TradeWork - A Tourist Based Peer to Peer Services Framework

By

BADAR AHMED 2004-NUST-BIT-233

Project Report in partial fulfillment of the requirements for the award of Bachelor of Information Technology degree

School of Electrical Engineering and Computer Science
National University of Sciences and Technology
Rawalpindi, Pakistan

2008

6
22

TradeWork - A Tourist Based Peer to Peer Services Framework

By

Badar Ahmed

(2004-NUST-BIT-233)

Project Report in partial fulfillment of

the requirements for the award of

Bachelor of Science degree in Information Technology (BIT)

In

School of Electrical Engineering & Computer Science (SEECS)

National University of Sciences and Technology (NUST)

Rawalpindi, Pakistan

(2008)

CERTIFICATE

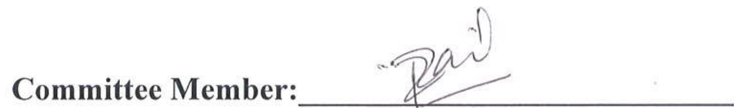
It is certified that the contents and form of thesis entitled "**TradeWork – A Tourist based P2P Services Framework"** submitted by Badar Ahmed have been found satisfactory for the requirement of the degree.

Advisor: ____________________

Director General: Professor Dr. Arshad Ali

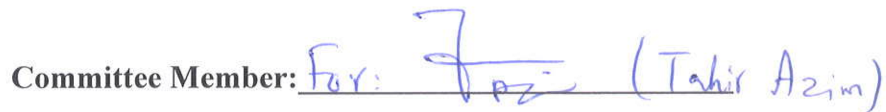
Co-Advisor: ____________________

Faculty Member (Mr. Tahir Azim)



Committee Member: ____________________

Faculty Member (Dr. Raihan-ur-Rasool)



Committee Member: ____________________

Faculty Member (Mr.Naeem Khalid)

DEDICATION

In the name of Allah, the Most Gracious, the Most Merciful.

To my dearest

Father, mother, brothers and sister!!

ACKNOWLEDGEMENTS

First and for all, I am extremely thankful to Allah the Almighty, for completion of this project. I am also thankful to my family members and especially my father who supported all of my expenses and my mother who motivated and prayed throughout the course of the project.

I am thankful to my project supervisor Prof. Dr Arshad Ali, Director General SEECS, for his support, and encouragement during the course of this project. I would like to thank my Co-Advisor Mr. Tahir Azim for providing immense guidance, always keeping me on my toes, and pushing me to work harder.

Also, I would like to pay special thanks to Mr. Faisal Khan, who was the original creator of this framework. Without his persistent help and technical guidance, the completion of this project would have been impossible for me.

TABLE OF CONTENTS

1. INTRODUCTION.....1

1.1 PROBLEM STATEMENT.....2

1.2 CHALLENGES & GOALS.....3

2. LITERATURE REVIEW.....4

2.1 JXTA.....4

2.2 OVERLAY ROUTING ALGORITHMS.....5

2.2.1 Chord.....5

2.2.2 Tapestry.....5

2.2.3 Pastry.....5

2.2.4 Kademlia.....5

2.2.5 Tourist.....6

2.3 STANDARD API FOR STRUCTURED PEER TO PEER OVERLAY NETWORK.....8

2.4 PEER TO PEER OVERLAY NETWORK VISUALIZATION.....8

2.4.1 Network Visualization API's.....9

2.5 OTHER IMPORTANT P2P RESEARCH.....10

2.5.1 Study of Skype Peer to Peer Internet Telephony Protocol.....10

2.5.2 Study of Peer to Peer Network Simulators.....11

2.5.3 Research Issues in P2P Overlays.....12

3. ANALYSIS.....13

3.1 CORE DESIGN.....13

3.1.1 Features of Tourist.....13

3.2 ARCHITECTURE.....15

4. IMPLEMENTATION.....20

4.1 MULTICAST ALGORITHM.....21

4.2 MESSAGE ROUTING 23
4.3 PEER-TO-PEER CHAT APPLICATION 24
4.4 TOOLS USED 25
4.5 CLASS DIAGRAMS 26
4.6 VISUALIZATION TOOL 30
5. RESULTS 32
5.1 UNIT TESTING 32
5.2 FRAMEWORK PERFORMANCE 32
5.3 TEST RUNS 34
6. CONCLUSION 38
7 FUTURE WORK 39
8. APPENDICES 40
8.1 APPLICATION CLASS DOCUMENTATION 40
8.1.1 Public Member Functions 40
8.1.2 Public Data Fields 42
9. REFERENCES 43

LIST OF FIGURES

Figure 1: TradeWork Architecture.....15
Figure 2: Local Event Logging.....18
Figure 3: Remote Event Logging.....18
Figure 4: Pseudo-Code of Multicast Algorithm.....21
Figure 5: Sample Suffix Multicast Tree.....22
Figure 6: Message Routing Algorithm.....23
Figure 7: Peer-to-Peer Chat Application based on TradeWork.....24
Figure 8: Class Diagram Part 1.....26
Figure 9: Class Diagram Part 2.....27
Figure 10: Class Diagram Part 3.....28
Figure 11: Class Diagram Part 4.....29
Figure 12: Visualization of 30 nodes with random levels.....31
Figure 13: TradeWork memory footprint for a single node.....33
Figure 14: TradeWork memory footprint for 30 nodes.....34
Figure 15: Test Run 1 Setup.....35
Figure 16: Test Run 1; 100 Nodes running successfully on 2 physical PCs.....35
Figure 17: Test Run 2 Setup.....36
Figure 18: Test Run 2; 50 Nodes running successfully on 2 physical PCs.....36
Figure 19: Test Run 3 Setup.....37
Figure 20: Test Run 3; 100 Nodes (level 5) on 2 physical PCs showing unsuccessful run 37

LIST OF ABBREVIATIONS & IMPORTANT TERMS

P2P: Peer-to-Peer.

SPOF: Single Point of Failure.

MPOF: Multiple Points of Failure.

Prefix: A prefix of a string $T = t_1 \ldots t_n$ is a string $\hat{T} = t_1 \ldots t_m$, where $m \leq n$

Suffix: A suffix of a string $T = t_1 \ldots t_n$ is a string $\hat{T} = t_{n-m+1} \ldots t_n$, where $m \leq n$ [1]

Chapter 1

INTRODUCTION

Peer-to-peer (P2P) networks work by building an overlay network on top of existing physical infrastructure of networks.

A pure peer-to-peer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. Thus there, salient feature is the decentralized architecture of the overlay network.[2]

Historically, such networks are mainly being used for content sharing, like Napster and KaZaA[3], and highly parallel idle cycle sharing networks, like SETI @HOME[4], LHC @HOME[5] and FightAIDS @HOME[6]. But, recently we saw its wide adaptation as a new computing paradigm, specifically suitable for applications requiring a great deal of scalability, fault tolerance and heterogeneity. Some of the applications leveraging P2P model of computing includes OceanStore[7] – global persistent storage, Tor[8] – network for anonymous virtual routers, MonaLISA[9] – global network monitoring, and many others.

Peer to peer systems are classified into several categories such as:

- Centralized (such as Napster)
- Decentralized (such as KaZaA)
- Hybrid (such as CAN)

- Structured (such as Gnutella)
- Unstructured (such as JXTA)

Each class & type has its own uses & differing architecture which are evident from their naming. Unstructured networks are mostly characterized by there use of flooding for information sharing in case of an event.

In a nutshell, the advantage of peer to peer systems is its MPOF (Multiple Point of Failures) as compared to the SPOF (Single Point of Failures) inherent in most server/client architectures.

In spite of wide adoption of P2P applications there is still a scarcity of generic frameworks to develop custom P2P applications. Existing frameworks like JXTA have their pitfalls as discussed in literature survey.

1.1 PROBLEM STATEMENT

The basic concept is to develop a framework for P2P application developement, which is leightweight, extendable and portable to resource constrained devices.

The aim of the major services to be built on top of the core framework is to utilize characteristics of P2P computing to build a platform, for enabling heterogeneous services to discover and interact with each other using underlying fabric of a structured P2P overlay network.

1.2 CHALLENGES & GOALS

a) A working P2P framework with supporting services like logging and P2P network visualization.

b) Deployment & testing of the basic implementation, so that claims made by Tourist algorithm are tested in a real world environment.

c) An application on top of the TradeWork framework, which utilizes the services offered by the framework.

Chapter 2

LITERATURE REVIEW

Although there are many related P2P systems which provides a heterogeneous structured overly environment but most of them lack feature implementation or are too specific to do a particular task.

2.1 JXTA

JXTA[10] by far provides the most complete implementation of their P2P framework in different languages with Java implementation as being most complete. The main hurdle in using JXTA is its use of heavy weight protocols due to use of XML.

This claim against JXTA is proved by benchmarking results, according to which the overhead in JXTA was specifically mentioned to be the large number of small control messages that caused performance trade-off in due to XML parsing overheads.[11]

Asides from JXTA, there are available implementations of most P2P routing algorithms like Pastry[12] and Chord[13]. But these implementation only provide the barebones functionality for overlay P2P routing, and do not provide a complete framework for P2P based application devlopement.

2.2 OVERLAY ROUTING ALGORITHMS

One of the fundamental problems in the construction of peer to peer overlays, is the overlay routing algorithm. One of the earliest and fundamental algorithm to solve this problem is Chord.

2.2.1 Chord

Chord maps each node to a key. This key is then used as the unique identifier in the overlay node ID space. Chord promises a O(log N) to O($\log^2$ N) hop routing.

2.2.2 Tapestry

Tapestry uses a Plaxton[14] style global mesh network. Plaxton is a data structure used by some overlay algorithms for efficient location of objects. Tapestry employs decentralized randomness to achieve both load distribution and routing locality. It promises O(log N) hop routing. [15]

2.2.3 Pastry

Pastry, like Tapestry, makes use of Plaxton-like prefix routing, to build a self-organizing decentralized overlay network, where each peer routes client requests and interacts with local instances of one or more applications. [16]

2.2.4 Kademlia

One of the key architectural decisions of Kademlia is the use of a novel XOR metric for distance between points in the key space. XOR is symmetric and it allows peers to receive lookup queries from precisely the same distribution of peers contained in their routing tables. [17]

2.2.5 Tourist

Tourist is a self-adaptive structured overlay, which can adapt itself to the changing environment dynamically. It derives most of its features from recent advancements in peer to peer routing algorithms, most notable from Kademlia (with respect to its XOR based distance metric).

On the one hand, Tourist can achieve 1-hop to 2-hop routing in most cases (e.g., in a 1,000,000-node system where nodes' average lifetime is only 1 hour). On the other hand, when the system size is extremely large or the nodes' churn rate is very high, Tourist can always guarantee O(logN)-hop routing for all the messages. Tourist nodes determine their routing table size autonomously nodes with different capacities hold routing tables with different sizes. This enables Tourist to sufficiently utilize all the nodes' allowable bandwidth to achieve as high routing efficiency as possible. Tourist also allows each node to adjust its routing table size dynamically, which is the essential reason for the self-adaptivity.

Tourist is an evolutionary work. It sums up the best features of many different overlay routing algorithms like XOR based distance metric from Kademlia. It is a work that has progressed past the previous mistakes made by researchers.

Hence, these great features promised by Tourist became the reason for choosing Tourist as the overlay routing algorithm in this framework.

2.2.5.1 Working of Tourist

In this sub-section some basic concepts regarding the Tourist routing algorithm are described such as nodeId, levels and XOR-based distance metric, prefix & suffix routing tables.

In a virtual space, each node occupies a point, denoted as *nodeId*. Each message is assigned a destination *key*, which is also a point in the virtual space. A structured

overlay guarantees that any message would be eventually sent to the node whose nodeId is the nearest to the key among all the live nodes.

Tourist nodes are allowed to adjust their levels dynamically to adapt themselves to the changing environment, which results in the self-adaptive routing efficiency from the view of the whole system. For example, when the churn rate rises, keeping original routing tables desires more bandwidth. To guard against bandwidth overloading, nodes will lower their levels, i.e., shrink their routing tables. Since nodes' average routing table size decreases, message routing requires more hops. Totally speaking, it is the flexible adjustment of the individual nodes' routing tables that makes Tourist self-adaptive.

Tourist uses numerical space, that is, every node has a 128-bit identifier *nodeId*. Each message has a destination *key*, also 128-bit long. Tourist uses XOR-based distance instead of commonly used numerical distance, i.e., the distance of two 128-bit numbers, say X and Y, are defined as the value of their XOR result.

Given a message with key k, Tourist guarantees that it would be eventually routed to the node whose nodeId is nearest to k, which is noted as the message's *root node*. Determining which node's nodeId is the "nearest" one depends on the XOR-based distance metric.

Each node has a **self-determined** value *level*, which should be an integer (0, 1, 2…). A node running at level l should keep two symmetric routing tables: a *prefix routing table* and a *suffix routing table*. The former comprises the pointers to all the nodes whose nodeId shares an l-bit common prefix with the local nodeId. Similarly, the latter comprises the pointers to all the nodes whose nodeId shares an l-bit common suffix with the local nodeId. It is easy to see that prefix routing table contains pointers to those nearby nodes in the nodeId space while suffix routing table contains pointers that scatter evenly in the nodeId space.

In Tourist, a pointer consists of the corresponding node's IP address, port number, nodeId, and level. To be convenient, we call the first *l* bits of a level-*l* node's nodeId the node's *prefix eigenstring*, and the last *l* bits its *suffix eigenstring*. The message routing and event multicast in Tourist, operate on the basis of matching of prefix and suffix *eigenstrings*. [18]

2.3 STANDARD API FOR STRUCTURED PEER TO PEER OVERLAY NETWORK

Dabek et al. in his paper, "Towards a Common API for Structure Peer-to-Peer Overlays" present a common API that can be used by structured P2P overlays like, Chord, Pastry and Tapestry. This standardized top level API would abstract the inner workings of different structured P2P algorithms and would provide with a consistent API for the application programmers to work with. [19]

2.4 PEER TO PEER OVERLAY NETWORK VISUALIZATION

Peer-to-peer networks form an overlay network and hence require an overlay visualization which is different from physical network visualization. Many different P2P researchers and developers use different techniques and API's for visualization of the P2P overlay.

Niko Kotilainen et al. in his paper "P2PStudio – Monitoring, Controlling, and Visualization Tool for Peer-to-Peer Networks Research" present an impressive tool called P2PStudio. P2PStudio not only does visualize the P2P overlay, but in fact it also monitors for different network parameters and also allows controlling of the P2P network through one single interface in P2PStudio. [20]

This aspect of P2PStudio has inspired us to make a similar tool, which could perform these three fold tasks of visualizing, monitoring and controlling the P2P network through the same tool.

2.4.1 Network Visualization API's

There is a huge number of different visualization API's, all of which have different pros and cons, and all of which are directed towards different needs. The visualization required in this project is mainly topology of network which would be showed in form of a graph.

Following is a brief review of some of the network visualization considered for this project:

2.4.1.1 GraphViz

AT&T's GraphViz API is very widely used in different kinds of visualization. It has language bindings in many different languages including, C/C++, Java and Python. It is a mature and widely used API. But, the graph algorithms have to be written from scratch. And the visual look and feel is not appealing.[21]

2.4.1.2 Tulip

David Auber's Tulip is a very powerful, visualization API with support for 3D visualization. It is implemented in C++ and is very memory efficient, highly scalable. But comparatively hard to setup and program. [22]

2.4.1.3 Touchgraph

Touchgraph is a Java API for simple visualizations. It is very easy to use. But, it's open source version was last updated in 2002, and it only supports a basic graph layout. [23]

2.4.1.4 GINY

GINY is another Java API, which supports many different graph algorithms & layouts. [24]

2.4.1.5 JUNG (Java Universal Network/Graph Framework)

JUNG is a library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. It supports many different graph algorithms & layouts. Scalability might be an issue with 10,000+ nodes. JUNG has only basic support for trees. [25]

2.4.1.6 Prefuse

Prefuse is another Java library. It produces highly visually attractive visualizations. But is not mature and is still in beta version. [26]

2.4.1.7 Selection

In the end, JUNG was selected for performing P2P network visualization, because of its mature API, large user base, easy to use nature, and good documentation.

2.5 OTHER IMPORTANT P2P RESEARCH

2.5.1 Study of Skype Peer to Peer Internet Telephony Protocol

Baset and Schulzrinne, in their paper by titled, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol" analyze the very widely used Skype software. As Skype's protocols are proprietary, the authors have reverse engineered Skype protocol through memory dumps and network activity loggers. The paper studies the behavior

of the Skype network which appears to be a P2P network with higher level super-nodes that are in fact Skype constructed highly available servers that allow new nodes to bootstrap. It also analyzes Skype's NAT traversal abilities, which allows Skype to work through virtually most of NATs and firewalls. [27]

2.5.2 Study of Peer to Peer Network Simulators

2.5.2.1 OverSim

OverSim is an open-source overlay network simulation framework for the OMNeT++/OMNEST simulation environment. The peer-to-peer simulator contains several models for structured (e.g. Chord, Kademlia, Pastry) and unstructured (e.g. GIA) P2P protocols. [28]

2.5.2.2 PeerSim

PeerSim has been developed with extreme scalability and support for dynamicity in mind. It is composed of two simulation engines, a simplified (cycle-based) one and event driven one. The engines are supported by many simple, extendable, and pluggable components, with a flexible configuration mechanism. [29]

2.5.2.3 ONSP

To improve the overlay network research efficiency ONSP was designed. It is a novel parallel overlay network simulation platform, which provides parallel discrete event simulation of overlay networks on high performance cluster. With this tool, overlay network simulation can be built in large scale easily and also tested in short time. [30]

ONSP was also used by Tourist for generation and verification of its results.

2.5.3 Research Issues in P2P Overlays

2.5.3.1 Location aware P2P network

Peer selection based on physical location of the peer. In this way a peer will always prefer a physically closer peer during the overlay routing. This issue has been addressed and thus implemented in various P2P routing algorithms. [31]

2.5.3.2 Efficient overlay routing based on peer lifespan

Peer selection on basis of peer lifespan, so that peers that have a longer lifespan in the P2P network get preferred treatment during overlay routing. [32]

2.5.3.3 Using P2P discovery in Grid

Grid discovery services can be improved with the use of a de-centralized mechanism of peer to peer discovery. This allows the grid to be more scalable and fault-tolerant in nature. [33]

Chapter 3

ANALYSIS

We were not motivated to use existing peer-to-peer solutions like JXTA for putting together components for our P2P engine. Evaluating the applicability of any P2P system we generally talk about characteristics like scalability, security, heterogeneity, fault tolerance, ubiquity etc. But, mainly, we were interested in having a cross platform, light weight implementation.

3.1 CORE DESIGN

We choose Tourist; which provides us with set of protocols to build a highly scalable and adaptive network of heterogeneous entities. One of the novel feature of the Tourist is its ability to dynamically adopt size of routing table to changing size of network giving 1-hop node discovery with smaller and static network, 2-hop node discovery in a larger but static network and O(log N)-hop node discovery in a very large and dynamic network of peers.

3.1.1 Features of Tourist

3.1.1.1 Adaptability

Tourist provides us with an adaptable routing mechanism based on the number of participating entities and their churn rate.

3.1.1.2 Routing table maintenance - selective tree multi cast

Tourist uses a very sophisticated multi cast technique in which routing table maintenance message is only send to set of nodes, from the whole network, which contains pointer to a node whose change in status (level change, join network, leave network) needs to be update everywhere.

3.1.1.3 Peer overload

For a node with large size of routing table, the associated maintenance cost will be relatively high. As tourist is a heterogeneous protocol, it allows nodes to run at different levels. This concept of levels is not entirely new and used many of other P2P systems like JXTA, FastTrack – used by the Kazaa , Grokster and iMesh file sharing programs, uses concept of super peers (or rendezvous peers in case of JXTA). The novelty of tourist in specifying the super node is that of its ability to dynamically change the role of a peer depending upon its resource usage (mostly bandwidth). A node running at higher levels is responsible for maintaining a very large portion of routing table and responsibilities decreases as the level decreases.

3.2 ARCHITECTURE

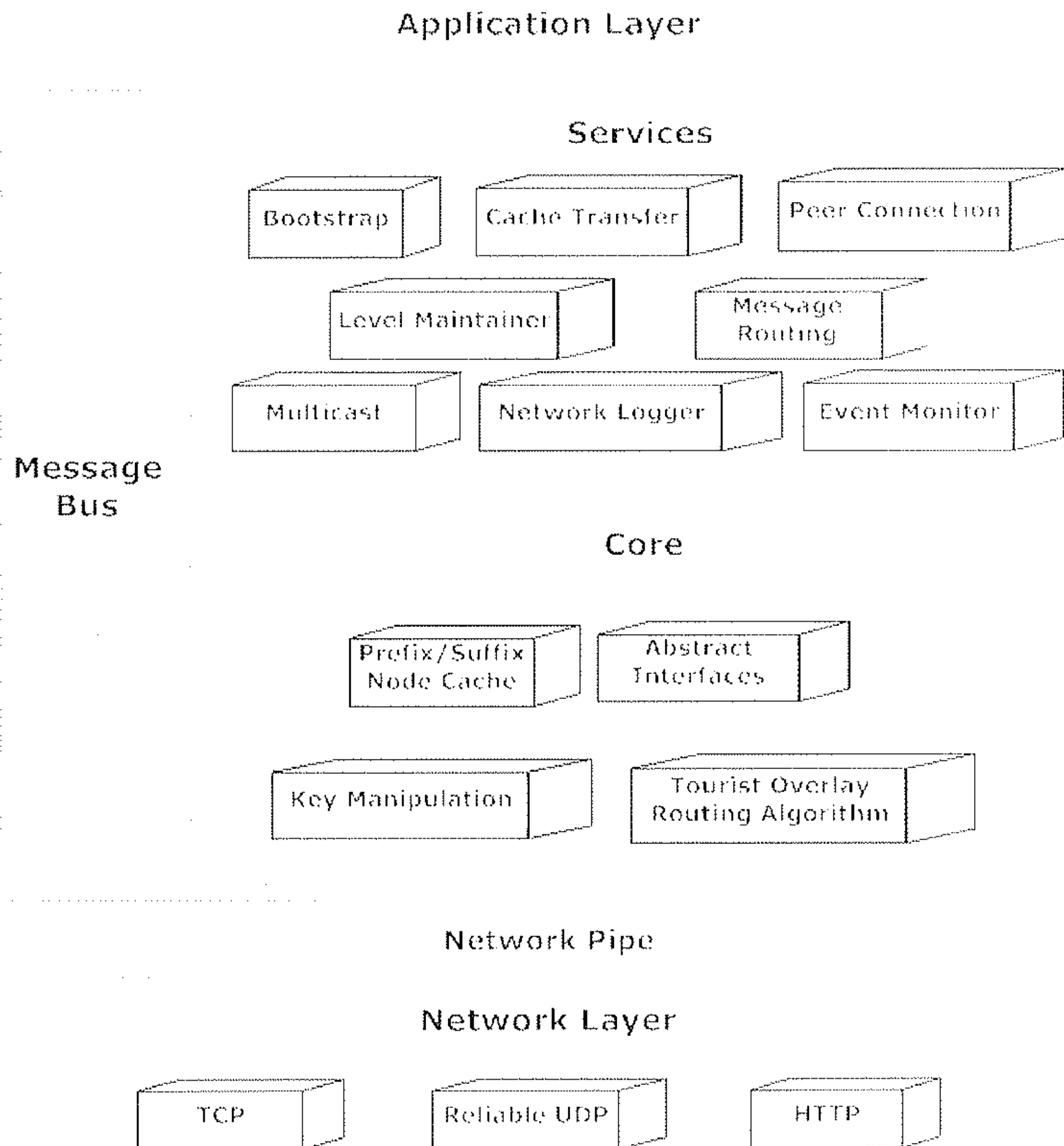


Figure 1: TradeWork Architecture

Many of the architectural components of our first iteration of P2P engine were derived directly from the requirement to implement Tourist's protocols. Here is a brief summary of each of these components and their interaction.

a) **Network layer:** Abstracts away the underlying physical network layer. We use concept of pipes which is very similar to one used in JXTA. Peers exchange messages among each other using these virtual pipes which can be tied to any of the available protocols. Currently supported networking protocols include TCP and reliable UDP – which is simple UDP with incorporation of application level sequencing and acknowledgments. The design supports addition of other transport mechanisms without disrupting the other components.

b) **Core:**

1. **Node Cache:** Maintains the routing table of the current node and, in addition, provide methods for searching, inserting and deleting routing entries, identifying next hops for routing message.

2. **Key Manipulation:** Provides utility methods for creating and manipulating 128-bit node identification. It performs the XOR operations between node IDs, which is essential part of the algorithm.

3. **Tourist Routing Algorithm:** This is the part where the Tourist routing algorithm is implemented. Peer selection for message routing is the responsibility of this module.

4. **Abstract Interfaces:** Interfaces that can be implemented by upper layer services, to extend the Core of the framework.

c) **Message Bus:** A system wide message bus for delivering message received via network hooks from the underlying transport system. Each interested component is required to register with this module in order to receive desired messages.

d) **Services:** The services based architecture is the real strength of our framework. It allows services to be written on top of the core framework implementation.

1. **Level Maintainer:** Responsible for changing the level of current node based on the pre-defined cost variable (currently based on bandwidth). Upon level increase it needs to increase the size of routing table, and vice versa for level decrease.

2. **Cache Transfer:** Enables peers exchange their routing information. This is required as part of bootstrap process and also when a node wants to move to a higher level.

3. **Bootstrap:** Helps a node to boot itself into the network, which involves contacting an arbitrary none which is part of the network and requesting certain nodes which are known to contain pointers this node should download in order to run at a specific virtual position in the network.

4. **Peer Connection:** Establishes connection to a specified peer, and provides a pipe between the two peers, for communication.

5. **Multicast:** A very important feature of the tourist algorithm is its ability to multicast event messages to only a specific set of peers that need their information to be updated. Multicast is performed in a tree based fashion, so that one node is not overloaded with sending event messages to a large number of peers.

6. **Event Monitor:** This module subscribes to event notifications published by the system. For example, when the node starts up, it publishes a node start event. This event is subscribed by the Event Monitor. It then performs necessary event logging on receiving an event notification. This module was designed to serve dual purposes. First, any one using the framework can subscribe to these system events for any purpose. Secondly, we are using this mechanism for event logging particularly. This event log is in turn used by the visualization tool.

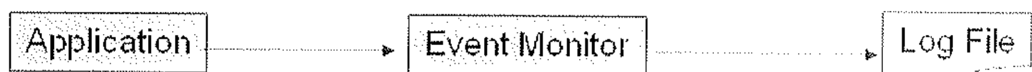


Figure 2: Local Event Logging

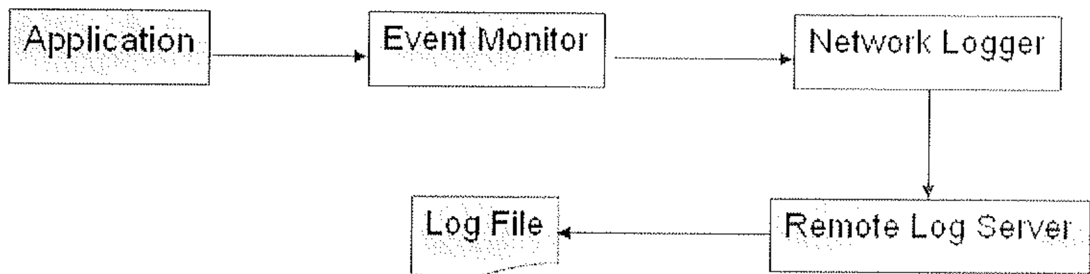


Figure 3: Remote Event Logging

7. **Network Logger:** Network logging facility is a service which enables peers in the system to report event log to a central log server. In this way,

the visualization tool can be run on the central log server which will have event information of all the peers in the system.

8. **Message Routing:** This is one of the main applications of a structured P2P overlay. The message routing module enables delivery of any message, in the form of an object inherited from AbstractMessage class. The API method routeMessage( ) has to be passed the message. The message routing module is always running as a separate thread in the client.

   After a call has been issued to route a message, first of all an appropriate next hop is calculated for the message delivery. Every node that receives this message, in turn calculates a new next hop, and then checks whether it is itself is the target node. In case of itself being the target node, it delivers the message to the application layer. Otherwise, the message is forwarded to the nearest node in ID space. In order to standardize the naming of message routing functions, the convention proposed by Dabek et al. is followed. [18]

e) **Application:** It's where all the different components are gelled together to make one single executable application.

f) **Visualization Tool:** Another very important part of the project which is not shown in the architecture of the framework is the visualization tool. The visualization tool is not part of the framework, but rather is developed so that the Tourist peer-to-peer network can be monitored and visualized effectively. The role of the tool also includes controlling the peer-to-peer network from a single interface.

Chapter 4

IMPLEMENTATION

The complete framework is developed using C++. In order to ensure availability of Tourist across wide variety of platforms we choose portable components, POCO[34], - a BOOST[35] like library, that make it easier to write cross platform code. All of the development and testing has been done on Linux platform.

Message exchange is done using Hessian[36] – a light weight, machine readable, binary and flexible message format. We prefer Hessian over XML because of overhead involved in parsing XML documents especially when we are dealing with exchange of lot of small control messages like heartbeat for checking the availability of a certain peer. As implementations of Hessian are available in wide variety of languages Java, C#, Python, we need not to worry about the computability of Tourist clients written in other languages.

The framework uses a multi-threaded architecture. All of the services layer components are running in separate threads.

Moreover, the core Tourist algorithm – creating node-id, storing nodes, calculating hops, is provided as a separate module in order to adopt it, with minimum effort, for resource constrained devices.

4.1 MULTICAST ALGORITHM

The ability to multicast efficiently, is one of the key features of TradeWork. The multicast algorithm implemented is according to one provided by Tourist.

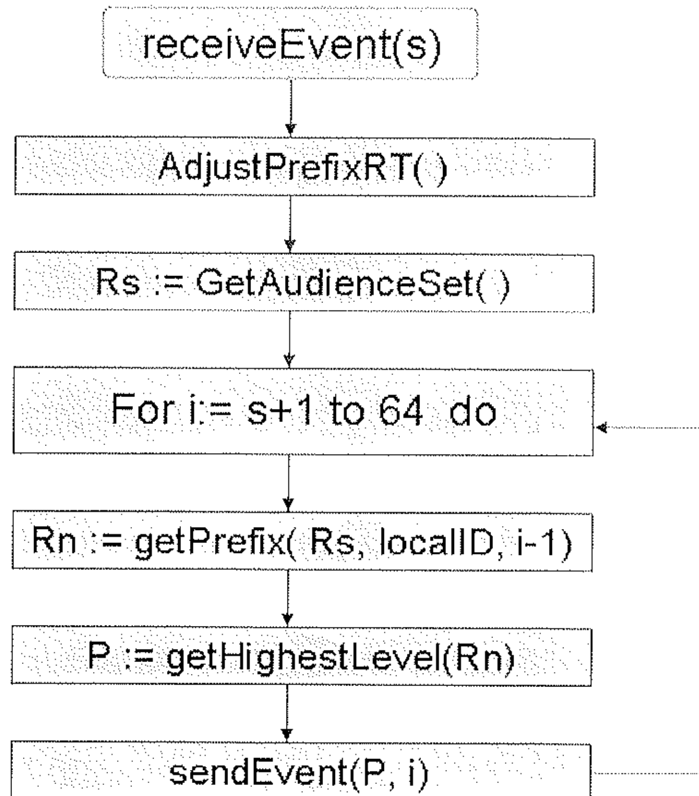


Figure 4: Pseudo-Code of Multicast Algorithm

getPrefix() Functionality:

At every step S, every node that receives the prefix event, should send event to node whose first S-1 (prefix) bits are same, but have different Sth bit. These nodes are audience set nodes.

Audience set nodes, are all those nodes which share the same prefix/suffix eigenstring. The prefix eigenstring for example, is defined as the prefix N bits that are common in the ID of the two nodes. N in this case, is the level of the node.

As a result a multicast tree is constructed, with one node passing the event report to a few others. This efficient way of distributed event notification saves a lot of bandwidth, as compared to unstructured overlay systems in which flooding is used for event notification.

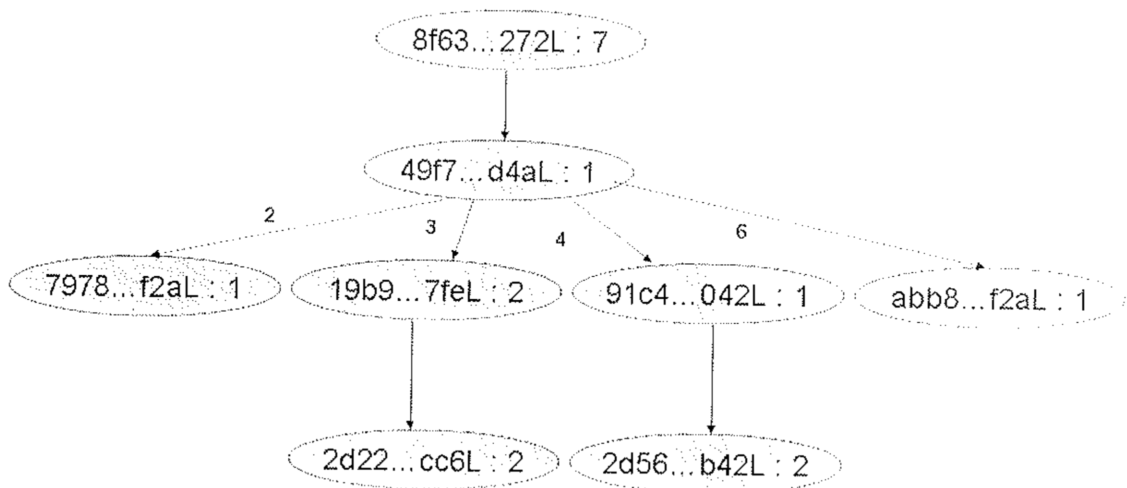


Figure 5: Sample Suffix Multicast Tree

4.2 MESSAGE ROUTING

Message Routing is one of the main features of the TradeWork framework as well as of the Tourist routing protocol.

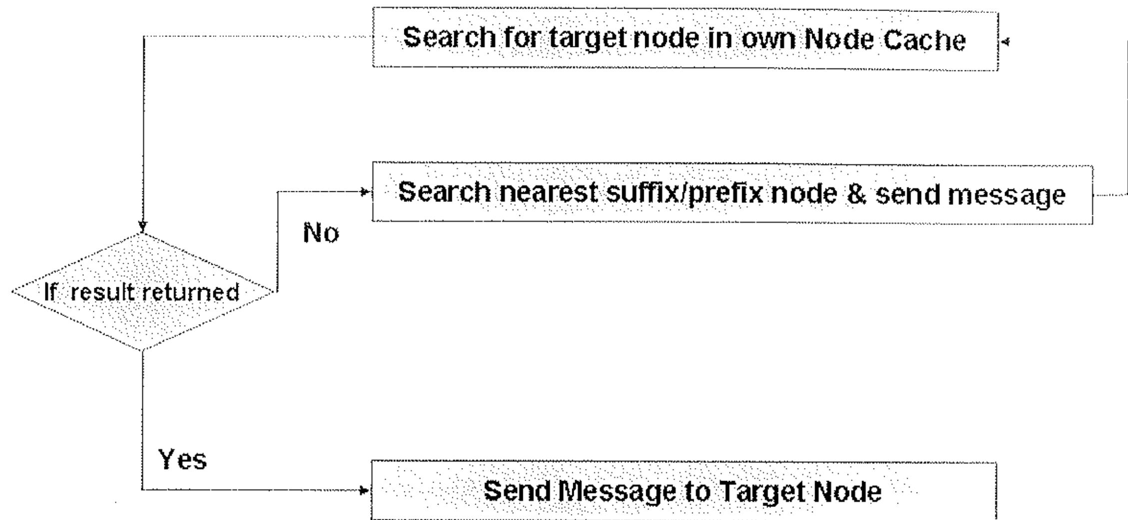


Figure 6: Message Routing Algorithm

Message routing is very simple. When a node receives a message, it:

1) Checks whether itself is the root node of the message. If so, reports it to upper applications; otherwise, turns to 2.

2) If the root node is in prefix routing table, forwards the message to it; otherwise turns to 3.

3) Selects a node from suffix routing table whose prefix routing table must contain the root node and forwards the message to it.

We can see that under this simple routing algorithm, a message`s routing has two possibilities: one-hop, directly via a pointer in prefix routing table or suffix routing table, and two-hop, the first one via a pointer in prefix routing table while the second one via a pointer in suffix routing table.

4.3 PEER-TO-PEER CHAT APPLICATION

A top level application, built on top of the TradeWork was needed which would utilize the framework & the underlying Tourist algorithm. Thus, a P2P chat application was developed, which is the first application to utilize the TradeWork framework. This application was developed in Qt.

Qt is a cross-platform application development framework, widely used for the development of GUI programs. It is widely used in Linux desktop development.

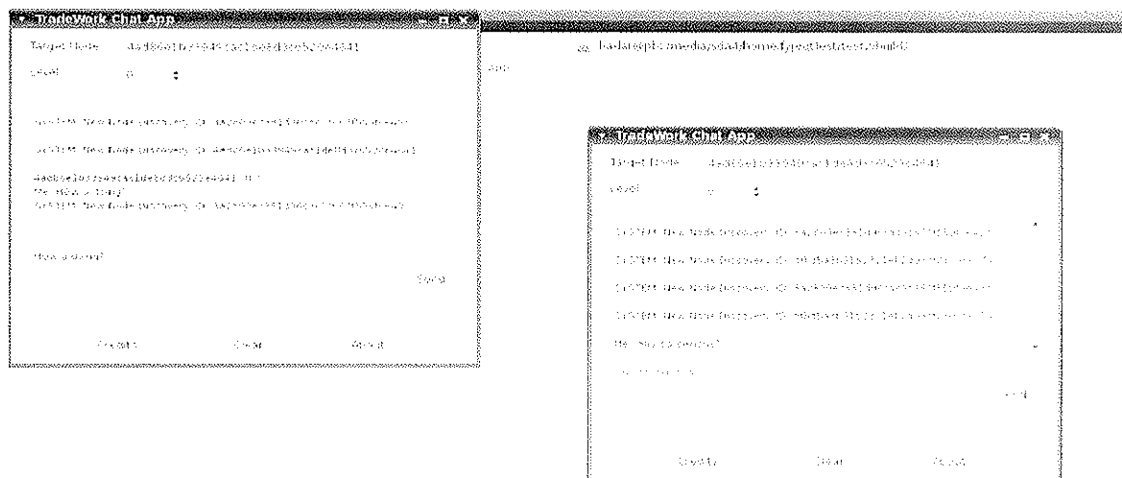


Figure 7: Peer-to-Peer Chat Application based on TradeWork

This P2P chat application allows a peer to send text messages to any other peer on the same P2P network. The destination peer's *nodeId* in the form of hex-string is needed by a peer to send message to any other peer on the network.

Whenever a new node joins the system, the framework multicasts the event report to nodes which share the same *eigenstring*. This event report is then passed on by the event monitoring module up to the chat application. Thus, we have the notion of system messages in the chat application. These system messages are in fact the discovered peers through received multicast messages.

Thus any number of people on the same P2P network using the TradeWork client & running its chat application will be able to exchange text messages with each other.

4.4 TOOLS USED

- CMake: A platform independent build tool. Much easier to use than GNU Make tools. Is used by some of the biggest open source projects, like KDE. [37]
- Assembla: Assembla is a software project management tool. It is used by this project, mostly for its SVN feature. [38]
- SVN: Subversion (SVN) is a version control system. The project is maintained on Assembla's SVN server at: http://svn2.assembla.com/svn/Tourist

4.5 CLASS DIAGRAMS

The class diagrams are split into three parts due to space constraints. In the first two diagrams the class LocalNode is common, whereas in the next two diagrams, the class Application is common.

These class diagrams only represent the major classes. There are a huge number of support classes that are involved in the framework. For brevity's sake, we only present here the most important classes that are the backbone of the framework. A complete list of classes can be seen in the appendices.

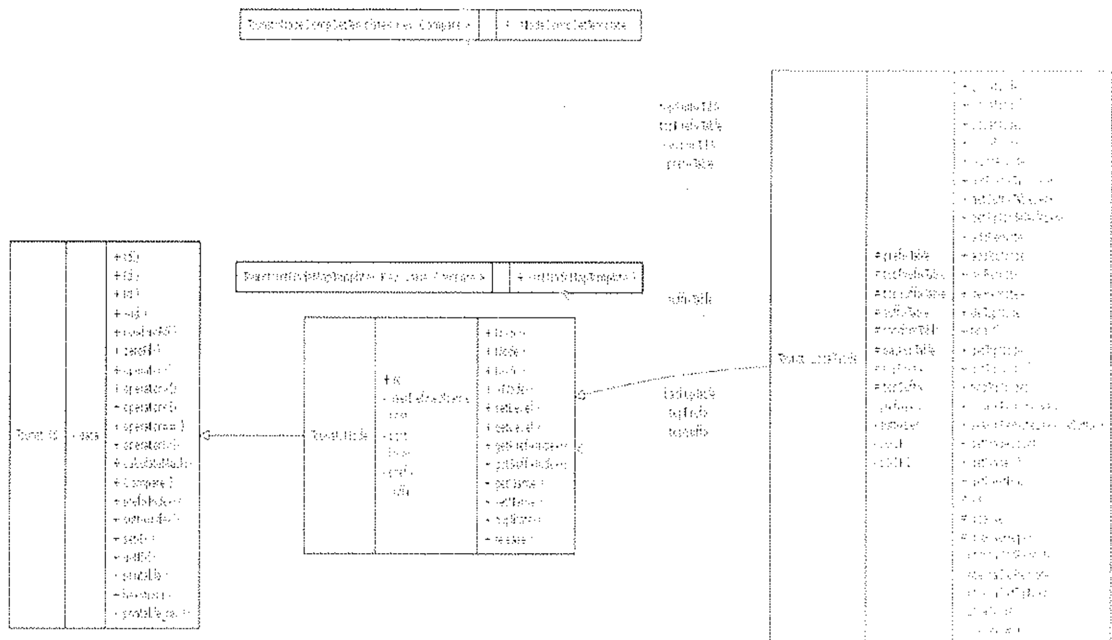


Figure 8: Class Diagram Part 1

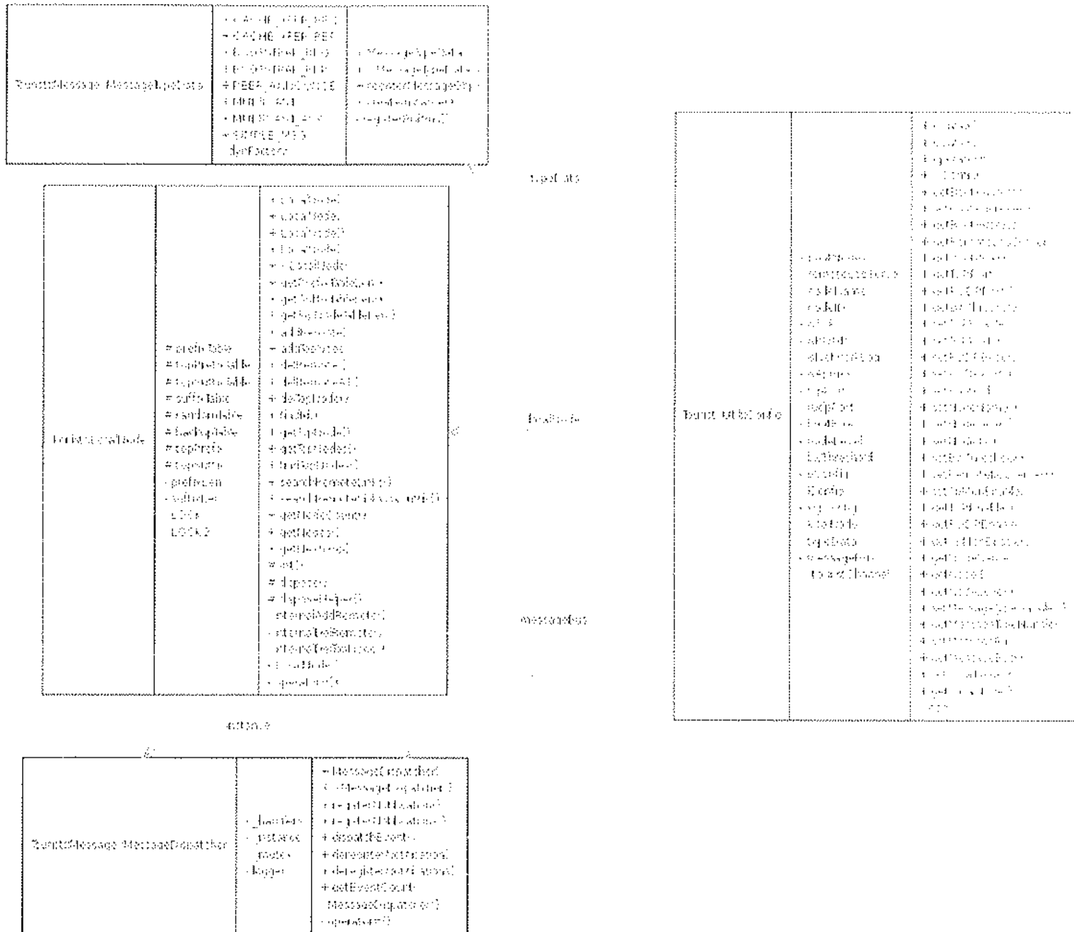


Figure 9: Class Diagram Part 2

Figure 10: Class Diagram Part 3

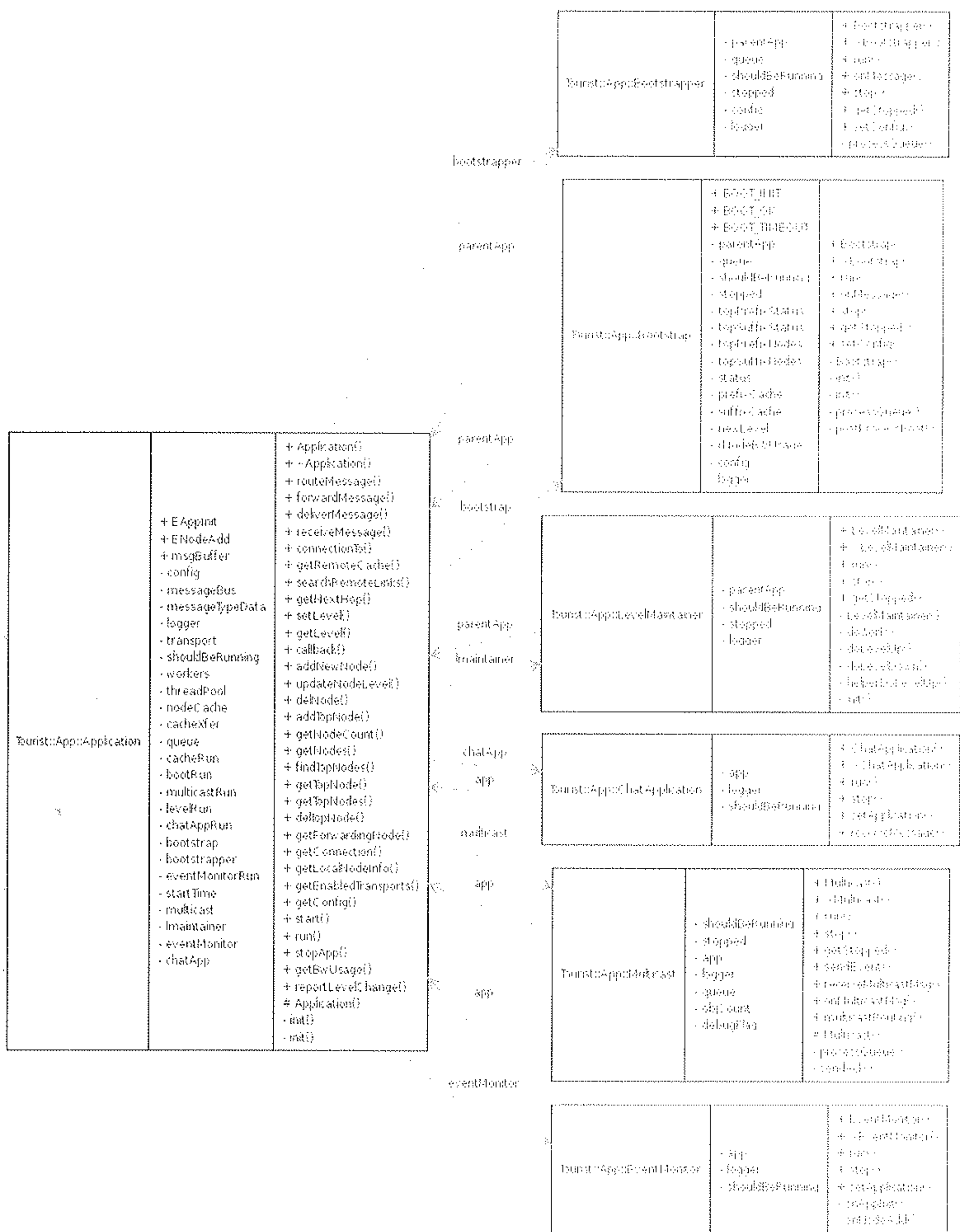


Figure 11: Class Diagram Part 4

4.6 VISUALIZATION TOOL

The TradeWorkViz visualization tool visualizes the topology of the P2P network. This tool is separately implemented using Java programming language, in contrast to the C++ code base of the rest of the framework. This is done because this tool per se is not a part of the TradeWork framework. Secondly, the JUNG API was evaluated as the most feature rich and easy to use API for visualization of this kind.

The tool consists of two parts, a log parser and graph drawer. The log parser parses the log file and draws the topology of the P2P network. The tool allows basic functions like, zooming, panning and selection of nodes. The identity of nodes can be seen in the form of a tool tip.

The levels are represented by different size of nodes (or circle). The higher the level, the bigger is the node size. For example, node at level 0 has the biggest node size in the topology graph.

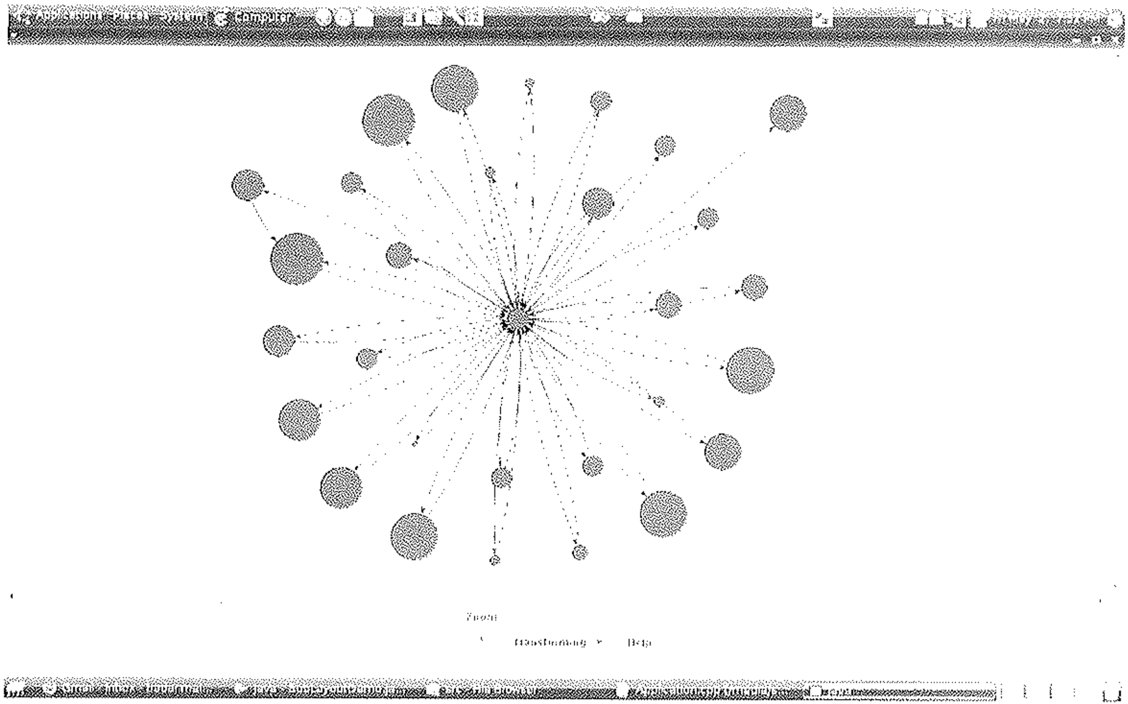


Figure 12: Visualization of 30 nodes with random levels

Chapter 5

RESULTS

5.1 UNIT TESTING

The development of a framework cannot be performed with hit and trial techniques. It requires focused approach towards software testing. Add to that the fact that the testing and debugging of a framework, which is for P2P application development, makes software testing and result gathering much more difficult.

Therefore, the practice of unit testing has been used to very good effect in this project. Unit tests assure that each component of the project is working perfectly fine in itself. Unit tests are written and run for most of the main features of each component, in this project.

The Portable Components (POCO) library's unit testing framework is used, which makes it easy to write and easily get unit tests running. Related test cases are added to a single test suite, which represent a framework class or particular component.

5.2 FRAMEWORK PERFORMANCE

One of the main goals of this framework, set very early on in the project, was it had to be lightweight. Thus, after performing memory consumption analysis of this framework we come to the conclusion that this goal has been fairly achieved. On our tests, TradeWork took memory footprint of 500 – 520 KB.

In contrast, Project JXTA's Java implementation takes up a process size of more than 25 MB. On the other hand, JXTA's pure C implementation called JXTA-C, takes up a process size of more than 150 KB.

Therefore, as you can see that TradeWork's memory performance is far superior to JXTA's Java implementation, but on the other hand is 5 times more than the C implementation of JXTA. The main reason for this is that, JXTA-C has been written from scratch for resource constrained devices, and has been highly optimized for it. Also, JXTA-C does not provide some services which are provided by JXTA's Java implementation.

As far as our framework is concerned, some extra services can be turned off to lower the memory footprint on resource constrained devices.

tourist_client	Sleeping	0	0	7807	524.0 KB

Figure 13: TradeWork memory footprint for a single node

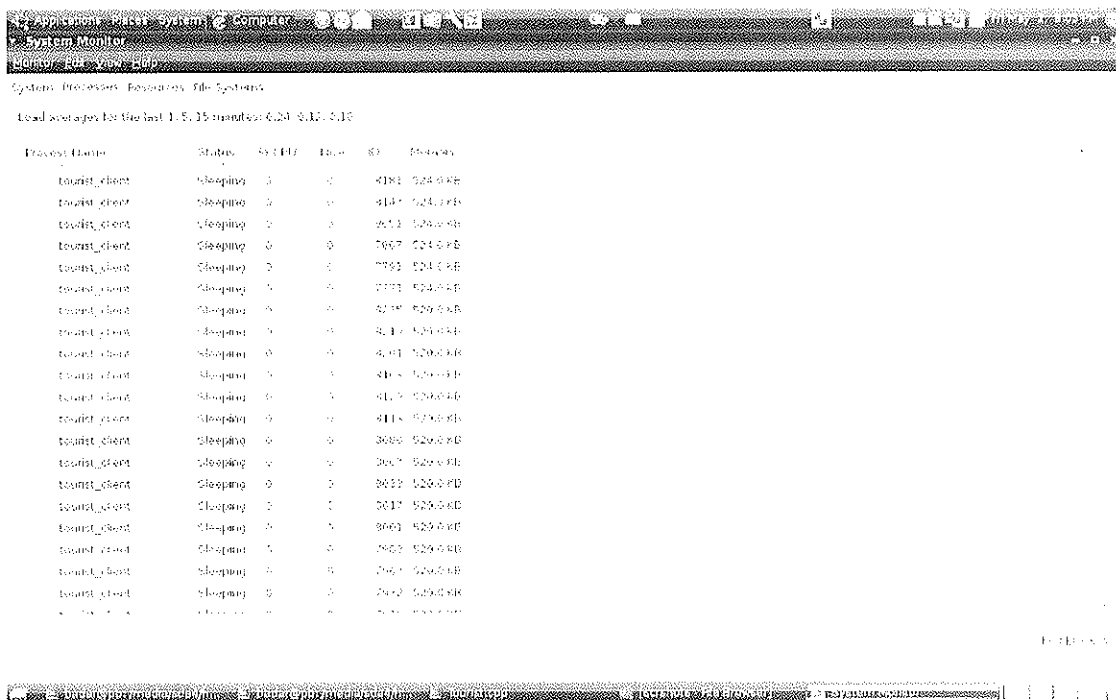


Figure 14: TradeWork memory footprint for 30 nodes

5.3 TEST RUNS

In order to facilitate test runs, a bash script is developed that launches specified numbers of nodes. Because of framework stability and memory management issues, our test runs have gone up to only 100 nodes successfully running and exchanging control messages with each other. It is to be noted that this 100 node system consists of all nodes running on level 0. The mesh like topology in test case 1, can be explained by the fact that all nodes are running on level 0.All nodes running on level 0, have a link to every other node in the system, thus the reason for a mesh like topology.

As you can see below, attempts to run up to 100 nodes, all running on level 5, have failed. The lack of links between nodes clearly illustrates this, in figure 20.

In all the three test cases, one PC was running only the boot node. Where as rest of the nodes were being run on a second PC.

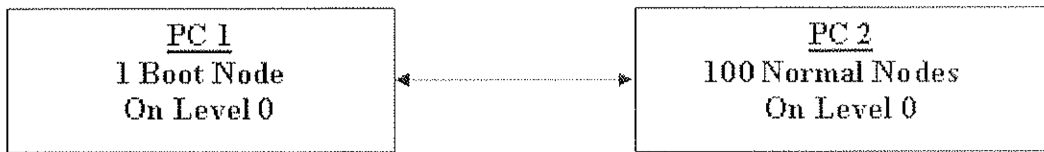


Figure 15: Test Run 1 Setup

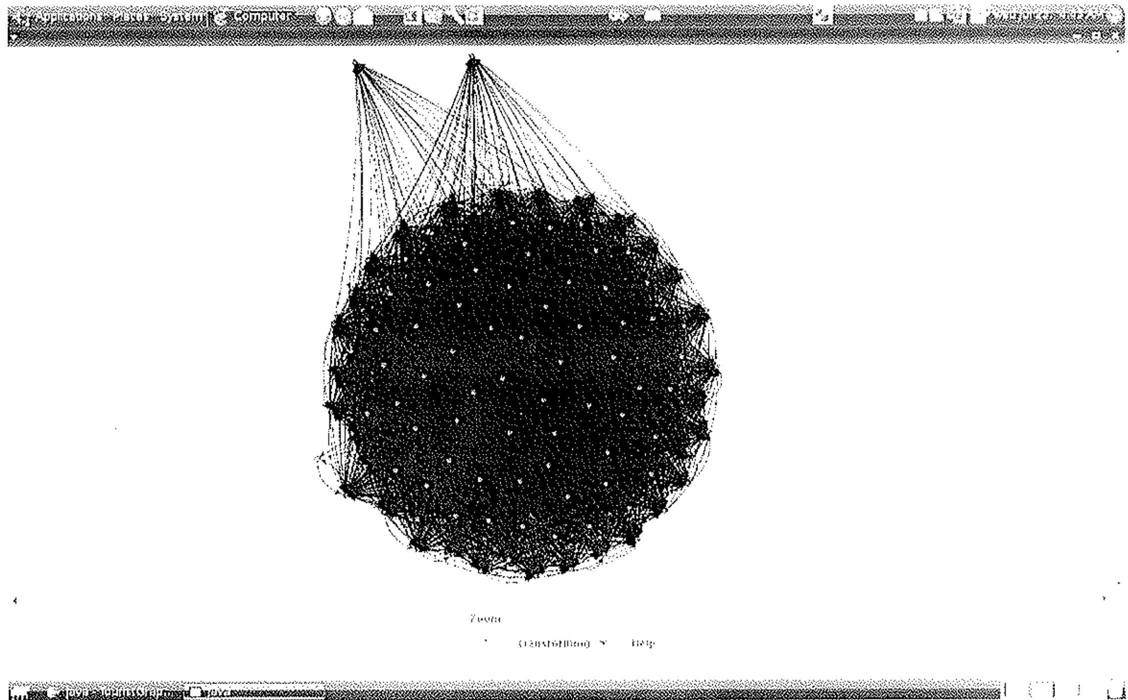


Figure 16: Test Run 1; 100 Nodes running successfully on 2 physical PCs

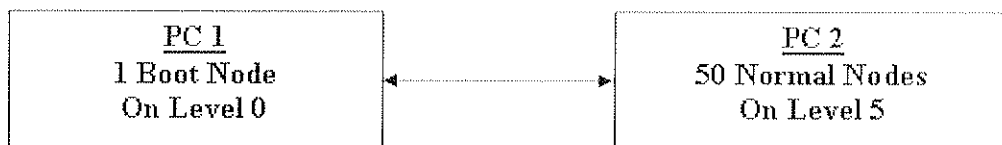


Figure 17: Test Run 2 Setup

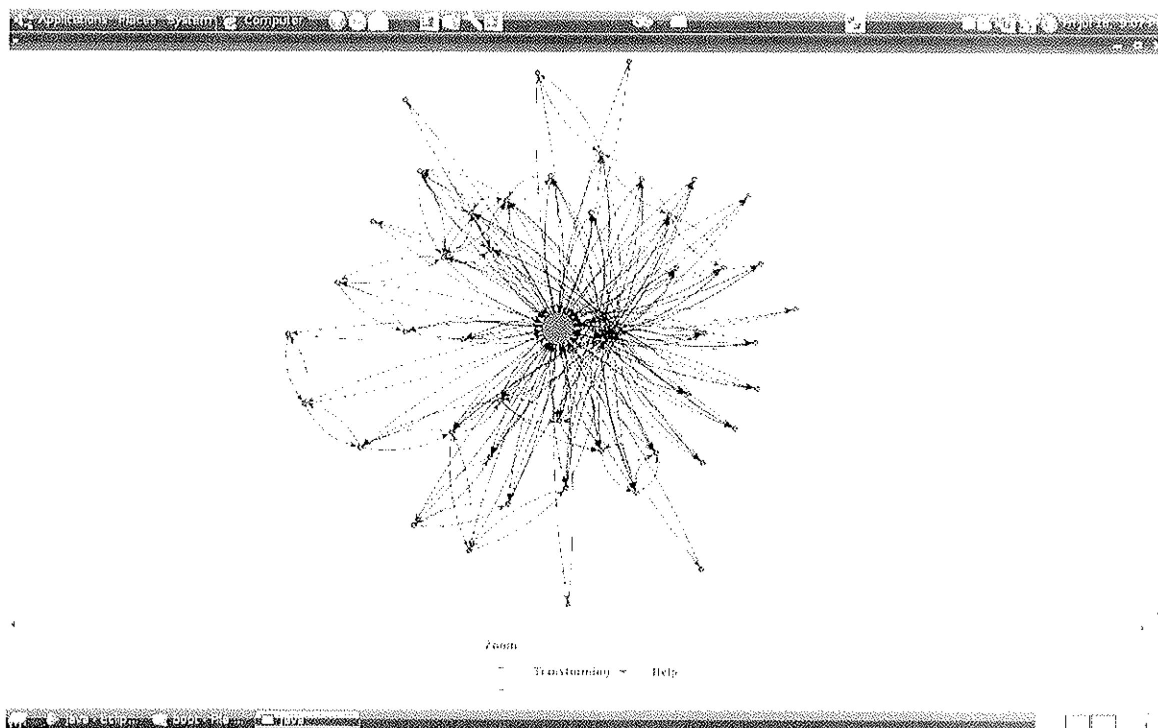


Figure 18: Test Run 2; 50 Nodes running successfully on 2 physical PCs

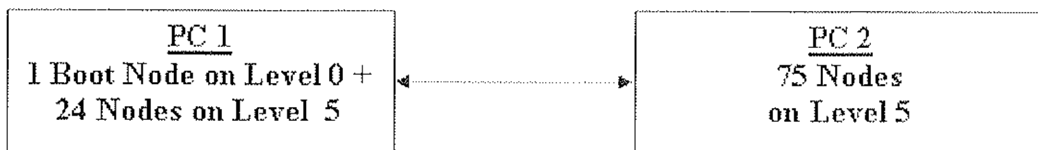


Figure 19: Test Run 3 Setup

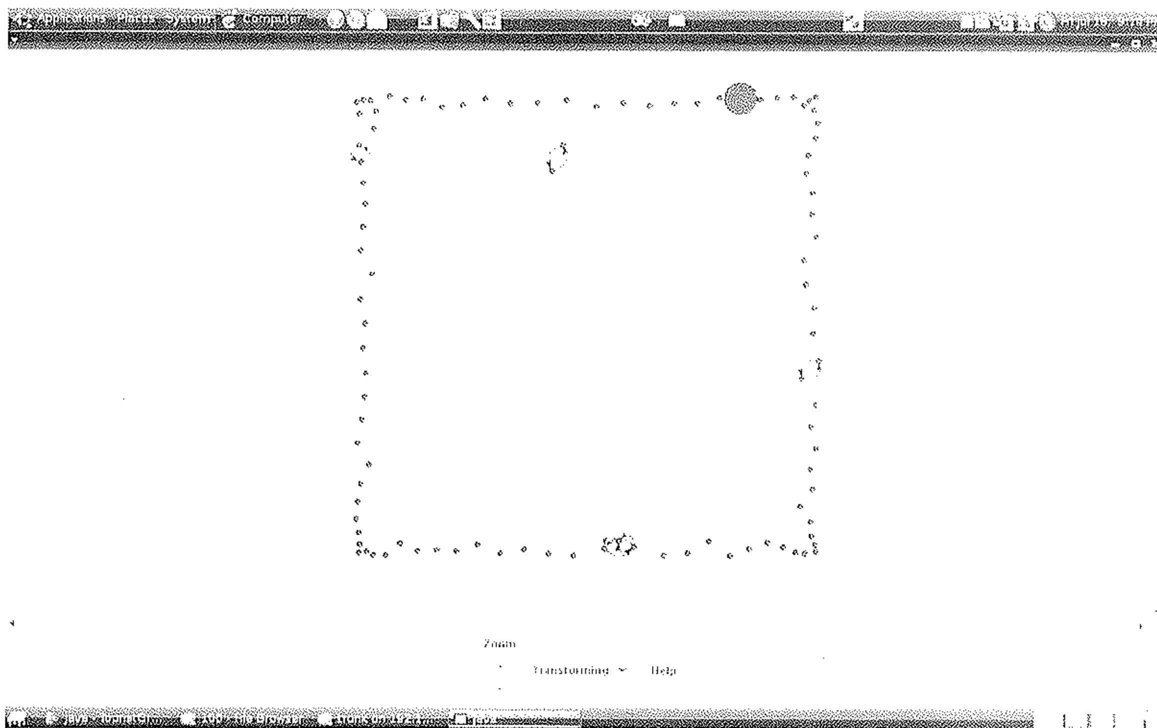


Figure 20: Test Run 3; 100 Nodes (level 5) on 2 physical PCs showing unsuccessful run

Chapter 6

CONCLUSION

TradeWork is still a work in progress. It promises to be a cross platform P2P framework built on top of the Tourist routing algorithm, which itself provides great routing effeciency, in terms of O(1) to O(2) hop routing in most cases.

There have been very few attempts at creating a fully fledged P2P application development framework. Therefore, TradeWork is an effort to create such a framework from ground up, providing with a lightweight and extendable framework in C++.

Chapter 7

FUTURE WORK

In its current state, TradeWork still has a fair amount of bugs, and it not still very stable and easy to use as an API. Hence, vigorous testing and bug fixing is required.

Also extensive performance evaluation needs to be carried out, so that we can compare the framework's routing performance with published results.

Once stability and correctness is achieved, TradeWork will serve as a great platform for P2P research. Up until till now, most of the P2P research is carried out on simulators, which despite being fairly accurate still are simulations. Hence, TradeWork will allow us to do research on a real test bed like Planetlab[39].

There are many research problems in P2P networks today. After the establishment of this test bed for TradeWork, we will look forward to work on these pure research problems. These research problems include next routing hop selection, location aware routing, peer lifespan based routing, P2P security & trust issues.

There is also a future plan to develop interfaces for other languages like Java and Python using either Facebook's Thrift platform [40] or Google's Protobuffers [41]. In this way application developers who prefer to develop in a language other than C++ can also benefit.

Chapter 8

APPENDICES

8.1 APPLICATION CLASS DOCUMENTATION

Application class provides the entry point into the TradeWork framework. It is the API which wraps many lower level functionalities of the framework components.

8.1.1 Public Member Functions

	Application (Tourist::Util::Config &conf)
	virtual **~Application** ( )
void	**forwardMessage** (const AutoPtr< Message::MessageNotification > ¬ification)
void	**deliverMessage** (AbstractMessage *msg)
void	**receiveMessage** ( )
int	**getRemoteCache** (Tourist::Message::RemoteNode *contactNode, int cacheType, int startLevel, int endLevel, int timeout, NodeSet &result)
bool	**searchRemoteLinks** (AutoPtr< Node > recipient, NodeSet &result, unsigned int flags = HIGHER_LEVELS|EQUAL_LEVELS|LOWER_LEVELS|FIND

<table>
<tr><td></td><td>SUFFIXES|FIND_PREFIXES|ONE_LEVEL_LOWER)</td></tr>
<tr><td>bool</td><td>**getNextHop** (NodeSet &audienceSet, AutoPtr< Node > source, AutoPtr< Node > *result, int step, int type)</td></tr>
<tr><td>void</td><td>**setLevel** (int newLevel)</td></tr>
<tr><td>int</td><td>**getLevel** ( )</td></tr>
<tr><td>void</td><td>**callback** (void *pipe)</td></tr>
<tr><td>void</td><td>**addNewNode** (AutoPtr< Tourist::Node > node)</td></tr>
<tr><td>void</td><td>**updateNodeLevel** (AutoPtr< Tourist::Node > query, int newLevel)</td></tr>
<tr><td>bool</td><td>**delNode** (AutoPtr< Tourist::Node > node)</td></tr>
<tr><td>int</td><td>**addTopNode** (AutoPtr< Tourist::Node > node, int type=PREFIX)</td></tr>
<tr><td>int</td><td>**getNodeCount** (int level, unsigned int flags=PREFIX|SUFFIX)</td></tr>
<tr><td>int</td><td>**getNodes** (int level, NodeSet &result, unsigned int flags=PREFIX|SUFFIX)</td></tr>
<tr><td>bool</td><td>**findTopNodes** (AutoPtr< Tourist::Node > query, NodeSet &result, const int type)</td></tr>
<tr><td>bool</td><td>**getTopNode** (int type, AutoPtr< Tourist::Node > *result)</td></tr>
<tr><td>bool</td><td>**getTopNodes** (int type, NodeSet &result)</td></tr>
<tr><td>bool</td><td>**delTopNode** (const AutoPtr< Node > topNode, int type)</td></tr>
<tr><td>bool</td><td>**getForwardingNode** (AutoPtr< Tourist::Node > query, AutoPtr< Tourist::Node > *result, int type)</td></tr>
</table>

bool	**getConnection** (AutoPtr< Tourist::Node > query, AutoPtr< Tourist::Node > *rConnection)
void	**getLocalNodeInfo** (Tourist::Message::RemoteNode &rn)
Tourist::Util::Config *	**getConfig** ( )
std::vector <TransportInfo>	**getEnabledTransports** ()
int	**start** ()
void	**run** ( )
void	**stopApp** ( )
int	**getBwUsage** ( )
int	**reportLevelChange** (int newLevel, int cacheType)

8.1.2 Public Data Fields

Poco::BasicEvent< int >	EAppInit
Poco::BasicEvent < std::pair <int, std::string> >	ENodeAdd

Chapter 9

REFERENCES

[1] Wikipedia: Substring
http://en.wikipedia.org/wiki/Substring

[2] Wikipedia: Peer-to-Peer
http://en.wikipedia.org/wiki/Peer-to-peer

[3] Sharman Network LTD. KaZaA Media Desktop, 2001.
http://www.kazaa.com/.

[4] SETI@ HOME
http://setiathome.berkeley.edu/

[5] LHC@ HOME
http://lhcathome.cern.ch/lhcathome/

[6] FightAIDS@ HOME
http://fightaidsathome.scripps.edu/

[7] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao (2000), OceanStore: An Architecture for Global-Scale Persistent Storage.

[8] TOR (The Onion Router)
http://tor.eff.org/overview.html.en

[9] I. Legrand et al. (2004), MonALISA: an Agent Based, Dynamic Service System to Monitor, Control and Optimize Grid Based Applications. CHEP, Interlaken, Switzerland.

[10] Project JXTA Home Page
http://www.jxta.org/

[11] Halepovic, E.; Deters, R. (2003), The costs of using JXTA, Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference, Volume , Issue , 1-3 Page(s): 160 – 167

[12] Pastry – A substrate for peer-to-peer applications.
http://research.microsoft.com/~antr/Pastry/

[13] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan (2003), Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, IEEE/ACM Transactions on Networking. Vol. 11, No. 1, pp. 17-32.

[14] A. Goal, H. Zhang, and R. Govindan (2003), Incrementally improving lookup latency in distributed hash table systems. In ACM Sigmetrics.

[15] Ben Zhao, John Kubiatowicz, and Anthony Joseph (2001), Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley.

[16] A. Rowstron, P. Druschel (2001), Pastry: Scalable, Decentralized Object Location and Routing for LargeScale Peer-to-Peer Systems, 18 Conference on Distributed Systems Platforms, Heidelberg (D).

[17] P. Maymounkov and D. Mazieres. Kademlia(2002), A peer-to-peer information system based on the XOR metric. In Proceedings of IPTPS02, Cambridge, USA.

[18] Huanan Zhang, Jinfeng Hu, Chunhui Hong, Dongsheng Wang (2007), Tourist: Self-Adaptive Structured Overlay. icn, p. 30, Sixth International Conference on Networking (ICN'07).

[19] Dabek, F., Zhao, B., Druschel, P., Stoica, I. (2003), Towards a common API for structured peer-to-peer overlays. 2nd International Workshop on Peer-to-Peer Systems.

[20] Kotilainen, N., Vapa, M., Auvinen, A., Weber, M., and Vuori, J. (2006), P2PStudio: monitoring, controlling and visualization tool for peer-to-peer networks research. Proceedings of the ACM international Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks.

[21] GraphViz
http://www.graphviz.org/

[22] Tulip
http://tulip.labri.fr/

[23] Touchgraph
http://touchgraph.sourceforge.net/

[24] GINY
http://csbi.sourceforge.net/

[25] JUNG (Java Universal Network/Graph Framework)
http://jung.sourceforge.net/

[26] Prefuse
http://prefuse.org/

[27] Salman A. Baset and Henning Schulzrinne (2006), An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. IEEE Infocom 2006.

[28] I Baumgart, B Heep, S Krause (2007). OverSim: A Flexible Overlay Network Simulation Framework. IEEE Global Internet Symposium.

[29] PeerSim – P2P network Simulator.
http://peersim.sourceforge.net

[30] Yinghui Wu, Ming Li, Weimin Zheng (2004): ONSP: Parallel Overlay Network Simulation Platform.

[31] Y Liu, X Liu, L Xiao, LM Ni, X Zhang (2004), Location-Aware Topology Matching in P2P Systems. Proceedings of IEEE INFOCOM.

[32] Bustamante, F., and Qiao, Y. (2003), Friendships that last: peer lifespan and its role in p2p protocols. In International Workshop on Web Content Caching and Distribution.

[33] Eddy Caron, Frédéric Desprez and Cédric Tedeschi (2007), Enhancing Computational Grids with Peer-to-Peer Technology for Large Scale Service Discovery.

[34] POCO (POrtable COmponents).
http://www.appinf.com/poco/

[35] BOOST C++ Libraries.
http://www.boost.org

[36] Hessian – Binary web-service protocol.
http://hessian.caucho.com

[37] CMake – Cross-platform compilation tool.
http://www.cmake.org/

[38] Assembla – SVN/Trac hosting service.
http://www.assembla.com/

[39] Planetlab – Distributed Overlay testbed.
http://www.planet-lab.org/

[40] Facebook Thrift.
http://developers.facebook.com/thrift/

[41] Google ProtoBuffers.
http://code.google.com/apis/protocolbuffers/docs/overview.html

PR. 722
378.242
28.11.2008
BAD
BIT 6
NIIT LIBRARY