local file on the client's machine, the client code could include extra code which modifies the price lists prior to communication with the pricing server. This type of price list tampering would not be detected because the price lists would match and still be encrypted in the appropriate keys. A closer analysis of security issues in the Billing Server project would detail these problems and attempt to provide solutions.

Utility processes to clean up expired data and inconsistencies in the database.

An example of data that could expire is the agreement price list tuples and other tuples associated with a session between a service provider and an end user. The authorization number for a given session expires after eight hours. If the service provider does not send an abort message, these tuples will remain indefinitely in the database. In general, there is a need to develop database consistency and integrity programs to maintain an efficient use of memory and avoid possible side effects from junk tuples.

Communication connections across server crashes need to be strengthened.

The current prototype version of the Internet Billing Server requires the restart of client processes when the Billing Server crashes. This can be easily remedied by reviewing the status of communication calls and performing a reconnect if the status warrants. This fix was implemented for the connection between the Kerberos server and the Billing Server, but was not performed for the client to Billing Server connections, due to time constraints.

Business Plan.

Many business processes need to be formalized. For example, the policies concerning invoice adjustments and the deactivation of delinquent accounts are not currently specified. The Internet Billing Server prototype includes the ability to accept or deny requests for adjustments, but there is no policy on how to evaluate the adjustment request. In addition, the prototype treats delinquent accounts in an automated fashion at statement generation time. Accounts which remain delinquent for two billing cycles are deactivated, and then closed after an additional two billing cycles in treatment. The two billing cycle choice was selected to allow end users sufficient time to submit adjustment requests on invoices involving delinquent service provider accounts. A business plan would include issues such as the daily management of the Billing Server.

An economic feasibility study is needed.

The scope of the Internet Billing Server project did not include the evaluation of the profitability of a Billing Server product or a strategy for introducing it to the marketplace. The scalability research document discusses the concept of varying levels of reliability and availability in a Billing Server. An economic analysis would indicate the appropriate level of reliability and scalability to use when introducing the Billing Server. As revenues increase, the functionality, reliability and performance of the Billing Server can be increased. This project demonstrates the technical feasibility of a Billing Server. The successful marketing of a product also requires an analysis of the product's economic feasibility, in order to minimize the risks associated with developing and introducing the new product.

5 Conclusion

The issues discussed in this document provide an overview of the scope of the Internet Billing Server Project. In addition, areas where additional work are required have been discussed.

The objectives of the Internet Billing Server Project were to create a comprehensive object-oriented design of a billing server, and to develop a prototype in order to demonstrate technical feasibility. Specifically, the team wanted to demonstrate how the Billing Server would work with the Mercury Library System, which would then validate the object-oriented design. These objectives have been met. In many ways, the work documented in the following references could be used by individuals or organizations planning on developing a production version of an Internet Billing Server[5].

6 Additional References

Reference (1) *The Internet Billing Server Requirements Document*, Information Networking Institute MS3 Project[6], Carnegie Mellon University, July 1992, INI TR 1992-1.

Reference (2) *The Internet Billing Server Business Plan*, Information Networking Institute Student MS3 Project, Carnegie Mellon University, July 1992, INI TR 1992-3.

Reference (3) *The Internet Billing Server Scope Document*, Information Networking Institute Student MS3 Project, Carnegie Mellon University, July 1992, INI TR 1992-2.

Reference (4) Project Athena Technical Plan Section E.2.1, *Kerberos Authentication and Authorization System* by S. P. Miller, B. C. Neuman, J. I. Schiller and J. H. Saltzer, M.I.T. Project Athena, Cambridge, Massachusetts.

Reference (5) *The Internet Billing Server Requirements Document*, Information Networking Institute MS4 Student Project, Carnegie Mellon University, October 1993, INI TR 1993-2.

5. For additional information on the prototype and related documents, please contact Prof. Marvin Sirbu, Chairman, Information Networking Institute, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail sirbu+@cmu.edu. Phone 412-268-7195.

6. MS3 Student project was completed by a group of ten students to meet the group summer project requirements for Masters in Information Networking at the Information Networking Institute, Carnegie Mellon University. These students were one of the groups in the third batch of graduates from the Information Networking Institute and they graduated in 1992.

Reference (6) *The Internet Billing Server Design Document*, Information Networking Institute Student MS4 Project, Carnegie Mellon University, October 1993, INI TR 1993-3.

Reference (7) *Analysis of Distributed Computing and Cross Platform Issues*, Information Networking Institute MS4 Student Project, Carnegie Mellon University, October 1993, INI TR 1993-4.

Reference (8) *Availability, Reliability & Scalability Issues in the Internet Billing Server Design & Prototype;* Information Networking Institute MS4 Student Project, Carnegie Mellon University, October 1993, INI TR 1993-5.

Reference (9) *Invoice Transaction Set (810),* American National Standard for Electronic Business Data Interchange, ANSI, 1986.