Journal of Computer and Communications, 2015, 3, 30-39
Published Online April 2015 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2015.34004

A Load Service System with Reputation Proof in P2P Networks

Ming-Chang Huang

Department of Business Information Systems & Operations Management, University of North Carolina at Charlotte, Charlotte, USA
Email: mhuang5@uncc.edu

Received 19 March 2015; accepted 9 April 2015; published 10 April 2015

Abstract

In this paper, a new system design for load services in computer networks with a new reputation system is constructed. The use of the reputation system is to address the free-rider problem. Database systems are used by directory agents to save information provided by load-server agents. Protocols are built for how a host finds available servers for load service or load transfer, especially when a host moves to a new region. Detailed procedures include how a directory agent builds its database, how a load-server agent provides services, and how a load-client agent receives its desired services. The system uses the fuzzy logic control method to transfer loads for load balancing, instead of the method of a fixed threshold level. For an ad-hoc (wireless) computer network framework, this new system structure is aimed to provide efficient ways for hosts to communicate with one another and to access resources in the system. This will also help the users of networks locate resources in a most effective and secure manner.

Keywords

Load Service, Peer-to-Peer Network, Directory Agent, Load-Server Agent, Load-Client Agent, Reputation System

1. Introduction

Computer networks provide parallel computation and services. It is important that hosts send their loads to other hosts for certain function implementation through network transfer. With the increasing popularity of mobile communications and computing, the demand for load services and load balancing grows significantly. When a computer is overloaded or needs special services, it sends requests to other computers for load transfer or load services. For example, a computer may send requests to other computers when it receives certain jobs that require a higher quality of service or a shorter time that its processor is able to provide. Since wireless networks have been widely used in recent years, how a host transfers its loads to other nodes has becomes a very important issue because not all wireless hosts have the ability to manipulate all their loads. For instance, a host with low battery power cannot finish all its jobs on time and should transfer some of them to other hosts. Currently, most of load balancing algorithms are based on wired network environments. It is important to find an efficient way for load service and load transfer purposes in wireless environments.

Before a wireless host transfers its loads to other hosts or asks for load services from other hosts, it needs to find available hosts using resource allocation algorithms. Several resource allocation protocols have been developed: for example, the IEFT Service Location Protocol (SLP) [1] and Jini [2] software package from Microsystems. However, these protocols address how to find resources in wired networks, not in wireless networks. Maab [3] develops a location information server for location-aware applications based on the X.500 directory service and the lightweight directory access protocol LDAP [4]. However, it does not address certain important issues regarding the movements of mobile hosts such as how to generate a new directory service, how a host receives new services, and when a directory agent moves away from its original region. In an ad-hoc network, system structure is dynamic and hosts can join or leave any time. Therefore, how to provide load services and how to find available hosts to fulfill load service requests are valuable topics in an ad-hoc network system.

To find a host which can fulfill a load service request, the requesting host needs to ensure that the host under consideration has good reputation in load services. Hosts with good reputation share not only request resources from other hosts, but also share their resources with others. It is called the "free-riding" problem if a host requests resources from other hosts without sharing its resources with others. Measurement study of the free-riding problem on Gnutella was first reported by [5] in 2000, which indicated that approximately 70% of Gnutella users did not share files with other users and nearly 50% of the queries were responded by the top 1% peers. However, according to the most recent measurement study, the percentage of free-riders rises to 85% [6]. It is highly possible that a small number of peers who are willing to share information handle most of the job loadings in P2P networks. As a result, the prevalence of free riders may eventually downgrade the performance of the entire system and make the system vulnerable [7].

In this paper, I construct a system structure for load services with reputation checking in wireless ad-hoc network systems using the peer-to-peer concept [8] [9]. In ad-hoc network systems, hosts move dynamically without base stations for communication. The load service architecture provides special services upon requests from hosts and services such as resource location and load balancing. A host may send special requests to other hosts for load services or send its loads for load balancing. The requests include service types the host needs or the amount of loads to be sent to other hosts. For special service requests, the host should define the conditions on which other hosts may accept the services. For example, the request includes the price of job execution, the limit requirement of execution time, etc. Besides looking for the desired resources, the requesting host should also check the requested host's reputation to avoid the "free-riding" problem [10].

In Section 2, I discuss the system structure. Section 3 describes the details of the method. Sections 4 and 5 illustrate the information format for databases and scalability respectively. Section 6 presents the conclusion. Section 7 addresses the future work for this research.

2. System Structures

There are three components in my load service system—load-server agent, load-client agent, and directory agent. A load-server agent provides load services that are queried by other hosts (load-client agents) which require load services. These agents post the types of services periodically to their directory agents to update the services they can provide to load-client agents. A load-client agent is a host in the network that may be in need of services performed by other hosts. It sends requests to its directory agents to ask for services from load-server agents when it is heavily loaded or it needs special services that it does not have the ability to perform. A directory agent forms groups for the load-server agents and load-client agents respectively and builds a database for service queries from load-client agents.

Figure 1 shows an example based on the architecture of my load service system. **Figure 2** shows the structure of the reputation system (FuzRep) [10] that is applied in the load service system. Each directory agent has a query database, which stores all the query information from load-server agents. Load-server agents and load-client agents may join directory agents upon requests. In **Figure 1**, for example, Load-server Agent 1 and

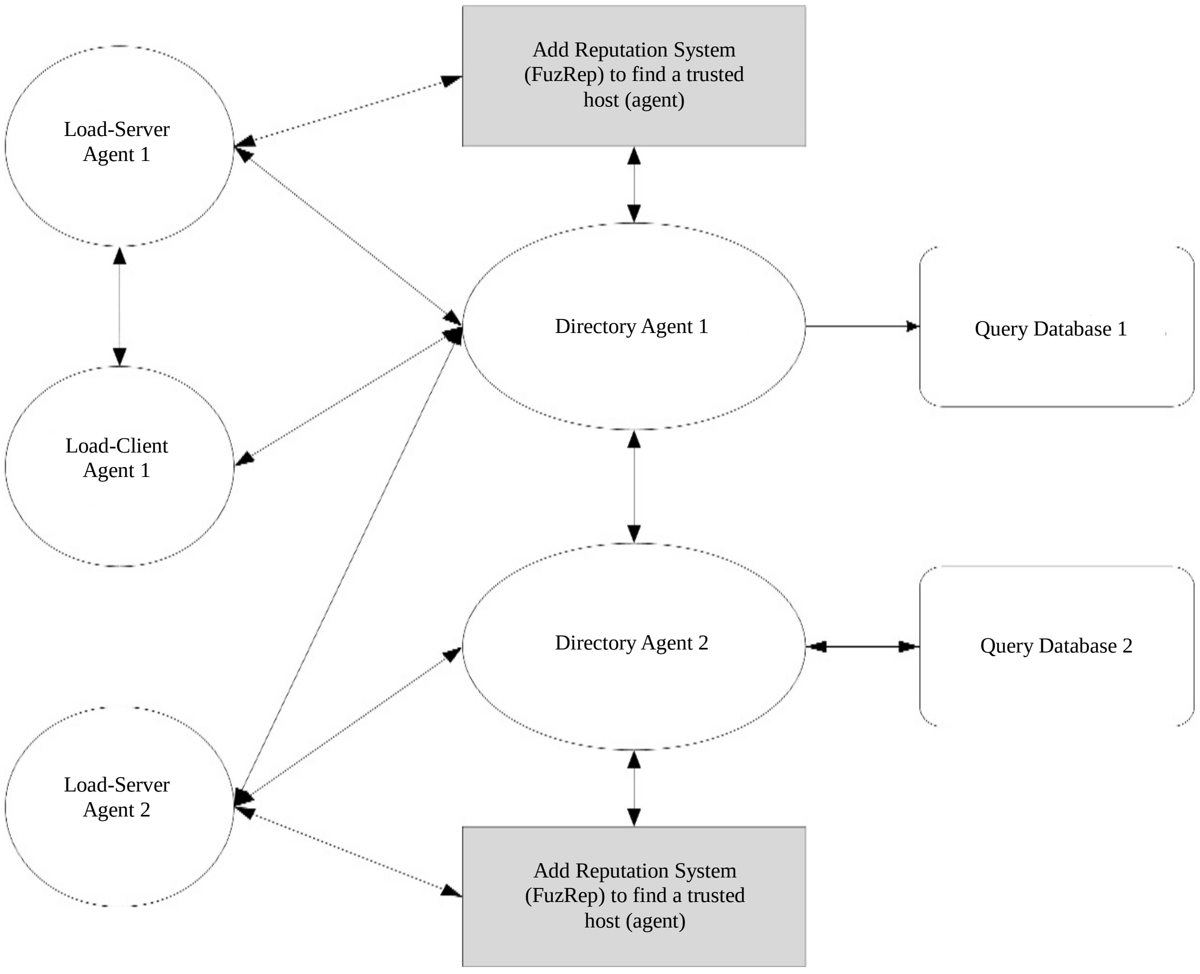

Figure 1. Load service system architecture with FuzRep reputation system.

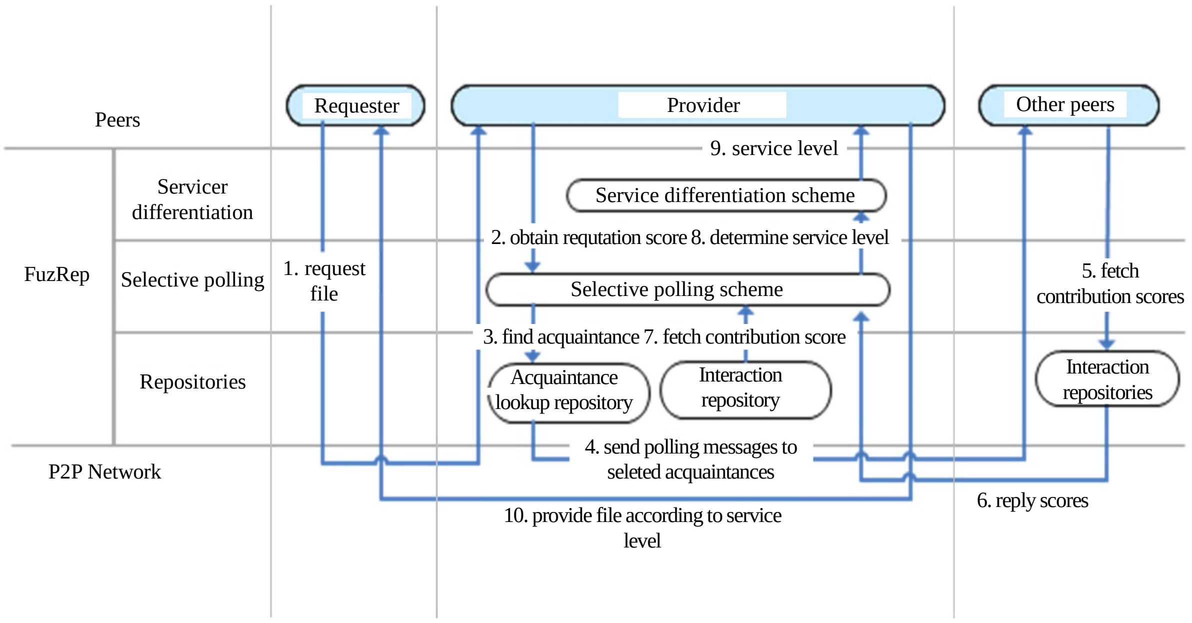
Peers
Requester
Provider
Other peers
9. service level
FuzRep
Servicer differentiation
Service differentiation scheme
2. obtain requtation score 8. determine service level
Selective polling
1. request file
Selective polling scheme
5. fetch contribution scores
3. find acquaintance 7. fetch contribution score
Repositories
Acquaintance lookup repository
Interaction repository
Interaction repositories
P2P Network
4. send polling messages to seleted acquaintances
6. reply scores
10. provide file according to service level

Figure 2. FuzRep architecture and operational processes.

Load-client Agent 1 register with Directory Agent 1; Load-server Agent 2 registers with Directory Agent 1 and Directory Agent 2 at the same time. Load-client Agent 1 may send requests to Directory Agent 1 for querying load services and Directory Agent 1 checks its database and the reputation system to find the matching load-server agents and sends the addresses of the available load-server agents to Load-client Agent 1. The matched load-server agents can be Load-server Agent 1, Load-server Agent 2, or both. Load-client Agent 1 can select one load-server agent based on its best convenience; or both for special purposes. Note that it is entirely possible that no suitable load-server agents can be found.

FuzRep is a design of a fuzzy-based reputation system for P2P networks. It includes three techniques—reputation determination, selective polling, and service differentiation. I describe how FuzRep works by revealing answers to the following questions.

- How does one determine a peer's reputation level? What are the criteria? How does one transfer a crisp score to a reputation level and maintain its level?
- How and when does one share the contribution information?
- How does one encourage information sharing and discourage free-riding? How does one differentiate different service levels?

A peer's reputation is determined by its contributions to the communities in the design of FuzRep [10]. A peer saves the transaction information in the local transaction repository, including the IDs of the requesters or providers and the accumulated contribution scores. The transaction repository is updated after every successful transaction. The initial local contribution score is set to zero for each of the previously unknown peers at its first interaction. A global accumulated contribution score reflects the corresponding peer's reputation, which is determined by the following two methods—personal reputation inference and global reputation deduction. Personal reputation inference simply extracts a peer's contribution score from its transaction repository. If a file provider chooses to determine a file requester's reputation solely based on its experience, personal reputation inference fits that purpose. Otherwise, the file provider should run the global reputation deduction process using the selective polling reputation sharing process [10].

3. Algorithm for Wireless Ad-Hoc Load Services

I consider several issues when designing the system architecture: 1) how a directory agent asks a host to register with its database, 2) the effects of the movement of mobile hosts on the joining of the load-server and load-client agents, and 3) the fault tolerance of the system. Below I explain how hosts join or leave directory agents and how directory agents form their databases when they move. In addition, I describe how a load-client agent should pay the load-server agents that it requests services from, how hosts in the system gain tokens to pay for the services it needs, and lastly how loads are transferred between load-server agents and load-client agents.

3.1. A Directory Agent Asks Hosts for Registration

In order to collect load service information from other hosts and provide results for queries, a directory agent builds a query database. The information in the database includes the addresses of load-server agents, service types, and the loads that load-server agents can accept. The host can be a desktop or laptop computer as long as it meets the minimum performance requirements; for example, the computer should have high-speed processors and sufficient capacity and speed for communication. The method for a directory agent to request a host for registration is discussed below.

1) A directory agent broadcasts a message to the other hosts within a given range that its power can reach.

2) A host, which receives the broadcast message from a directory agent and is willing to register with the directory agent's database as a load-server agent, sends an ACK message to the directory agent for registration. The ACK message includes information regarding the service types it can perform, the loads it can accept, and other pertinent details.

3) The directory agent keeps the ACK information in its query database and builds a link from itself to the load-server agent sending the ACK message.

4) To check if a load-server agent is available in the database, a directory agent periodically sends multicast messages to all load-server agents with query information in its database. The purpose for doing so is to maintain the most updated database because load-server agents may move away at any point in time. When a load-server agent receives a query message from a directory agent, it sends back a response to the directory agent to

indicate its availability. If a directory agent does not receive an acknowledgement from a given load-server agent, the directory agent deletes the information provided by that load-server agent from its database and removes the link between them. **Figure 3** demonstrates the steps regarding how a directory agent builds its query database.

a) A directory agent sends requests to hosts for registration.
b) Hosts, which are willing to register as load-server agents, send ACKs back to the directory agent.
c) The directory agent saves all the information in the ACKs to its database for future use.
d) The directory agent builds links between itself and its load-server agents.

3.2. A Host Joins a Directory Agent's Database as a Load-Server Agent

A mobile host may join a directory agent's database as a *load-server* agent when it has the ability to provide services, or it is lightly loaded and is willing to accept loads from other hosts. Not only can a load-server agent join the database of one directory agent, it can also join the databases of other directory agents. A load-server agent can join a directory agent's database in two ways.

Method 1: In the first method, a load-server agent sends messages to ask to register with directory agents within its power range and waits for replies. After receiving acknowledgements from directory agents, the mobile host registers with the directory agents by sending its address, service types it can provide, and the amount of loads it can accept for load transfer. A mobile host can register with multiple directory agents at the same time; in other words, it can join several databases simultaneously. **Figure 4** shows the steps how a host joins a directory agent's database as a load-server agent based on this method.

Method 2: The second method is similar to the method described in Section 3.1. In particular, a mobile host receives messages from directory agents requesting it to join their databases. The mobile host joins the databases by sending acknowledgements (ACKs) to the directory agents, who then add the information within the ACKs to their databases.

After the directory agents receive the ACKs from load-server agents, they build links between them. The following figure illustrates the procedures of Method 1 for how a load-server agent joins a directory agent database.

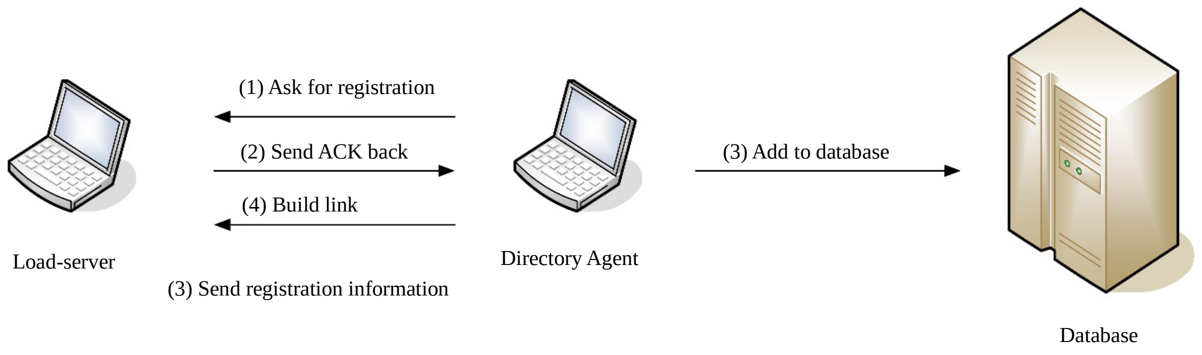

Figure 3. The procedures for how a directory agent requests for registration.

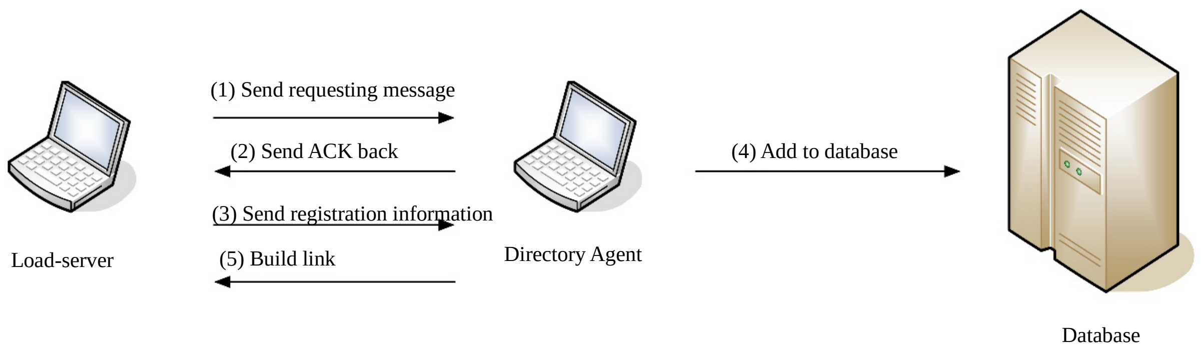

Figure 4. How a load-server joins a directory agent's database based on method 1.

1) A host sends requests to directory agents for registering as a load-server agent.
2) Directory agents send ACKs back to the host to allow it to join their databases.
3) The host sends registration information to the directory agents once it receives the ACKs.
4) The directory agents add the information to their databases.
5) The directory agents build links between themselves and the load-server agent.

3.3. Queries from Load-Client Agents

A mobile host may join directory agents' databases as a *load-client* agent when it needs services from other hosts. Since directory agents broadcast their addresses periodically to ask the mobile hosts to register for services, a load-client agent can find the addresses of directory agents from those broadcasting messages. When a load-client agent needs load services, it sends queries to directory agents and waits for replies. The contents of the replies include the addresses of available load-server agents that can provide the services the load-client agent requests for. The load-client agent may find several available load-server agents at the same time and it chooses the one that best fits it needs. If the load-client agent receives no replies from directory agents within a specified period of time, it sends out its requests again.

A load-client agent selects the best-fit load-server agent based on the service conditions it requests. For example, it may choose the one that satisfies the price the load-client agent asks for. Oncea load-client agent selects a load-server agent, it sends the service requirements or loads to the chosen load-server agent. **Figure 5** illustrates the steps.

1) A load-client agent sends query to directory agents to request for services
2) Directory agents apply the FuzRep reputation system to the requesting host for a free-rider check. The Directory agents then search their database for the desired services requested by the load-client agent. During the search, the Directory agents also apply the FuzRep reputation system to the load-server agents to detectany-free-riding problem.
3) Directory agents send replies to the load-client agent with the search result.
4) The load-client agent receives the services it needs from the load-server agents.

3.4. Movement of Directory Agents

When a directory agent moves to another region, it loses all the information in its database about load-server agents and its peer directory agents. How a directory agent notifies all the other agents about its movement becomes an important issue. There are two ways that other agents can detect the departure of a directory agent. The first is that the directory agent sends a message to notify the hosts about its movement. Hosts receiving the message will stop sending queries to this directory agent and remove the links between them.

The second method is based on the fact that hosts can no longer detect the existence of a departed directory

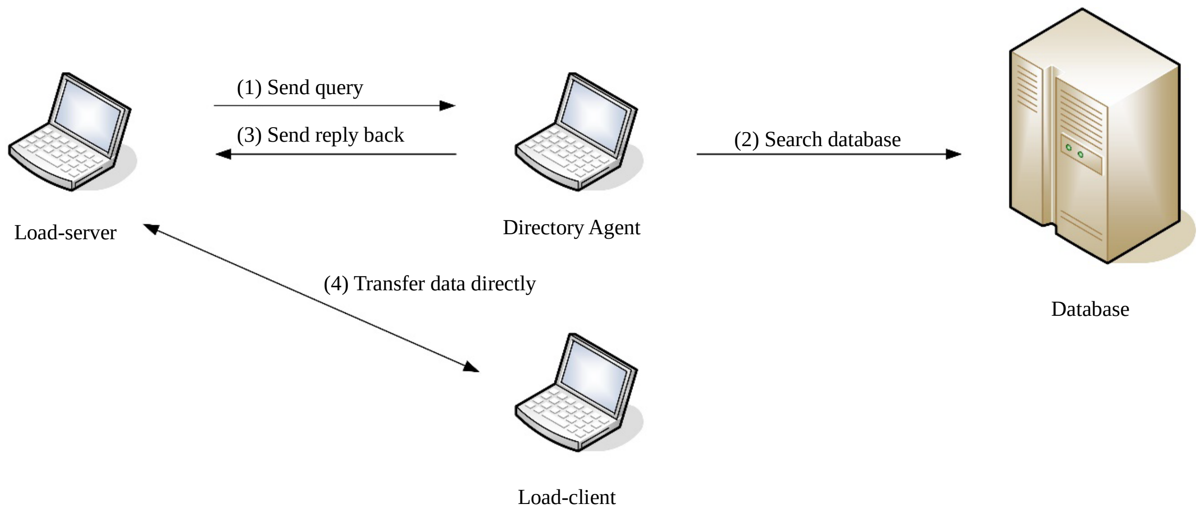

Figure 5. How a load-client agent sends queries.

agent. As load-server agents send updated information to directory agents periodically, they are able to recognize the departure of a directory agent when no reply is received from that directory agent. In addition, a load-client agent can detect the departure of a directory agent when it does not receive broadcast messages from that agent for a specified period of time. In this case, the load-client agent deletes the link to that directory agent. After moving to a new region, a directory agent sends messages to hosts within the range it can reach to ask the hosts to join its database for load services as discussed in Section 3.1. It is possible for certain hosts to not have any directory agents to contact once a given directory agent moves away. Those hosts will continue sending messages to other hosts to search for new directory agents as described in Sections 3.2 and 3.3.

3.5. Movement of Load-Server Agents

When a load-server agent moves to a new region, it may lose its original directory agents and need to establish links to new directory agents as described in Section 3.2. If a directory agent does not receive updated information from a departed load-server agent for a period of time, it deletes the information on that load-server agent from its database and removes the link between them.

3.6. An Example

Figure 6 illustrates how a directory agent, a load-server agent, and a load-client agent communicate among each other. Steps 1) through 6) describe the details of the communication processes.

1) A Directory Agent broadcasts a message to hosts to ask them to join its database.

2) A Load-Server Agent replies with an acknowledgement to that Directory Agent that it wishes to join the database.

3) The Directory Agent saves the information in its database.

4) The Load-Server Agent sends requests to the Directory Agent for load services.

5) The Directory Agent sends the address of the Load-Server Agent who is suitable for load service.

6) The Load-Client Agent communicates with the Load-Server Agent directly.

3.7. Load Transfer

A host may transfer loads to other hosts when it is heavily loaded. Instead of using a fixed threshold method to decide whether a host is heavily loaded, I employ the fuzzy logic control method to improve the performance. First, the host finds an available host by sending a service request. Once it finds a host that accepts its request for load transfer, it transfers its loads to the selected host. The amount of loads to be transferred is equal to half of

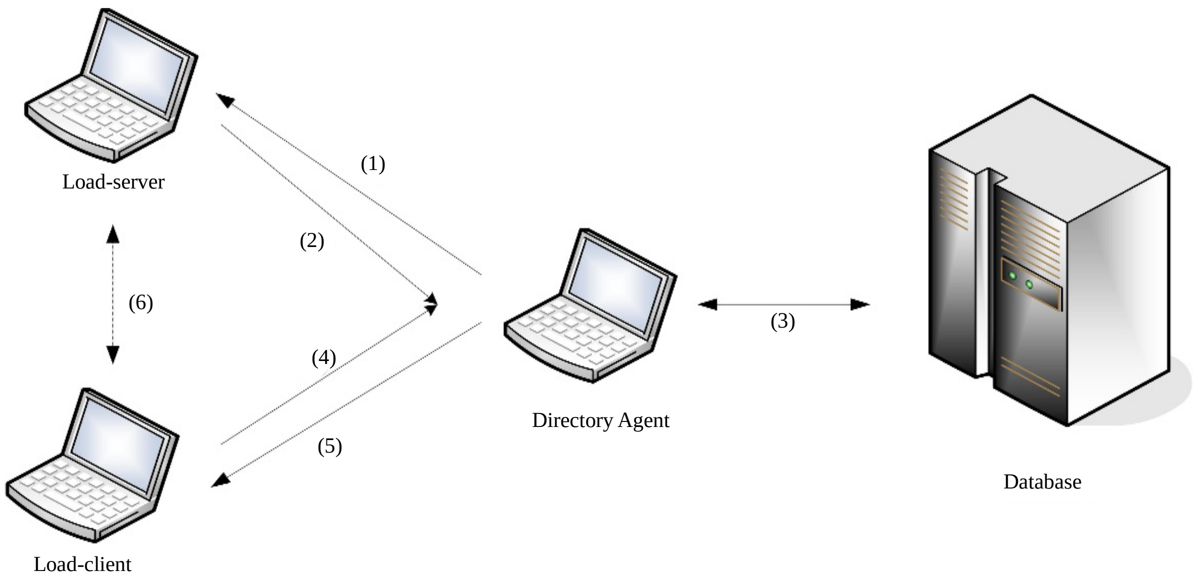

Figure 6. An Example for communications between agents.

the difference of loads between the load-client agent and the load-server agent. It is possible that there are several load-server agents that can satisfy the request by a load-client agent. To minimize the moving cost, a load-client chooses the load-server agent closest to it. In **Figure 7**, I use an example to illustrate the process. The circle in the figure represents the range that load-client agent C can reach. Within the range, there are three load-server agents within this range—S1, S2, and S3—which can satisfy the request from agent C. Since S1 is the closest one to C, it is selected to receive the load transfer fromC.

The following steps show the details of a load transfer.

1) When a host detects that it is heavily loaded, it broadcasts a request message to hosts within its range to request a load transfer service. Instead of using a fixed threshold level to check if it is lightly loaded or heavily loaded, I use the fuzzy logic control [11] to check its queue status to improve the performance. This method is mentioned in [12] [13].

2) Upon receiving the request, hosts check their queue status using the fuzzy logic control method. Lightly loaded hosts will return ACKs to the load-client agent initiating the request.

3) When the load-client agent receives the ACKs from load-server agents, it chooses the load-server agent that is the first to respond for load transfer. That means that the load-client agent chooses the closest load-server agent to maximize performance.

4) If no available hosts exist within the range of the load-client agent, the load-client agent sends request to its directory agents to look for registered load-server agents for load transfer. Then it waits for the responses from its directory agents.

5) The directory agents find available (lightly-loaded) load-server agents when they receive requests from a load-client agent. The directory agents send addresses of these available load-server agents to the load-client agent for load transfer. The load-client agent chooses the best-matched host for load transfer.

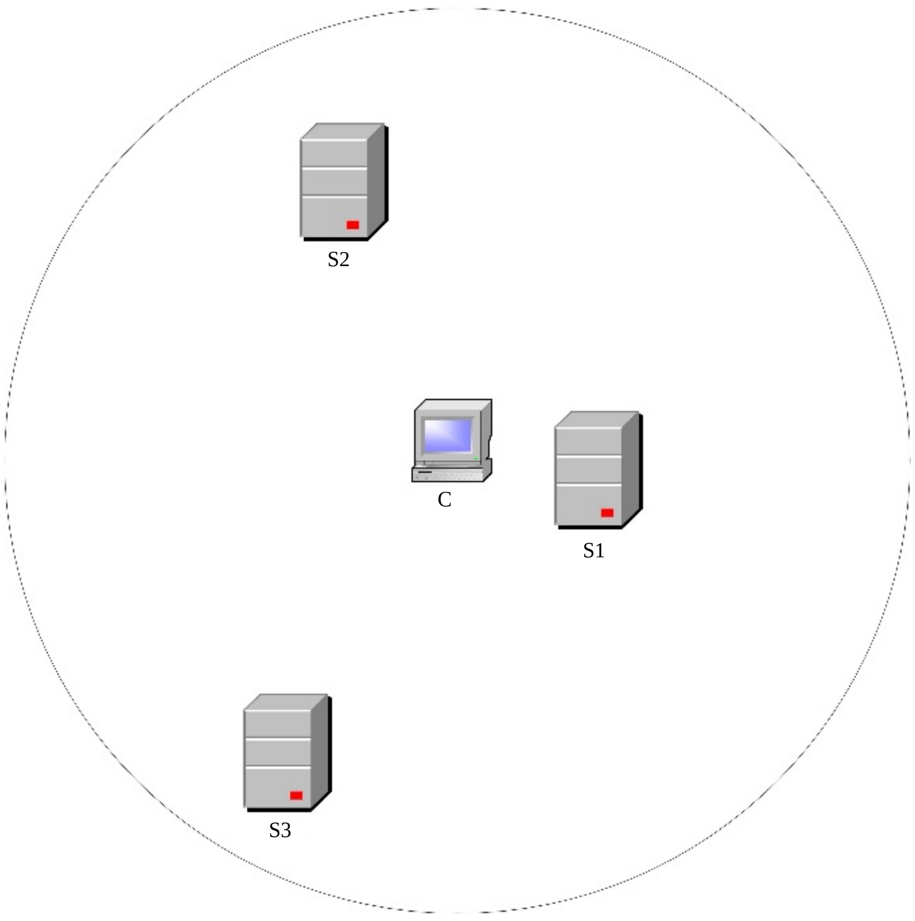

Figure 7. An example of how a load-client agent chooses a best-fit load-server agent.

4. Service Information Format and Service Price

In the future, it is possible that hosts are required to pay for service requests. In this section, I discuss this scenario, define the format of service information in load services including the price of service. This format is designed to represent what information is stored by a directory agent in its database. **Figure 8** shows the format of service information that contains 4 fields—*address*, *service-type*, *number-of-tokens*, and *load*.

The *address* field is the address of a load-server agent. The *service-type* field indicates which type(s) of service that a load-server agent can provide. The *number-of-tokens* shows the price of service, and the *load* field shows the current load information of a load-server agent. When a load-server agent communicates with the directory agents, it provides them with the information about the type(s) of services it can provide, the tokens (price) of service, its current load status and address. The service request of a load-client agent can be fulfilled only when a load-server agent is found to provide the requested service type at a price that the load-client agent can afford (or is willing to pay). Otherwise, the load-client agent will search for available (lightly-loaded) load-server agents for the purpose of load transfer and select the one that charges the lowest price.

The architecture of hosts is based on the following assumptions:

1) A load-client agent is required to pay a load-server agent for load services provided.

2) When sending a request for a load service, a host loses tokens as the price for asking the load service.

3) To increase the number of tokens and therefore the ability to request for services, a host must try its best to gain tokens. There are two possible ways to gain tokens. First, a host can provide services to other hosts. Second, a host should avoid sending useless requests to network to save tokens. This can be implemented by increasing the waiting time for a load-client agent to send requests. By reducing the number of messages sent, the problem of network congestion can also be alleviated.

4) A load-client agent may find several available load-server agents for a particular request such that those load-server agents satisfy the requirements set by the load-client agent. The client host chooses the load-server agent that best fit the need of the host.

5) If a host does not have enough tokens to pay a load-server agent to complete the load services, it will stop sending requests to its directory agents until it has sufficient tokens.

When a load-client agent require load services to be completed by other hosts, it sends the directory agent a service request indicating the service needed and the price it can afford. The directory agent, upon receiving the request, finds available load-server agents by matching by the key feature(s) and pricing of the requested service. For example, a host sends a request for a service of high speed calculation along with the price it can afford. The directory agent considers the speed of each load-server agent's processor and the price charged by each load-server agent for providing the service of high speed calculation. Matching by processor speed and price, the directory agent identify all available load-server agents. The addresses of these load-server agents are sent to the requesting load-client agent. Upon receiving the addresses, the load-client agent chooses one and sends jobs directly to that load-server agent. Typically, the load-client agent chooses the load-server agent that charges the smallest number of tokens (or lowest price) for performing the requested service.

5. Scalability

As the number of clients and servers increases, the burden on a network system becomes significantly heavier due to a larger number of messages for service discovery and request. When a host joins or roams into a network, it sends out requests. If there are numerous hosts that move in and out of a system frequently, the flood of requests may lead to network congestion. Therefore, careful consideration of scalability issues is very important to the design of protocols. In our system, I use the number of tokens (the price to pay) to control for the scalability of load-server agents registered with directory agents and load-client agents sending load service requests. For example, a client host cannot send requests to directory agents for services if it does not have enough tokens. It should provide its services to other hosts to gain enough tokens before it sends a request.

6. Conclusions

In this paper, I present a new load service structure in wireless ad-hoc networks using a reputation system to

address	*service-type*	*number-of-tokens*	*load*

Figure 8. Service information format.

check for nodes' reputation. Since the hosts in a wireless ad-hoc network can move to anywhere at any time, it is a challenge for a host to find other hosts for the purpose of load service or load transfer. I discuss the detailed process for a directory agent to ask hosts to register as load-server agents, a load-server agent to register with directory agents' databases, and a load-client agent to identify available load-server agents when it needs load services. In the structure, the directory agents are tasked with finding load-servers which can provide services the clients need. Also, the directory agents apply the FuzRep reputation system to check for the clients' and servers' reputations when matching load services and transfer requests. This is to avoid the free-riding problem in P2P network systems.

In addition, I introduce a new concept that a host should pay for the load service requested and the use of tokens as the price to pay by the clients (or the price to be charged by the servers) in P2P networks. The token concept can also be used to control for the scalability of networks and congestion of network flows. Finally, fuzzy logic control is used to check for load status of hosts in the load transfer protocol.

7. Future Work

There are some works to be done in the future. In order to compare the performances between the systems with and without using the FuzRep reputation system, I will build a heterogeneous system and run simulations for system performance comparisons. I also have to construct a system for the service information and service price structure. What described in this paper is an initial concept for how to charge Web services in P2P networks. The system will be built with the supports of Web services using XML and Web 2.0 concepts.

References

[1] Guttman, E., Perkins, C., Veizades, J. and Day, M. (1998) Service Location Protocol. Version 2, IEFT, RFC 2165.

[2] Waldo, J. (1999) The Jini Architecture for Network-Centric Computing. *Communication of the ACM*, **42**, 76-82. http://dx.doi.org/10.1145/306549.306582

[3] Maab, H. (1997) Location-Aware Mobile Application Based on Directory Services. MOBICOM 97, 23-33.

[4] Yeong, W., Howes, T. and Kille, S. (1995) Lightweight Directory Access Protocol. RFC 1777.

[5] Adar, E. and Huberman, B.A. (2000) Free Riding on Gnutella.

[6] Hughes, D., Coulson, G. and Walkerdine, J. (2005) Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, **6**, 6. http://dx.doi.org/10.1109/MDSO.2005.31

[7] Ramaswamy, L. and Liu, L. (2003) Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems. *Proceedings of* 36*th Hawaii International Conference on System Sciences* (*HICSS*'03), 6-9 January 2003. http://dx.doi.org/10.1109/HICSS.2003.1174583

[8] Oram, A., *et al*. (2001) Peer-to-Peer: Harnessing the Power of Disruptive Technologies. Oreilly.

[9] Androutsellis-Theotokis, S. and Spinellis, D. (2004) A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, **36**, 335-371. http://dx.doi.org/10.1145/1041680.1041681

[10] Lin, Y.-K. and Huang, M.-C. (2006) A P2P Reputation System with a Fuzzy Technique for Incentive Provision. *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems* (*IASTED PDCS*), Dallas, November 2006, 220-225.

[11] Ross, T.J. (1995) Fuzzy Logic with Engineering Applications. McGraw Hill, New York.

[12] Huang, M., Hosseini, S.H. and Vairavan, K. (2002) Load Balancing in Computer Networks. *Proceedings of ISCA* 15*th International Conference on Parallel and Distributed Computing Systems* (*PDCS*-2002), *Special Session in Network Communication and Protocols*, Louisville, 19-21 September 2002.

[13] Ross, T.J. (1995) Fuzzy Logic with Engineering Applications. McGraw Hill, New York.