Transactions on Engineering and Computing Sciences - Vol. 13, No. 01
Publication Date: February 25, 2025
DOI:10.14738/tmlai.1301.18272.
Huang, M.-C. (2025). Apply Reputation Proof with Load Services in Ad-hoc Networks. Transactions on Engineering and Computing Sciences, 13(01). 87-98.

Apply Reputation Proof with Load Services in Ad-hoc Networks

Ming-Chang Huang
Department of Business Information Systems / Operation Management University of North Carolina at Charlotte

ABSTRACT
In this paper, a new system design for load services in computer networks with a new reputation system is constructed to check available host reputation. Database systems are used for directory agents to save information provided by load-server agents and protocols are built how a host can find available hosts for load service and load transfer purposes when the host moves to a new region. This includes how a directory agent builds its database, how a load-server agent provides its services, and how a load-client agent gets its desired services. The system uses the fuzzy logic control method to transfer loads for load balancing, instead of fixed threshold level methods. The purpose of this new system structure is to provide efficient ways in building communication and accessing resources in ad-hoc computer network systems and helps users to find resources easily and securely.

Keywords: Load Service, Ad-Hoc Network, Directory Agent, Load-server Agent, Loadclient Agent, Peer-to-Peer, Reputation System.

INTRODUCTION

Computer networks can 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 mobile computing, the demand for load services and load balancing grows. When a computer is overloaded or it needs special services from other computers, it may send requests to other computers for load transfer or load services. For example, a computer may need some jobs to be performed with higher quality of services or it needs some jobs to be done within a brief period of time. If its processor is too slow to perform the jobs, it may need to send part jobs to other computers with higher speed of processors. Since wireless networks have been wild used in recent years, how a host transfers its loads to other nodes has become a critical 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 purposes.

Before a wireless host transfers its loads to other hosts or asks for load services from other hosts, it has to find available hosts using resource allocation algorithms. There are several resource allocation protocols that have been developed, for example, IEFT Service Location Protocol (SLP) [1] and Jini [2] software package from Microsystems. However, these protocols address how to find the 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]; while it does not cover some important issues about the movements of mobile hosts, for example, how to generate a new directory service and how a host gets the new services, when a directory agent moves away 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 providing load services become importance issues in an Ad-Hoc network system.

To find a host which can fulfill the load service purpose, the requesting host also has to make sure that the host it is looking for has good reputation in load services. For good reputation hosts, they will have to share their resources as well besides just requesting resources from other hosts. It is called the “free-riding” situation if a host only requests resources from other hosts without sharing it resources to others. Measurement study of free-riding on Gnutella was first reported by [10] in 2000 which indicated that approximately 70% of Gnutella users did not share any files and nearly 50% queries were responded to by top 1% peers. However, according to the most recent measurement study, the percentage of free riders rises to 85% [11]. It is very possible that a small number of peers who are willing to share information take most of the job loadings in P2P networks. As a result, the prevalence of free riders will eventually downgrade the performance of entire system and would make the system vulnerable [12].

In this paper, a system structure is going to be constructed for load services with reputation checking in wireless Ad-Hoc network systems using 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 these services, e.g., include resource location services and load balancing services. A host may send its special requests to other hosts for load services or send its loads for load balancing. The requests include service types of the host needs or the amount of loads to be sent to other hosts. For those special services, the host should define the conditions that 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 also check the requested host’s reputation to avoid “free-riding” cases [7] [13].

In Section 2, I discuss the system structure. Section 3 expresses the details of the method. Section 4 and section 5 illustrate the information format for databases, and the scalability, respectively. Section 6 presents the conclusion.

SYSTEM STRUCTURES

There are three basic components in my load service system – directory agent, loadserver agent, and load-client agent. A load-server agent provides load services that are queried by other hosts (load-client agents) which require load services. Load-server 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, which may need some 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 some special services, which it does not have the ability to perform. A directory agent forms groups for both 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) [7] which the system is going to apply. 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 request. In Figure 1, for example, Load-server Agent 1 and Loadclient 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 fitted load-server agents and sends those available load-server agent addresses to Load-client Agent 1. The fitted loadserver agents can be Load-server Agent 1, Load-server Agent 2, or both. Load-client Agent 1 can choose one of them based on its best convenience; or it can choose both of them for special purposes. Of course, it is possible that none of the load-server agents can be found.

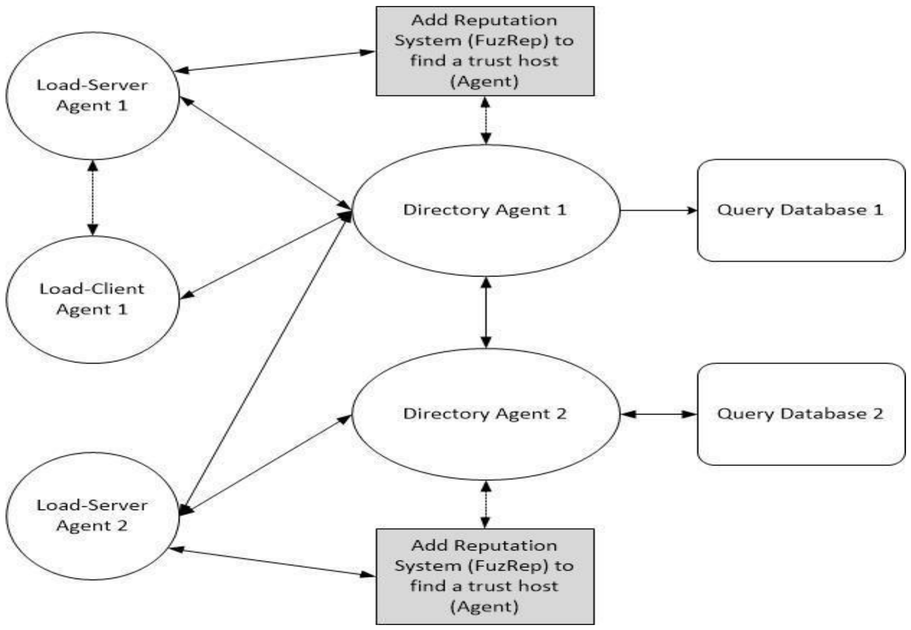

Figure 1: Load service system architecture with FuzRep Reputation System

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 am going to describe how FuzRep works by revealing answers of the following questions.

- How to determine a peer’s reputation level? What are the criteria? How to transfer a crisp score to a reputation level? How to maintain it?
- How and when to share the contribution information?
- How to encourage sharing and discourage free riding? How to differentiate the service level?

In FuzRep_M1, a peer's reputation is determined by its contributions to the communities. A peer saves transaction information into local transaction repository, including requesters or providers' IDs, and accumulated contribution scores. The transaction repository is updated after every successful transaction. The initial local contribution score is set to zero originally for pre-unknown peers at their first interactions. A global accumulated contribution score is used to determine corresponding peer's reputation. It is built on two phase computes – personal reputation inference and global reputation deduction. Personal reputation inference simply fetches a peer's contribution score from its transaction repository. If a file provider chooses to determine a file request's reputation simply 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. [7]

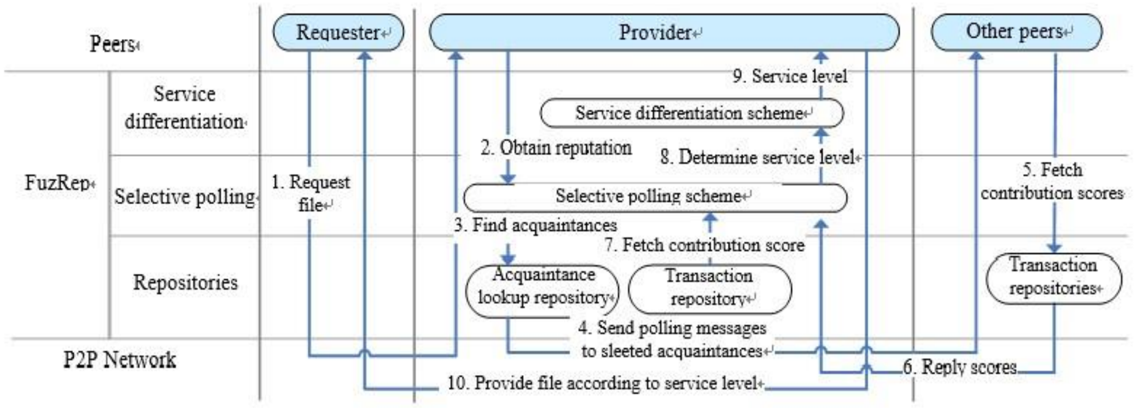

Figure 2: FuzRep architecture and operational processes

ALGORITHM FOR WIRELESS AD-HOC LOAD SERVICES

There are several issues that I consider when designing our system architecture, which includes, for example, how a directory agent asks a host to register with its database, the effects of the movement of mobile hosts to the joining of load-server and load-client agents, and 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.

I also describe how a load-client agent should pay load-server agents that it asks the services from and how hosts in the system gain tokens in order to pay for the services it need. How to transfer loads between load-server agents and load-client agents is also mentioned in this section.

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 which provide information, the service types, or the loads which load-server agents can accept. The host can be a desk computer or a laptop once it has the ability; for example, it has high-speed processors, enough power for communication, etc. The method of how a directory agent asks for registration is discussed below.

1. A directory agent broadcasts a message to the other hosts within the 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, such as the service types it can perform and/or the loads it can accept, etc., provided by a load-server agent.
3. The directory agent keeps the ACK information in its query database and therefore builds a link from itself to the load-server agent sending the ACK message.
4. To check if a load-server agent is still available in the database, a directory agent periodically sends multicast messages to all the load-server agents, which have query information in its database. This purpose for this is for database information update because load-server agents might move away anytime. When a load-server agent receives a query message from a directory agent, it should send back a response to the directory agent to indicate that it still exists in the directory agent's power range. If the directory agent does not get the acknowledgement from a load-server agent that has query information in the database, it deletes the information provided by that load-server agent from its database and therefore deletes the link between them.

Figure 3 demonstrates the steps in how a directory agent builds its query database.

1. A directory agent sends requests to hosts for registration.
2. Hosts, which are willing to register as load-server agents, send ACKs back to the directory agent.
3. The directory agent saves all the information in those ACKs on its database for future use.
4. The directory agent also builds links between itself and its load-server agents.

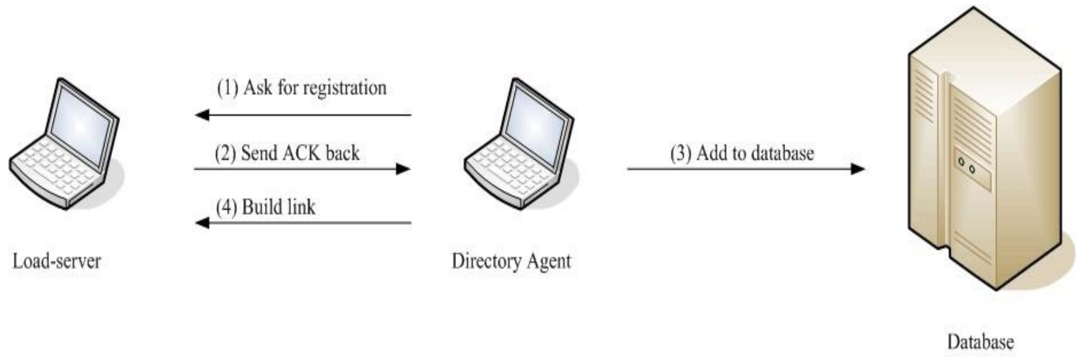

Figure 3: The procedures for a directory agent asks for registration

A Host Join Directory Agent's Databases as A Load-Server Agents

A mobile host may join directory agents' databases 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 may a load-server agent join a directory agent, but also it may join multiple directory agents. A load-server agent joins directory agent's databases in two ways.

- Method 1: The first method is that it sends out messages to ask for registering with directory agents within its power range and waits for the replies from those directory agents. After receiving acknowledgements from directory agents, the mobile host registers with the databases of those directory agents by sending its address, the service types it can provide, and the amount of loads it can accept for load transfer. A mobile host can register with several directory agents at the same time; which means a mobile host can join several databases simultaneously.
- Method 2: The second method, like the method in Section 3.1, is that a mobile host receives messages from some directory agents for requesting joining their databases. Thereafter, the mobile host may join those databases by replying to acknowledgements (ACKs) back to those directory agents and the directory agents add the ACKs into 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 a load-server agent to a directory agent database.

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

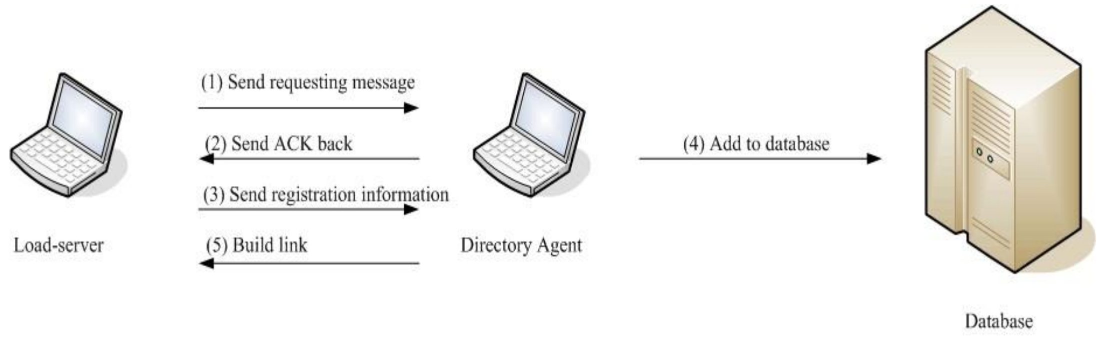

Figure 4: How a load-server joins a directory agent database for Method 1

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 for 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 that it can contact and waits for replies from them. The contents of these replies include the addresses of available load-server agents that can provide the services the load-client agent asks for. The load-client agent may receive several replies from different load-server agents at the same time and it chooses the best-fit one. If it cannot find available load-server agents (without any reply from directory agents in a period of time), it waits for a certain period of time and sends queries 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 loadclient agent asks. When a load-client agent selects the best-fit load-server agent, it directly sends service requirements or loads to the chosen load-server agent. Figure 5 shows the steps.

1. A load-client agent sends query to directory agents to request services
2. Directory agents apply the FuzRep reputation system to the requesting host for a freerider check. Then the Directory agents search their database for the desired services requested by the load-client agent. Before the Directory agents send back searching information, they also apply the FuzRep reputation system to the load-server agents to avoid free-riding.
3. Directory agents send replies back, which indicate the information they have in the databases.
4. The load-client agent gets the services it needs from load-server agents.

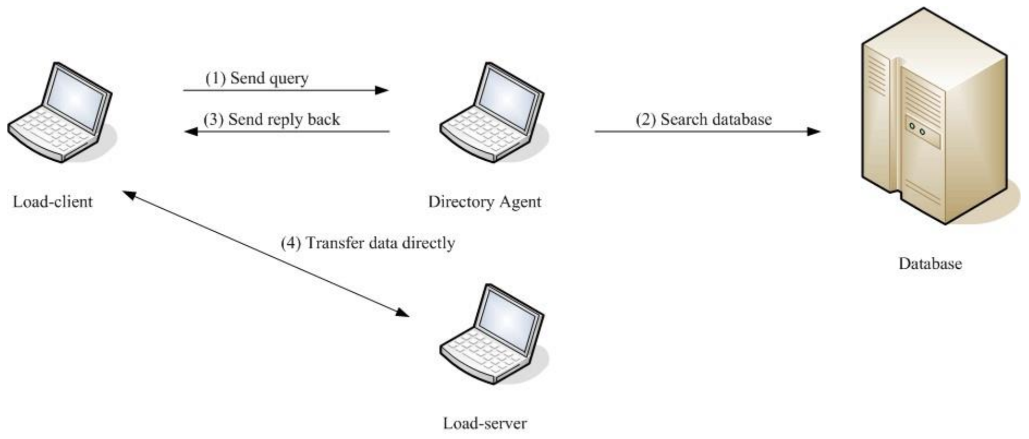

Figure 5: How a load-client agent sends queries

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 a prominent issue. There are two ways that other agents can detect the leave of a directory agent. The first is that the directory agent sends a message to notify other 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 to use the fact that hosts cannot detect the existence of a directory agent. Since load-server agents send updated information to a directory agent periodically, load-server agents can notice that a directory agent does not exist in the region if hosts do not get the reply from that directory agent. For a load-client agent to detect the existence of a directory agent, if it does not receive any broadcast message during a period of time, then it deletes the link to that directory agent.

After moving to a new region, a directory agent sends messages to hosts in the power range it can reach to ask for hosts to join its database for load services as discussed in section 3.1. It may happen that some hosts do not have any directory agent to contact once a directory agent moves away. Those hosts will keep sending messages to other hosts to find new directory agents as described in section 3.2 and 3.3.

Movement of Load-Server Agent

When a load-server agent moves to a new region, it may lose its original directory agents, and it has to establish new links to its new directory agents as described in section 3.2. Once a directory agent does not receive updated information from a load-server agent for a period of time, it deletes the information about that load-server agent from its database and therefore deletes the link between them.

An Example

Figure 6 illustrates steps how a directory agent, load-server agent, and load-client agent communicates each other. (1), (2), (3), (4), and (5) indicate the procedures for setting up the processes.

1. A Directory Agent broadcasts join message to hosts.
2. Load-Server Agent sends an acknowledgement to reply to that Directory Agent to join the database.
3. Directory Agent saves the information on its database.
4. Load-Server Agent sends requests to Directory Agent for load services.
5. Directory sends the address of the Load-Server Agent if Load-Server Agent is suitable for load service.
6. Load-Client Agent communicates with Load-Server Agent directly.

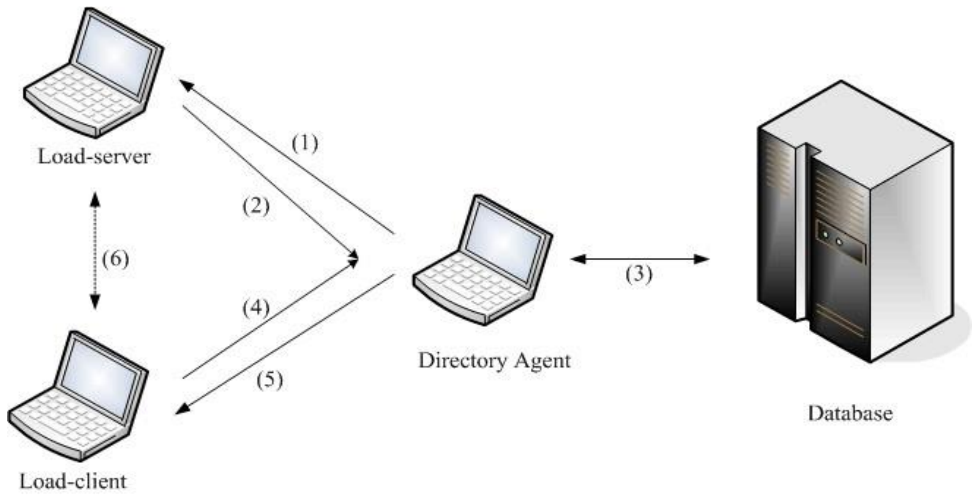

Figure 6: An Example of Communications between Agents

Load Transfer

A host may transfer loads to other hosts when it is heavily loaded. Instead of using fixed threshold method to decide whether a host is heavily loaded, I use fuzzy logic control method to improve the performance. First, the host finds an available host by sending service request as I mentioned before. Once it finds a host that accepts its request for load transfer, it transfers its loads to the selected host. The number of loads to be transferred is equal to half of the difference of loads between the load-client agent and the load-server agent. It is possible that there are several server load-server agents, which satisfy the request by a load-client agent. In order to reduce the distance and movement effect, a load-client chooses the load-server agent that is the closest one to it. Figure 7, for example, shows the power range that load-client agent C can reach and there are three load-server agents – S1, S2, and S3 – which satisfy the request from agent C. Since S1 is the closest one to C, it is chosen which C will transfer its loads to.

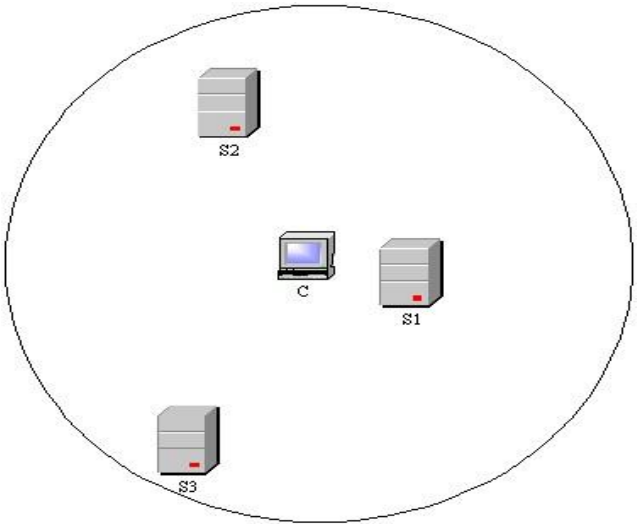

Figure 7: An example for a load-client agent to choose a best-fit load-server agent

The following steps show the details of load transfer.

1. When a host detects that it is heavily loaded, it broadcasts a request message to hosts in its power range to ask for load transfer service. Instead of using fixed threshold levels to check if it is lightly loaded or heavily loaded, I use fuzzy logic control [5] to check its queue status to improve the performance. This method is mentioned in [6, 7].
2. Hosts, which receive the request, check their queue status using fuzzy logic control method, and returns ACKs, if they are lightly loaded, to the load-client agent that sent the request.
3. When the load-client agent gets the ACKs from load-server agents, it chooses the load-server agent, which is the first one to send its ACK, for load transfer. That means that the load-client agent chooses the closest one in order to improve performance.
4. If there are no available hosts in the load-client agent power range, the load-client agent sends requests to its directory agents to look for the 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. Then, the directory agents send addresses of these available load-server agents to that load-client agent for load transfer. The load-client chooses the best host to transfer its loads to the selected host.

SERVICE TYPE FORMAT AND SERVICE PRICE

In the future, it is possible that hosts will have to pay if they ask for service from other hosts. This situation is discussed in the section, and the service type format for load services and the price for each service are also defined. This format is for a directory agent to store the information in its database. Figure 8 shows the format that there are 4 fields in it – *address, service-type, number-of-tokens,* and *load*.

The *address* field is the address of a load-server agent, so that a load-client can directly connect to it. The *service-type* field indicates which kind of services a load-server agent provides. The *number-of-tokens* shows the price of a service for a load-client to pay, and the *load* field shows the current load for a load-server agent. When a load-server agent provides load services to directory agents, it provides directory agents the information about the type(s) of services it can provide, the tokens (price) for a loadclient agent to take the service, and the current load status and address for the load-server agent. A load-client agent can get the service only if it matches the service type, and the price that the load-server agents ask, or it can find an available load-server agent for load transfer purposes if the load-server agent is lightly loaded, and the load-client agent can pay the price.

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

Figure 8: Service Type Format Stored

There are some assumptions in our architecture for hosts.

1. A load-client agent has to pay a load-server agent when it needs load services from that load-server agent.
2. When sending a request to a directory agent, a host loses tokens as the price for asking load service.
3. In order to increase the number of tokens and therefore increase the ability to ask for services, a host must try its best to gain tokens. There are two ways to implement it. First, a host can provide the services to other hosts to gain tokens. 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. This also may avoid network congestion because the number of messages is reduced.
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 for the loadclient agent. Then the client host has to choose the best-fit one.
5. If a host does not have enough tokens to find a load-server agent for load services, it should stop sending requests to its directory agents asking for load services until it can provide enough tokens.

The request message, which a load-client agent sends out when it needs a service, includes a price that the load-client agent can pay. The directory agent, which receives the message, finds available load-server agents by comparing the key words and the prices. For example, if a host needs a service with higher speed calculation, it sends requests to its directory agents. In these requests, the speed of the load-server agent's processor and the price the load-client agent can provide are included. Directory agents match these requirements to the information via the key words and the number of tokens in their database and therefore find the available load-server agents. The addresses of those load-server agents are sent to the load-client agent which sent the request. Upon receiving those addresses, the load-client agent chooses one available and sends jobs directly to that load-server agent. To choose an available load-server agent from those addresses by directory agents, the load-client agent may choose the one which asks for the lowest number of tokens for performing the requested service.

SCALABILITY

As the number of clients and servers in the network system increases, so does the burden to the system because of the increases of messages for service discovery and request. When a host joins or roams into a network, it sends out requests. If there are too many hosts that move too frequently, they may send many requests, which may cause the congestion of the network. Therefore, careful consideration of scalability issues is especially important to the design of the protocols. In our system, I use the number of tokens (the price to pay) to control 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 requests.

CONCLUSION

In the paper, I introduce a new load service method in wireless ad-hoc networks using a reputation system to check nodes' reputation. Since the hosts in a wireless ad-hoc network can move anywhere at any time, it is difficult for a host to find other hosts for load service or load transfer purposes. I discuss several issues about a directory agent asking for hosts to register as load-server agents, a load-server agent registering with directory agents' databases, and a load-client agent finding available load-server agents when it need load services. The Directory agents can find the available load-servers which provide services the clients need. Also, the Directory agents apply the FuzRep reputation system to check the clients and servers' reputation to load and services requests. This is to avoid free-riding situation in P2P network systems.

I also mention a new concept that a host should pay the price when it needs services from other hosts in networks and how it works by using token as the price in the networks. The token concept is also used to control the scalability of networks and congestion control of network flow. I also want to use Fuzzy logic control to check load status of hosts in the load transfer protocol.

References

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

[2]. J. Waldo, "The Jini Architecture for network-centric computing," Communication of the ACM, pp 76-82, July 1999.

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

[4]. W. Yeong, T. Howes, and S. Kille, “Lightweight Directory Access Protocol,” RFC 1777, March 1995.

[5]. Ross, T. J., Fuzzy Logic with Engineering Applications, McGraw Hill, 1995.

[6]. Huang, M., S. H. Hosseini, and K. Vairavan, “Load Balancing in Computer Networks,” Proceedings of ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS-2002), Special session in Network Communication and Protocols. Held in the GALT HOUSE Hotel, Louisville, Kentucky, Sep. 19 - 21.

[7]. Ross, T. J., Fuzzy Logic with Engineering Applications, McGraw Hill, 1995.

[8]. Andy Oram et al., “Peer-to-Peer:Harnessing the Power of Disruptive Technologies,” Oreilly 2001.

[9]. Stephanos Androutsellis-Theotokis and Diomidis Spinellis, “A survey of peer-topeer content distribution technologies,” ACM Computing Surveys, 36(4):335–371, December 2004. doi:10.1145/1041680.1041681.

[10]. Adar, E., and Huberman, B. A. Free riding on Gnutella. First Monday 5, 10 (October 2000).

[11]. Hughes, D., Coulson, G., and Walkerdine, J. Free riding on Gnutella revisited: the bell tolls? In IEEE Distributed Systems Online 6, 6 (June 2005).

[12]. Ramaswamy, L. and Liu, L. Free riding: A new challenge to peer-to-peer file sharing systems. In Proceedings of 36th Hawaii International Conference on

[13]. System Sciences (HICSS’03) (January 2003)

[14]. Ming-Chang Huang, A Peer-to-Peer Reputation Evaluation System, the ISCA journal (International Journal of Computers and Their Applications) special issue on SEDE, pp. 3-13, Vol. 27, No. 1, March 2020.