Rt
490
c.1

Bank Intra Routing Protocol

BIRP

by

Rabie Sadek

Thesis submitted in partial fulfillment of the requirements for the Degree of

Master of Science in Computer Science

Division of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

2006

LEBANESE AMERICAN UNIVERSITY

Thesis Approval Form

Student Name:	Rabie Sadek	I.D.: 200302014
Thesis Title :	Bank Intra Routing Protocol BIRP	
Program :	M.S. in Computer Science	
Division/Dept :	Computer Science and Mathematics	
School :	Arts and Sciences - Beirut	

Approved/Signed by:

Thesis Advisor	Dr. Ramzi Haraty	[redacted]
Member	Dr. Samer Habre	
Member	Dr. Abdul Nasser Kassar	[redacted]

Date: August 21, 2006

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Rabie Sadek

Signature: Date:

To my parents

Acknowledgment

I would like to thank my advisor for his guidance and the committee members throughout my Thesis work. I would like to express my sincere gratitude to the Lebanese American University whose financial support during my graduate studies made it all possible.

Finally, I would like to thank my friends and family for their long support.

Abstract

SWIFT is a financial institution which provides an interface for banks in order communicate and transfer money worldwide. All banks over the world and all their branches are directly connected to SWIFT, so banks that have many branches in several countries will have lots of expenses including security infrastructure, SAA package license, yearly maintenance, and message costs. Banks always look for a good investment and the SWIFT routing protocol does not deal with intra bank message processing as a separate case. If a message is sent for either a branch or another bank, it must pass through SWIFT before it continues to the correspondent.

In this work, I will provide a new protocol Bank Intra Routing Protocol (BIRP) that intra routes all messages which are exchanged between the main branch and all its branches.

Contents

Page

I - Introduction 12

1.1 Overview 12
1.2 Need for the Study 13
1.3 Structure of the thesis 14

II - SWIFT Background and History 15

III - SWIFT Alliance Access (SAA) 18

3.1 SWIFT Alliance Overview 19
3.2 Structure of SWIFT Message 19
3.2.1 Header 19
3.2.2 Text Block 20
3.2.3 Instance Types 20
3.2.4 Intervention 21
3.2.5 Appendix 21
3.3 The Life Flow of a Message 21
3.4 SWIFT Alliance Access Services 23
3.5 Access Control Application 23
3.5.1 Calendar Application 25
3.5.2 Correspondent Information File Application 25
3.5.3 Event Journal Application 26
3.5.4 Message File Application 26
3.5.5 Monitoring Application 27
3.5.6 Routing Application 27
3.5.7 Security Definition Application 28
3.5.8 System Management Application 28
3.5.9 BK Management Application 29
3.5.10 Message Preparation Application 29
3.5.11 Message Creation Application 29
3.5.12 Message Approval Application 30
3.5.13 Message Modification Application 30
3.6 Security, Reliability, Recovery 32
3.6.1 Who are the Security Officers? 32
3.6.2 Alliance Security Installation 33
3.6.3Creating and Managing Operators 34

3.7 Alliance Developer Kit (ADK) 36
3.8 Disadvantages of SWIFT Routing Protocol According To Intra Banking Message Processing 38

IV - Enhanced Bank Intra Routing Protocol (BIRP) 39

V - Experimental Results 44
5.1 Costs $ 44
5.2 Comparison between both Protocols 45
5.3 Delay 46
5.4 Connectivity 47

VI - Conclusion 48
6.1 Limitations of the Research 49
6.2 Future Work 49

VII - References 50

VIII - BIRP Implementation 53

List of Figures

Figure 3.1 The Life Flow of a Message 22

Figure 3.2 How Messages are being Processed 22

Figure 3.3 Access Control Application 24

Figure 4.1 Bank Intra Routing Protocol BIRP 41

Figure 4.2 Scenario 1 Receiver is a Branch 42

Figure 4.3 Scenario 2 Receiver is not a Branch 43

Figure 5.1 Comparison between both Protocols 46

List of Tables

Table 3.1 Summary of all Access Control Applications 31

Table 3.2 LSO and RSO Installation Steps 33

Table 3.3 Steps used for Creating Operators by LSO and RSO 35

Chapter1

Introduction

1.1 Overview

SWIFT is a financial organization that provides all banks with a dynamic graphical interface known as SWIFT Alliance Access (SAA) in order to communicate and transfer money all over the world.

SWIFT Alliance Access is a financial message switch used to interface to multiple networks in order to transfer money in the most secure and consistent way.

The main goal of SWIFT is to create a world wide message processing and a common language for all international transactions.

All banks are directly connected to SWIFT via SWIFT net secure links so that they can communicate with each other. Every message sent by the sender must go to SWIFT and from there it will be forwarded to the correspondent.
SWIFT messages pass through many queues before they are moved to SWIFT. First, they are created in the message creation; then after that they are routed to both the verification and the authorization queues where they can be verified and authorized before continuing their tour to SWIFT. A message can be modified by sending it back to the modification queue, after that it can continue its path normally.

1.2 Need for the Study

The main problem is that every message sent to a branch must pass through SWIFT before going to the receiver and this leads to additional costs to the bank, besides that, if the connection is down then all branch messages will fail to be received and must wait until the connection is up again, and this delay will increase because of network traffic that is why all these problems must be avoided in order to have a better bank investment.

Institutions that have many branches in several countries have lots of expenses including security infrastructure, SAA package license, yearly maintenance, and message costs. Banks always look for a good investment and that can be achieved only if:

-message costs are saved

-messages are received with less delay

-connectivity is trusted

The SWIFT routing protocol does not deal with intra messages exchanged between branches as a separate case, so all messages will have the same flow regardless of whether the receiver is a branch or any other bank.

That is why I will provide a new protocol Bank Intra Routing Protocol (BIRP) that intra routes all messages which are exchanged between the main branch and all its branches.

The main goal of this routing protocol is: to make all branches centralized to the main branch, to save message costs transferred between branches, to decrease delay, and to provide better connectivity.

1.3 Structure of the Thesis

First in chapter two, I will provide a background related work and literature review; Second in chapter three, I will give a SWIFT Alliance overview then I will briefly give an explanation of the structure of a SWIFT message, its life flow in SWIFT, and the different services it provides. After that I will discuss all access control applications such as the routing application, message file application, message approval application, event journal application, message creation application etc...

In this chapter I will talk about the security, reliability and recovery that SWIFT provides for its banks. Then I will talk about the disadvantage of SWIFT routing protocol regarding intra message processing. Finally, in chapter four a new enhanced protocol called Bank Intra Routing Protocol BIRP for intra routing system will be proposed. Before concluding the thesis in chapter six, I have made a case study in chapter five on the enhanced BIRP protocol, where I have made a comparison between both protocols in terms of costs, delay, and connectivity.

Chapter2

SWIFT Background and History

SWIFT is a wide financial industry which provides a secure message exchange for around 8000 financial institutions in more than 200 countries.
SWIFT always provides the best services to its customers and this is clear from the products it offers, the security it maintains, and the reliability it provides.

In Belgium 1973, SWIFT was established by a group of people supported by 239 banks in 15 countries [1]. The Society for Worldwide Inter bank Financial Telecommunication SWIFT started the goal of creating a worldwide message processing [1] and a common language for global financial transactions. At that time all communications where via telex and there was not very strong security. In 1974, large organizations planned to be members of SWIFT. Initially, it has emphasized its security and reliability.
In 1977, SWIFT started to be active and the first message was sent by King Albert [1]. In that year about 3,000,000 messages were exchanged between 518 banks in 21 countries. In 1978, the first ten million messages were sent in less than 12 months.

In the 1980s, Hong Kong and Singapore [1] were ready to exchange messages live. SWIFT also introduced the ST100 interface application [1], [2] which was used by 900 customers. With the SWIFT geographical expansion, it introduced new message standards. In order to support traffic growth, SWIFT provided satellite communications for enhancing the financial services.

In 1991, SWIFT received the Computerworld Smithsonian Information technology Award [1] for its excellent work in maintaining financial telecommunication in an advanced way.

In 1995, SWIFT opened its office in Germany. Its target was to improve security and reduce costs for customers. In 1996, about 3,000,000 million messages were processed every day.

In 2001, SWIFT's aim was to gain the trust of all banks worldwide and to process messages throughout the net, thus the idea SWIFT net was established.
In 2002, SWIFT net Release 4.0 specified in [1] [10] went live and concurrently the first SWIFT net message was sent.

Now most of the banks around the world communicate with each other throughout SWIFT alliance access SAA interface in order to transfer money worldwide in the most secure way. Nowadays, SWIFT can be considered as one of the largest organizations which provide for financial institutions the best worldwide message processing.

In the annual report published on 11 August 2006 Johan Kestens [1] [11], the head of marketing and member of the SWIFT Executive Steering Group, said, "We are encouraged by the results customers continue to value our performance in the critical areas of
security, reliability and standards, with scores of 9.0 or higher. However, we should not be complacent; the results also pointed to a number of areas for improvement."

On Monday 26 September 2005, the launch of the first international money exchange company located between Western Europe and East Asia took place which is known by Dubai International Financial Exchange (DIFX) [1] [12]. Their aim is to become the leading message exchange. They want to have customers from all banks over the world.

Dr. Omar Bin Suleiman [12], The general manager of the DIFX, said, "This is a historic day for Dubai, the Middle East, and the entire region the DIFX will serve. The exchange will be a catalyst for economic growth and prosperity. It will strengthen the ties between the countries of the DIFX region, as well as between the region and the rest of the world." "The DIFX has been able to invest in the latest technology and, as a result, is highly automated, enabling it to act upon trades immediately."

According to Simon Eacott [13], Managing Director of Corporate Banking Services "SWIFT is becoming the 'Microsoft' of financial messaging – the benchmark against which everything else is measured."

According to Lori Hricik [13], Executive Vice President and Head of Treasury Services "SWIFT is a critical part of the value chain we deliver to our clients. When thinking about the role that SWIFT plays for them, my paramount concern is the need for resiliency and efficiency."

Chapter3

SWIFT Alliance Access (SAA)

SWIFT Alliance Access is a financial message interface [2], built in a way in order to facilitate the LAN and WAN network communication and transfer money securely.

SWIFT Alliance Access is made up of several applications each one has its own importance and specific role. For example, the routing application has to deal with routing rules and the processing of messages, while the role of the event journal application is to report any action that has been done by the operator in the system, the role of the message approval is to verify and authorize messages, and the role of the system management is to create operators and start/stop components etc…

Banks over the world are directly connected to SWIFT through SAA interface via different wired and wireless lines:

Wired connections:
-Dial up
-Leased line
-ISDN
-Internet

Wireless connections:
-Microwave
-VSAT
etc….

3.1 SWIFT Alliance Overview

SWIFT Alliance Access [2] runs on the server using the Windows 2000 or XP platform where operators work at different machines and communicate with the central server. The central server is the one that controls the message flow.

In SWIFT, messages are classified into two subformats:

Input: every message created at the sender side is called input message which is going to be forwarded to the correspondent.

Output: messages that are sent to the correspondent are called output messages.

3.2 Structure of the SWIFT Message

It is made up of:

1-Header
2-Text block
3-Instances
4-Interventions
5-Appendix

3.2.1 Header

It is the essential part of the message which consists of the sender and correspondent bank identifier code [2] [3]. The Bic [8], Bank Identifier Code, uniquely identifies the bank and differentiates it from other banks. It is made up of 11 characters.

Such as: JTBKLBBEXXX

The first four characters are for the bank name, the fifth and sixth characters are for country code, the seventh and eighth characters are for the city, and the last three are for branches. For example, XXX is the main branch and B01 for a branch.

Each message has its own, unique message identifier, UMID which is automatically generated by the system. It is used internally by Alliance as the identifier of the message. It forms the primary key of the message.

It comes in the following form: IJTBKLBBEXXX103REFERENCE

The first character is for the subformat, next for the correspondent bic code then the message type and finally the reference of the message.

3.2.2 Text Block

The text block [2] [3] contains the text details of the message. Each message type has its own text block syntax. It is made up of many field numbers and each field has its syntax.

Such as:
20 is for the sender reference
32 is for the amount transferred
59 is for the beneficiary customer name and address
52 is for the ordering institution
Etc…

3.2.3 Instance Types

The SWIFT message is made up of three instances original, copy and notification message instances as defined in [2].

When a message enters the system an original message instance is created. An important role of Alliance it is that messages can have copies and their process is independent from the original message, they can be created by specifying in routing rules. This allows, for example, an original message to enter the system, to be processed and then completed and at the same time for a copy to be processed to another queue.

3.2.4 Intervention

All events [2] that occur in the life of the message from the time it is created till its completion will be reported as interventions inside the message. They are automatically generated by SWIFT Alliance.

3.2.5 Appendix

The main purpose of appendix [2] [3] is to record all network interactions such as a message has been failed to be authenticated, or a message has been acknowledged, or a message was rejected so it returns as a non acknowledge etc...

Every reception from a network and every emission to a network must have one appendix attached to the message in order to record this interaction.

3.3 The Life Flow of a Message

A SWIFT message during its life passes through many queues before being routed to SWIFT. It is first created in the message creation queue then it is moved to the message verification then to the authorization. Now if the authorizer found that a message should be modified then it will be moved to the modification queue otherwise it continues to sitoswift queue and from there it will be finally processed to SWIFT. The life flow of the message is shown on figure 3.1 in detail.

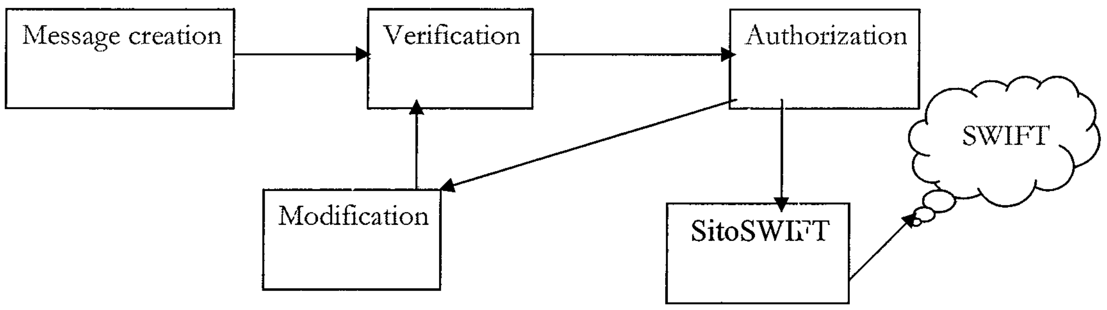


Figure 3.1 Life Flow of a Message

When a message is forwarded to SWIFT it is an input message, after an output message has been created SWIFT sends it to the correspondent returning either an acknowledgment or a non acknowledgment to the sender as shown in the figure 3.2.

The original message stays live till a notification acknowledgment or a non acknowledgment is received by SWIFT.

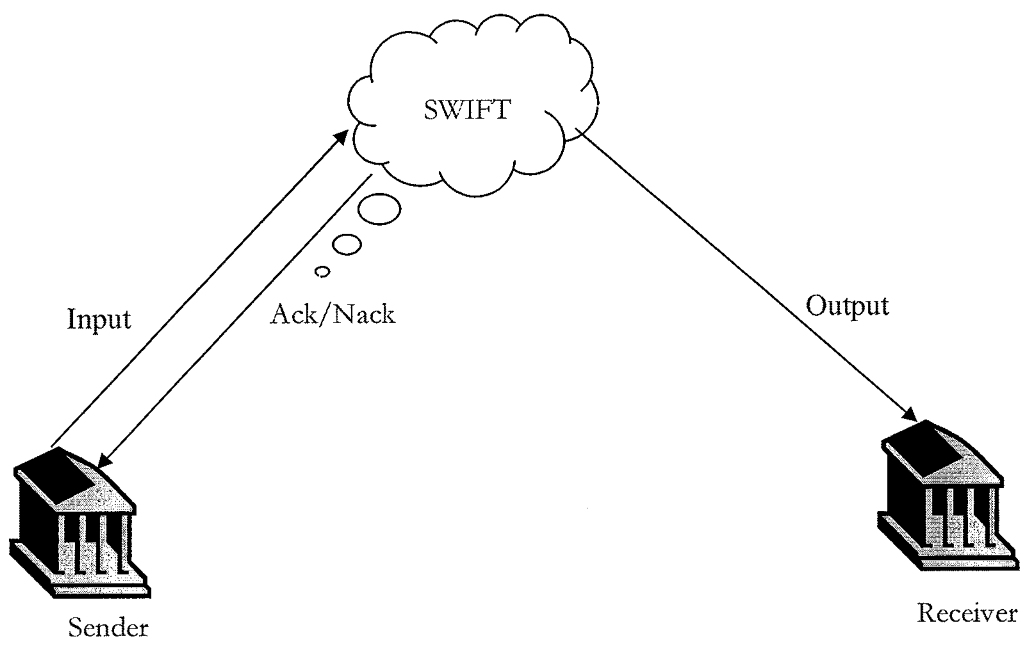


Figure 3.2 How Messages are being Processed

There are 999 types of SWIFT messages [2] each one has its own syntax. Some which financial, acknowledgment, confirmation and free text messages.

MT 100,110,200,201,202,330,340,.500,504...600,630,...800...910,912,...999.

3.4 SWIFT Alliance Access Services

The SWIFT Alliance interface application [2] provides several services to users and to the interface applications itself.

There are a number of different services that SWIFTAlliance provides:

_ Access control application: is the alliance interface

_ Bank information: used in order to search for bank details such as bank name

_ Message processing: for checking and routing messages

_ Routing definition: deals with queues and routing rules

_Security definition: deals with creating operators and changing passwords

_ System management: for start and stop components

3.5 Access Control Application

The Access Control application is the key into all SWIFT Alliance applications. It controls all SWIFT Alliance applications and functions.

When the user logs on into SWIFT Alliance, the applications that the user can use appear in the Access Control window. (figure 3.3)

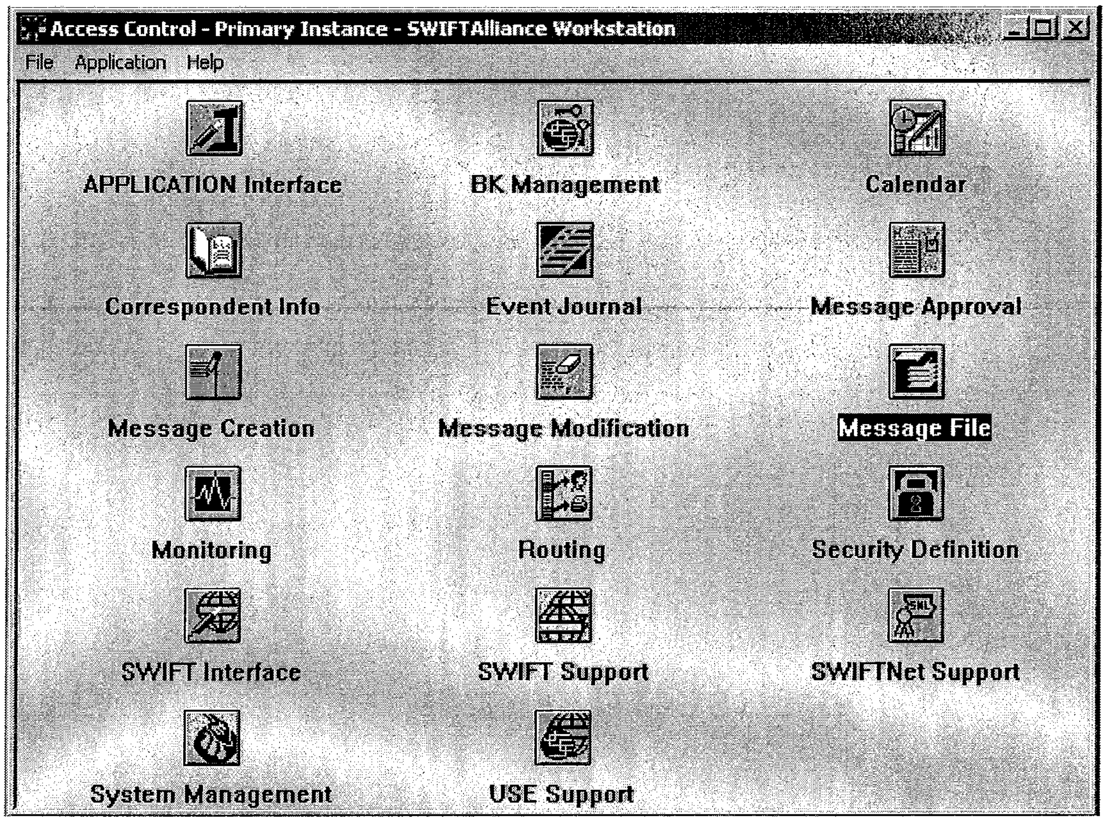


Figure 3.3 Access Control Application

These are all the applications that are provided by SAA [2]. The user profile defines which applications he can use. The user cannot sign on to SWIFTAlliance unless the administrator has allowed him to use the Access Control application.

The Security Officer or System Administrator is responsible for creating user operators and gives them privileges through the Security application in order to define the user's profile.

Users are not permitted to access the interface application unless they have authenticated username and password.

3.5.1 Calendar Application

The Calendar application [2] is used to:

_Create a yearly calendar

_ Modify a yearly calendar

_ Schedule SWIFT alliance applications in order to run them on different times.

For example, all messages sent and received are stored in alliance and in order to avoid the database from growing too large it is better to archive every time by scheduling the message file application and this can be done either manually or automatically.

3.5.2 Correspondent Information File Application

The Correspondent Information File (CIF) [2] [7], [8] contains essential data about correspondents. Each correspondent includes details such as the correspondent bic code, bank name, city, country, address, and so on.

The Correspondent file application is used to:

_ Search for banks based on a specific criterion such as bank name.

_ Add records

_ Remove records

_ Modify records that are in the list

The CIF is updated every four months in order to accommodate banks that have been recently included then the list in alliance will be updated and add all additional information.

3.5.3 Event Journal Application

All actions done by users, and by the SWIFT Alliance system, are recorded as events in the Event Journal application [2]. This file provides all actions performed in SWIFTAlliance.

Every event has number of details such as:

_ The date and time of the event

_ The name of the user that is using the system

_ The importance of the event.

The user can search for events in the event journal application based on specific criteria. Events can be implemented as alarms. Alarm events are used in order to alert the operator that a serious action must take place.

3.5.4 Message File Application

The message file application [2] is used in order to display all incoming and outgoing messages that are in alliance database.

Every message is made up of many instances, it can be original, copy, and notification. Each instance has many interventions which are the history of the instance and many appendixes that are the network interactions of the message.

In this application the user can:

_Look at the message details including interventions, appendixes

_Complete a message

_Route a message

_Reactivate a message

_Move a message

_Archive all messages that are completed in order to save database space.

3.5.5 Monitoring Application

This application is used in order to control the SWIFT Alliance system and ensure that it is running smoothly and if any problem or error occurs the user can control and manipulate it. The Monitoring application [2] provides an updated status for message queues, events, and processes.

For example, when a user wants to check how many messages are in a specific queue in order to avoid being full, or the administrator wants to check which user is accessing which queue.

3.5.6 Routing Application

The flow of messages within SWIFTAlliance is controlled by a routing schema. The routing application is made up of many queues where messages accumulate.

Each queue can have many routing rules defined by the user; they are used in order to control the flow of messages. The routing schema is activated when no more changes are needed in the routing rules.

In the routing application [2] the user can:

_Create routing rules

_Delete routing rules

_Activate routing schemas

_Deactivate routing schemas

All messages are controlled by the message processing function MPFN, its role is to process all messages from one queue to another.

3.5.7 Security Definition Application

The role of Security Definition application [2] is to define which SWIFTAlliance Application each user can access in order to work on it. This is done by assigning a user profile to each user.

The Security Definition application is used to:

_ Create SWIFT Alliance users

_ Modify SWIFT Alliance users

_ Remove SWIFT Alliance users

_ Configure security parameters, such as the number of days which a user password has to change the password.

_ Approve user in order to be able to sign on to alliance.

The user that is created must be approved by the administrator in order to have access to the alliance application.

3.5.8 System Management Application

The role of System Management application [2] is to control SWIFTAlliance.

It is used to:

_Change date/time formats

_ Shut down the system

_ Create queues

_ Back up and restore SWIFT Alliance database

_ Stop and start a protocol application

_ Define events that need to be set as alarms

3.5.9 BK Management Application

The main job of bilateral management key [2], [9], [15] is to authenticate messages that are created by the sender and that are going to be sent to the receiver via SWIFT net.

Sets of bilateral keys are used by correspondents in order to legally exchange messages. When the user sends a message, SWIFT alliance uses the bilateral key agreed on in order to calculate the MAC and attach it to the message, now after the receiver receives it he calculates the MAC based on the bilateral key, if they match then this means that the message arrived without errors while if the MAC did not match it means that the message is not reliable, it was invaded.

3.5.10 Message Preparation Applications

There are three message preparation applications [2] in the life of a message:
Message Creation, Approval, and Modification. These applications are used in order to create, verify, authorize and modify incoming and outgoing messages inside SWIFT Alliance application.

3.5.11 Message Creation Application

The Message Creation application [2] is the base application since there we create all message types, we should fill all fields before routing the messages in order to continue the process.

We should fill the sender, receiver, text block, network if it is SWIFT or telex. The messages can be saved as templates in order to be used next time.

3.5.12 Message Approval Application

The Message Approval application [2] is used in order to verify and authorize messages before going to SWIFT. This is used for security reasons in order to be sure that for example the amount transferred to the beneficiary X is correctly sent.

In banks there are users whose jobs are only to authorize or verify messages.

3.5.13 Message Modification Application

Every message that is wrongly created or it has failed to be authenticated and verified is directly sent to the message modification application [2] so that it can be checked again in detail and correct the wrong field values entered.

After the user has modified successfully the message, he can route it so that it can be verified and authorized again before it continues its tour to SWIFT.

In the table 3.1 below there is a summary of all access control applications and their roles.

Table 3.1 Summary of all Access Control Applications

<u>Access Control Applications</u>	<u>Role</u>
System Management Application	Start and stop components
Calendar Application	Provides calendar for the system in order to schedule applications
Correspondent Information File Application	Searches correspondent details based on specific criterion
Event Journal Application	Displays all events and alarms occurred in the system
Message File Application	Displays all messages that are inside alliance database
Monitoring Application	Controls the alliance system
Routing Application	Routes messages from queue to another
Security Definition Application	Creates, modifies, deletes operators and assigns profiles for them
BK Management Application	Authenticates messages exchanged between two parties
Message Creation Application	Creates messages
Message Modification Application	Modifies messages
Message Approval Application	Authenticates and verifies messages

3.6 Security, Reliability, Recovery

3.6.1 Who are the Security Officers?

Security Officers [2] play the main role in configuring and managing the security functions of SWIFT Alliance.

Two Security Officers are defined in SWIFT Alliance [2] the Left Security Officer LSO and the Right Security Officer RSO; each one has a specific role. Both Security Officers can create operator definitions, both must approve before changes are taken into consideration. Users are unable to enter alliance unless approved by both security officers. Assuming that the responsibility of user1 is to authorize messages so the security officer provides for him the message approval application in order to authorize and route messages.

Both LSO and RSO security officers are responsible for:

- entering both the SWIFT Alliance initialization passwords and the SWIFT Alliance master password when SWIFT Alliance is installed for the first time.
- creating new operators and assigning for them specific applications to be accessed.
- modifying operator privileges.
- removing operators.
- approving operators so that the user can authentically access the alliance application.
- resetting user passwords in case a user forgot his password.

3.6.2 Alliance Security Installation

As seen in table 3.2 below before installation SWIFT provides the security officers with generated initialization and master passwords and they are sent each part of the password to LSO and RSO [2], [21]. They store the password securely because based on them SAA are installed on the system.

During installation LSO enters part1 of the initialization password and RSO enters part2. LSO signs on using Part 1 of the Master Password as the operator password in order to enter the system for the first time, and then changes the LSO operator's password after he enters the first time. Also RSO does the same job and in this way they can enter the system and create users and assign them profiles.

Table 3.2 LSO and RSO Installation Steps

Before installation	SWIFT generates an Initialization Password and Master Password, and sends one part of each password to the LSO and RSO
	LSO and RSO store the passwords until they are required
During Installation	LSO enters Part 1 of the Initialization Password. RSO enters Part 2 of the Initialization Password
After Installation	LSO signs on using Part 1 of the Master Password as the operator password, and then changes the LSO operator password.
	RSO signs on using Part 2 of the Master Password as the operator password, and then changes the RSO operator password.

3.6.3 Creating and managing Operators

Both LSO and RSO security officers are responsible for creating operators [2]. In the first step, LSO must enter the security definition application, then he creates a new user name, assigns a profile for the user and approves him after having memorized the left part of the password. Now the RSO signs in so that it can approves and displays the right part of the password.

After the operator has been approved, The LSO and RSO give the left and the right part of the passwords to the user so that he can change them from the first log in and enters the system normally. The steps are summarized in the table 3.3 as seen below.

Table 3.3 Steps used for Creating Operators by LSO and RSO

LSO signs on access control application and opens the Security Definition Application
LSO enters the new user full name
LSO assigns profile(s) to the operator
LSO approves the operator
LSO displays the left part of the user's system password
LSO signs off
RSO signs on access control application and opens the Security Definition Application

RSO approves the operator. Now that both security officers have approved the operator, the operator is automatically enabled.

RSO displays the right part of the user's system password

RSO signs off

Both LSO and RSO give left and right part of the passwords to the user so that he can change them from the first log in

In order to have a reliable application, SWIFT has provided two SWIFT Alliance databases, the live database and the shadow database.

The live SWIFT Alliance database contains all operational files such as the event journal and the message file, they are continuously updated.

The shadow SWIFT Alliance database consists of a copy of the live database. It is used for backup and recovery purposes so that if a crash occurs then the alliance can recover by restoring the shadow database to live and it will return to its initial state before the crash.

3.7 Alliance Developer Kit (ADK)

The SWIFT Alliance system is composed of a collection of components that are integrated inside alliance. Each one has its own role, functionality, and importance inside SWIFT. Developers are able to create new components by ADK.

The Alliance Developers Kit (ADK) [3] is a package which allows developers outside of the SWIFT Alliance development team to implement new protocols. These protocols that have been developed using the ADK can be smoothly integrated with SWIFT Alliance. They will have the same security which is available for the standard SWIFT components.

The ADK provides libraries that contain APIs. APIs are functions or procedures that are implemented in C programming language and these procedures are called via remote procedure calls RPC [3]. There are classes for every part of the message such as the header, text block, interventions, and appendix. Each one has a set of parameters which defines each class. For instance if I want to read a message from SWIFT I use mfsgetmessage ( ), if specifically I decided to read only the header or the text block I will use mfsgetheader ( ), mfsgettext ( ).

Some of the APIs are listed below:

-Mfsaddheader ( )

-Mfsupdateheader ( )

-Mfsgetheader ( )

-Mfsgettext ( )

-MfsgetIntervention ( )

-Mfsupdateintervention ( )

-Mfsgetappendix ( )

-Mfsupdateappendix ( )

In order to have all APIs operational, they must be registered first before implementation otherwise the functions will not work. The full implementation is available in Appendix A.

3.8-Disadvantages of SWIFT Routing Protocol According To Intra Banking Message Processing

The problem is that every branch is directly connected to SWIFT and so lots of expenses and problems arise including Security infrastructure, SAA package license, yearly maintenance, especially message costs. In case SWIFT connectivity is down then all banks are disconnected. If the correspondent bank is a branch or not it should pass to SWIFT. So I have found that if I create a new protocol whereby the main branch takes the job and intra routes the message to the receiver it will be great.

Our main goal is to provide a new protocol having the following enhancements.

- All branches centralized to the main branch.
- Save costs: in this way banks are saving messages costs that are intra to their branches.
- Less delay: since it takes less than 1 sec for a message to reach the correspondent because of real time exchange.
- Fault tolerance: since all branches will be centralized to the main branch this means that if it is disconnected from SWIFT then all branches will continue to operate normally.

Chapter 4

Bank Intra Routing Protocol (BIRP)

Intra Routing Protocol was created as an enhancement to SWIFT routing protocol in order to take every input message from the routing point BIRP Input and checks if the Bic Receiver is a branch or not now there are two possibilities:

If the correspondent is a branch then it creates two messages an output message which is sent to queue Output and process it to the correspondent and an Acknowledge notification instance sent to queue Acked telling the sender that the message has been received successfully by the correspondent.

Otherwise if the receiver is not a branch then only an instance non acknowledge is created and the message is routed to queue Nacked so that it can be forwarded to Sitoswift by a routing rule specified on this queue.
After that the message is directed to SWIFT for processing an output message to the correspondent.

The original instance will always be complete so it is free from any routing point while both the notification and the output messages are live placed in the corresponding queues BIRPAcked and BIRPOutput. All events are reported as interventions inside every message.

There are four different BIRP routing points created in order to handle the incoming messages in an efficient way. They are listed as follows: BIRPinput, BIRPoutput, BIRPacked, and BIRPnacked.

The BIRPinput is the central queue of the component where every incoming message is read by the BIRP protocol. Routing rules have been created on this queue so that if the message instance is an acknowledgment then it is sent to queue BIRPacked, while if the message instance is a non acknowledgment then it will be sent to BIRPnacked and the output message will be sent to BIRPoutput in order to be forwarded to the correspondent by a specified routing rule.

All messages that pass through these routing points are automatically handled by the message processing function mpfn whose role is to process and control messages. The BIRP protocol is integrated in the main branch system and all branches are linked to it.
So all messages will pass through the main branch and its role is like SWIFT to decide whether to create an output message and transfer it to the branch or to forward it to SWIFT so that it can send the message to the specified correspondent. This protocol can be started or stopped by the system administrator.

In this way all messages that are exchanged between branches will not be forwarded to SWIFT. All the communication will take place between the branches and the main branch independent of SWIFT net.

Branches 1 and 2 are only linked to the main branch as seen in figure 4.1.

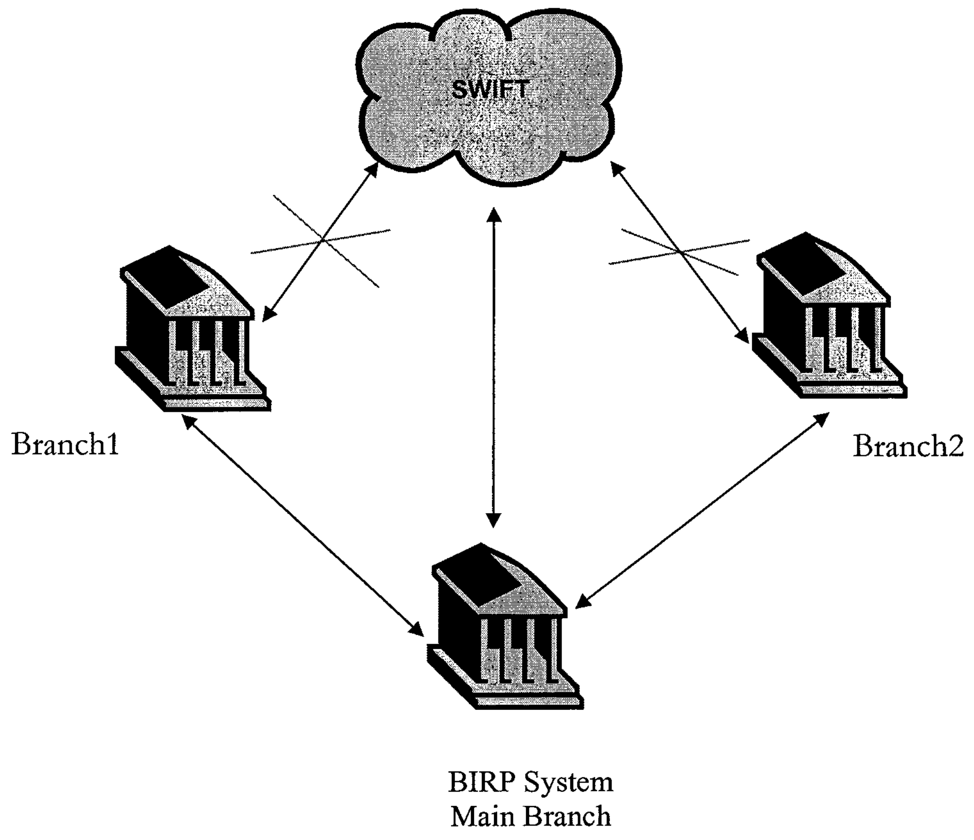


Figure 4.1 BIRP Processing

In figure 4.2 we can see the first case if the receiver is a branch so the routing will be intra the main branch, the message will go from branch1 "sender" to the main branch it is verified and authorized and then it enters the BIRPinput queue where the protocol starts to scan the message.

Since the correspondent is a branch "branch2" an output message is created and processed to queue BIRPoutput which will then be forwarded to branch2 and an acknowledge notification will be created confirming that the message has been received successfully by branch2.

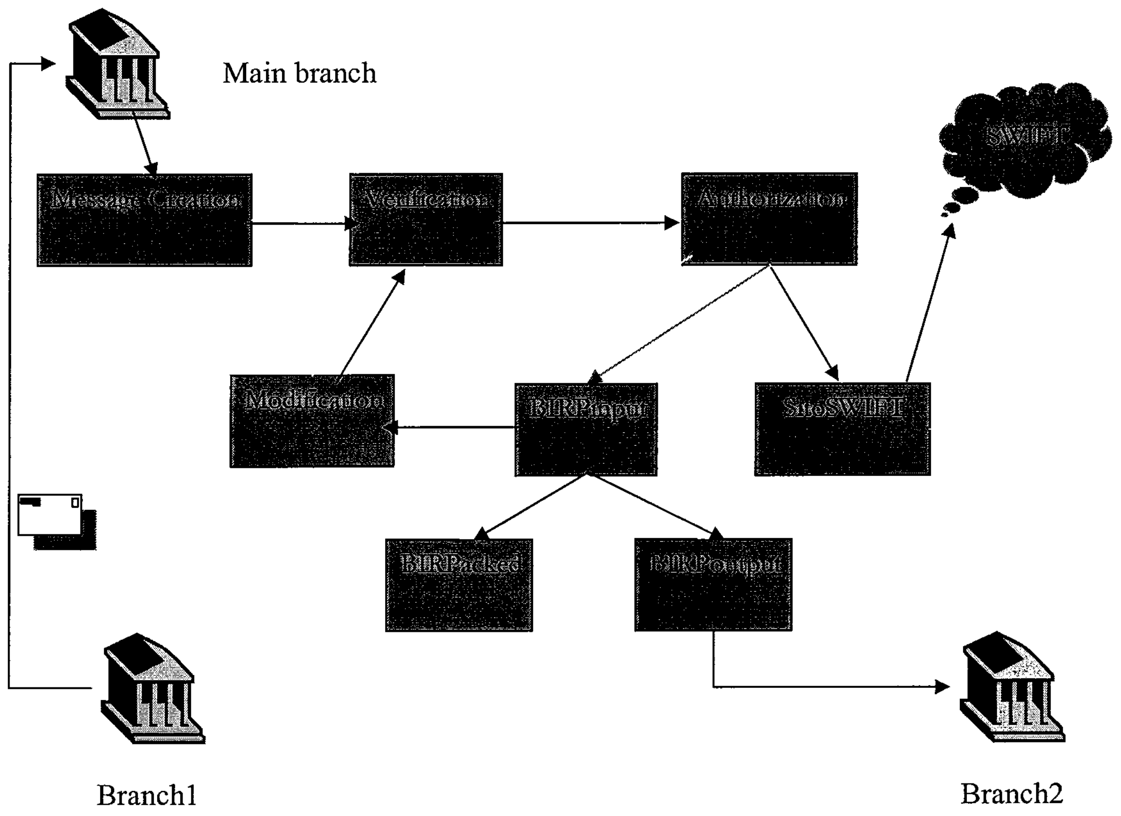


Figure 4.2 Scenario 1 Receiver is a Branch

Now in the second scenario as shown in figure 4.3 the receiver is not a branch so the BIRP routing protocol creates a non acknowledgement notification sending it to the BIRP nacked queue. By a defined routing rule, the message will be forwarded to SitoSWIFT queue and then it continues its tour to SWIFT.

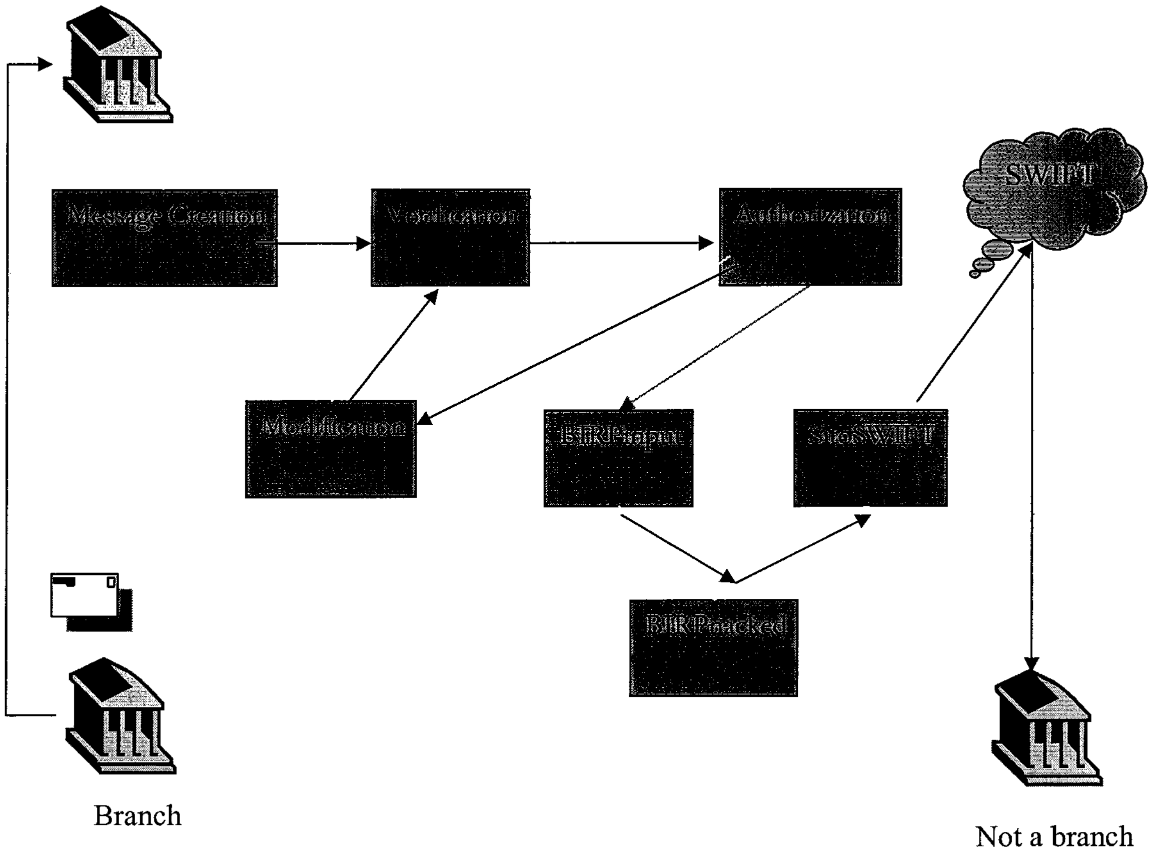


Figure 4.3 Scenario 2 Receiver is not a Branch

Chapter 5

Experimental Results

The BIRP routing protocol has been applied to ECOBANK of AFRICA which is the largest bank in Africa. It has branches in 12 countries. That is why I decided to do a case study on this bank and I have reached the following results according to:

5.1 Cost ($)

A message sent charges 0.25 cents per day. If the ECOBANK sends 9000 messages per day all over the world and around 4000 of them are for their branches in different countries then we calculate the annual costs in order to see how much savings per year.

Annual Message Costs for Every Branch=

4000 messages *0.25=1000EUR per day*317*1.28$=405.760$ saving costs per year

As we can see the increase of branches leads to the increase of saving costs to the bank. In ECOBANK there are 12 countries worldwide so in this case 12*405.760$ which is around 5 million dollars a year is clearly worth to be saved.

5.2 Comparison Between Both Protocols

If we take a look at the expenses in the old protocol we find that the total yearly expenses for every branch connected to SWIFT is as follows:

Maintenance + SAA License + Security Infrastructure = 40,000EUR = 51,200$

Total Yearly Expenses = 51,200$ + 405,760$ messages = 456,960$ for every branch

While the expenses for the new BIRP routing protocol are:

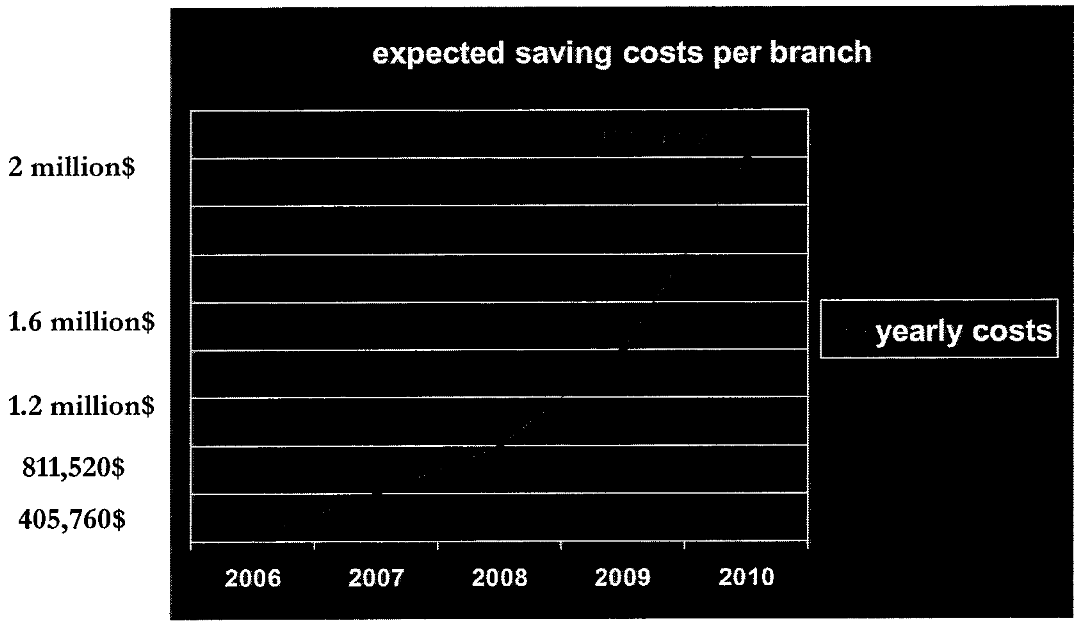
Total Yearly Expenses = 5000 EUR License + 0 Security Infrastructure + 0 (since no more message costs with the new system)

We can see the difference in message costs saved and it is worth to use the new protocol because no security infrastructure is needed since it is already implemented in the main branch.

Statistics projections the next 5 years and how much costs will be saved for the bank.

In the figure below 1,260,000 messages are sent yearly to branches and these cost 405,760$ per year. In 2010, we are expecting the bank to save over 2 million dollars, which is worth a good investment and it corresponds to 6,340,000 messages and this is all for one branch.

According to ECOBANK around 25 million dollars are saved in 5 years for 12 countries. This is an excellent investment for ECOBANK.



1,268,000 messages per year

Saving around 24 million$
in 5 years for 12 branches

Figure 5.1 Comparison between both Protocols

5.3 Delay

The estimated time taken for a message to go to SWIFT then forwarded to the correspondent takes an estimation minimum of 3 to 5 seconds depending to the network traffic and while with BIRP less than 1 second because of real time exchange between branches.

There are several aspects that affect delay:

- Distance.

-Physical medium

In SWIFT protocol the sender is connected to SWIFT through routers, servers, vpn box [14] and the delay is increased with the increase in distance, while there is no such delay in BIRP protocol since the connection is direct.

5.4 Connectivity

It is a good solution for connectivity since branches will not be directly connected to SWIFT, and any problem to SWIFT connectivity, BIRP routing protocol will stay working normally. While if branches are connected to SWIFT then any failure in the connectivity will stop the message processing till the connection comes up again.

As we can see in the below figure 5.2 in case 1 which is the BIRP protocol all branches are connected to the main branch while in case 2 all banks and branches are connected to SWIFT.

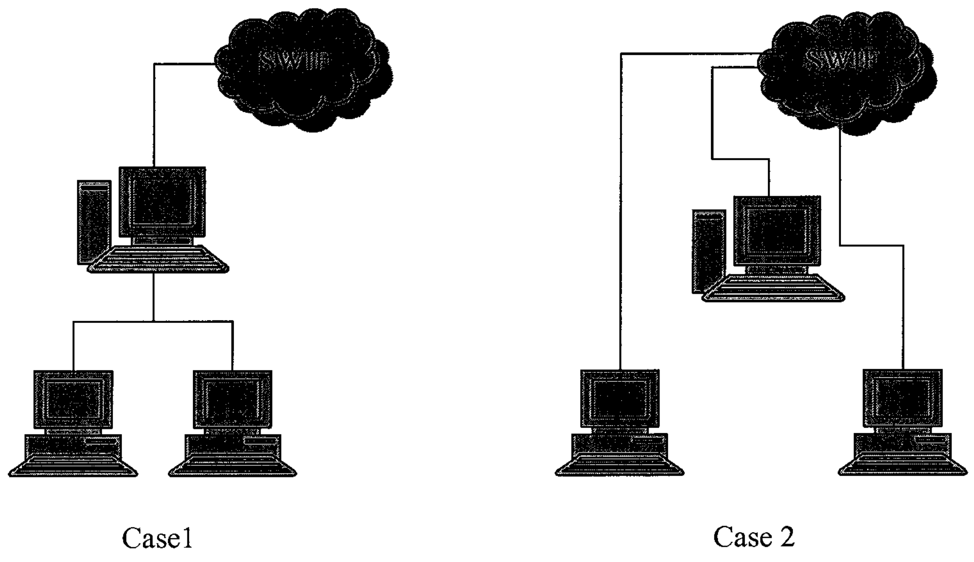


Figure 5.2 BIRP connectivity

Chapter 6

Conclusion

This thesis discusses how to enhance the intra message processing system and this was achieved by the new BIRP routing protocol. The final conclusion of this thesis can be summarized by three main results.

First, the thesis presented the BIRP Bank Intra Routing Protocol which has proved its power in exchanging messages in zero cost and zero security infrastructure so that all messages are intra processed between the main branch and the branch; receiving messages in less delay because of a shorter distance since there is a difference if a message flows through SWIFT or only through the main branch; and providing better connectivity.

Second, the thesis presented an analysis of the new BIRP protocol, its procedure, and how it efficiently works shown in Appenix A.

Third, the thesis presented a case study done on the Ecobank of Africa and I have tried to provide accurate statistics regarding total yearly message costs saved.

Comparing to yearly costs I have found that it is strongly worth to use the BIRP routing protocol and its disadvantage can be overcome since it is fine to increase bandwidth. Banks can find this protocol as an asset for their investment. Finally, we can say that BIRP can be considered an efficient and dynamic real time exchange protocol for intra routing banks.

6.1 Limitations of the Research

Despite the major advantages that BIRP has regarding the costs it can save for the bank, it still has some limitations that need more bandwidth.

In SWIFT protocol each branch is directly connected to SWIFT and the database is distributed. The Bandwidth is 8kb/s and the connection is leased line LL provided by one of these providers (EQUANT, COLT, INFONET, AT&T).

While BIRP needs higher bandwidth, since the database now is centralized to the main branch and the delay will increase because of continuous access to the database and banks almost print all the incoming messages so a huge load will occur in the main system that is why it is better to have either:

Band of 128kb/s, it costs 30EUR/month

Band of 64kb/s, it costs 15EUR/month

Also this protocol is limited only to intra processing between branches while the SWIFT protocol process to all banks.

6.2 Future Work

Many additional features can be added to the new protocol implementation so that it can be a virtual SWIFT routing protocol having all its functionalities, such as letting all messages be intra processed regardless of the receiver, but this is prohibited in SWIFT since amounts that are transferred to other banks must be authorized and verified by the SWIFT institution.

References

[1] SWIFTHistory, SWIFT routing protocol [Online]. Available: http://www.SWIFT.com

[2] SWIFT team Belgium (2004). Getting Started with SWIFT Alliance Access (Release 5.5)
Document version 2.0

[3] SWIFT Team (2005). Alliance Developer Kit ADK (Release 5.5)

[4]DirkBogaert,[Online]. Available http://www.mkssoftware.com

[5]SWIFTAllianceproducts[Online]Available
http://www.perago.com/Products/services/SWIFT.html

[6]SWIFTPartners[Online]Available
http://www.swissrisk.com/partners/index_SWIFT.php

[7] Bic Directory [Online] Available http://www.SWIFT.com/biconline/

[8] SWIFT Team (2005). Bic Directory

[9]BKE Bilateral Key Management [Online]Available
http://public.logicacmg.com/~fsd/Prod_descr/Bke.PDF

[10] SWIFT Team. SWIFT Net Phase 2

[11]MakingiteasiertodobusinesswithSWIFT,[Online]. Available: http://www.SWIFT.com, 11 August 2006.

[12] Dubai bridges the gap between East and West, [Online]. Available: http://www.SWIFT.com, 15 November 2005.

[13] Kumar Parekh. Society for Worldwide Inter bank Financial Telecommunication.

[14] Alwin Thomas and George Kelley.
Cost-Effective VPN-Based Remote Network Connectivity over the Internet

[15] SWIFT Team. BESS Bilateral Key Exchange (BKE)

Appendix A: Implementation of The Bank Intra Routing Protocol

```
/////////////////***************************//////////////////
                                   BIRP
//////////////////***************************//////////////////

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "cifs.h"
#include "rs.h"
#include "mfs.h"
#include "adk_field.h"
#include "pcs.h"
#include "ejs.h"
#include "adk.h"
#include "adk_trace.h"


#define LOG_SUCCESS (1)


int process_running = 1;

////////////////////////////////////////////////////////////////////////

int Search_Receiver(MfsAddressX1String_t name1,FILE *fptr,char *s1[17]){


char buffer1[13];

int i=0,j=0;

int result1,result2;


while(j<17){

    result1=strcmp(name1,s1[j]);


  if(result1==0)

   {         i=1;

   fprintf(fptr,"Sender_BIC_message  %s\n",name1);

  }j++;

  }

return i;}
```

```
//creating an output message subformat

void add(MfsHeader_t *mesg,MfsHeader_t *mesg1,MfsText_t

*text1,MfsInstance_t *inst,MfsInstance_t *inst2,FILE* fptr){

    AdkStatus_t rc;
    MfsMessageDiagnostic_t diagnostic;
    MfsMessageSubFormat_t subformat=ADK_OUTPUT;
    MfsMessageFormatString_t sformat;
    MfsAddressX1String_t sender={0},receiver={0};
    CifsCorrespondentType_t corrtype,correspondent_type;
    bool_t live,partial;
    MfsMessageFormatString_t format;
    MfsMessageTypeString_t type;
    MfsNetworkPriority_t priority;
    MfsSWIFTAddressString_t add,add1,N1,N2;
    MfsNetworkApplicationIndexString_t applindex;
    MfsUnitNameString_t unit;
    MfsRpNameString_t routing;
    MfsInstanceType_t instance_type;
    char i;


    FieldGetValue(mesg,"mesg_type", 0, type,
                       "mesg_is_live", 0, &live,
                       "mesg_is_partial", 0, &partial,
                        "mesg_sender_corr_type", 0, &corrtype,
                        "mesg_sender_X1", 0, sender,
                        "mesg_frmt_name", 0, format,
                        "mesg_network_priority", 0, &priority,
                        "mesg_sender_SWIFT_address", 0, add,
                        "mesg_receiver_SWIFT_address", 0, add1,
                        NULL);


   FieldGetValue(mesg,"mesg_network_appl_ind", 0, applindex, NULL);

i=add[8];
add[8]='X';
add1[8]=i;
/////////////////////////////////////////////////////


   FieldSetValue(mesg1,"mesg_sub_format", 0, &subformat,
                       "mesg_s_umid",0,mesg->mesg_s_umid,
                       "mesg_class",0,&(mesg->mesg_class),
                       "mesg_is_live", 0, &live,
                       "mesg_is_partial", 0, &partial,
                       "mesg_sender_corr_type", 0, &corrtype,
                       "mesg_sender_X1", 0, sender,
                       "mesg_frmt_name", 0, format,
                       "mesg_type", 0, type,
                       "mesg_network_priority", 0, &priority,
                        NULL);

      fprintf(fptr,"\nSENDER_ADDRESS    %s\n",add);
      fprintf(fptr,"  RECEIVER_ADDRESS  %s\n",add1);
    strcpy(routing,inst->inst_rp_name);

    strcpy(unit,inst->inst_unit_name);



    FieldGetValue(inst, "inst_type", 0, &instance_type,
                        "inst_sm2000_priority", 0, &priority,
                        "inst_rp_name", 0, routing,
                        "inst_unit_name", 0,unit,
                        "inst_receiver_X1", 0, receiver,
                         NULL);

    fprintf(fptr,"\nSENDER%s      \n",sender);

    fprintf(fptr,"RECEIVER%s    \n",receiver);


    FieldSetValue(inst2, "inst_s_umid", 0, &(mesg->mesg_s_umid),

                        "inst_type", 0, &instance_type,

                        "inst_sm2000_priority", 0, &priority,

                        "inst_rp_name", 0, routing,
                        "inst_unit_name", 0,unit

                       "inst_receiver_X1", 0, receiver,
                        NULL);


  FieldGetValue(mesg1,"mesg_sender_SWIFT_address", 0, N1,

                      "mesg_receiver_SWIFT_address", 0, N2,NULL);


  rc = MfsAddMessage(mesg1, text1, inst2, NULL, NULL, &diagnostic);


  rc = RsRouteInstance(inst2->inst_s_umid, 0,    "IBRSINPUT",
ADK_R_SUCCESS, 0, 0);
}//end of add






/********************** MAIN ***************************/
 void main(int argc, char *argv[]){

    AdkStatus_t rc,status;
    MfsSumidString_t s_umid;
    int32_t inst_num;
    MfsHeader_t mesg,mesg1;
    MfsText_t text,text1;
    MfsIntervention_t intv;
    MfsInstance_t inst,inst1,inst2;
    bool_t yes=TRUE,RESTARTED;
    MfsInstanceType_t instance_type = ADK_INST_TYPE_NOTIFICATION;
    MfsReceiveDeliveryStatus_t delivery=ADK_EM_DELIVERED;
    MfsPriority_t priority = ADK_PRIORITY_1;

    MfsInstanceType_t inst_type1;
    FILE *fptr,*fptr1;
    int process=1;
    char rout[ADK_RP_NAME_LEN]={0};
    AdkExitStatus_t     exit_status=ADK_SUCCESS_EXIT;
    MfsMessageSubFormat_t subformat=ADK_OUTPUT;
    MfsAppendix_t appe;
    MfsAppendixType_t appe_type = ADK_APPE_RECEPTION;
    MfsAppendixSessionHolderString_t sess_holder;
    MfsAppendixSessionNbrString_t sess_nbr;
      MfsNetworkDeliveryStatus_t deliv_status =
ADK_DLV_ACKED,deliv_status1 =ADK_DLV_NACKED ;

      MfsReceiveDeliveryStatus_t
non=ADK_EM_ABORTED,non2=ADK_EM_DELIVERED;
      MfsAuthResult_t RES=ADK_AUTH_NO_KEY;
      MfsMessageDiagnostic_t diagnostic;
      MfsAddressX1String_t receiver,sender1,receiver1;
      MfsRpNameString_t routing,rou;
      MfsUnitNameString_t unit;
      MfsInstanceStatus_t insta=ADK_COMPLETED;
      MfsAppendixSeqNbrString_t seq_nbr;
      MfsNackReasonString_t ap;

      int Sender11=0,Receiver11=0;
      char temp,*temp1,*temp2,*temp3,*temp4,
      *temp5,*temp6,*temp7,*temp8,*temp9,
      *temp10,*temp11,*temp12,*temp13,*temp14,*temp15,*temp16;
      char *MAINBRANCH[17];



      fptr=fopen("news.txt","w");








    /////////////////////////////////////////////////////////////////////
   temp=(char *)malloc(20*sizeof(char));
   temp1=(char *)malloc(20*sizeof(char));
   temp2=(char *)malloc(20*sizeof(char));
   temp3=(char *)malloc(20*sizeof(char));
   temp4=(char *)malloc(20*sizeof(char));
   temp5=(char *)malloc(20*sizeof(char));
   temp6=(char *)malloc(20*sizeof(char));
   temp7=(char *)malloc(20*sizeof(char));
   temp8=(char *)malloc(20*sizeof(char));
   temp9=(char *)malloc(20*sizeof(char));
   temp10=(char *)malloc(20*sizeof(char));
```

```
temp11=(char *)malloc(20*sizeof(char));
temp12=(char *)malloc(20*sizeof(char));
temp13=(char *)malloc(20*sizeof(char));
temp14=(char *)malloc(20*sizeof(char));
temp15=(char *)malloc(20*sizeof(char));
temp16=(char *)malloc(20*sizeof(char));

//**************************************************

//these are the 12 bic code branches that ends XXX and the others

// are branches inside these countries


strcpy(temp,"ECOCGHACXXX");

strcpy(temp1,"ECOCGHACTMA");

strcpy(temp2,"ECOCGHACTDI");

strcpy(temp3,"ECOCGHACKSI");

strcpy(temp4,"ECOCGNCNXXX");

strcpy(temp5,"ECOCNGLAXXX");

strcpy(temp6,"ECOCNENIXXX");


strcpy(temp7,"ECOCTGTGXXX");

strcpy(temp8,"ECOCTGTGETI");

strcpy(temp9,"ECOCTGTGWPS");

strcpy(temp10,"ECOCBJBJXXX");

strcpy(temp11,"ECOCLRLMXXX");

strcpy(temp12,"ECOCMLBAXXX");

strcpy(temp13,"ECOCSNDAXXX");

strcpy(temp14,"ECOCBFBFXXX");

strcpy(temp15,"ECOCCMCXXXX");

strcpy(temp16,"ECOCCIABXXX");


MAINBRANCH[0]=(char *)malloc((strlen(temp)+1)*sizeof(char));

strcpy(MAINBRANCH[0],temp);

MAINBRANCH[1]=(char *)malloc((strlen(temp1)+1)*sizeof(char));

strcpy(MAINBRANCH[1],temp1);

MAINBRANCH[2]=(char *)malloc((strlen(temp2)+1)*sizeof(char));
```

```
strcpy(MAINBRANCH[2],temp2);

MAINBRANCH[3]=(char *)malloc((strlen(temp3)+1)*sizeof(char));

strcpy(MAINBRANCH[3],temp3);

MAINBRANCH[4]=(char *)malloc((strlen(temp4)+1)*sizeof(char));

strcpy(MAINBRANCH[4],temp4);

MAINBRANCH[5]=(char *)malloc((strlen(temp5)+1)*sizeof(char));

strcpy(MAINBRANCH[5],temp5);

MAINBRANCH[6]=(char *)malloc((strlen(temp6)+1)*sizeof(char));

strcpy(MAINBRANCH[6],temp6);

MAINBRANCH[7]=(char *)malloc((strlen(temp7)+1)*sizeof(char));

strcpy(MAINBRANCH[7],temp7);

MAINBRANCH[8]=(char *)malloc((strlen(temp8)+1)*sizeof(char));

strcpy(MAINBRANCH[8],temp8);

MAINBRANCH[9]=(char *)malloc((strlen(temp9)+1)*sizeof(char));

strcpy(MAINBRANCH[9],temp9);

MAINBRANCH[10]=(char *)malloc((strlen(temp10)+1)*sizeof(char));

strcpy(MAINBRANCH[10],temp10);

MAINBRANCH[11]=(char *)malloc((strlen(temp11)+1)*sizeof(char));


strcpy(MAINBRANCH[11],temp11);

MAINBRANCH[12]=(char *)malloc((strlen(temp12)+1)*sizeof(char));

strcpy(MAINBRANCH[12],temp12);

MAINBRANCH[13]=(char *)malloc((strlen(temp13)+1)*sizeof(char));
strcpy(MAINBRANCH[13],temp13);

MAINBRANCH[14]=(char *)malloc((strlen(temp14)+1)*sizeof(char));
strcpy(MAINBRANCH[14],temp14);

MAINBRANCH[15]=(char *)malloc((strlen(temp15)+1)*sizeof(char));
strcpy(MAINBRANCH[15],temp15);

MAINBRANCH[16]=(char *)malloc((strlen(temp16)+1)*sizeof(char));
strcpy(MAINBRANCH[16],temp16);


          MfsInstanceInit(&inst);

          MfsInstanceInit(&inst1);
```

```
            MfsHeaderInit(&mesg);

            MfsHeaderInit(&mesg1);

            MfsTextInit(&text);

            MfsTextInit(&text1);

            MfsAppendixInit(&appe);

            MfsInstanceInit(&inst2);

            strcpy(rout,"BIRPINPUT");


///message should be reserved in order to read its contents after that

//we free it by unreserved or make the message complete

rc = RsReserveInstanceNext(s_umid, &inst_num,rout,ADK_PRIORITY_1,"");

            strcpy(inst.inst_s_umid,s_umid);

            inst.inst_num=0;

//to read the message

 rc = MfsGetMessage(&mesg, &text, &inst);

///////////////////////////////////////////////////////////////////////

     FieldGetValue(&inst,"inst_receiver_X1", 0, receiver1,
                                NULL);

     FieldGetValue(&mesg,"mesg_sender_X1", 0, sender1,
                                NULL);

      Receiver11=Search_Receiver(receiver1,fptr,MAINBRANCH);


//check if receiver is a branch

       if(Receiver11==1){

        add(&mesg,&mesg1,&text,&inst,&inst2,fptr);

        FieldGetValue(&inst,"inst_receiver_X1", 0, receiver, NULL);

        strcpy(routing,inst.inst_rp_name);
        strcpy(unit,inst.inst_unit_name);

        FieldSetValue(&inst1, "inst_s_umid", 0, &(mesg.mesg_s_umid),
                         "inst_type", 0, &instance_type,
                         "inst_sm2000_priority", 0, &priority,
                         "inst_rp_name", 0, routing,
                         "inst_unit_name", 0,unit,
                        "inst_receiver_X1", 0, receiver,
                         NULL);


   inst1.inst_notification_type=ADK_INST_NOTIFICATION_TRANSMISSION;



   rc = MfsAddInstance(&inst1);

            strcpy (sess_holder, "BIRP");
            strcpy (sess_nbr, "1234");
            strcpy (seq_nbr, "123456");
            FieldSetValue(&appe,
                  "appe_s_umid", 0,mesg.mesg_s_umid,
                  "appe_inst_num", 0, &inst1.inst_num,
                  "appe_iapp_name", 0, "BIRP:BIRP_A",
                  "appe_type", 0, &appe_type,
            "appe_session_holder", 0, sess_holder,
            "appe_session_nbr", 0, sess_nbr,
            "appe_sequence_nbr", 0, seq_nbr,
            "appe_network_delivery_status", 0, &deliv_status,
            "appe_ack_nack_text",0,"Network Delivery Status:NetworkAck",
                  "appe_rcv_delivery_status",0,&non2,
                  NULL);


//ADD ACK notification to the message


       rc = MfsAddAppendix(&appe);

//complete the original message

RsMoveInstance(mesg.mesg_s_umid,inst.inst_num,ADK_ACTION_TYPE_COMPLETE,
routing, ADK_R_SUCCESS);

//route the notification instance to BIRPINPUT

rc = RsRouteInstance(mesg.mesg_s_umid, inst1.inst_num,

"BIRPINPUT",ADK_R_SUCCESS, 0, 0);

        }

// If receiver not a branch create nack appendix in the notification

//instance
```

```
  else{

      FieldGetValue(&inst,"inst_receiver_X1", 0, receiver, NULL);


      strcpy(routing,inst.inst_rp_name);

      strcpy(unit,inst.inst_unit_name);


      FieldSetValue(&inst1, "inst_s_umid", 0, &(mesg.mesg_s_umid),
                       "inst_type", 0, &instance_type,
                       "inst_sm2000_priority", 0, &priority,
                       "inst_rp_name", 0, routing,
                       "inst_unit_name", 0,unit,
                       "inst_receiver_X1", 0, receiver,
                       NULL);

   inst1.inst_notification_type=ADK_INST_NOTIFICATION_TRANSMISSION;

    rc = MfsAddInstance(&inst1);

              strcpy (sess_holder, "BIRP");

              strcpy (sess_nbr, "1234");

              strcpy (seq_nbr, "123456");



FieldSetValue(&appe,
                  "appe_s_umid", 0,mesg.mesg_s_umid,
                  "appe_inst_num", 0, &inst1.inst_num,
                  "appe_iapp_name", 0, "BIRP:BIRP_A",
                  "appe_type", 0, &appe_type,
                  "appe_session_holder", 0, sess_holder,
                  "appe_session_nbr", 0, sess_nbr,
                  "appe_sequence_nbr", 0, seq_nbr,
                  "appe_network_delivery_status", 0, &deliv_status1,
                  "appe_ack_nack_text",0,"NACKED NOT LICENSED",
                  "appe_rcv_delivery_status",0,&non,
                  "appe_auth_result",0,&RES,
                  NULL);

      FieldGetValue(&appe,"appe_nak_reason", 0, ap, NULL);

       rc = MfsAddAppendix(&appe);

RsMoveInstance(mesg.mesg_s_umid,inst.inst_num,ADK_ACTION_TYPE_COMPLETE,
routing, ADK_R_SUCCESS);

RsRouteInstance(mesg.mesg_s_umid,
inst1.inst_num,"BIRPINPUT",ADK_R_SUCCESS,

0, 0);

///////////////////////////////END/////////////////////////////////////
```