University of Twente
department of computer science

Centrum voor Wiskunde en Informatica
INS1 (databases and data mining)

Design of Peer-to-Peer Protocol for AmbientDB

Brahmananda Sapkota

Thesis for a Master of Science Degree in Telematics
from the University of Twente,
Enschede, the Netherlands

Graduation Committee:

dr. P. A. Boncz
dr. ir. D. A. C. Quartel
ir. R. van de Meent

Enschede, the Netherlands
August, 2003

Acknowledgments

I am indebted to a number of people for their enormous support in the completion of my master's thesis.

I am very grateful to Dr. Peter Boncz for supervising me during the entire length of the project, and for providing an opportunity to carry out this master's thesis at CWI. His support through review and correction in the completion of this thesis report will be a guide to me in constructing a scientific documentation in my future endeavors.

I would like to express my gratitude to the members of my graduation committee, namely Dr. Ir. Dick Quartel, and Ir. Remco van de Meent for their constructive feedback and continuous guidance not only in the completion of this research, but also in helping me formulate a well structured documentation of my research. A well appreciated thanks for your time and energy in reviewing and correcting my report.

I would like to express my sincere gratitude to my parents, for encouraging me towards the completion of my masters program, and for guiding me in shaping the direction I chose in life. My brothers are equally thankful for their suggestions and for carrying me through the difficult times of filled with confusions and frustrations. I am also thankful to my sisters for their continuous love and cares, making me feel closer to home even when I am miles away.

I would like to thank Belinda Knol from UT for helping me with administration, and to Matthijs Mourits from CWI for helping me with software installations and fixing machine bugs.

Many thanks go to my friends: Hong (Jimmie) Chen, Katarzyna Wac, Larassetyo Wibowo, G. Mamo Alemu, Maneesh Khattri, Sachendra Shrestha, Rameshwor Shrestha, Gaurav Pradhan and Xiaobo He for their invaluable encouragement and their support during the hard times. Thank you all for being good friend.

Abstract

This thesis describes the result of a Masters of Science assignment at Centrum voor Wiskunde en Informatica (CWI). This assignment has been carried out at CWI located in Amsterdam in cooperation with the Architecture of Distributed Systems group of the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) of the University of Twente.

In this master thesis, a P2P protocol is designed for AmbientDB. AmbientDB is an unified data management framework that aims to provide query processing functionalities for ad-hoc network of consumer electronic devices [22]. The ever increasingly 'intelligent' devices can talk to each other when they are in each other's neighborhood. The P2P network enabled by this project envisions a networking protocol that extends the pure Gnutella protocol. The main problem with the Gnutella protocol is scalability because of query flooding. As each Gnutella query has to visit every node in the network, the number of queries in the network increases with increasing network size. Therefore, as the network size increases, the query rate per node increases until it is limited by node resources, usually network bandwidth. To avoid this problem Gnutella uses a fixed TTL (Time To Live) and delimit the network search diameter. This implies that the query answers located at nodes farther than TTL hops can not be found, which leads to poor recall. In AmbientDB, we want to provide better recall.

The main idea of the AmbientDB P2P networking protocol (Adb/NP), designed in this research work, is to create a 'good' overlay network of participating nodes and minimize the average query response time. A 'good' overlay network created by this protocol somehow resembles the underlying physical network and reduces possible bottlenecks. Also, the protocol defines the roles of the participating nodes in the network dynamically. The nodes with more resources allow the nodes with fewer resources to transfer their data. The basic idea behind transferring the data from lower resource nodes to the higher resource nodes is to reduce the number of nodes that receive and process the queries.

We evaluate our Adb/NP protocol by simulating it in the ns-2 network simulator. We compare the performance of this protocol against that of the pure Gnutella protocol.

Keywords: P2P systems, overlay network, super-node, simulation, performance, query cost.

Table of contents

Table of figures I
List of Abbreviations II
1. Introduction 1
1.1 P2P Systems 1
1.1.1 Types of P2P Systems 1
1.2 Motivation 2
1.3 Context 2
1.4 AmbientDB 3
1.4.1 AmbientDB Goals 3
1.4.2 AmbientDB Assumptions 3
1.4.3 AmbientDB Architecture 3
1.5 Objectives 4
1.6 Approach 5
2. Related Work 6
2.1 Overlay Network 6
2.2 Review of Existing P2P Architectures 6
2.2.1 Napster 7
2.2.2 Gnutella 7
2.2.3 KaZaA 8
2.2.4 FreeNet 9
2.2.5 CAN 10
2.2.6 Chord 11
2.2.7 OceanStore 12
2.2.8 Comparative Analysis of Existing P2P Architectures 13
2.3 P2P Databases 14
3. Problem Definition 16
3.1 Problem Statement 16
3.1.1 Goals 16
3.1.2 Assumptions 17
3.2 AmbientDB Network Structure 18
3.2.1 Node Structure 19
3.2.2 Query Processing 19
3.3 Formal Model 20
4. Protocol Design 22
4.1 Joining the AmbientDB Overlay 22
4.1.1 Super-node Selection 22
4.1.2 Nearest Neighbor Selection 23
4.2 AmbientDB Service 24
4.2.1 Service User 25
4.2.2 Service Definition 25
4.2.3 Usage Scenario 27
4.2.4 Service Behavior 28
4.3 AmbientDB P2P Protocol 30

4.3.1 Low Level Service 30
4.4 Protocol Functions 31
4.4.1 Participation Administration 31
4.4.2 Query (Message) Exchange 31
4.5 Protocol Data Unit 32
4.5.1 PDU Types 32
4.5.2 Protocol Behavior 35
4.5.3 Error Situations 37
4.5.4 Complete Behavior 38
4.6 Addressing and Initialization 38
5. Simulation 40
5.1 Simulation Goals, Assumptions and Requirements 40
5.2 Simulation Environment 41
5.3 Simulation Strategy 41
5.3.1 Topology Creation 42
5.3.2 Node Initialization 42
5.3.3 Message Exchange 42
5.4 Simulation Setup 42
6. Evaluation 44
6.1 Cost Metrics 44
6.2 Benchmark Parameters 44
6.3 Simulation 45
6.3.1 Uniform data distribution 45
6.3.2 Increasing heterogeneity 47
7. Conclusion and Future Work 49
7.1 Conclusion 49
7.2 Future Work 49
8. References 50
9. Appendix I 52
10. Appendix II 59
10.1 Network Simulator-2 (ns-2) 59
10.1.1 Ns-2 Node 60
10.1.2 Ns-2 Link 61
10.1.3 Ns-2 Agent 62
10.1.4 Ns-2 Header 62
10.1.5 Otcl Library 62
10.2 Ns-2 extension for AmbientDB 62
10.2.1 C++ extension 62
10.2.2 Otcl extension: ns-lib.tcl 63
10.2.3 New Header Type 64

Table of figures

Figure 1: Example pure and hybrid P2P systems 2
Figure 2: AmbientDB Architecture 4
Figure 3: An example heterogeneous network of users 4
Figure 4: An example overlay network 6
Figure 5: Napster system 7
Figure 6: Gnutella System 8
Figure 7: KaZaA System 9
Figure 8: Freenet request sequence 10
Figure 9: CAN ID-Space (2-d) with 5 (left) and 7 nodes (right) 11
Figure 10: Chord circular ID space and routing 12
Figure 11: Query processing in OceanStore 13
Figure 12: AmbientDB Network Structure with two independent nodes 18
Figure 13: Query processing scenario 19
Figure 14: Neighbor circle 24
Figure 15: AmbientDB layered architecture. 24
Figure 16: AmbientDB P2P usage scenario 27
Figure 17: Instance of local behavior 29
Figure 18: Instance of remote behavior 29
Figure 19: Generic PDU type 32
Figure 20: Adb/NP protocol behavior 36
Figure 21: an example simulation environment 41
Figure 22: The AmbientDB simulation strategy 41
Figure 23A: A Adb/NP simulation setup 43
Figure 24B: the AmbientDB agent initialization 43
Figure 25: The ns-2 structure 59
Figure 26: An ns-2 class hierarchy 60
Figure 27: TclClass instantiation 60
Figure 28: ns-2 node structure 61
Figure 29: ns-2 link structure 61
Figure 30: AmbientDB agent inheritance structure 63
Figure 31: Node-agent relationship in ns-2 63

List of Abbreviations

3G	Third Generations
Adb/NP	AmbientDB Networking Protocol
ADB	AmbientDB
AmI	Ambient Intelligence
CAN	Content Addressable Network
CPU	Central Processing Unit
CWI	Centrum voor Wiskunde en Informatica
DBMS	Database Management System
FTP	File Transfer Protocol
ID	Identity
IP	Internet Protocol
P2P	Peer-to-Peer
PC	Personal Computer
PDA	Personal Digital Assistance
PDU	Protocol Data Unit
PE	Protocol Entity
QP	Query Processor
RDBMS	Relational Database Management System
SAP	Service Access Point
SDU	Service Data Unit
SQL	Structured query language
TCP	Transmission Control Protocol
TSS	Telematics Systems and Services
UDP	User Datagram Protocol
VDLB	Very Large Database

1. Introduction

This chapter describes P2P systems and their types, and represents the motivation, objectives, and the structure of this thesis.

1.1 P2P Systems

The exact definition of a P2P System [1][20] is debatable. Some describe it as an extension of traditional client/server systems and some other describe it as a system without servers. A P2P System is a network of nodes or peers where each node behaves both as a server and a client. Gnutella uses the term servants, where nodes can be both SERVers and clieNTS. Each node in the P2P systems can join and leave the network at will.

All P2P systems, in principle, are based on three common characteristics: resource sharing, decentralization, and self-organization. The resources that can be shared among participating nodes can be physical resources such as storage space, processor cycles, and the network bandwidth or a logical resource such as the knowledge about their neighbors. The decentralization characteristic makes each participant autonomous in P2P system. This characteristic makes a P2P system free of a single point of failure. As the nodes in a P2P system are decentralized, they will self-organize themselves in the network interacting with their reachable neighbors.

P2P systems gained a lot of attention both from the commercial and academic fields, shortly after the introduction of the file sharing system Napster and have already proved its potential in different resource sharing areas:

- CPU: sharing the CPU resources between different participants in the network. For example, SETI@HOME [28], Entropia [7], United Devices [32], etc.

- File/Storage: sharing the storage between different participants in the network. For example, Napster [17], Gnutella [10], KaZaA [15], Freenet [9], etc.

1.1.1 Types of P2P Systems

Current P2P systems appear in different categories such as pure P2P and hybrid P2P systems [2]. A P2P system where each node can communicate with each other without the need of centralized server is defined to be a pure P2P system. Nodes cooperate with each other to find other nodes in the network. This type of system is fully decentralized and each node in the network has an equal role. Gnutella and Freenet are examples of pure P2P systems.

A P2P system where a server is designated to keep track of other participants is called a hybrid P2P system. In a hybrid P2P system, one node discovers another node with the help of a server but the communication between peers takes place independent of the server. Figure 1 shows an example of a pure and a hybrid P2P system.

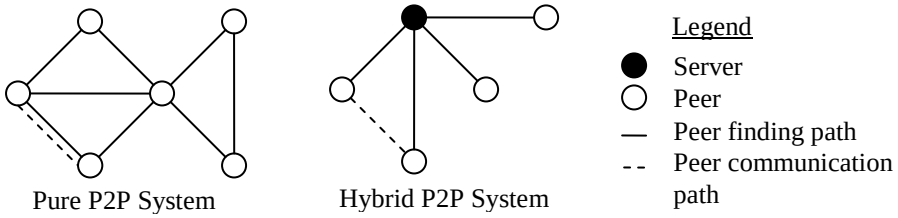


Figure 1: Example pure and hybrid P2P systems

A hybrid P2P system can be hierarchical or centralized. In a centralized P2P system, there exists only one server, in principle, in the network. In a hierarchical P2P system, nodes are organized in a hierarchy of groups, where each group is coordinated by a local server also called a group leader. In this system, communication between groups takes place through group leaders. SETI@HOME is an example of a centralized P2P System. The domain name system is an example of a hierarchical system.

1.2 Motivation

The most popular end user P2P applications are Napster and KaZaA. These P2P systems share a flat, unstructured data model (basically, a list of files with some properties) on which they provide keyword-based exact matching lookup services. These systems allow end users to share their files with a relatively good response time for search.

All the existing P2P systems are targeted at providing a better way of sharing a large number of files between end users. However, they are not able to address data management problems such as managing complexly structured data objects, content update, data semantics and the relationships between data. Furthermore, because of data management problems, P2P applications still lack scalability [14]. If an aggregate query, for example, is asked, it must be forwarded to every node in the network to get a better recall. Database management systems on the other hand, provide functionalities like query processing, query optimization, views, and integrity constraints to consider the relationships between data. These functionalities can be used to define, retrieve and process only the required set of data. An integration of database management technology and P2P technology would seem beneficial.

1.3 Context

This master's project is done in the context of the AmbientDB project at CWI [21]. AmbientDB is a P2P database management system (DBMS) targeted at addressing data management issues in an ad-hoc

network of consumer electronic devices. The motivation behind AmbientDB is to make it easier to create intelligent cooperative applications, as envisioned in Ambient Intelligence (AmI) [24]. AmI refers to the vision of pervasive and obtrusive intelligent applications in our surrounding environment that support the activities and interaction of (mobile) users.

Since the AmI vision demands services/applications to adapt to any (mobile) device and any context (available content, time, place, mood, etc.), it is difficult to hardcode all context information management facilities in each application to allow intelligent interaction between them. AmbientDB offers a single data management facility in highly distributed, heterogeneous, and ad-hoc organized ambient applications, such as context aware applications.

1.4 AmbientDB

AmbientDB aims to provide a unified data management framework for ad-hoc network of consumer electronic devices, including query processing functionalities [21]. For example, an intelligent assistant application at a university that assists students according to their needs based on their mood, situation, location, etc incorporating data stored in their PDAs, Laptops, 3G phones, etc. The data management issue in distributed, heterogeneous and ad-hoc organized devices motivates the use of a P2P architecture in AmbientDB.

1.4.1 AmbientDB Goals

AmbientDB aims to provide full relational database functionality for standalone operation in ad-hoc networks of consumer electronic devices [21]. Devices may be mobile and disconnected for a long period of time. When connected, these devices can communicate in their neighborhood.

1.4.2 AmbientDB Assumptions

AmbientDB takes into account the following assumptions to delimit the scope of the project [21]:

- devices are heterogeneous in their resources (network access, storage, data)
- a large number of devices will cooperate with each other
- all devices cooperate under a common global schema, i.e., they use global relational tables to share data
- during the execution of a single query, the nodes involved in answering it won't leave the network

1.4.3 AmbientDB Architecture

The AmbientDB allows 'intelligent' devices to cooperate with each other in a P2P fashion. The use of P2P architecture eliminates the

need of centralized administration and the cost associated with it. Figure 2 shows the AmbientDB architecture.

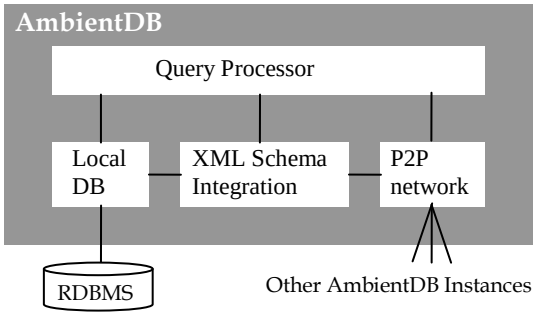


Figure 2: AmbientDB Architecture

The heterogeneity among peer's resources motivates us to use an application level networking protocol thereby forming an overlay network of participating users. The use of an application level protocol is flexible as it facilitates an application level routing. Therefore, it makes an application level communication easier. Figure 3 shows an example heterogeneous network of users.

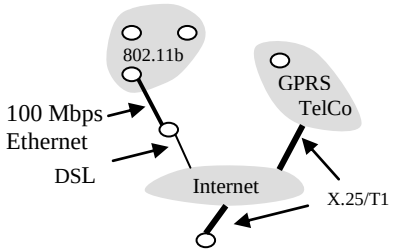


Figure 3: An example of heterogeneous network of users

1.5 Objectives

The ad-hoc query processing facility aimed by AmbientDB requires Gnutella-like query flooding because devices connect and disconnect at will such that there is no (static) knowledge on where data items are located. The query flooding mechanism broadcasts each query message to all (reachable) nodes in the network. However, such query flooding lacks scalability because each search query must be broadcasted to all nodes. Therefore, as the network size increases the rate of query arrival per node also increases. The increased query arrival rate per node leads the system to saturation. This master's project focuses on just one of the many challenges in constructing AmbientDB, namely the creation of a 'good' P2P logical overlay networking layer to prevent such saturation.

In this project, an overlay network is created to reduce the average query response time in AmbientDB network. Related with AmbientDB concerns, two main research questions are formulated:

- How to create a 'good' overlay network?, and

- How to map the overlay network onto the underlying physical network?

Secondly, though many P2P protocols including Gnutella create an overlay network of participating nodes, they disregard the physical network structure/resources. The possibility of creating an overlay network that somehow maps onto the underlying physical network is studied. This mapping will help in reducing the network traffic while executing queries in the AmbientDB.

1.6 Approach

In this project the following approach is taken.

- Some existing P2P systems/protocols are reviewed to understand the existing P2P architectures presented in chapter two of this report. Through the review, the possibility of using the existing P2P architecture in the context of AmbientDB is studied.

- The goal of the project is detailed in chapter three of this report. The goal of the project is defined to limit the scope of this research work.

- In chapter four of this report, the P2P architecture in AmbientDB context is designed. This architecture creates a 'good' overlay network of participating nodes in the AmbientDB network. In order to construct the overlay network of participating nodes in AmbientDB we need a protocol. This overlay protocol will work on top of exiting data transmission protocols and create an overlay network.

- The network simulation tool ns-2 is extended for the AmbientDB P2P protocol in chapter five of this report.

- The AmbientDB P2P protocol is simulated and evaluated with a network simulator tool ns-2 in chapter six of this report. Through the simulation, the effectiveness of the AmbientDB P2P protocol is evaluated against that of the pure Gnutella protocol, in terms of query response time.

- Finally, chapter seven concludes the work of this master's thesis providing the simulation results and some future works.

2. Related Work

This chapter describes the most influential work that has been done in the P2P domain. The review has been done in four different problem fields: Overlay Network, P2P systems, P2Peer file sharing architectures, and P2P databases. Through review, we try to find how existing P2P overlay networks work, what is their query response time, and if they are heterogeneous and scalable.

2.1 Overlay Network

The overlay network consists of an abstract (sub) set of nodes from the underlying network that play an active role in a particular application domain. Figure 4 shows an example overlay network.

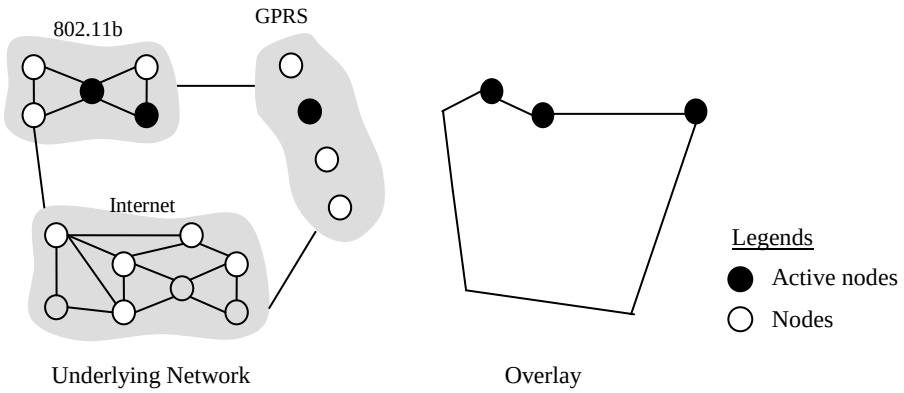


Figure 4: An example overlay network

The overlay network consists only of the active nodes. In the overlay network the end-to-end path between active nodes can be different than that in the underlying network. Therefore, one single link in the overlay network can include multiple links in the underlying network. In principle, the overlay network makes the application level routing possible, irrespective of the underlying network. One of the advantages of using an overlay network is that the users can be mapped onto a topology that corresponds with their available resources (e.g., data, network bandwidth, storage space). This can be useful to allow a participating user to connect to a similar user that is already a part of the network.

2.2 Review of Existing P2P Architectures

A review of some of the most influential P2P systems is presented here. The existing P2P systems and protocols are reviewed to observe how scalable they are, and how they can be used in the context of AmbientDB. Their mechanisms to process and forward queries are evaluated to find their average query response time.

2.2.1 Napster

Napster [17] was one of the first file sharing P2P systems. It was designed to share and swap MP3 files between users. Figure 5 depicts the Napster system model. Napster used central servers to keep track of shared files among the users as well as to create flat namespace of its host addresses. The Napster server maintained the index of shared files and the host information of the active peers in the network. Napster did not replicate the data but used "keepalives" to test client liveliness. The index was updated as the peer joined and left the network. There could be several central servers connected to each other, each of them forming a shared community. Search in Napster was keyword based. Once a file has been found, the download took place from the owner of the file, i.e., P2P downloading. The cost for a node to join and search in Napster was O(1) and O(N) respectively. Where, N is the total number of Napster servers. Thus Napster used a centralized server, violating one of the characteristics of the P2P architecture and thus was prone to central point of failure. When legal action closed down its central servers, the Napster system thus immediately vanished from the Internet.

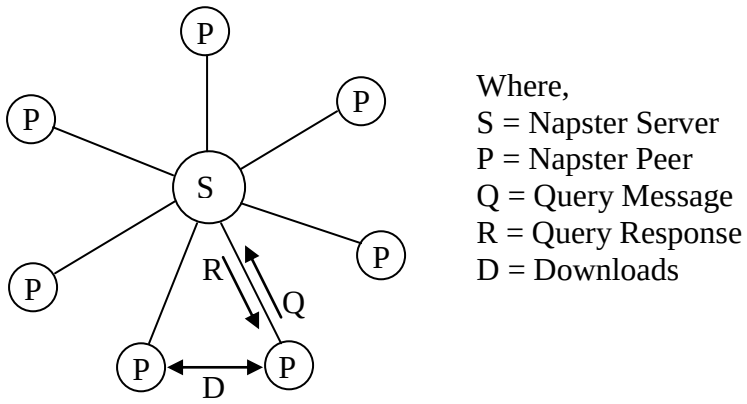


Figure 5: Napster system

Advantages:

Consumes less network resources
Files can be found in lower cost

Disadvantages:

Prone to central point of failure
Expensive to scale the central server

2.2.2 Gnutella

Gnutella is another popular P2P system. Like Napster, it facilitates locating and exchanging files between peers [29]. Unlike Napster, Gnutella does not use any centralized servers and is fully decentralized [10]. The peers are identified by their IP address and therefore a new peer willing to participate must know at least one

peer already in the Network. Each peer keeps a set of connections with its neighbors thereby forming an overlay network [29]. Figure 6 shows an example of the Gnutella system. It uses PING and PONG messages to discover other peers in the network. A joining node sends a PING message to one of the known nodes in the network to discover other nodes. It receives a PONG message in response to a PING. The PONG message contains information about the node such as IP address, port number and number of files shared. Gnutella search is keyword based and the routing is flooding based. Some Gnutella applications use TTL limited query flooding to reduce the scalability problem. The cost for joining and searching in Gnutella is O(1) and O(N) respectively, where N is the total number of participating nodes. Though Gnutella is able to remove the problem of central point of failure, it lacks the scalability characteristics because of this query flooding. As the number of nodes increases, the number of queries also increases in higher magnitude. It is possible in Gnutella to have several disjoint Gnutella overlays.

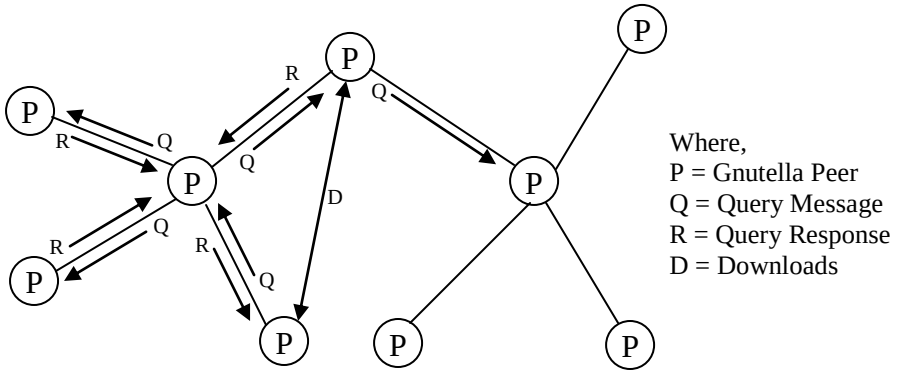


Figure 6: Gnutella System

Advantages:

Fully decentralized
Increased system reliability

Disadvantages:

Low scalability
Consumes more network resources
TTL limited query flooding decreases recall

2.2.3 KaZaA

KaZaA [15] is another kind of P2P network that falls in between Napster and Gnutella which implements the FastTrack protocol [8]. FastTrack groups the nodes as SuperNodes or Nodes. Any node in KaZaA network can become a SuperNode if they are computationally powerful and have fastest internet connection. Figure 7 gives an example of the KaZaA system. The SuperNodes communicate amongst themselves handling search queries. Nodes connect to one of their nearest SuperNodes. SuperNodes allow their neighbor to upload a small list of files, handle their queries, thereby minimizing the

query response time. Query routing in FastTrack is accomplished by broadcasting among SuperNodes. Routing the query result follows the Gnutella principle, i.e., the query results are routed back along the query path. The download in FastTrack is P2P. KaZaA clients need to know at least one super-node in the network. However, KaZaA installation comes with a built in list of KaZaA super-nodes. The details of KaZaA system and the FastTrack protocol are not publicly available.

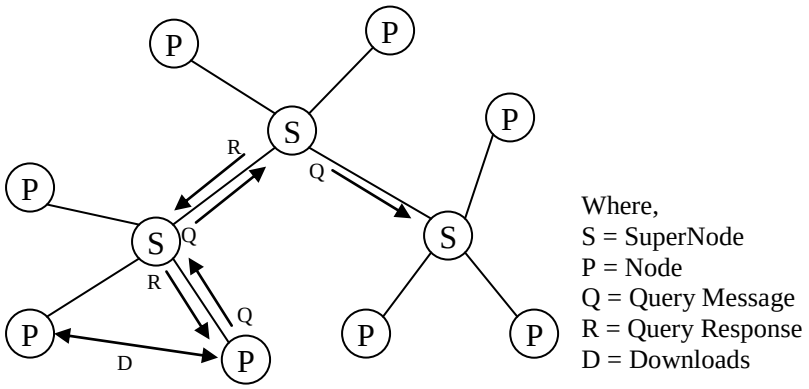


Figure 7: KaZaA System

Advantages:

Efficient query response time
Nodes self-organize in the network
Load balancing
Scalable

Disadvantages:

It is not extensible as the protocol is not publicly available

2.2.4 FreeNet

FreeNet [9] is a distributed information storage and retrieval system designed to provide privacy and availability of data. It uses a SHA-1 function to obtain a location independent key for each file in the system. Each node provides a shared data store which can be used to upload and download files. Also, each node maintains an individual dynamic routing table that contains the addresses of other nodes and the keys of the files they are sharing. With each request message a hops-to-live limit is assigned to prevent infinite loops. Each node is also assigned a pseudo-random identifier, to enable nodes to reject a request that has been seen already. When a request is made for a key k at node n, it looks at its local storage and if found it returns the result back to the user. If the requested key is not found in the local data storage, then the node looks into the routing table and finds the nearest key and forwards the request to the corresponding node. When there is no node to forward the request, the node returns the

backtracking failure message and so the request originator finds another nearest key from the routing table and sends the request to the corresponding node. Thus, the routing in FreeNet is depth first search (DFS) with backtracking. FreeNet uses a lazy replication mechanism [26]. This means that the request result takes the reverse path and makes a replica in each node visited along the path from the request source to the destination. Figure 8 depicts a typical sequence of request message where node A is issuing a request for the data owned by node D.

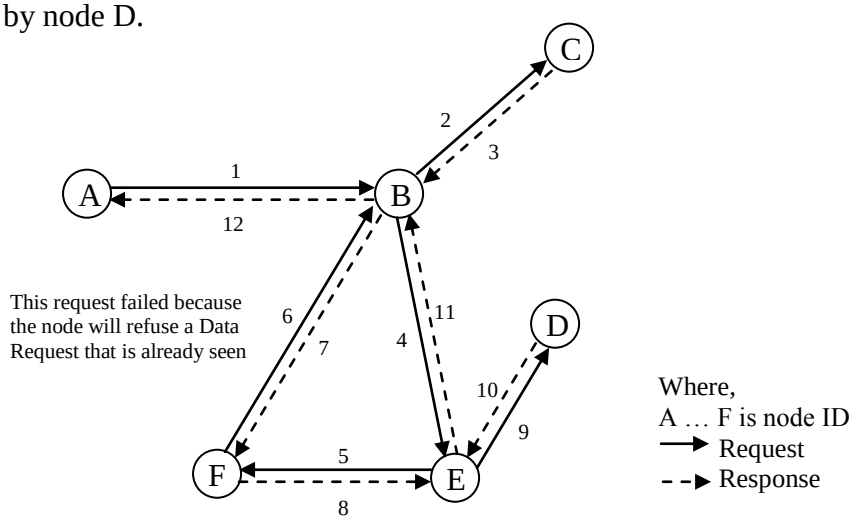


Figure 8: Freenet request sequence

Advantages:

Provides data and user anonymity
Neighbor knowledge is sufficient to find the other peer

Disadvantages:

Due to compressed key, it might select a wrong destination

2.2.5 CAN

Content Addressable Network (CAN) is an indexing mechanism that provides hash table functionality to locate the desired file [31]. It uses a d-dimensional Cartesian coordinate space to make routing possible in a dynamic P2P network. The d-dimensional Cartesian coordinate space looks like the one shown in Figure 9. Each data record has its unique Key which is mapped onto a point in d-dimensional Cartesian coordinate space using a hash function. Each CAN node stores a chunk of an entire hash table also known as zone and also maintains a coordinate routing table that contains the IP address and coordinate zones of its immediate neighbors. In d-dimensional coordinate space, two nodes are immediate neighbors if their coordinate spans overlap along d-1 dimensions and adjoin along one dimension. To find the required key, the requests would be routed through the intermediate nodes towards the node whose zone contains that key. Efficient routing is a critical aspect of CAN. When a new node joins the CAN,

a node with largest volume gives half of its zone to a new node. The node that shares its zone appends 0 to its original virtual ID and the new node appends 1 to the virtual ID of original occupant to form its own virtual ID. When a node departs, then it hands over its zone and associated virtual routing table to one of its neighbor whose zone is smaller. For a d-dimensional space, node insertion affects only O(d) neighbors and the path length is $O(dN^{1/d})$ for an overlay network with N peers.

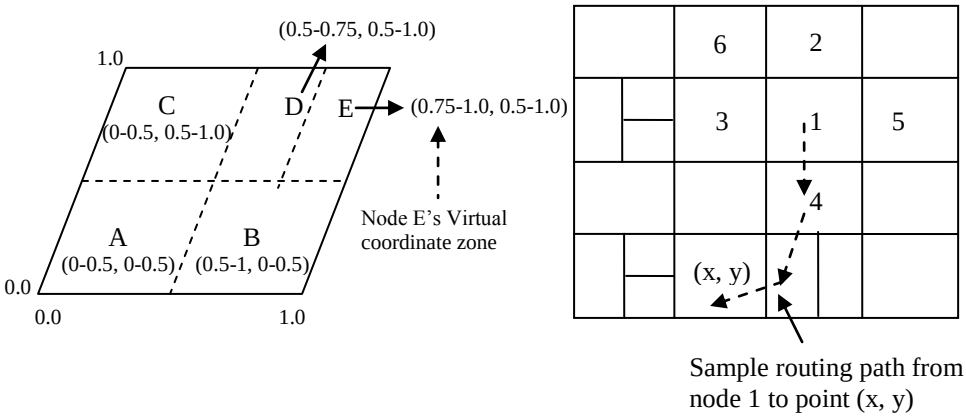


Figure 9: CAN ID-Space (2-d) with 5 (left) and 7 nodes (right)

Advantages:

Scalable and robust
Lower query cost

Disadvantages:

Data must be placed by the system

2.2.6 Chord

Chord [11] is a distributed node lookup protocol which works in a similar way as CAN, mapping keys to nodes that are responsible for them. It uses hash mechanism like SHA-1 to map keys to nodes. Instead of using a d-dimensional Cartesian coordinate space, Chord routes the queries in a circular fashion through the nodes. Each node is identified by an *m-bit* identifier and is responsible for storing the key ID of their closest neighbor in the network. Each data is assigned an *m-bit* ID obtained by hashing its key and is used to locate them. In Chord, nodes are ordered in a circle according to their ID. The data with ID *k* is stored in the closest node before *k* in the circular space. Figure 10 illustrates the Chord circle. In order to find a node efficiently, every node in the network maintain *m-entry* key routing table also called a finger table. The routing table entries consist of a node identifier and its network address. It contains the direct successor as well as additional nodes that have an exponentially increasing distance to it. Query routing sends queries just the node to the closest finger smaller or equal than *k*. In an *N* node network each node only has to have the knowledge of log *N* neighbors and resolves

all lookups, on average, with only log *N* messages to other nodes. This shows that the communication cost of querying data logarithmically scale with the number of nodes in the network. When a node *n* fails, the query is forwarded to the successor of n. To achieve fault tolerant storage, each node in Chord replicates the data to its *r* nearest nodes. So if each node holds information of its *r* successor nodes, failures can be detected and recovered in O(log *N*) time.

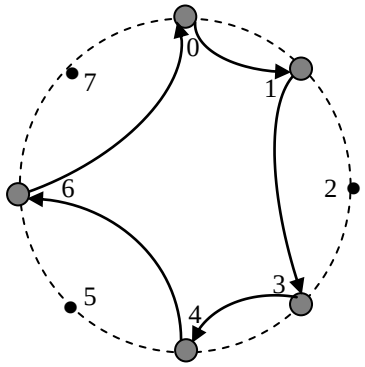


Figure 10: Chord circular ID space and routing

Advantages:

Logarithmic communication cost
Scalable and fault tolerant

Disadvantages:

Data placement must be dictated by system

2.2.7 OceanStore

OceanStore is a distributed storage architecture that provides continuous access to information [13]. It is wide area network (WAN) oriented. OceanStore set two main goals: to cope with an untrusted infrastructure, meaning that the information should be freely accessible irrespective of the strength of the infrastructure or system crash. The second is to support nomadic data, meaning that the data should be location independent. It uses the term persistent object to refer to name of the data, replicas, access control list and data archival fragments. Each persistent object is identified by a globally unique identifier GUID. The OceanStore forms a highly connected "pools" among which data is allowed to flow freely. "Pools" are the servers for the clients connected to them. OceanStore replicates the data in multiple pools. Each node in OceanStore is given an ID according to the Plaxton scheme [13]. The objects are mapped to a single node whose ID matches the object's GUID in the most significant bits. Each node maintains a list of their neighbor's bloom filter and its own bloom filter. It uses Attenuated Bloom filters for query routing. If the bloom filter fails, then it uses the Plaxton routing algorithm. Figure 11 demonstrates the OceanStore Attenuated Bloom filter routing

mechanism. Routing in the OceanStore thus consumes O(N) time, where N is the total number of pools.

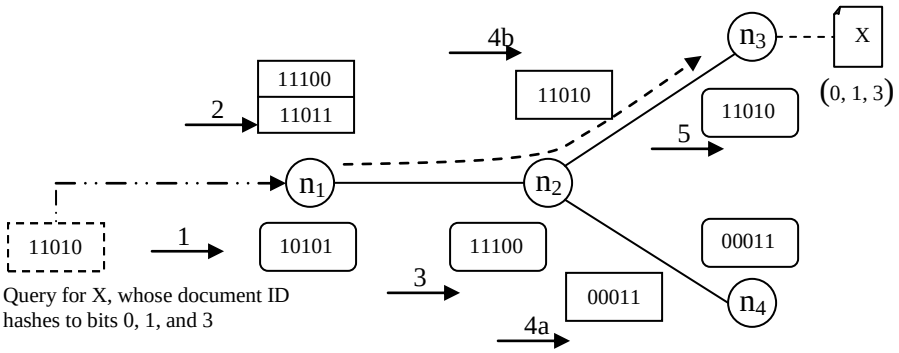


Figure 11: Query processing in OceanStore

Advantages:

Highly distributed storage
Secure

Disadvantages:

Bloom filters may lead to false positives
No dynamicity of nodes, fixed set of "pools" in the system

2.2.8 Comparative Analysis of Existing P2P Architectures

All P2P systems discussed above are designed to facilitate sharing and storing files in a P2P network. These systems, however, share a flat, unstructured data model (basically, a list of files with some properties) on which they have implemented exact matching techniques, ignoring the problems associated with complexly structured data objects, content update, data semantics and the relationships between data. Beside these, they have several other shortcomings in their architecture. For example, the Napster has a central point of failure. Gnutella is not able to provide scalability to large numbers of connected nodes. CAN, Chord and FreeNet use Distributed Hash Tables (DHTs), to minimize the scalability problem of the Gnutella system. However, the problem with DHT is that data location is system dictated. The same sort of problem is apparent in OceanStore. KaZaA, using its concept of SuperNodes, is able to reduce the scalability problems of Gnutella. In KaZaA, peers can choose a SuperNode to index their files, if they find themselves too poor to process queries for other peers in the network. This system is, therefore, similar to the AmbientDB vision. Unfortunately, KaZaA has not unveiled its architecture making it difficult to analyze if it could be used in AmbientDB.

The following table summarizes the features and the cost of the systems discussed above.

Table 1: Features of the Systems

Features	**Napster**	**Gnutella**	**KaZaA**	**FreeNet**	**CAN**	**Chord**	**OceanStore**
Decentralized	No	Yes	Yes	Yes	Yes	Yes	Yes
Query Cost	O(1)	O(1)	O(S)	O(N)	$O(N^{1/d})$	O(log N)	O(P)
Failure Cost	-	O(log N)	-	O(N)	O(1)	O(log N)	-
Participation	O(1)	O(N)	O(1)	O(N)	O(d)	O(log N)	O(1)
Load Balance	-	No	Yes	No	Yes	-	Yes
Locality Aware	Yes	No	No	No	No	No	Yes
System dictated data placement	No	No	No	Yes	Yes	Yes	Yes

Where,

N = Total number of nodes in the network
S = Total number of SuperNodes in the network
P = Total number of pools in the network
d = The dimension of Cartesian Coordinate

In summary, existing P2P systems/protocols lack support for databases. In existing P2P systems user's can not control the data transfer. Though some of these systems are opting to provide better file sharing service, they lack mapping between physical and overlay network to reduce network traffic caused by query transfer.

2.3 P2P Databases

The P2P architecture has been gaining popularity for sharing information in some specific domains between the active peers in the network. As the peers in a P2P system can join and leave the system at will, it is difficult to predict availability and data consistency. In [30] Gribble et al, states the need of database management in P2P systems, as the existing systems ignore the semantics of data and their relationships. Database management is required to provide finer data granularity and preserve the semantics of data and their relationships. Two fundamental problems are visible in P2P database management systems: answering queries from the whole network, and minimizing the query response time.

In [1], Abhishek et. al proposed an architecture for P2P data sharing system using database systems. Their architecture assumes that peers can cache horizontal partitions of various relations and database schema is global similar to that in AmbientDB. However, they allow the 'select' on a relation based on one attribute at a time. To find the nodes with relevant data partitions, hashing would be used. For this purpose, [1] in their architecture have used Chord. Each node, therefore, stores partitions of similar type. To answer a query, a query plan is used and all the selects are moved towards the leaves as much as possible. Each leaf then evaluates the query, and returns the answer for that particular select. The node/intermediate node issuing the query can now compute the remaining query by Cartesian product of all the results. This architecture is opting for schema integration, but it does not address the problem of heterogeneity among the peer resources.

The above mentioned architectures and principles indicate that integration of database technology and P2P systems may help in addressing data management issues in networked applications. However, a major problem is to find a suitable location and indexing mechanisms. The data location is a crucial aspect as the P2P system uses ad-hoc overlay topology and there is no mapping between data location and overlay topology. This results in unstructured data management [25]. Introducing the super node concept similar to that of KaZaA might help to control the scalability problem of Gnutella. Beside the scalability problem, most important is to make the system able to handle structured data, preserving the relationships between data and their semantics.

3. Problem Definition

The review of existing P2P protocols and systems, in chapter two, revealed that the existing P2P systems lack heterogeneity, scalability, and user control. This chapter states the problem statement of the AmbientDB P2P protocol and presents the AmbientDB network structure.

3.1 Problem Statement

The ultimate goal of the AmbientDB project is to share and query database relations with complex structure in an ad-hoc network of consumer electronics. The devices that participate in the network are heterogeneous in their resources (e.g., network bandwidth, network latency, storage space and stored data). The lower resource devices place their data at the devices with higher resources. These devices which have higher resources will work on behalf of lower resource devices. Every device in the network retains the control over data placement. However, user dictated data placement can not use DHTs as in existing P2P systems. To enable query processing facility in AmbientDB we use query flooding as in Gnutella. As Gnutella flooding inherently leads to scalability problem, a KaZaA-like approach can be used to. Also, it is necessary to optimize the use of the physical topology to reduce network traffic.

3.1.1 Goals

We define our goal for this project to create a 'good' self-organized overlay network topology. In this project, we define a good overlay network as the overlay network that would have minimal query response time given a set of assumptions such as the network size, node resource distribution, data distribution and query distribution.

In order to create this optimal overlay network, every node that wants to join the network would have to find a suitable place for it in the AmbientDB P2P network. The type of available resources among the participating nodes and their stored data determines this suitable place. Nodes with lower resources will have to be able to transfer its data to a node with higher resources. Also, the nodes will have to be able to decide whether to delegate their query handling to other nodes.

To support the goals of this project we define two sub-goals as follows:

- Automatically assign roles to the participating nodes i.e., super-node or independent node or normal node;

- Create an overlay topology that maps onto the underlying physical topology to reduce network traffic caused by query transfer.

3.1.2 Assumptions

It is difficult to evaluate the performance and scalability of P2P systems because of the presence of several uncontrollable factors such as data distribution, query frequencies, and network heterogeneity. Also, the network traffic created by other networked applications can affect P2P communication.

In order to be able to do some quantitative comparisons we made the following assumptions.

Query Cost: the overall network cost dominates the query processing cost. The average network cost, therefore, determines the average query processing cost. While calculating network cost, we ignore the network traffic caused by other networked applications. Also, we ignore the super-node initialization cost. That is when a normal node joins a super node, data transfer cost will be ignored.

Queries: AmbientDB queries can be arbitrarily complex. We concentrate on aggregate queries in this project. Each query in AmbientDB network is broadcasted to all super-nodes which will send a response back for each query.

Message exchange: we use the TCP transfer protocol for exchanging the messages between nodes. The UDP broadcast will be used in limited segments of the P2P network in order to find an existing node in the network. When transferring a large blob of data from a normal node to a super-node, we use FTP protocol.

Failure resistance: we assume that nodes do not leave the network as long as they have waiting queries to respond.

Network limitations: each node activity is independent of the network bandwidth between two nodes in the sense that node activities do not affect the data transfer.

Data and query distribution: we will experiment for the moment with a uniform data and query distribution in the network.

We acknowledge that our assumptions are strong. However, they do allow us to evaluate the P2P systems to some extent. This should be taken as a first step towards investigation and evaluation of the P2P systems. Further refinement of the evaluation taking into account more factors that affect the performance of the P2P systems is considered as future work and is out of the scope of this master's thesis.

3.2 AmbientDB Network Structure

The AmbientDB P2P logical overlay structure is a spanning tree, where participating peers are grouped in clusters. Each cluster consists of a leader that is responsible for handling queries on behalf of others. The leader is selected dynamically based on its available resources e.g., available storage, network bandwidth and network latency. This means that the cluster leader is resource rich and can hold the data of other nodes in the same cluster. Two clusters can overlap if the leader of one cluster can hold the data of the leader of another cluster. If the neighboring clusters are non-overlapping then their leaders are independent to each other. The communication between two clusters takes place through their leaders. Members of one cluster communicate to each other through the leader of the same cluster. The clustering structure can be seen as a recursive structure. That is, a cluster leader can be a member of another cluster. Two cluster leaders can have direct communication with each other only if there is a direct (logical) connection between them. Figure 12 shows a clustered structure of an AmbientDB P2P network.

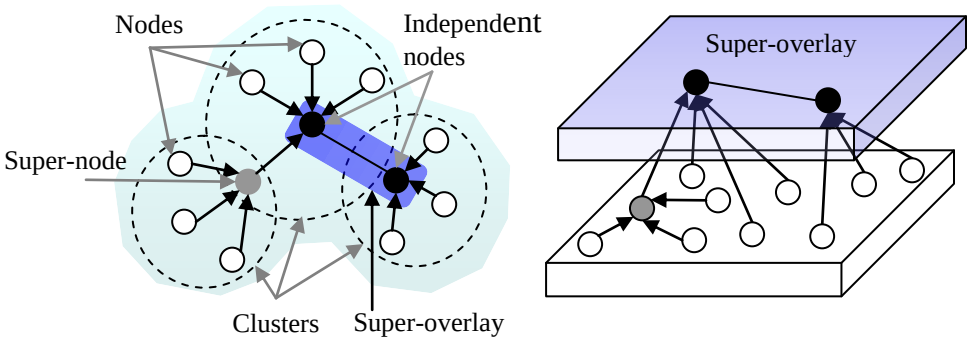


Figure 12: AmbientDB Network Structure with two independent nodes

As indicated in the figure, we allow hierarchical clustering connected via their leaders. In this structure, we call a cluster leader a super-node and the members the nodes of the AmbientDB network. Therefore, as explained before a super-node can have another super-node. However, independent super-nodes may exist if the cluster leaders are independent to each other. The directed edge in the figure indicates the direction of superiority between two nodes. This means that a node at the tail of a directed edge transfer its data to other node at the head of the edge. The independent nodes are 'equal' in their resources and therefore form a 'cloud', called a super overlay. In a super overlay, there are no directed edges as the nodes are independent super-nodes and do not transfer data to other nodes. Like in Gnutella, the super-overlay does not have a root. A node in the super-overlay that receives a query is considered a root node of the 'cloud' for that moment.

Leafs of the tree consist of the nodes with poor resources. A node that is using lower network bandwidth and has smaller storage space than

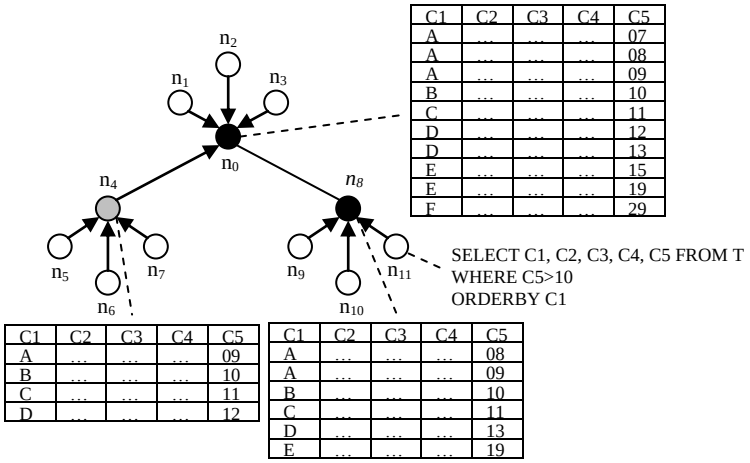
their neighbors is defined as a poor resource node. The intermediate nodes of the tree consist of the higher resource nodes (super-nodes or independent nodes). If a node is not a poor resource node then it is called a higher resource node. It is possible that two neighbor nodes can have higher resources and become independent super-node of each other. The Adb/NP protocol, therefore, dynamically assigns roles to the participating nodes as good as possible.

3.2.1 Node Structure

Each intermediate node in the AmbientDB overlay tree may provide storage space to its immediate children to store their data. These intermediate nodes also maintain a dynamic neighbor table that contains information about their immediate neighbors. For an intermediate node, both its parent and children are immediate neighbors. The neighbor table primarily consists of neighbor type, neighbor status, neighbor's storage capacity, neighbor's stored data, network latency, and network bandwidth. The neighbor status can be a super-node, an independent node or a node.

3.2.2 Query Processing

In AmbientDB, we define a node that sends a query as a query initiator and a node that receives a query as a query receiver. The basic query processing scenario in AmbientDB is the following. If the query initiator is not a super-node then it forwards the query to the super-node in the same cluster. When the super node receives the query, it floods the query over the super-overlay. When a query receiver gets the answer from the super-overlay, the query initiator can retrieve the query results from its super-node. The basic query processing scenario is explained below with the help of Figure 13.



C1	C2	C3	C4	C5
A	…	…	…	07
A	…	…	…	08
A	…	…	…	09
B	…	…	…	10
C	…	…	…	11
D	…	…	…	12
D	…	…	…	13
E	…	…	…	15
E	…	…	…	19
F	…	…	…	29

C1	C2	C3	C4	C5
A	…	…	…	09
B	…	…	…	10
C	…	…	…	11
D	…	…	…	12

C1	C2	C3	C4	C5
A	…	…	…	08
A	…	…	…	09
B	…	…	…	10
C	…	…	…	11
D	…	…	…	13
E	…	…	…	19

Figure 13: Query processing scenario

In the figure, node n_{11} is the query initiator. The query goes to the super-node n_8, being super node of n_{11}. This query is flooded in the super-overlay i.e., among n_8, n_0. This Gnutella-like flooding takes place as follows.

1 n_8 receives the query from n_{11}, executes the query locally and forwards the query to its independent super-node n_0 and waits for the result from n_0.
2 n_0 executes the query locally and returns the result tuples to n_8. As n_0 does not have any other (independent) super node, it may not forward the query to any other node, such as n_4.
3 Upon receiving the query results from n_0, n_8 merges this result with its local results and sends the result tuples to n_{11}.
4 n_{11} receives the result from n_8.

3.3 Formal Model

AmbientDB P2P networking protocol creates an overlay structure of participating nodes. We formally represent the AmbientDB P2P network topology by a spanning tree whose nodes represent the participating devices and the edges represent the logical connection between participants.

AmbientDB nodes: an AmbientDB node is denoted as n(*r*, *d*), where *r* represents the storage space and *d* represents the stored data in that node.

AmbientDB edges: an AmbientDB edge is denoted as *e*(*b*, *l*), where *b* represents the network bandwidth and *l* represents the network latency between two nodes.

AmbientDB super-nodes: a node n_1 is a super-node with respect to a node n_2, if and only if there is a directed edge *e* from node n_2 to node n_1.

AmbientDB independent-nodes: two super-nodes n_1 and n_2 are independent to each other if there is an undirected edge *e* connecting n_1 and n_2.

AmbientDB overlay: formally we define the AmbientDB overlay as a tree T (N, E), where N is the set of nodes in the network and E is the set of edges between nodes $u, v \in N$.

AmbientDB super-overlay: formally we define the AmbientDB super-overlay as a sub-tree T_S (N_S, E_S) | $T_S \subseteq T$, $N_S \subseteq N$ and $E_S \subseteq E$, where N_S is the set of independent nodes and E_S is the set of edges between independent nodes $u_S, v_S \in N_S$. The AmbientDB super-overlay is used to answer the queries.

AmbientDB query: an AmbientDB query is an aggregate query like:

```
SELECT      count(*), genre
FROM    songLog
GROUP BY    genre
```

AmbientDB query cost: an AmbientDB query cost is calculated as the sum of the network cost:

- to transfer a query from a query initiator to the super-overlay,
- (because of) query flooding in the super-overlay, and
- to transfer the result from super-overlay to the query cost.

Each node in the super-overlay can have more than one neighbor. The neighbors of a node may not be in equal distance. Therefore, the query response time for a node is the maximum of the response time of its neighbors. This implies that the query response time in the AmbientDB network can be calculated recursively. Formally, we define the query response time as follows:

$C_0(y) = 0$
$C_i(y) = \max \{C_{i-1}(x) + 2\, L_y(x) \mid x \in N_y\}$

Where,
$N_y = \{x \mid x \text{ is neighbor of } y\}$
$L_y(x)$ = latency between y and x

The average query cost is calculated as:

$$C_{avg} = \frac{1}{n}\sum_{i=0}^{n} C(y_i)$$

Where,
n is the total number of queries.

AmbientDB neighbors: two nodes n_1 and n_2 ($n_1, n_2 \in N$) are neighbors to each other if there is an edge $e \in E$ between them.

AmbientDB neighborhood: the neighborhood of a node n_1 in the AmbientDB network is denoted by $F(n_1)$, where $F(n_1) = \{n \mid (n_1, n) \in E\}$.

AmbientDB participating devices: the AmbientDB participating devices falls in two categories dynamic devices and static devices represented by *M* and *I* respectively ($M \subseteq N$, $I \subseteq N \mid M \cup I = N$). We define highly mobile and lower resource devices e.g., PDA, MP3 Player, and 3G Phone as dynamic devices. The static devices include the devices that are not mobile but have higher resources, e.g., PC and Laptop.

Each node in the AmbientDB P2P network is identified uniquely by its IP-Address.

4. Protocol Design

This chapter describes the strategies used to build the AmbientDB overlay, and the architecture of the AmbientDB networking protocol.

4.1 Joining the AmbientDB Overlay

When joining the AmbientDB network, a node has to find a suitable place to join the AmbientDB overlay thereby deciding whether it will become a super-node of other nodes or even be part of the super-overlay. In order to join the network a joining node must know at least one node that has already joined the overlay. The join process is the following.

Let the node that joins the network be *n* and *n*' is the node it knows and is already a member of the AmbientDB overlay. The following steps take place to find a suitable place for node *n* in the AmbientDB overlay.

1. *n* contacts and asks *n*' for its membership in the overlay.
2. *n*' locates and returns the address of the super-node *S* belonging to a cluster *C* using the principle described in the section 4.1.1.
3. if *n* sees *S* as its super node it joins the cluster *C* as a normal node and transfers its data to *S*.
4. if *n* sees *S* as a normal node it joins with *S* and becomes the new super-node of the cluster *C* and receives data from *S*. *S* now becomes a normal node of the cluster *C*.
5. if *n* sees *S* neither as its super node nor as a normal node, *n* creates a new cluster *C*', *n* being the only member and super node of that cluster, and becoming an independent neighbor of *S*.

4.1.1 Super-node Selection

Finding a concrete algorithm for selecting a super-node is difficult because of the presence of uncontrollable multi dimensional factors such as data size, storage space, available network bandwidth, ad-hoc participation of the nodes, etc. We use the following heuristic that works similar to the depth-first search algorithm to select a super-node in the network.

When a node receives a join request it does the following to select the super-node:

1. Measures the candidate latency with the Ping-Pong message. Candidate latency is the latency between *n* and *n*'.
2. If the candidate latency is higher than the previous candidate latency *n'* returns null. The previous candidate latency for the original join request is ∞.

3. If the candidate latency is lower than the previous candidate latency *n*' compares the new node with respect to their resources and stored data.
4. If *n*' can become the super-node for *n* do 5 else do 7 to select even a better candidate super-node from its neighbor table.
5. If the neighbor table *T* is not empty, select a set of candidate super-nodes $L|L = \{x|\ x \in \text{neighbor (T)}\}$ using the principle described in section 4.1.2.
   a. Select a subset of nodes $L'|\ L' \subseteq \{L\text{-forwarding node}\}$ that have higher network bandwidth. Initially, forwarding node = $\{\phi\}$.
   b. If *L*' is not empty, select a subset of nodes $L'' \mid L'' \subseteq L'$ that have storage to store the data from the joining node, else return the current node as a candidate super-node.
   c. If *L*'' is not empty, select the lowest network latency node $n''|\ n'' \in L''$ else return the current node as a candidate super-node.
   d. Forward the join request to *n*'' and wait for the address and other information of *S*. forwarding node = forwarding node $\cup$ {b}
   e. If *n*'' can not return the address of *S* that can be a super-node for a new node, repeat *c* with next lowest latency node *n*'', else candidate super-node = candidate super-node $\cup$ {*S*}. Initially, the set candidate super-node = $\{\phi\}$.
6. If the neighbor table is empty or none of the neighbors can find the super-node, return the current node as a candidate super-node.
7. If *n*' can not become the super-node for *n*, it does the following:
   a. If *n*' is a normal node it delegates the request to its super-node and waits for the address and other information of *S*.
   b. If *n*' is the super-node of its cluster, repeat 5
8. If the candidate super-node = $\{\phi\}$, *n*' returns the lowest latency node as.

The pseudo-code of this algorithm is available in **Appendix I**.

4.1.2 Nearest Neighbor Selection

In AmbientDB, we use a heuristic that is similar to the principle used in [6] to find the nearest neighbors of a node, as explained below.

Let the node that joins the network be *p* and *q* be the node it knows and is already in the network. Let *N* be the total number of nodes in the network. *q* selects another participant *r* from its neighbor table T, $r \in N$, such that *latency* (*p*, *r*) $\leq$ *latency* (*p*, *q*). Let, *l* = *latency* (*p*, *q*) and *l*' = *latency* (*r*, *q*). Where, the function *latency* (*x*, *y*) measures the latency between two participants *x* and *y*.

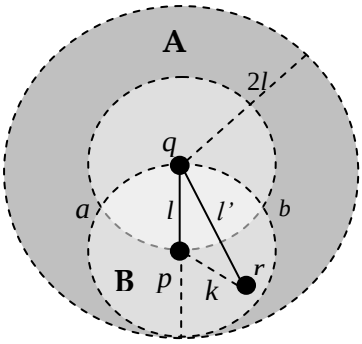


Figure 14: Neighbor circle

In

Figure 14, a joining node *p* contacts *q* that is already in the network. After receiving the join request, *q* measures the latency *l* between *p* and *q*, with Ping-Pong messages. It selects all the nodes from its neighbor table that are in the neighborhood of *p*. We assume that all the nodes within the latency distance *l* from *p* (lower inner circle in the figure) are closer to *p*. If we draw a circle of radius *l* around *q*, the nodes in region **B** can not be assumed nearer to *p*. To cover this left out region, *q* assumes that all the nodes within 2*l* latency distance are closer to *p*. This assumption, however, considers nodes that are far from *p* (all the nodes from region **A**, in the figure above) as closer to *p*. Therefore, this assumption alone can not guarantee that *r*, for example, is the nearest neighbor of *p*. However, this assumption is used to select some nodes from *p*'s neighbor table. To come to precision, our heuristic works as follows:

q sends *l* to its neighbor *r*, $r \in$ N within distance *l*', such that $l' < 2l$. *r* measures the real latency *k* between *p* and itself. If $k < l$, *r* executes the requests otherwise it send a negative response to *q*. After receiving all the responses from its neighbors, *q* can determine a node is closer to *p*.

4.2 AmbientDB Service

The AmbientDB service is a P2P query processing service provided by an AmbientDB service provider. Figure 15 shows the AmbientDB layered architecture.

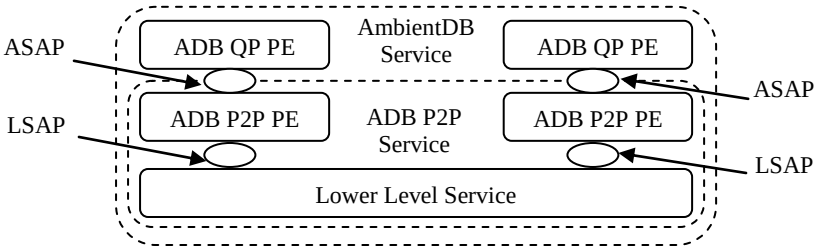


Figure 15: AmbientDB layered architecture.

The definition and identification of the AmbientDB Service and its users is outside the scope of this thesis. However, we assume that the AmbientDB Query Processor (ADB–QP-PE) protocol entities make use of AmbientDB P2P (ADB-P2P) service provider to communicate with each other. The AmbientDB query processor protocol entities (ADB-QP-PEs) interact with the AmbientDB P2P service provider through an AmbientDB P2P service access point (ASAP). A service access point is an interaction point that marks the boundary between a service user and a service provider [5]. The AmbientDB P2P protocol entities (ADB-P2P-PEs) make use of an underlying point-to-point service to communicate with each other. Each ADB-P2P-PE communicates with the underlying service provider through a lower level service access point (LSAP).

4.2.1 Service User

The AmbientDB query processors are the users of the AmbientDB P2P service provider. The query processors can be running in different machines and are connected with each other through different networks. The machine characteristics and network characteristics together determine the role of query processors in an AmbientDB query processing scenario. AmbientDB query processors running in lower resource machines do not participate in the query processing scenario. However, they can send queries to query processors running in higher resource machines. The AmbientDB query processors register themselves to the AmbientDB P2P service provider and communicate with other AmbientDB query processors in their neighborhood.

4.2.2 Service Definition

We define and concentrate on the AmbientDB P2P service. The AmbientDB P2P service is the service provided by the AmbientDB P2P service provider. The AmbientDB P2P service allows its users to join the AmbientDB network, send and receive data to and from another user, view the list of other users in their neighborhood, create logical connections between them, exchange message in the scope of these connections, and leave the network. We assume that a user should join the network before (viewing and) exchanging messages with other users in the network.

The AmbientDB service provider has to deal with two service concerns that are addressed by two service elements: user participation and message exchange. The user participating service element handles join requests from the users and data transfer from one user to another, if necessary, when a user leaves or joins the network. To perform user participation, the AmbientDB service provider has the following service primitives.

joinReq: a user specifies the destination address and joins the network.
joinConf: a user receives a join confirmation.
joinInd: a user receives a join indication.
transferReq: a user transfers its stored data to another user in the network.
transferInd: a user receives data from another user in the network.
leave: a user leaves the network.

The message exchange service element handles the exchange of messages between users. To perform message exchange, the AmbientDB service provider has the following service primitives.

dataReq: a user sends a message to another user in the network.
dataInd: a user receives data request from another user in the network.
dataResp: a user sends a response to the data received from another user in the network.
dataConf: a user receives a response from another user in the network.

The message exchange service element is used to send queries from one user to another and to return answers to the received queries. A user should provide the destination of the receiving user while sending a message. The AmbientDB P2P service provider has the following service elements to provide a neighbor list of a user:

nbrListReq: a user requests the list of its neighbors.
nbrListConf: a user gets the list of its neighbors.

Table 2 shows the service primitives and their parameters.

Service Primitives	**Parameters**
joinReq	destAddr, storageSpace, dataSize, bandwidth
joinConf	status (super-node, node, independent-node)
joinInd	status (super-node, node, independent-node)
transferReq	data
transferInd	data
nbrListReq	-
nbrListConf	neighbor list
dataReq	destAddr, message id, data (typically a query)
dataIndd	srcAddr, message id, data
dataResp	destAddr, message id, response (typically a query-result)
dataConf	srcAddr, message id, response
Leave	-

Table 2: AmbientDB P2P service primitives and their parameters

Users inform their machine characteristics at the time they perform a *joinReq* service primitive. The information provided with a *joinReq* service primitive is used to determine the user's location in the overlay network.

A user can perform a join request at it's ASAP. The AmbientDB service provider finds a suitable location in the overlay network, creates a connection to an existing user, and confirms the connection. After the join has been established, the requesting user either requests a data transfer in case it is a normal node or receives a data transfer indication in case it becomes a super-node. After transferring the data, a user can perform a neighbors list request at it's ASAP and send (or receive) queries (or query-results) to (or from) its neighbor(s) for a number of times. After the join has been established, a user can leave the overlay network at any time in case it is a normal node, otherwise it can not leave the network unless there are pending requests.

4.2.3 Usage Scenario

Figure 16 illustrates the basic usage scenario of the AmbientDB P2P service.

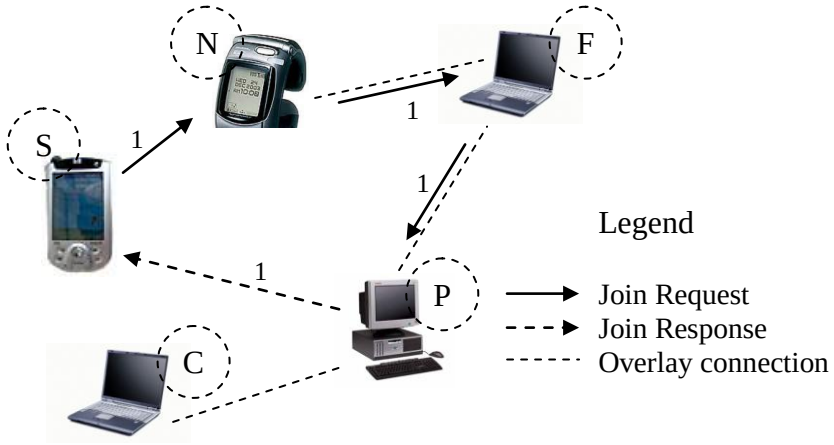


Figure 16: AmbientDB P2P usage scenario

A student S while doing her home work, during midnight, needs some information about protocol architecture. She wants to refer to papers about protocol architectures A, B, and Z written by three famous researchers X, Y and Z either individually or together. She picks up her PDA and announces to her network about her willingness to join the network. A 3G phone used by one of her neighbors called N, detects that S wants to join the network. This 3G phone is already in the community and is connected with a Laptop used by his friend F. As a 3G phone can not guarantee that it can provide services that S's PDA might asks for, it forwards the request to F's Laptop. F's Laptop finds that S's PDA has significantly lower processing power, storage space and is very far from F. F's Laptop then forwards the request to its friend P's PC which is located closer to S and has higher storage spaces and network bandwidth. P's PC also detects that there is

another Laptop C which is located even closer to S, but is not capable to store S's data (if needed). P's PC then decides to grant S's PDA permission to join the network through it and sends the join response to S's PDA and adds S's PDA in its neighbor list. When S's PDA gets permission to join the network, it knows the storage space and bandwidth of P's PC are very good. Knowing this, S's PDA decides to transfer its data to P's PC. P's PC is now super node for S's PDA.

By this time, S knows that she has access to the network and sends a query to P via her PDA. After receiving a query from S's PDA, P's PC looks into its database and stores the result temporarily. At the same time, it passes the query to C and F's Laptop. P's PC, after receiving the results from C and F's Laptop, merges them with its local result and sends the overall result to S. Finally S will be able to see the title of the papers available about protocol architectures A, B, and C written by professors X, Y, and Z. S now has sufficient information so she leaves the community. When she leave the community, S's PDA informs P's PC that S is leaving. P's PC now removes S's PDA from its neighbor list.

4.2.4 Service Behavior

The execution of service primitives at distinct SAPs determines the behavior of the AmbientDB P2P service. The interaction between a user and the AmbientDB service provider at a SAP may effect the interaction between the AmbientDB service provider and other users at other SAPs. The interaction between a user and the AmbientDB service provider at a SAP is defined as the local interaction and the interaction caused because of the local interaction between the AmbientDB service provider and other users at other SAPs is defined as the remote interaction. We define the service behavior corresponding with the local interaction as the local service behavior and that corresponding with the remote interaction as the remote service behavior.

The local behavior at ASAP of the AmbientDB service in one instance of communication has the following characteristics.

- a user is only allowed to perform a *joinReq* to the network.
- after performing a *joinReq*, a user receives a *joinConf*.
- after receiving join confirmation by performing a *joinConf*, a user is only allowed to transfer or receive data by performing a *transferReq* or *transferInd* respectively.
- after performing a *transferReq* or *transferInd*, a user may perform a *nbrListReq* or receive a data request by performing a *dataInd*.
- after receiving a data request by performing a *dataInd*, a user sends a response by performing a *dataResp*.
- after performing *nbrListReq*, a user receives a neighbor list by performing a *nbrListConf*.

- after performing a *nbrListConf*, a user may send a data (typically a query) and receive a data (typically a query-response) by performing a *dataReq* and *dataInd* respectively.
- after performing a *dataReq*, a user may receive a data response by performing a *dataConf*.
- A user can leave the network by perform a *leave* at any time after a performing a *joinReq*.

Figure 17 depicts an arbitrary instance of the local behavior of the AmbientDB P2P service.

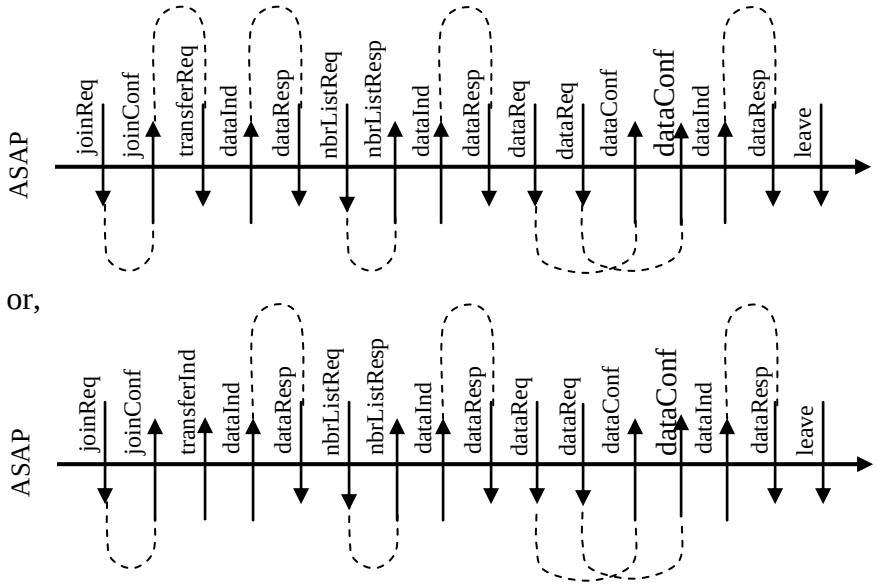


Figure 17: Instance of local behavior

The remote behavior at ASAP of the AmbientDB P2P service in one instance of communication has the following characteristics.

- each *joinReq* causes a *joinInd* to be issued to the destination user. We assume that the destination user has already performed a *joinReq* and has not performed a *leave* after that.
- each *transferReq* causes a *transferInd* to be issued to the destination user before the sending user performs a *leave*.
- each *dataReq* causes a *dataInd* to be issued to the destination user before the sending user performs a *leave*.
- each *dataResp* causes a *dataConf* to be issued to the destination user before the sending user performs a *leave*.

Figure 18 depicts an instance of remote behavior of AmbientDB P2P service.

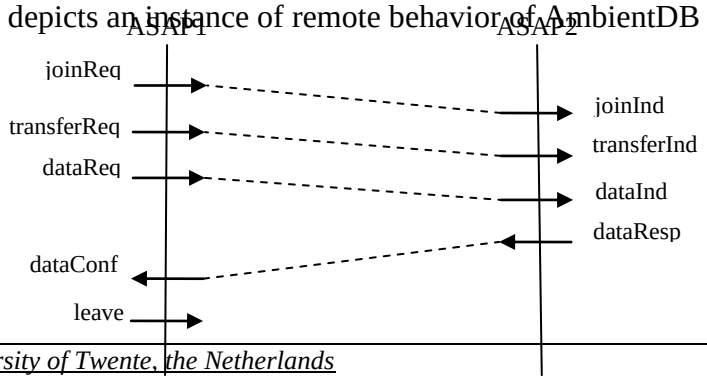


Figure 18: Instance of remote behavior

4.3 AmbientDB P2P Protocol

The AmbientDB P2P protocol provides the AmbientDB P2P service as defined before. The AmbientDB P2P protocol is responsible for two main concerns: firstly the administration of the participants and secondly the transfer of message between participants in the overlay network. To manage these service concerns, two main service elements and a lower level service is identified. These service elements and lower level service together forms the AmbientDB P2P protocol. The main service elements identified for this protocol are: attachment control and message exchange. The attachment control service element controls the new participation in the network. The message exchange service element controls the exchange of messages between the users.

4.3.1 Low Level Service

To make protocol entities able to communicate with each other, the point-to-point data transfer service provided by separate layers (TCP, UDP, FTP) on top of IP is identified as lower service. For the exchange of a join request messages between AmbientDB P2P protocol entities, the service provided by UDP is identified as a suitable service. The service primitives and their parameters of this lower level service is listed in Table 3:

Service Primitives	**Parameters**
sendReq	destination IP, SDU
recvReq	source IP, SDU

Table 3: Service primitives of the lower level service, UDP

A user sends a message to another user by executing a sendReq service primitive and a user receives the message by executing a recvReq service primitive.

For the exchange of a data request (typically a query and query-response), the service provided by UDP is identified as a suitable service. The service primitives and their parameters of this lower level service is listed in Table 4:

Service Primitives	**Parameters**
connReq	destination IP, connection id
connInd	source IP, connection id
connResp	destination IP, connection id
connConf	source IP, connection id

dataReq	destination IP, connection id, SDU
dataReq	source IP, connection id, SDU
connRelease	connection id
releaseConf	-

Table 4: Service primitives of the lower level service, TCP

Similarly to exchange a transfer request (typically a data from a normal node to a super-node), the service provided by FTP is identified as a suitable service. The service primitives and their parameters of this lower level service is listed in Table 5.

Service Primitives	**Parameters**
ftpConnReq	destination IP, connection id
ftpConnInd	source IP, connection id
ftpConnResp	destination IP, connection id
ftpConnConf	source IP, connection id
ftpSendReq	destination IP, connection id, SDU
ftpRecvReq	source IP, connection id, SDU
ftpConnRelease	connection id
ftpRealeaseConf	-

Table 5: Service primitives of the lower level service, FTP

4.4 Protocol Functions

As the service provided by the lower level service, UDP, is unreliable, to guarantee reliability, the AmbientDB P2P protocol can use a positive acknowledgement together with a time out. When a positive acknowledgement is not received within a time $t+\Delta t$ for a message sent, it can be sent again. This method is suitable to ensure that a message indeed reaches the intended protocol entity. It is also simple to implement and sufficient to ensure reliability.

4.4.1 Participation Administration

The participation administration function keeps track of new participants in the network. When a join request is arrived, the participation administration function pushes the request to the joinRequest queue and sends a ping message to the requesting node. When it receives the pong message in response to the ping message, it removes the join request from the joinRequest queue, and evaluates the capacity of the requesting node using the algorithm explained in section 4.1. If the requesting node has lower resources than that of the receiving node, the neighbor table is updated and a joinConf is sent to the requesting node. Otherwise, the participation administration forwards the requests to its neighbor that can allow the new node to join the network. When a node receives the joinConf, it updates its neighbor table.

4.4.2 Query (Message) Exchange

The message exchange function handles the exchange of messages between two nodes in the network. When a request message has arrived it is pushed to the messageRequest queue and it is forwarded to its (independent) super-node(s) and is also delivered to the user by performing dataInd service primitive. When an answer to the request message is arrived, the request message is removed from the messageRequest queue and the answer is forwarded to the requesting node.

4.5 Protocol Data Unit

The AmbientDB P2P protocol uses a generic format for the exchange of protocol data units (PDUs). The generic PDU format consists of a PDU type, a source address, a destination address, a time stamp, a set of specific fields and a sequence number. The PDU type must be set each time a PDU is sent.

0	1	5	9	13	2xn+13
Type	Src	Dest	TimeStamp	Specific Fields	Seq. no
1 Octet	4 Octets	4 Octets	2 Octets	2xn Octets	2 Octets

Figure 19: Generic PDU type

In this generic PDU format, PDU type represents the type of the PDU. The PDU type can be a *joinPDU*, a *confPDU*, a *transferPDU*, a *leavePDU*, a *reqPDU*, a *ping*PDU, a *pong*PDU, or an *answerPDU*. A description of each PDU type is presented in this section. The source and destination address designate the originator and the intended consumer of this PDU, a requester time stamp is used to specify the time when the PDU is sent from the sender to the receiver. If the time difference between the generation and consumption of a PDU is not important, we can ignore it. The sequence number is used to uniquely identify the message sent to another protocol entity.

4.5.1 PDU Types

The AmbientDB P2P protocol entities use the following PDU types.

joinPDU: informs the receiving protocol entity that the sending protocol entity wants to join the network. This PDU consists of the available storage space, available network bandwidth, size of data being shared, and the device type of the sending protocol entity.

pingPDU: informs the receiving protocol entity that the sending protocol entity wants to measure the latency. This PDU consists of the time the PDU is sent.

pongPDU: informs the receiving protocol entity that the sending protocol entity has received the *ping*PDU. This PDU

consists of the time the *ping*PDU has originally been sent.

confPDU: informs the receiving protocol entity that the sending protocol entity granted permission to the receiving protocol entity to join the network. This PDU consists of the available storage space of the sending protocol entity, latency between the sending and receiving protocol entities, shared data size, and the status of the sending protocol entity with respect to the receiving protocol entity.

transferPDU: informs the receiving protocol entity that the sending protocol entity wants to use its available storage space. This PDU consists of the data to be transferred from the sending user to the receiving user.

reqPDU: informs the receiving protocol entity that the sending protocol entity request the answer to a query. This PDU consists of a query message from sending user to the receiving user.

answerPDU: answers a protocol entity that has sent a reqPDU. This PDU consists of the answer message from sending user to the receiving user.

leavePDU: informs the receiving protocol entity that the sending protocol entity left the network.

These PDU types are encoded as follows:

joinPDU:

0 1 5 9 11 13 15 17 19

Type	Src	Dest	BW	Storage	Data Size	Device type	Seq. no
0000 0001	4 Octets	4 Octets	2 Octets	2 Octets	2 Octets	2 Octets	2 Octets

confPDU:

0 1 5 9 11 13 15 17 19

Type	Src	Dest	BW	Storage	Data Size	Device type	Seq. no
0000 0010	4 Octets	4 Octets	2 Octets	2 Octets	2 Octets	2 Octets	2 Octets

transferPDU:

0 1 5 9 2 x m+9

Type	Src	Dest	Data	Seq. no
0000 0011	4 Octets	4 Octets	2 x m Octets	2 Octets

reqPDU:

0 1 5 9 2x m+9

Type	Src	Dest	Data	Seq. no
0000 0110	4 Octets	4 Octets	2 x m Octets	2 Octets

respPDU:

0 1 5 2x m+9

Type	Src	Dest	Data	Seq. no
0000 0111	4 Octets	4 Octets	2 x m Octets	2 Octets

leavePDU:

0 1 5 9 11

Type	Src	Dest	Seq. no
0000 1000	4 Octets	4 Octets	2 Octets

pingPDU:

0 1 5 9 11 13

Type	Src	Dest	timeStamp	Seq. no
0000 1001	4 Octets	4 Octets	2 Octets	2 Octets

pongPDU:

0 1 5 9 11 13

Type	Src	Dest	timeStamp	Seq. no
0000 1010	4 Octets	4 Octets	2 Octets	2 Octets

ftpConnPDU:

0 1 5 9 11 13

Type	Src	Dest	DataSize	Seq. no
0000 1011	4 Octets	4 Octets	2 Octets	2 Octets

ftpConfPDU:

0 1 5 9

Type	Src	Dest	Seq. no
0000 1101	4 Octets	4 Octets	2 Octets

concPDU:

0 1 5 9

Type	Src	Dest	Seq. no
0000 1111	4 Octets	4 Octets	2 Octets

confPDU:

0 1 5 9

Type	Src	Dest	Seq. no
0001 0000	4 Octets	4 Octets	2 Octets

The PDU type is represented with binary encoding. For example, the PDU type *ping* is represented by 00001001. The source and the destination addresses are standard IP-addresses. The network bandwidth, storage and data size are represented with binary encoding

of integer values. Similarly, the sequence number and timestamp are encoded is also represented with binary encoding. The device type is represented with binary equivalent of an integer value. For example, static devices are represented by 0000 0000 0000 0001. Data (typically a query and a query answer) are ASCII characters, right aligned and with padding zeros. In all PDU encodings we prescribe that most significant bits are placed in the beginning of each byte.

4.5.2 Protocol Behavior

Each protocol entity maintains a neighbor table, a rescue table and a pointer to its super-node. All of these tables store a set of pairs. Each pair consists of an IP address, bandwidth, latency, available storage space and shared database size of a neighbor. When a user wants to join the network, the AmbientDB P2P protocol exhibits the following behavior:

- each protocol entity that performs a joinReq service primitive sends a joinPDU to destination address established in the joinReq primitive through a sendReq primitive provided by the lower level service, UDP.
- each protocol entity that receives a joinPDU, updates its request queue and sends a pingPDU to the source of this joinPDU.
- each protocol entity that receives a pingPDU sends a pongPDU to the source of this pingPDU.
- each protocol entity that receives a pongPDU, extracts the joinPDU received from the source of this pongPDU from its request queue, updates the time stamp, and either forwards this joinPDU to its neighbor or sends a confPDU to the source of this joinPDU. The joinPDU is forwarded to its neighbor if the neighbor has more resources.
- each protocol entity that receives a confPDU either transfers its data to the sending protocol entity by executing a transferReq primitive, or receives data from another protocol entity by executing a transferInd primitive.
- each protocol entity that performs a transferReq primitive sends a transferPDU to the destination address established in the joinConf primitive through a sendReq primitive.
- each protocol entity that performs a dataReq service primitive sends a reqPDU to the destination of message id established in the dataReq primitive through a dataReq primitive provided by the lower level service.
- each protocol entity that receives a transferPDU via a receiveReq primitive, delivers data to its user through a transferInd primitive.
- each protocol entity that performs a dataResp primitive sends a respPDU to the connectionId established in the dataResp primitive.
- each protocol entity that performs a leave primitive sends a leavePDU to all its neighbors.

- each protocol entity that receive that executes a dataReq requests a connection to the destination address established in the dataReq primitive by executing the connReq service primitive provided by the lower level service.
- each protocol entity that executes a tranferReq primitive requests a ftp connection to the destination address established in the transferReq primitive by executing the ftpConnReq service provided by the lower level service.

Figure 20 shows these rules with an arbitrary instance of behavior.

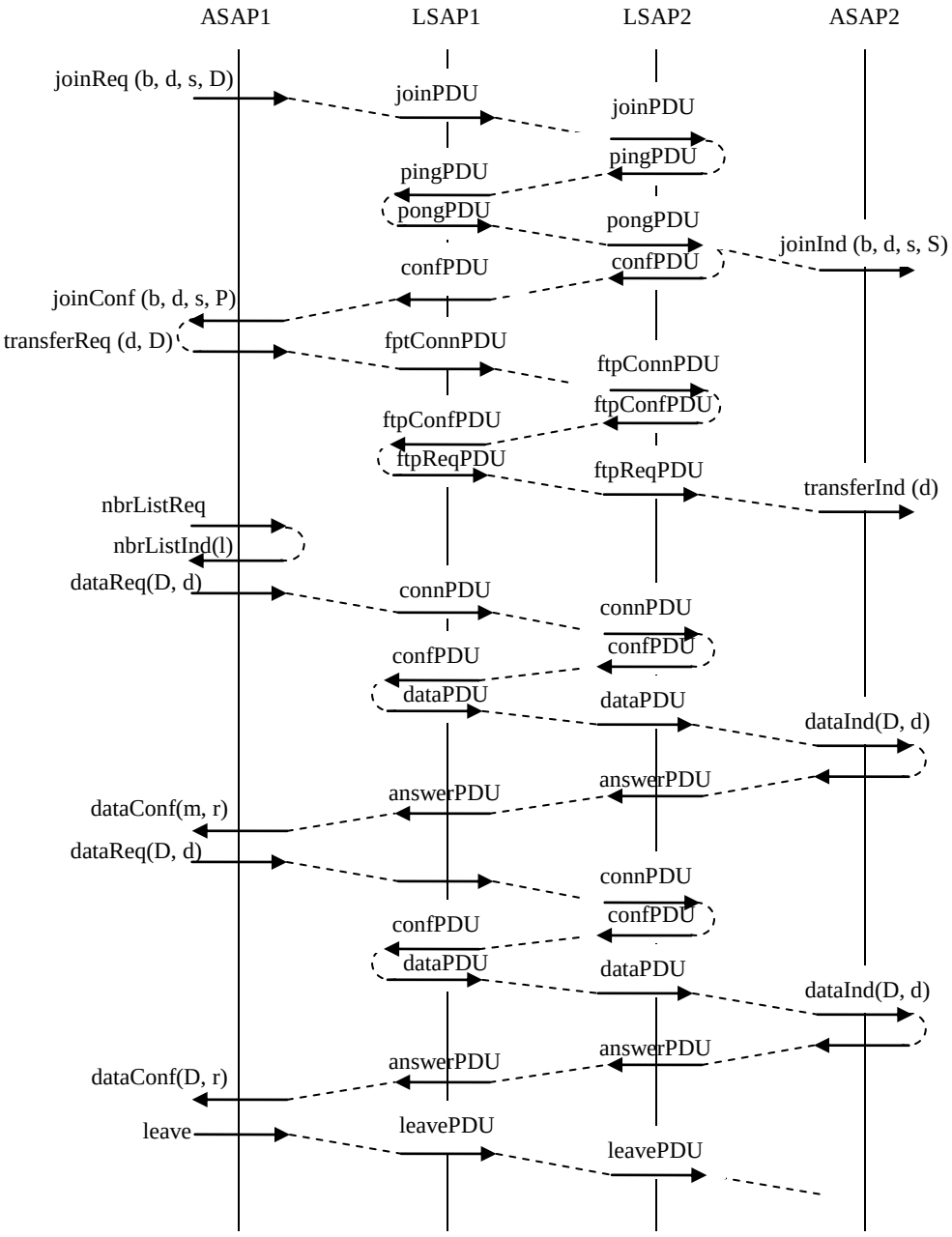


Figure 20: Adb/NP protocol behavior

- a node S at ASAP1 executes a joinReq to another node D with its network bandwidth b, storage space s, and stored data d. A joinPDU is generated and sent to ASAP2 via LSAP1. The protocol entity at node D after receiving a joinPDU generates a pingPDU and sends it back to ASAP1. The protocol entity at node S after receiving the pingPDU responds back with a pongPDU. When the protocol entity at node D receives the pongPDU, a confPDU is generated and sent to the source of this pongPDU. Also, the node D is notified about a new join by executing a joinInd primitive. When the protocol entity at node S receives a confPDU, it notifies the node S of the join by executing a joinConf primitive.
- a node S at ASAP1 executes a transferReq to send data d to node D. A transferPDU is generated and sent to ASAP2 via LSAP1. The protocol entity at node D after receiving a transferPDU, opens a FTP connection and delivers the data to node D through a transferInd.
- a node at ASAP1 executes a nbrListReq primitive to receive a list of its neighbors. A neighbor list, *l*, is generated and delivered to the node S at ASAP1 by executing nbrListInd primitive.
- a node at ASAP1 executes a dataReq primitive to send data d to its neighbor(s). A TCP connection is established (if it does not exist) and a reqPDU is generated and sent over this TCP connection. When a protocol entity at node D receives reqPDU, it delivers the data d to the node D by executing a dataInd primitive.
- a node at ASAP2 executes a dataResp primitive to send the response message r to another node at ASAP1. An answerPDU is generated and sent to another node at ASAP1 over the TCP connection established at dataInd. The protocol entity at node S after receiving an answerPDU delivers the response message r to node S through a dataConf.
- a node at ASAP1 executes a leave primitive. A leavePDU is generated and sent to its neighbor D through LSAP1. When a protocol entity at node D receives a leavePDU it removes S from its neighbor list.

4.5.3 Error Situations

The situations illustrated in Figure 20 consider only normal behavior. However, PDUs sent using the lower level service may get lost. The consequences of losing a PDU are:

joinPDU: if a joinPDU is lost, the destination protocol entity does not know that the source protocol entity has sent the joinPDU. This implies that the join request sent by this protocol entity will never reach the other. The consequence is that the sending protocol entity will wait forever for a confPDU.

pingPDU: if a pingPDU is lost, the destination protocol entity does not know that the source protocol entity has send the pingPDU. This implies that the ping request sent by this protocol entity will never reach the other. The consequence is that the sending protocol entity will wait forever for a pongPDU.

pongPDU: if a pongPDU is lost, the destination protocol entity does not know the existence of pongPDU. The consequence is that this protocol entity will wait forever for a pongPDU.

confPDU: if a confPDU is lost, there is loss of synchronization between two protocol entities. This implies that the joining protocol entity does not receive the confPDU, while the answering protocol entity has the joining protocol entity in its neighbor table. The consequence is that the joining protocol entity is not officially joined in the network and can not send message to another protocol entity. This means that a protocol entity may receive a message from a protocol entity that does not belong to its neighbor table.

leavePDU: if a leave PDU is lost, the destination protocol entity is not informed that the leaving protocol entity has left the network. This means that messages could be sent to protocol entities that no longer participate in the network.

The following protocol functions could be defined in order to recover from the loss of PDUs.

- in order to recover from the loss of a PDU we can use the positive acknowledgement. If a positive acknowledgement is not received in time $t+\Delta t$, we could re-send the PDU. Still there is the possibility that the acknowledgement will be lost. This means that there is the possibility of receiving duplicate PDUs. When a duplicate PDU is received, we can discard the duplicates and re-send a positive acknowledgement. A duplicate PDU can be identified using a sequence number.

- in order to recover from the loss of a leavePDU, we could re-send pingPDUs to all the neighbors from time to time. If neither a positive acknowledgement nor a pongPDU is received, a neighbor can be removed from the neighbor list.

4.5.4 Complete Behavior

The complete behavior of an AmbientDB P2P protocol entity includes some extra rules defined in section 4.5.3 together with the behavior discussed in section 4.5.2.

4.6 Addressing and Initialization

Each node in the AmbientDB P2P networking protocol is identified by its IP address.

All protocol entities are allowed to function independently. This means that a node may or may not want to join the network. If a node does not join the network, then it is sufficient to initialize the protocol entity with a shared database only. If the node wants to join the network, then it must be initialized with the destination address and the shared database. Node initialization without known destination is required if there are no other nodes in the network. It seems that the protocol entity does not need to function when the network is a single node network. However, the initialization is important because other nodes may want to join this node at a later time.

5. Simulation

This chapter introduces the simulation goals and assumptions, simulation setup and simulation environment. In this section, the necessary extension of the ns-2 simulation tool for AmbientDB is also described.

5.1 Simulation Goals, Assumptions and Requirements

To evaluate the complexity and efficiency of the AmbientDB P2P networking protocol (Adb/NP) and compare it with pure Gnutella, we adapted the ns-2 network simulator. ns-2 is an object oriented, event driven network simulator suitable for physical network simulation. The main goal of this simulation is to study and verify the improvements offered by Adb/NP protocol over pure Gnutella. The main improvements expected from the Adb/NP protocol are:

- lower average query response time under a given set of assumptions such as network access, storage space, and stored data, and
- better scalability, especially in heterogeneous physical networks.

To make it possible to simulate the Adb/NP protocol ns-2 needs to provide support for the following:

- measurement of various time factors e.g. average query time.
- creation of hierarchical network topologies, both physical and overlay.
- making routing decisions at each level of the hierarchy.

With the aforementioned goals, we take into account the following assumptions to design the simulation environment:

- the ns-2 nodes connected with ns-2 IP links represent the physical topology.
- the ns-2 agents attached to ns-2 nodes represent the AmbientDB nodes.
- the ns-2 agents connected to each other represent the AmbientDB overlay topology.
- link and node characteristics are read from a configuration file.
- the ns-2 topology file is generated automatically from outside the ns-2 network simulator.
- not all nodes in the physical network participate in the AmbientDB network.
- traffic generated by nodes that do not participate in the AmbientDB network is ignored.

5.2 Simulation Environment

The simulation consists of two different topologies: a physical topology and an overlay topology. The physical topology uses the IP routing mechanisms and other physical level operations. The overlay topology is constructed over the physical topology and uses the Adb/NP protocol. Figure 21 shows both topologies of our simulation.

In the figure, a thick line represents a connection between two overlay nodes and a thin line represents the connection between two physical nodes. In this diagram only three out of eleven physical nodes are participating in the AmbientDB overlay network.

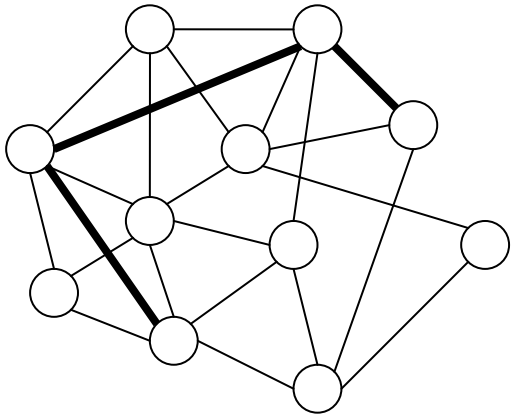


Figure 21: an example simulation environment

We generate the physical topology as a random mesh network topology. This topology is re-used to create the AmbientDB P2P overlay topologies. The AmbientDB P2P overlay topology consists of a subset of randomly chosen nodes from the physical network. We create different overlay topologies with varying number of nodes.

5.3 Simulation Strategy

To study the behavior of the Adb/NP protocol, the simulation is carried out in three different steps: creating both the physical and overlay topologies, initializing the AmbientDB nodes (i.e., network participation), and exchanging messages. Figure 23 shows these simulation steps.

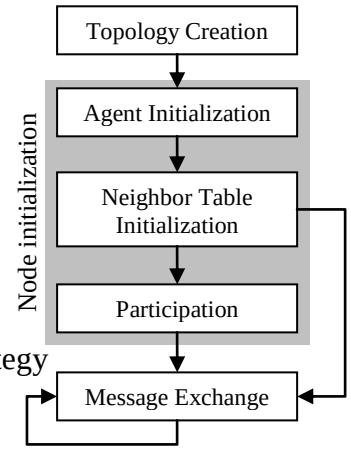


Figure 22: The AmbientDB simulation strategy

5.3.1 Topology Creation

We create a certain physical topology consisting of N nodes. We take a sub-list of M < N nodes that want to participate in an AmbientDB network and a node that is already a part of the AmbientDB overlay network to create the overlay topology. This known node is used as the node that receives the join request from all other joining nodes.

5.3.2 Node Initialization

In this step, we randomly initialize the AmbientDB node with the parameters stored in a node configuration file. These nodes at they join the network update their neighbor table that are empty initially.

5.3.3 Message Exchange

After all M < N nodes participate in the AmbientDB network, each node sends a request message to their (independent) super-node sequentially and receives a corresponding response. In this step, we measure the average query cost. For the simulation purpose we use a query that is as simple as possible, e.g., an aggregate query.

We perform a number of sets of simulation of the Adb/NP protocol. For each set of the simulations the physical and overlay topologies are fixed. However we change the node and link related parameters to observe the behavior of the Adb/NP protocol with increasing heterogeneity and shared data.

5.4 Simulation Setup

External to the ns-2 simulator, we use two simple programs written in C++ to randomly generate physical and overlay topologies. Given a network size N and node and link parameters, the topology generator generates a physical network topology (PT) and stores it in a file. The physical topology file together with overlay size O ($| O | \leq | N |$) and a known node *n* (*n* is supposed to be one of the node in overlay network) is given as an input to the overlay topology generator. The overlay topology generator randomly chooses |O| nodes from the physical topology, generates an overlay network topology (OT) and stores it in a file. This OT file is then used to specify the overlay network for the purpose of simulation. The ns-2 takes the OT file as an input, adds one node at a time to the overlay network and initializes the AmbientDB agents. Figure 24 shows an initialization process. When all the nodes are added to the overlay node, each node sends a request to and receives the answer from its (independent) super-node sequentially. The ns-2 then simulates the query-response operation of the Adb/NP protocol and stores the result (Rs) in a file. We use the Rs file to calculate the average query response time. Figure 23 shows the different states in our simulation setup as described above.

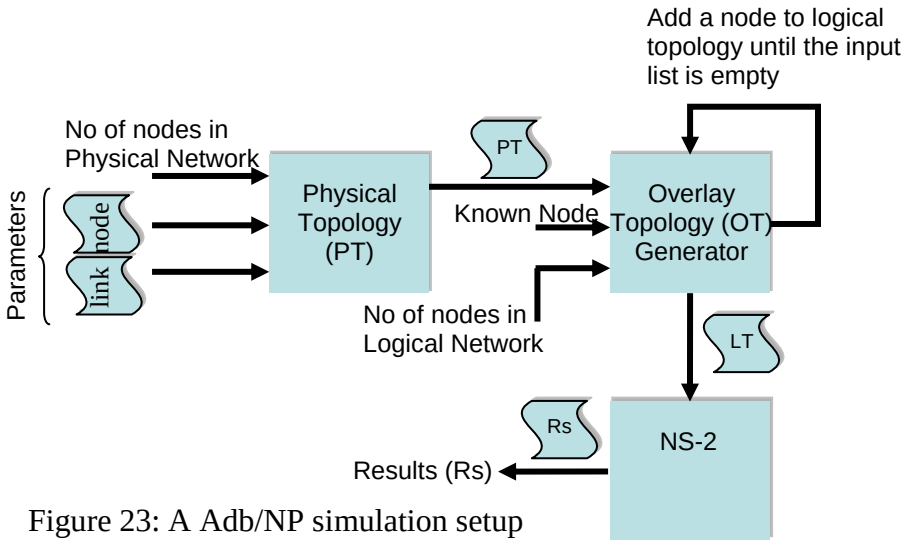


Figure 23: A Adb/NP simulation setup

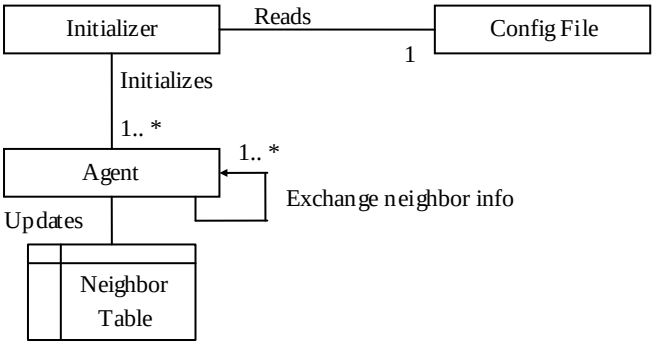


Figure 24: the AmbientDB agent initialization

6. Evaluation

This chapter describes the benchmark parameters used to compare and evaluate the AmbientDB P2P networking protocol with respect to the pure Gnutella protocol. In this section, the performance of the adb/NP is evaluated using our benchmark parameters.

6.1 Cost Metrics

We define cost metrics in order to evaluate the effectiveness of the AmbientDB P2P protocol. We look at cost metrics that directly affect the performance of the AmbientDB P2P protocol.

As defined earlier in 3.3, the query cost is the minimum response time and is calculated as a sum of the transfer and the processing costs. However, the processing cost is dominated by the network cost as a query forwarded to all nodes in the super overlay. Therefore, the query response time is determined mainly by the network cost. The network cost can be calculated in terms of available network latency and bandwidth.

Latency: the network latency is the transmission delay between two nodes.

Bandwidth: the available network bandwidth for each pair of nodes is not symmetric. The incoming bandwidth can be different than the outgoing bandwidth, but for simplicity we assume that it is symmetric. According to our assumption in section 3.3, if a query can be evaluated at a single node, the cost of a given query is not influenced by the available bandwidth. If a query can not be evaluated at a single node, the available bandwidth can influence the query cost of a given query. This is because the available bandwidth can be different between different pairs of nodes, and the query (query-answer) that goes from one node to another can be bigger.

6.2 Benchmark Parameters

We define different benchmark parameters to evaluate the nodes participating in the AmbientDB overlay network. The main parameters we take into account include the node type, their resources, and their participation. These parameters are used to define a node evaluation benchmarking.

Node type: we want to maximize heterogeneity in the network. The ever growing ‘intelligent’ consumer electronic device can also participate in the network. We distinguish the types of the nodes participating in the network as either mobile or static. If the nodes are consumer electronic devices (e.g., 3G phones, PDA, MP3 players, etc) they are considered as mobile devices and other devices like PC, Laptop are considered as static devices.

Participation: devices can join and leave the network at will. This ad-hoc participation makes a variable sized network.

Storage space: we want a query to visit a minimal number of nodes to get the answer. If a node has higher available storage space it can hold the data from other nodes that have lower storage space. The available storage space of any node in the network is one of the parameters to select the candidate super-node for a node that wants to join the network.

Stored data: we want to minimize the query response time by transferring data from lower resource nodes to higher resource nodes. The stored data of a node is one of the parameters to determine if a node can transfer its data to another node.

6.3 Simulation

We compare the performance of the Adb/NP networking protocol and the pure Gnutella protocol through simulation. We run simulations to evaluate different scenarios including home environments (10 nodes physical network) and somewhat Internet like environments (100 nodes physical network). To analyze the performance of the Adb/NP over the pure Gnutella, we performed the following experiments:

- with uniform data distribution over all the nodes.
- with increasing heterogeneity.

From these two different experiments we evaluated the effectiveness of Adb/NP over Gnutella in terms of average query response time, bandwidth consumption, scalability and heterogeneity.

6.3.1 Uniform data distribution

To evaluate the performance of Adb/NP, we created an overlay network where data and queries are distributed uniformly over nodes. We added the nodes in the AmbientDB network using two different strategies:

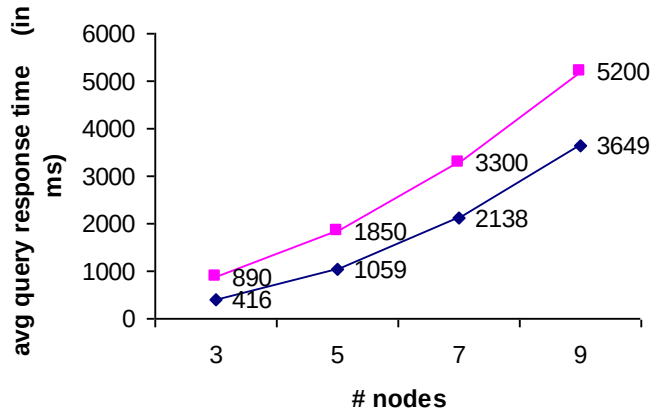
- intelligent join without data transfer from normal nodes to their super-nodes;
- intelligent join with data transfer from normal nodes to their super-nodes.

Using these strategies, we observed the contribution of intelligent join. During the experiment, the simulation ran for two nights, one for each strategy. Through the experiment, we observed that the first strategy maintained on average a three node super-overlay for home environments and a 22 node super-overlay for internet environments. In the home scenario, we used five dynamic nodes and five static nodes. Though the number of nodes in a super-overlay is

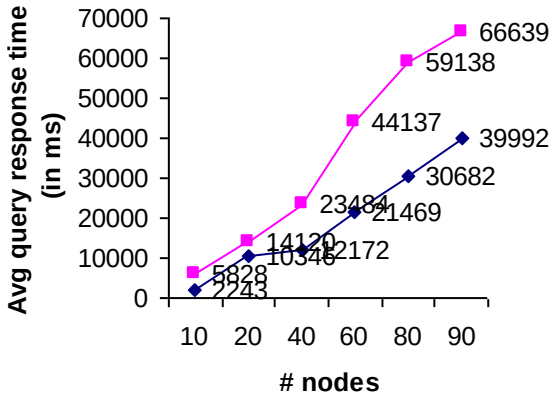
unpredictable as it depends on the type and resources of the node, with our configuration the result is satisfactory.

Experiments with intelligent join without data transfer

The average response time in an AmbientDB network is found increasing with increasing number of nodes, as in pure Gnutella network. However, because the nodes join the network through another node that is in its neighbuorhood, the average query response time is relatively smaller than that in pure Gnutella. Graph below shows the query response time in an AmbientDB vs Gnutella home network.

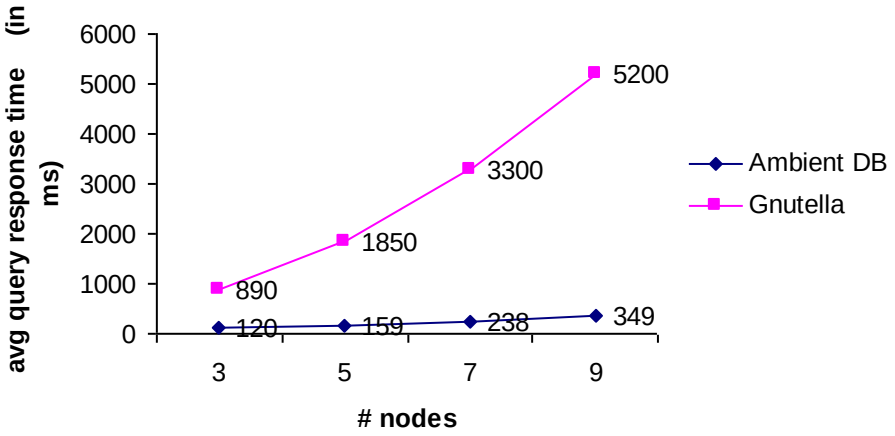


Similarly, the graph below shows the query response time in an AmbientDB vs Gnutella Internet-like network.

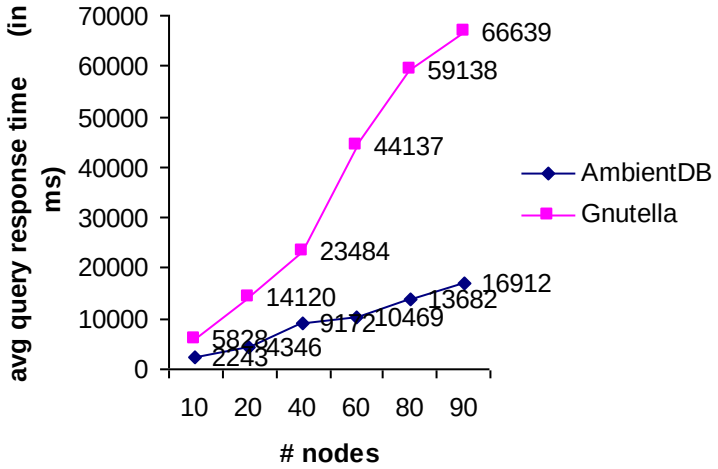


Experiments with intelligent join and data transfer

The average query response time in an AmbientDB is found significantly smaller than that in the pure Gnutella network. Graph below shows the query response time in an AmbientDB vs Gnutella home network.



Similarly, the average query response time in an AmbientDB Internet-like environment is also lower than that in the Gnutella network. The graph below shows the query response time in an AmbientDB vs Gnutella Interne–like network.



The big difference in query response time is because of the size of the super-overlay. In a pure Gnutella network, each query is flooded over the entire overlay network, where as in an AmbientDB network it is flooded only over the super-overlay.

The response time, as stated above, is largely dominated by network time. When a query is asked, only the network latency affects the response time. But when the answer is sent, the size of the message increases in each node it is visited. Therefore, the bandwidth also influences the response time. The bandwidth consumed to execute queries depends on the size of the message.

6.3.2 Increasing heterogeneity

The results of the experiments presented in the previous section are based on the number of nodes and their available resources. While performing the experiments, the number of nodes that participate in the network is varied. Similarly, the type (mobile or static) of each participating node and their resources are changed to ensure the maximum heterogeneity of the nodes. The ratio of the mobile and

static nodes is increased gradually from 50% to 90%. The result presented in the previous section presents the worst case average response time of the AmbientDB network, with node configuration defined by ourselves.

7. Conclusion and Future Work

This chapter concludes the work done in this master thesis and states the future work.

7.1 Conclusion

In this master thesis we designed and simulated an AmbientDB P2P networking protocol. This protocol creates a 'good' overlay network of the AmbientDB nodes as good as possible. The nodes that have more resources work for other nodes that have fewer resources. It transfers the data from fewer resource nodes to the bigger resource nodes, and reduces the number of nodes that process a query. While participating in the network, nodes join through another node that is in a closer latency distance, has larger network bandwidth, larger storage and is 'static'. No mobile node can become a super-node for any other new node no matter whether it is a mobile node or a static node. The AmbientDB P2P protocol can be used with any ad-hoc connected heterogeneous network of consumer electronic devices. It supports the execution of a common global database in an ad-hoc network of heterogeneous devices there by providing lower query response time than the pure Gnutella protocol.

From the simulation result, it is seen that the AmbientDB network does scale with the growing number of nodes (tested up to 100 nodes). It also informs that the query response time is indeed lower than that of the pure Gnutella protocol. The simulation results are indeed as per our expectations. However, the results are based on our own node configurations. The average query response time is influenced by the total number of nodes in the super-overlay. Smaller the super-overlay size, smaller is the average query response time. In our simulation results, the super-overlay size is seen smaller for our node and link configuration.

7.2 Future Work

In this research work, we focused our attention only on the join algorithms and left other parts for future work. The join algorithm, however, takes IP-address as an unique identifier of each user. This assumption does not work in case of shared IP-addresses. For the complete evaluation of this protocol we need to look at failure resilience. Also, we analyzed only the average query cost. For the complete analysis, we need to analyze the join performance and its efficiency. Our simulation result does not incorporate the effect of other loads in the network. In the actual network, the IP traffic might affect the performance of this protocol. The analysis of the protocol in presence of IP traffic is also left for future work.

8. References

[1] Abhishek Gupta, Divyakant Agrawal, and Amr EI Abbadi, *Approximate Range Selection Queries in Peer-to-Peer Systems*, In Proceedings of the 2003 CIDR Conference, California 2003

[2] Beverly Yang and Hector Garcia-Molina, *Comparing Hybrid Peer-to-Peer Systems*, In the VLDB Journal, pages 561-570, September 2001

[3] Baverly Yang and Hector Garcia-Molina, *Designing a Super-Peer Network*, In the Proceedings of the 19th International Conference of Data Engineering (ICDE), Banglore, India, March 2003

[4] C. Greg Plaxton, Rajmohan Rajaraman, Andrea W. Richa, *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*, In proceedings of ACM Symposium on Parallel Algorithms and Architectures, 1997

[5] Chris A. Vissers, L. Ferreira Pires, Dick A.. C. Quartel, Marteen J. v. Sinderen, *The Architectural Design of Distributed Systems, Lecture Notes(265100)*, University of Twente, Enschede, the Netherlands, March 2002

[6] Christopher Kommareddy, Narendar Shankar, Bobby Bhattacharjee, *Finding Close Friends on the Internet*, In the Proceedings of ICNP, November 2001

[7] Entropia, *http://www.entropia.com*

[8] FastTrack – P2P Technology

[9] FreeNet, *http://freenetproject.org*

[10] Gnutelliums, *http://gnutell.wego.com*

[11] Ion Stocia, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, Chord: *A scalable Peer-to-Peer Lookup Service for Internet Applications*, In Proceedings of the 2001 ACM SIGCOMM Conference, pages 149-160, 2001

[12] Jens Mache, Melaine Gilbert, Jason Guchereau, Jeff Lesh, Felix Ramli and Matthew Wilkinson, *Request Algorithms in Freenet-style Peer-to-Peer Systems*, In the Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P'02)

[13] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Partick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao, *OceanStore: An Architecture for Global-Scale Persistent Storage*, In Proceeding of ACM ASPLOS, ACM November 2000

[14] Jordan Ritter, *Why Gnutella Can't Scale. No, Really*, http://www.darkridge.com/~jpr5/doc/gnutella.html

[15] KaZaA, *http://www.kazaa.com*

[16] L. Ferreira Pires, Dick A. C. Quartel, *Protocol Engineering, Lecture Notes (214004)*, University of Twente, Enschede, the Netherlands, August 2001

[17] Napster, *http://www.napster.com*

[18] NS-2 manual, http://www.isi.edu/nsnam/ns/doc

[19] O'Reilly P2P Directory, *http://open2p.com/pub/q/p2p_category*

[20] Peer-to-Peer Working Group, *http://www.peer-to-peer.org*

[21] Peter Boncz, Caspar Treijtel, *AmbientDB: Relational Query Processing in a P2P Network*, Technical Report INS-R0305, CWI, Amsterdam, June 2003

[22] Peter Boncz and Caspar Treijtel, *AmbientDB: Relational Query Processing over P2P Network*, In proceedings of the International Workshop on Databases,

Information Systems and Peer-to-Peer Computing, Humboldt University, Berlin, Germany, September 8, 2003

[23] Peter Triantafillou, Chrysanni Xiruhaki, Manolis Koubarakis, Kikolaos Ntarmos, *Towards High Performance Peer-to-Peer Content and Resource Sharing Systems*, In Proceedings of the 2003 CIDR Conference

[24] Promoting Ambient Intelligence, *http://www.ambientintelligence.net*

[25] Qin Lv, Sylvia Ratnasamy, and Scott Shanker, *Can Heterogeneity Make Gnutella Scalable?* In Proceedings of the First International Workshop on Peer-to-Peer Systems 2002

[26] Rivka Ladin, Barbara Liskov, Liuba Shira, and Sanjay Ghemewat, *Providing High Availability Using Lazy Replication*, ACM Transaction on Computer Systems, 10(4): 360-391, 1992

[27] S. A. Thomas, *IPng and the TCP/IP protocols*, John Wiley & Sons, Inc. USA, 1996

[28] SETI@HOME, *http://www.setiathome.ssl.berkeley.edu*

[29] Stafen Sariou, P Krishna Gummadi, and Steven D. Gribble, *A Measurement Study of Peer-to-Peer File Sharing Systems*, In Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA, January

[30] Steven Gribble, Alon Halevy, Zachery Ives, Maya Rodrig, and Dan Suciu, *What Can Peer-to-Peer do for Databases, and Vice Versa?* In Proceedings of the Fourth International Workshop on the Web and Databases(WebDB 2001), Santa Barbara, California, USA, May 2001

[31] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shanker, *A Scalable Content Addressable Network*, In Proceedings of SIGCOM 2001, San Diego, August 2001

[32] United Devices, *http://www.ud.com*

9. Appendix I

1. AmbientDB P2P Network, Peer Comparing Algorithm

We assume that the participants in the AmbientDB P2P Network are very heterogeneous in terms of their resources. We define the devices like 3G phone, PDA, MP3 Players, etc as a mobile device and the devices like PC, Laptop, etc as a static device. Let *m* and *s* denote the mobile node and the static node respectively. Also, we assume that participants wish to share as much resources as possible along the path of higher available network bandwidth and lower network latency. The network bandwidth and the storage space play an important role to distinguish the strength of a participant. We give the network bandwidth a higher priority and the storage space the lowest, because the queries are flooded between the super-nodes. The mobile nodes are highly dynamic and it is unlikely that they have large storage space therefore they can not be super-node nodes. As there are two possible types for each device, we can have four different combinations: *mm*, *ms*, *sm*, *ss* in the order of *p* and *q*, where *p* and *q* are a new and an existing node respectively. If both of *p* and *q* are of type *m*, then they can neither be a super-node nor a simple node of each other, so they remain independent to each other.

```
f_c (p, q)
1.    if p.t = m and q.t = m  // if both p and q are mobile, no one
                              // can be a super-node
2.       return independent
3.    end if
```

On the other hand if either *p* or *q* is *m*, it is sufficient for a node of type *s* to be a super-node of another node of type *m*, if it can hold the data of another node of type *m*. If that is not the case then they remain independent of each other.

```
4.    if p.t = m and q.t = s        // if p is mobile, check if q can
hold p's data
5.       if q.s ≥ p.d
6.          return super-node
7.       else
8.          return independent      // if q can not hold p's data
                                    // then they become independent
9.       end if                     // as a static node q can not
                                    // become a slave of a mobile
                                    // node p
10.      end if
11.         if q.t = m and p.t = s  // if q is mobile, check if p can
                                    // hold q's data
12.         if p.s ≥ q.d
13.               return slave
14.         else
15.               return independent // if p can not hold q's data
                                      // then they become independent
16.         end if                    // as a static node p can not be
                                      // slave of a mobile node q
17.      end if
```

In case if both p and q are both of type s, then one can become the super-node of other if it has higher or equal bandwidth of the other and it has sufficient space to accommodate other's data. In other case, they also remain independent of each other.

```
18.         if q.t = s and p.t = s    // if both p and q are static,
                                      // they are equally likely to
19.         if q.b ≥ p.b and q.s ≥ p.d  // become a super-node of
                                      // each other iff one has
                                      // higher or equal bandwidth
20.               return super-node   // and sufficient storage
                                      // space to hold other's data
21.               end if
22.         if p.b < q.b and p.s < p.d
23.               return slave
24.         end if
25.               return independent
26.         end if
```

In peer evaluation algorithm explained above, other parameter's such as: memory size and processor speed are not considered as overall query processing time will be dominated by network infrastructure.

2. AmbientDB P2P Network, Join Algorithm

We assume that a new node *p*, by some means, knows at least one existing node *q* in the AmbientDB P2P Network. Also, we assume that there exists a cost function $f_{c,}$ as explained above. Given the parameters of two nodes *p* and *q* cost function f_c returns either slave, or independent or super-node representing *p* happens to be the super-node of *q*, or *p* and *q* are similar, or *q* happens to be the super-node of *p* respectively. Furthermore, we assume that every node in the AmbientDB P2P network has the following state variables.

```
status ∈ {super-node, slave, independent, void}, initially void
N: neighbor list, (initially null for new participant), sorted in
ascending order on latency
N[i].status: status of i^th neighbor {slave, independent}; i ≥ 0
b: super-node, initially null
l: latency, initially zero
s: storage space
t: device type
c: bandwidth
d: data size
temp: temporary super-node, initially null
f: request, f ∈ {true, false} // if the request is new
k: temporary variable, initially null
o: originator, initially self
```

In this algorithm, we assume that it is the responsibility of the network to find a place to join for a new participant. It is important to note that each participant in the network can be a super-node, a slave or an independent super-node for a new participant. When an existing node receives a join request from a new node, it measures the latency between the new node and itself.

```
request_super-node (p, q)
1. if p.f
2.    p.l = echo (p, q), p.f = false // get latency between p and q
3. else
4.    k = echo (p, q)
5. end if
6. if k < p.l
7.    p.l = k
```

After measuring the latency, it checks the aforementioned possibilities. When it finds that it can be a super-node for the new participant, then we assume that there is still a possibilities that its neighbors can still be a super-node for the new participant. The node that receives the join request appends its address at the start of the originator's *originator* variable and forwards the join request to its neighbors who it sees as a candidate super-node for a new node and waits for the response. As soon as it gets the response from the neighbor it checks, if the neighbor is willing to be a super-node for the new node or not. If that is the case, it simply replies back the originator that the super-node has been found. If the neighbor is not willing to be a super-node, then it forwards the same message to another neighbor. This process continues until there are still some other potential super-nodes for new participant in the receiver's neighbor list.

```
8.    switch f_c (p, q)
9.      case: super-node       // q is a super-node for p
10.         if N is not null
11.              for i:= 0 to |N|-1 // check if my neighbors can
                                    // become a super-node for p
12.                 if N[i].l ≤ 2*p.l and  f_c(p, N[i]) == super-node
13.                      p.l = k
14.                      p.o = q concat p.o
15.                   temp = request_super-node(p, N[i])
16.                   if (temp.status = super-node)
17.                   break for
18.                 end if
19.             end if
20            end for
21.         else
22.             temp = q
23.             temp.status = super-node //q is a super-node of p
24.         end if
25        if length (p.o = (p.o – q)) = 1
26.             p.status = slave
27.             q.s = q.s – p.d
28.           N.addElement (p) // add p in neighbor list as slave
29        end if
30.         return temp to p.o  // return to sender
```

If the receiver finds that the responding node can not be a super-node but a slave node for a new node, then still we assume that the super-node of receiver could also be closer to the new node therefore, if it happens to be so, the receiver simply forwards the join request to its super-node appending its address at the start of the originator's *originator* variable. If the receiving node finds that its super-node is not near to the requester, then it informs the requester that it can be a slave for the new node but the new node can join to the receiver's super-node as well.

```
31.         case: slave
32.         if b is not null and b.l ≤ 2*p.l
33.            p.l = k
34.             p.o = q concat p.o
35.             request_super-node(p, b)
36.             exit
37.          endif
38.          if b is not null and b.l > 2*p.l
39.             return b    // p can join b
40.             p.b = q     // p becomes my super-node
41.             inform b that p left
42.             inform q that p is slave for q
43.             send data to q
44.             exit
45.          end if
```

```
46.               temp q
47.               temp.status = slave
48.               return temp
49.               send data to q
50.               exit
```

Similarly, if the receiver finds that they are independent super-nodes, then the receiver may think that its super-node can be a super-node, so it forwards the request to the super-node. If it thinks that its super-node is not near to the sender, then it simply send the requester that they can remain independent super-node of each other.

```
51.          case: independent
52.               repeat steps 31 -34
53.               temp = q
54.               temp.status = independent
55.               p.status = independent
56.               N.addElement (p)
57.               return temp
58.           end if
59.               exit
```

3. AmbientDB P2P Network, Peer Joining Alternate Algorithm I

In this alternative algorithm, we add an extra assumption than in the previous algorithm. Each node also has a temporary potential super-node list. And it uses the breadth first search (BFS) algorithm to find the potential super-nodes. In this algorithm, instead of sending the response back to the intermediate node, each node directly response to the originator node of the join request.

```
status ∈ {super-node, slave, independent, void}, initially void
N: neighbor list, (initially null for new participant), sorted in
ascending order on latency
N[i].status: status of i^th neighbor {slave, independent}; i ≥ 0
b: super-node, initially null
l: latency, initially zero
s: storage space
t: device type
c: bandwidth
d: data size
temp: temporary super-node, initially null
f: request, f ∈ {true, false} // if the request is new
k: temporary variable, initially null
N_t: temporary potential super-node list, initially null
```

As in the previous algorithm, when a node receives a join request it checks several possibilities. If it finds that it can become a super-node for a new node then checks its neighbors it they can become a super-node for a new node and creates a temporary list of all of potential super-node for a new node. The receiving node then forwards this join request to all nodes in the temporary list, also informs the new node that it can also become a super-node. After receiving the response from all the potential super-nodes, the new participant chooses the best one as its super-node and joins the network. This process however creates a lot of network traffic but reduces the extra burden of keeping pending message list at all receiving participants.

```
request_super-node (p, q)
1. if p.f == true
2.    p.l = echo (p, q), p.f = false // get latency between p and q
3. else
4.    k = echo (p, q)
```

```
5. end if
6. if k < p.l
7.    p.l = k
8.    switch f_c (p, q)
9.      case: super-node       // q is a super-node for p
10.          if N is not null
11.               for i:= 0 to |N|-1 // check if my neighbors can
                                     // become a super-node for p
12.                 if N[i].l ≤ 2*p.l and  f_c(p, N[i]) == super-node
13.                     N_t[i] = N[i]
14                       break for
15.                 end if
16.             forward message to all in N_t
17.          end if
18.          temp = q
19.          temp.status = super-node
20.           return temp to p.o            // return to sender
```

If the receiving node can only become a slave for a new node, we assume that it is still possible that the super-node of the receiving node can become a super-node for the new participant. To check this possibility, the receiving participant forwards this join request to its super-node as a new message from the participant. Furthermore, if the receiving node finds that the new node is nearer from it than its existing super-node, then it simply changes its super-node and informs its existing super-node that it changed the super-node.

```
21.          case: slave
22.          if b is not null and b.l ≤ 2*p.l
23.              p.f = true
24.               forward message to b
25.               if b.l > k
26.                     inform b that q changed super-node
27.                     b = p
28.                     inform p that q is slave for p
29.                     send data to p
30.               end if
31.           end if
32.           if b is not null and b.l > 2*p.l
33.               inform p that b is super-node for p
34.               inform p that q is slave for p
35.               inform b that q changed the super-node
36.               b = p
37.               send data to p
38.               exit
39.           end if
40.               temp q
41.               temp.status = slave
42.               return temp
43.               send data to q
44.               exit
```

Similarly, if the receiver can neither become a super-node nor a slave for a new node, we assume that it is still possible that the super-node of the receiving node can become a super-node for the new node. The receiving node checks if the super-node can become a super-node of a new node and is in nearby location, then it forwards this join message to its super-node. If the super-node is found far from the new node, then the receiving node informs the new node that they can remain independent super-node for each other. This means that they do not transfer data to each other.

```
45.          case: independent
46.               repeat steps 22 -31
47.               temp = q
```

```
48.                temp.status = independent
49.                p.status = independent
50.                N.addElement (p)
51.                return temp
52.           end if
53.                exit
```

This algorithm requires an extra management from all the super-node nodes. When a neighbor changes its super-node, then the previous super-node has to remove the migrating neighbor from its neighbor list and the corresponding data from its database. This requires an additional network overhead of data transfer. This involves the freedom of selecting a super-node dynamically and the total network cost. That might ultimately increase the query response time.

4. AmbientDB P2P Network, Peer Joining Alternate Algorithm II

As in the previous algorithms, the initial assumptions in this algorithm are same.

```
status ∈ {super-node, slave, independent, void}, initially void
N: neighbor list, (initially null for new participant), sorted in
ascending order on latency
N[i].status: status of i^th neighbor {slave, independent}; i ≥ 0
b: super-node, initially null
l: latency, initially zero
s: storage space
t: device type
c: bandwidth
d: data size
temp: temporary super-node, initially null
f: request, f ∈ {true, false} // if the request is new
k: temporary variable, initially null
N_t: temporary potential super-node list, initially null
```

In this algorithm, we assume it is the responsibility of the participating node *p* to find and join at a suitable place in the network. To find a suitable place, a participating node first makes a temporary connection with the known participant *q*. After making a temporary connection with *q*, *p* requests for the list of all participants that are directly connected with *q* including *q*. When *p* receives the list of all the participants connected with *q*, it filters them according to their available resources. It then measures the latency between those selected participants and itself and selects the best one as its place to join the network.

With the aforementioned assumptions, following algorithm is used to find a so-called super-node for a new participant *p*.

```
request_super-node (p, q)
while temp is null
1.    join temporarily to q
2.    N = ask q for its neighbor list including q
3.      for i:= 0 to |N|-1     // check if neighbors can become a
                               // super-node for p
4.            if N[i].l ≤ 2*p.l and  f_c(p, N[i]) != slave
5.               N_t[i] = N[i]
6.            N_t[i].l = echo (N_t[i])
7.         end if
8.      end for
9.      if N_t is not null
8.         for i:=0 to |N_t|-1
```

```
9.          temp = N_t [i] such that latency is minimum
10.         end for
11.         join temp
12.         transfer data to temp
13.       end if
14.          if N_t is null
15.         q = super-node of q
repeat
```

This algorithm, though looks simple, requires more than two connections to join the network. This multiple connections introduce high network traffic.

10. Appendix II

10.1 Network Simulator-2 (ns-2)

The network simulator 2 (ns-2) is a discrete object oriented event driven simulator [18] designed to simulate the network behaviors. The ns-2 provides supports the simulation of TCP, routing, and multicast protocols over wired and wireless networks. It also supports the simulation of several ad hoc routing protocols and propagation models except cellular phones, data diffusion and satellite networks.

The ns-2 translates physical activities into events and time advances as the events are processed. It allows creating nodes, building routes, monitoring events, and generating traffic. Because of these features, ns-2 has been chosen as a simulation platform to simulate the Adb/NP protocol. ns-2 is written in C++ with an OTcl interpreter as a front end. It supports a class hierarchy in C++ and a similar class hierarchy within the Otcl interpreter. These two class hierarchies are closely related to each other and can communicate with each other via Tclcl. The ns-2 Tcl interpreter structure can be seen as follows:

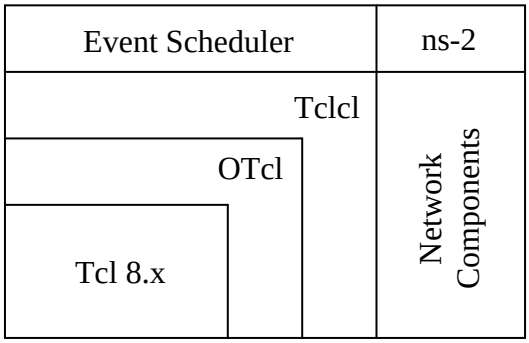


Figure 25: The ns-2 structure

The event scheduler schedules the queued events to be processed by the simulator. Otcl is an object oriented support for tcl, and tclcl is used to glue the tcl objects and C++ objects and vice versa. The network components consist of the basic network components like nodes, links, routers, etc.

All control operations including topology creation are implemented mostly in Otcl and other core components are implemented in C++. To simulate the Adb/NP protocol, we extend both the C++ and OTcl classes, particularly ns-lib.tcl and agent classes.

Figure 26 shows an ns-2 class hierarchy.

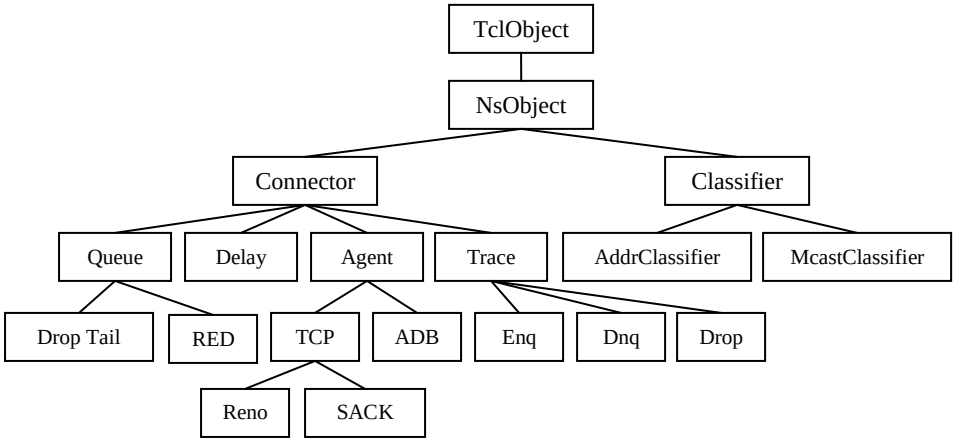


Figure 26: An ns-2 class hierarchy

The root of any simulation is TclObject. Users create simulator objects through the interpreter. These objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the C++ class hierarchy. The C++ class hierarchy is automatically instantiated through the methods defined in the class TclClass as in the following figure:

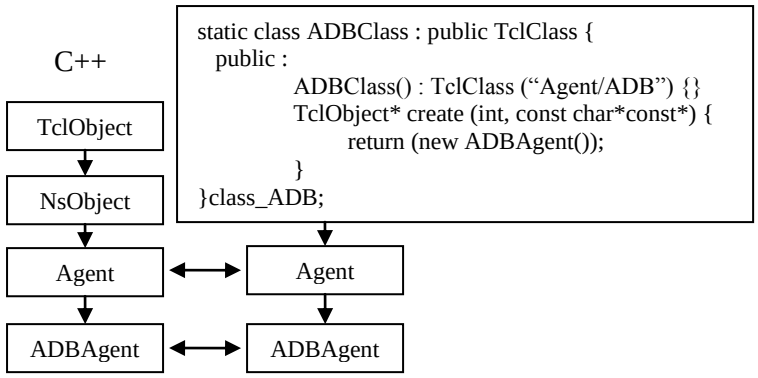


Figure 27: TclClass instantiation

10.1.1 Ns-2 Node

The figure below shows the internal structure of a simple node in ns-2. Each node consists of two classifiers: port and address classifiers. These classifiers are used to do the routing in the network topology. Besides routing, the classifiers can store the pointers to another classifier, agent, link, etc. The address classifier routes the packet to the right link or to the port classifier depending on the target address contained in the received packet. The address consists of the node number and the agent id also called as port number.

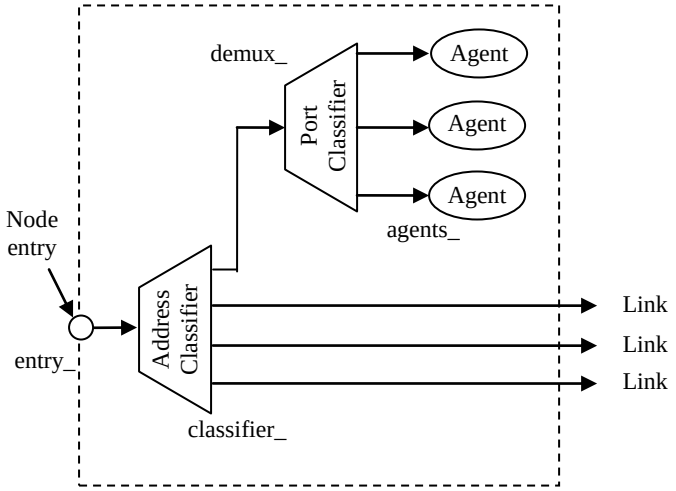


Figure 28: ns-2 node structure

10.1.2 Ns-2 Link

The ns-2 link represents the physical link of the physical network. Figure below shows the internal structure of a simple point to point link in ns-2. A simple link consists of a sequence of connecters. The connectors are linked together via their target_ pointer. Connectors generate data for one recipient, either the packet is delivered to the next connector or it is dropped from the link. In both cases the connector can produce new events and tell the scheduler to insert it in the event queue.

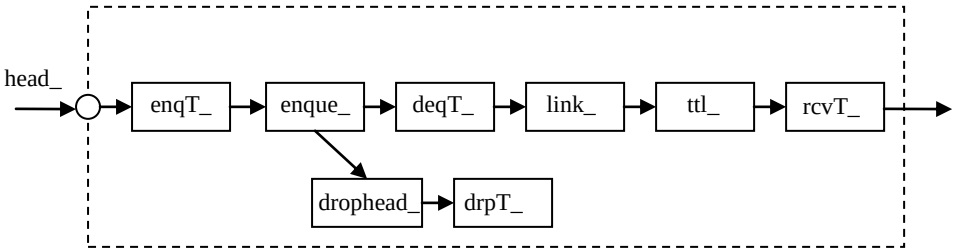


Figure 29: ns-2 link structure

Where,

enqT_ = reference to the enqueue trace. All trace connectors aim at producing one line of the output nam file.

queue_ = reference to the queue manager. It mainly forwards the packets from the enqueue trace to the packet queue or from the packet queue to the dequeue trace

drophead_ = reference to the packet queue. It mainly stores the waiting packets and drops some packets when it is full

drpT_ = reference to the drop trace. It traces the packets that are dropped

deq_ = reference to the dequeue trace. It traces the packets that are dequeued and will continue to the dalay connector.

link_ = reference to link delay and bandwidth. Given a packet, the delay connector schedule a new event which will occur in ttl checker at current time + delay of the link

ttl_ = reference to ttl checker. Every packet has a time-to-live variable. Every time the packet goes through a link it is decreased. If it comes to be null, the packet is dropped.

rcvT_ = reference to receive trace. It traces the arrival of packets in a node.

10.1.3 Ns-2 Agent

The ns-2 agent represents the end points where network-layer packets are consumed or produced. The ns-2 agents contain methods to create new packets, receive new packets and subclass specific time out methods. These agents can be connected with nodes. Agents also allow including extra information that is needed for any application specific simulation.

10.1.4 Ns-2 Header

To exchange information between two communicating agents ns-2 Packets are used. ns-2 consists of header fields but no data. Thus, the header field forms the main ingredient of ns-2 Packet. They are abstract class and are defined for each new application according to their requirements. Data needed to be exchanged between two communicating entities are exchanged as header fields.

10.1.5 Otcl Library

The Otcl library contains all the methods required to configure nodes, links, agents, etc to initialize the whole topology at ns initialization. As the topology objects are C++ objects and are instantiated using the create function of the TclClass class, the methods in Otcl library are required to link C++ and Tcl hierarchies.

10.2 Ns-2 extension for AmbientDB

In order to simulate the Adb/NP protocol behavior, we extend the ns-2 agents to ADBAgents to represent AmbientDB nodes. The extension includes extra node parameters. These parameters include bw_, hdd_, data_, and lat_ representing network bandwidth the node is currently using, its available storage, stored data, and the network latency between two nodes. Also we define our new header types so as to transfer node specific parameters between communicating nodes when necessary. Details about necessary extensions are explained in the following sub sections.

10.2.1 C++ extension

ADB Agent is an extension of the agent class. It consists of the additional node parameters, methods to receive and send the packets, and the method to analyze the packet. Following picture depicts the ADB Agent structure.

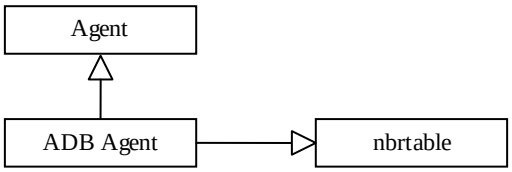


Figure 30: AmbientDB agent inheritance structure

The ADB Agent inherits the properties from Agent and nbrtable classes. The nbrtable class provides methods to build and manipulate a neighbor table. Beside this, the ADB Agent provides methods to access the received packet header, its member variables, send and receive the packets. The ADB Agents uses the neighbor table to keep track of its neighbors in the Peer-to-Peer overlay topology. The following picture depicts the internal structure of the ADB Agent:

AmbientDB Agent

```
public:
    command () // this method is used as a hook to execute methods
    forward() // method to forward packet
    recv() // method to receive packet
protected:
    neighbor table* table_
    myId_, myBw_, myData_, myHdd_, myType_
```

Figure: AmbientDB agent example structure

In the ns-2 model of the Adb/NP protocol, agents are attached to nodes and work whenever there is some information flow between nodes. The picture below shows the relationships between nodes and gents.

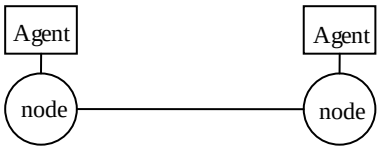


Figure 31: Node-agent relationship in ns-2

10.2.2 Otcl extension: ns-lib.tcl

The ns-lib.tcl contains all the methods required to configure nodes, links, agents, etc. One extra method has been added in ns-lib.tcl to configure the agents. This method could be invoked from nsObject.

10.2.3 New Header Type

The ns-2 Packets are combination of header fields but no data. It is not possible to include data in ns-2 Packets. As we want to compare different characteristic of the nodes, a new header type hdr_adb has been defined to include the node specific parameters: network bandwidth, storage space, stored data and device type.