Charge-based Flooding Algorithm for Detecting Multimedia Objects in Peer-to-Peer Overlay Networks

Kenichi Watanabe†, Tomoya Enokido†, Makoto Takizawa†, and Kane Kim††
†*Tokyo Denki University, Japan*
{*nabe, eno, taki*}*@takilab.k.dendai.ac.jp*
††*University of California, Irvine*
khkim@uci.edu

Abstract

Multimedia objects are distributed in peer-to-peer (P2P) overlay networks since objects are cached, downloaded, and personalized in peer computers (peers). An application has to find target peers which can support enough quality of service of objects. We discuss a new type of flooding algorithm to find target peers based on charge and acquaintance concepts so that areas in networks where target peers are expected to exist are more deeply searched. In addition, we discuss how peers can be granted access rights to manipulate the objects with help and cooperation of acquaintances. We evaluate the charge-based flooding algorithm compared with a TTL-based flooding algorithm in terms of the number of messages transmitted in networks.

1. Introduction

According to the development of common frameworks of computers and networks, various types and huge number of computers, especially personal computers (PCs) are now interconnected in *peer-to-peer* (P2P) overlay networks [6]. Service supported by multimedia objects is characterized by quality of service (QoS) like frame rate and number of colours. Multimedia objects are distributed in nature on multiple computers since objects are downloaded, cached, and personalized in local peer computers (peers). For example, a full-coloured video object is downloaded and changed to a monochromatic version in a personal computer. It is critical to support applications with sufficient, possibly necessary quality of service (QoS). In a P2P overlay network, the huge number of peers are included and the membership is dynamically changed. If a peer would like to obtain some service of an object, the peer has to find target peers which can manipulate the object with enough QoS. It is difficult, maybe impossible for each peer to perceive what objects are distributed to what peers.

An *acquaintance* of the peer c_i is another peer c_j which a peer c_i perceives. An acquaintance peer c_j has some information on where objects are stored and how objects can be manipulated. If a peer would like to manipulate objects, the peer first asks acquaintances to detect the objects. If objects which satisfy QoS requirements are not detected, each acquaintance furthermore asks its acquaintances. In the paper [3], acquaintance concepts are also used to detect a target computer. Here, requests are sequentially forwarded and only text documents are searched.

In flooding algorithms [2, 11] to find objects in P2P overlay networks, counters like TTL (time-to-live) [11] and HTL (hops-to-live) [2] are used to prevent indefinite circulation and explosion of request messages in networks. If there are multiple candidate routes to find target objects, some route with higher possibility to find the target objects should be more deeply searched. A request is first assigned with some *charge* which shows the total number of messages to be transmitted. The charge of the request is decreased each time the request is passed over a peer in a way similar to TTL and HTL. If there are multiple routes from a peer, a request is in parallel transmitted through the routes. A request to each route is assigned with a part of the charge. If there is larger possibility to find target objects in a route, a request forwarded in the route is charged with a more portion of the charge. In the evaluation, we show a target object can be found with larger possibility than the traditional flooding algorithm. Even if peers with target objects are found, the objects cannot be manipulated without access rights. In this paper, we newly discuss how to obtain access rights and to ask acquaintances which are granted access rights.

In section 2, we present the distribution of objects in peers. In section 3, we discuss acquaintances. In section 4, we discuss the charge-based flooding algorithm. In section 5, we evaluate the algorithm compared with other flooding algorithms in terms of the number of messages.

2. Distribution of Objects

2.1. Objects

An object [8] is an instantiation of a *class*. A class is composed of attributes and methods for manipulating its objects. An object is an encapsulation of values of attributes and the methods. Let $\mathbf{P}_o$ be a set of methods supported by an object o. A collection of attribute values are referred to as *state* of an object which is characterized by quality of service (QoS) like frame rate. Not only state but also QoS of an object are manipulated through methods. QoS supported by an object is changed as well as state of the object through methods. For example, some application degrades the frame rate of a video object by a method *degrade*. Even if a state t of an object supports enough QoS, an application cannot obtain enough QoS from the state t if a method *op* is not well facilitated to manipulate the object with its QoS. For example, a video object v is composed of fully coloured video data. However, if the object v supports only a monochromatic version of *display* method, an application can only watch the monochromatic video. Thus, QoS supported by an object depends on QoS of both state and methods. Nemoto and Takizawa [7] discuss *state* and *QoS* aspects of multimedia objects and various types of consistent relations among states of an object.

Each object is allowed to be manipulated by a subject only if the subject is granted an access right to manipulate the object. An access right (or permission) is specified in a pair $\langle o, op\rangle$ of an object o and a method op. Only a subject s who is granted an access right $\langle o, op\rangle$ is allowed to manipulate an object o through a method op. There are mandatory and discretionary approaches to authorizing access rights [9]. In the mandatory approach, only an authorizer grants and revokes access rights to and from subjects. In the discretionary approach, a subject s granted an access right can furthermore grant and revoke the access right into and from another subject s', respectively.

2.2. Layered structure

Objects are distributed to multiple peers on huge number of peers in a peer-to-peer (P2P) overlay network. Our system model is composed of two layers, *logical* and *physical* ones, as shown in Figure 1. A logical model is composed of logical objects which are independent of distribution of objects. An application manipulates logical objects with QoS requirement without being conscious of where the objects exist in networks and how to manipulate the objects. A logical request is specified in a form $\langle O, OP, Q\rangle$ for a logical object O, a method OP, and QoS requirement Q. A physical model is composed of physical objects. A *physical object* is a unit of distribution of a logical object O. A physical object may be a replica, part, and less-qualified version of a logical object O.

A *directory* shows a mapping information among the logical and physical models. The ontology-based directory [5] is also supported to find logical and physical objects from the properties. Users specifies their requests with keywords. Various types of relations on keywords and logical objects are specified in ontology. A user's request is translated into logical requests by using the ontology. Then, for a given identifier *id* of a logical object O, physical objects with location information are found through the directory.

In this paper, we assume that every peer knows a logical model and a logical object identifier of a target logical object is obtained through the directory. Each peer can only have a part of the mapping information and the physical model. In addition, peers may be faulty, leave and join the network, and change their physical objects. It takes time to propagate the charge of objects and distribution of objects to every peer in the network. Hence, some pair of peers may have inconsistent mapping information.

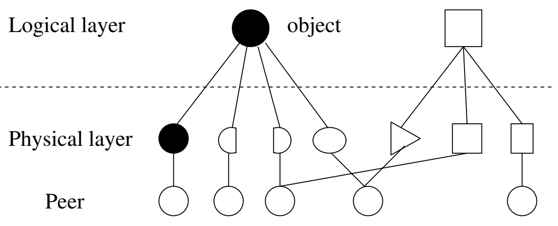

Figure 1. Layers.

2.3. Physical objects

A term "object" means a physical object for simplicity in this paper. Objects are stored in an object base OB_i of each peer c_i. Objects are related as follows:

1. If an object o_i has the same attributes as an object o, o_i is *fully instantiated* for o ($o_i \equiv^I o$). Otherwise, o_i is *partially instantiated* ($o_i \subseteq^I o$).
2. An object o_i is *fully equipped* for an object o ($o_i \equiv^E o$) if o_i supports a same set of methods as o. Otherwise, o_i is *partially equipped* ($o_i \subseteq^E o$).
3. If an object o_i supports the same QoS as an object o, o_i is *fully qualified* for o ($o_i \equiv^Q o$). If o_i supports lower and higher QoS than o, o_i is *less* and *more qualified* than o ($o_i \subseteq^Q o$ and $o \subseteq^Q o_i$), respectively.

An object o_i is *full* for an object o ($o_i \equiv o$) if o_i is fully instantiated, qualified, and equipped for o. Otherwise, o_i is *partial* for o. $o_i \equiv^{IEQ} o$ means that $o_i \equiv^I o$, $o_i \equiv^E o$, and $o_i \equiv^Q o$. $o_i \subseteq^{IEQ} o$ shows that $o_i \subseteq^I o$, $o_i \subseteq^E o$, and $o_i \subseteq^Q o$. $o_i \equiv o$ and $o_i \subseteq o$ indicate $o_i \equiv^{IEQ} o$ and $o_i \subseteq^{IEQ}$

o, respectively. Notations like $o_i \equiv^{IE} o$ and $o_i \subseteq^{EQ} o$ are similarly used for simplicity.

Let us consider an example of a *movie* object o which is composed of *plane* and *background* objects and which supports methods *display* and *delete* which are denoted by symbols $\bigcirc$ and $\triangle$, respectively [Figure 2]. An object o_1 is full for a logical object O ($o_1 \equiv O$). Then, an object o_2 is derived by copying the *background* object of o_1. o_2 has a same set of methods as o_1. Here, $o_2 \subseteq^I o_1$ and $o_2 \equiv^{EQ} o_1$. A pair of objects o_3 and o_4 are also derived from o_1. $o_3 \subseteq^E o_1$ and $o_3 \equiv^I o_1$. A *display* method is less qualified in o_3 than a *display* method of o_1. $o_3 \subseteq^Q o_1$. $o_4 \equiv^I o_1$ and $o_4 \subseteq^E o_1$. In addition, QoS of a *background* object in o_4 is more degraded than o_1. Here, $o_4 \subseteq^Q o_1$. Relations among objects are represented in an *object graph* as shown in Figure 2. Each node shows an object. For types of full relations $o_i \equiv o_j$, $o_i \equiv^I o_j$, $o_i \equiv^E o_j$, and $o_i \equiv^Q o_j$ among objects o_i and o_j, there is an edge $o_i \overset{\alpha}{-} o_j$ if $o_i \equiv^\alpha o_j$ for $\alpha \in 2^{\{I,Q,E\}}$. For types of partial relations $o_j \subseteq o_i$, $o_j \subseteq^I o_i$, $o_j \subseteq^E o_i$, and $o_j \subseteq^Q o_i$ among objects o_i and o_j, there is a directed edge $o_i \overset{\alpha}{\dashrightarrow} o_j$ if $o_i \supseteq^\alpha o_j$ for $\alpha \in 2^{\{I,Q,E\}}$.

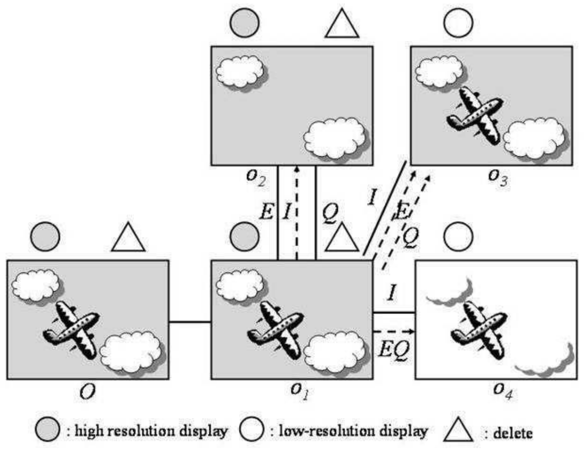

Figure 2. Relations among objects.

3. Acquaintances

3.1. Peer-to-object relation

We discuss how to find a peer which supports multimedia objects which satisfy QoS requirement and which can manipulate the objects in peer-to-peer (P2P) overlay networks. First, an application issues a logical request $\langle O, OP, Q\rangle$ with QoS requirement Q to a local peer c to manipulate a logical object O with a logical method OP. Then, the local peer first finds a target (physical) object o of the logical object O which satisfies the QoS requirement Q. An application has to find peers which can manipulate target objects, i.e. which is granted an access right on the target objects with QoS which satisfy their requirements. In addition, the type and quality of service supported by each peer are dynamically changed, e.g. due to faults, congestions, and version up. We discuss what kinds of relations exist among peers and objects:

- A peer c serves an object o ($c \mid o$) if the object o is stored in the peer c, i.e. c is a server of o.
- A peer c can manipulate an object o through a method *op* ($c \models_{op} o$), i.e. c is granted an access right $\langle o, op\rangle$.
- A peer c can grant an access right $\langle o, op\rangle$ to another peer ($c \vdash_{op} o$).

The server c of an object o is assumed to manipulate the object o for a method *op* ($c \models_{op} o$) if c serves o ($c \mid o$). A peer c can manipulate an object o ($c \models o$) if $c \models_{op} o$ for some method *op*.

In the discretionary control, a *granter* peer c can grant an access right $\langle o, op\rangle$ to another peer ($c \vdash_{op} o$) if the peer c is granted the access right $\langle o, op\rangle$. Another peer c' can ask the granter peer c to grant an access right $\langle o, op\rangle$ on the object o to the peer c' if the peer c' is not granted. A peer c can grant an access right on an object o to another peer ($c \vdash o$) if $c \vdash_{op} o$ for some method *op*. If an object o takes a mandatory access control, a peer c cannot grant an access right to other peers ($c \nvdash o$) even if $c \models o$. Only an owner peer of the object o can grant access rights to other peers. A peer c can do something on an object o ($c \triangle o$) iff one of the relations $c \mid o$, $c \models o$, and $c \vdash o$ holds but it is not sure which one of the relations holds. A perception relation $c \square o$ iff $c \mid o$, $c \models o$, $c \vdash o$, or $c \triangle o$ for a peer c and an object o. For a peer c and an object o, $c \rightarrow c \square o$ if $c \square o$.

3.2. Acquaintance relations

There are following types of acquaintances of a peer c: Knowledge on what a peer c knows is stored in an *acquaintance base* (AB) of the peer c.

- A relation $c \rightarrow c_i \mid o$ holds iff a peer c knows that another peer c_i serves an object o [Figure 3].
- $c \rightarrow c_i \models_{op} o$ iff a peer c knows that c_i can manipulate an object o through a method *op*. $c \rightarrow c_i \models o$ if $c \rightarrow c_i \models_{op} o$ for some method *op* [Figure 3].
- $c \rightarrow c_i \vdash_{op} o$ if a peer c knows that c_i can grant an access right $\langle o, op\rangle$. $c \rightarrow c_i \vdash o$ if $c \rightarrow c_i \vdash_{op} o$ for some method *op*.
- $c_1 \rightarrow^* c_2$ for a pair of peers c_1 and c_2 iff $c_1 \rightarrow c_2$ or $c_1 \rightarrow c_3 \rightarrow^* c_2$ for some peer c_3.
- $c_1 \square^* o$ iff $c_1 \square o$ or $c_1 \rightarrow^* c_2 \square o$ for some peer c_2, i.e. a peer c_1 directly or indirectly makes an access to an object o.
- $c_1 \square^+ o$ iff $c_1 \rightarrow^* c_2 \square o$ for some peer c_2, i.e. a peer c_1 accesses to an object through some peer.

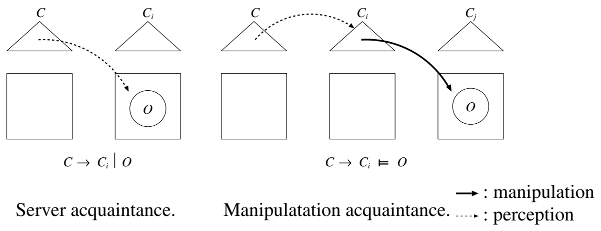

Figure 3. Acquaintance.

An *acquaintance* of a peer c_i is another peer c_j which knows where objects are served and can be manipulated:

- A peer c_j is an acquaintance of a peer c_i for an object o with respect to a perception relation $\Box$ ($c_i \Rightarrow_o^{\Box} c_j$) if $c_i \to c_j \Box^* o$ for an object o.
- A peer c_j is an acquaintance of c_i for o ($c_i \Rightarrow_o c_j$) if $c_i \to_o^{\Box} c_j$ for some perception relation $\Box$.
- A peer c_j is an *acquaintance* of c_i ($c_i \Rightarrow c_j$) if $c_i \Rightarrow_o c_j$ for some object o.

Let *view*(c_i) be a set $\{ c_j \mid c_i \Rightarrow c_j \}$ of acquaintance peers of a peer c_i. The acquaintance relation $\Rightarrow$ is reflexive but is neither symmetric nor transitive. For a peer c_i and an object o, the following sets of objects and peers are defined.

- $\mathbf{O}(c_i) = \{ o \mid c_i \Rightarrow_o c_j$ for some $c_j \}$.
- $\mathbf{S}(c_i, o) = \{ c_j \mid c_i \Rightarrow_o^{\mid} c_j$, i.e. $c_i \to c_j \mid^* o \}$.
- $\mathbf{M}(c_i, o) = \{ c_j \mid c_i \Rightarrow_o^{\models} c_j \}$.
- $\mathbf{G}(c_i, o) = \{ c_j \mid c_i \Rightarrow_o^{\vdash} c_j \}$.
- $\mathbf{U}(c_i, o) = \{ c_j \mid c_i \Rightarrow_o^{\triangle} c_j$, i.e. $c_i \Rightarrow c_j \triangle^* o \}$
  $= \mathbf{S}(c_i, o) \cup \mathbf{M}(c_i, o) \cup \mathbf{G}(c_i, o)$.
- $\mathbf{A}(c_i, o) = \{ c_j \mid c_i \Rightarrow_o c_j \}$ ($\subseteq$ *view*(c_i))
  $= \mathbf{S}(c_i, o) \cup \mathbf{M}(c_i, o) \cup \mathbf{G}(c_i, o) \cup \mathbf{U}(c_i, o)$.

A peer which is granted an access right on an object is a *surrogate* peer of the object. If the peer c_i has a surrogate c_j of a server c_k, the peer c_i can ask the surrogate c_j to make an access to the server c_k [Figure 4]. The surrogate c_j issues requests to the server c_k on behalf of c_i. Secondly, the acquaintance c_k notifies c_i of the granter c_k. Here, the peer c_k is now an acquaintance of the peer c_i ($c_i \to c_k \vdash o$). The information is added to the acquaintance base AB_i of the peer c_i and is propagated to the acquaintances. Then, the peer c_i asks the granter c_k to grant an access right on the object o to c_i. If granted, c_i manipulates o.

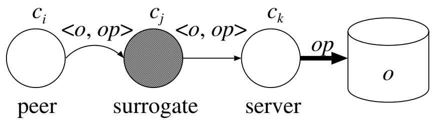

Figure 4. Surrogate.

There are multiple acquaintance peers on an object o for each peer c_i. The peer c_i has to find some acquaintance c_j which knows something about an object o. The acquaintance relation "$c_i \Rightarrow c_j$" is weighted by the *trustworthy* factor f (= $W(c_i, c_j)$) ($c_i \overset{f}{\Rightarrow} c_j$). Suppose $c_i \overset{f_1}{\Rightarrow} c_j$ and $c_i \overset{f_2}{\Rightarrow} c_k$. If $f_1 > f_2$, the peer c_i considers a peer c_j to be more *trustworthy* than c_k. Here, the weight $|c \Box o|$ of a relation $c \Box o$ is defined to be as $|c \vdash o|$ = 3, $|c \models o|$ = 2, $|c \mid o|$ = 1, and $|c_1 \to c_2 \Box^* o| = |c_2 \Box^* o| + 1$. The trustworthy factor f_j of $c_i \Rightarrow_o c_j$ is defined to be 1 / $|c_j \Box^* o|$. The larger the trustworthy weight of an acquaintance c_j is, the larger possibility c_j owns an object.

The acquaintance base AB_i of a peer c_i is composed of information on what objects are stored in what computers, $\mathbf{S}(c_i, o)$ for every object o in $\mathbf{O}(c_i)$.

4. Charge-based Flooding Algorithm

An application issues a logical request $\langle O, OP, Q \rangle$ where O is a logical object, OP is a method of the logical object O, and Q is QoS requirement. We have to find target objects of the logical object O supporting a method *op* of the logical method OP which satisfies QoS requirement Q. There are many discussions on how to find target objects in a P2P overlay network. In the centralized way like Napster [1], an index showing locations of (physical) objects is stored in one peer. In the flooding algorithms [2, 11], a peer asks some number of other peers if they have objects which the peer would like to get. If not, each of the peers furthermore asks other peers. In order to resolve indefinite circulation and explosion of messages, each peer sends a request to only a limited number of peers and each request is assigned with a counter like TTL (time-to-live) [11] and HTL (hops-to-live) [2]. Each time a request is forwarded to another peer, the counter is decremented. If the counter gets zero, the request message is thrown away. In the distributed hashing way [10, 12–14], peers are totally ordered, e.g. by their addresses. Peers to which a request is issued are selected by a hashing function.

A request is first sent to some peers in the flooding algorithm. Then, the request is forwarded to other peers if objects satisfying the requirement are not obtained in the peer. Here, suppose there are multiple peers to which the request can be sent to find the objects. Even if there might be bigger possibility to find a solution in one way, a same integer value of TTL or HTL is assigned for a request on every way in traditional flooding algorithms. We newly introduce a concept of *charge* which is given to a request and which shows the total amount of allowable communication overheads, i.e. number of messages to be transmitted. The more a request is charged, the more number of peers can be accessed.

For each logical request $\langle O, OP, Q\rangle$, a physical request $A = \langle o, op, Q\rangle$ is obtained through the directory and ontology. The application tries to find a target peer as follows:

1. First, a surrogate which is granted an access right $\langle o, op\rangle$ and can support enough QoS Q is found in a P2P overlay network. If found, the application negotiates with the surrogate peer to manipulate the object o through the method *op*.
2. If not found or no surrogate agrees on manipulating the object o, a granter peer of the object o is searched. If found, the application negotiates with the granter peer to grant to the access right $\langle o, op\rangle$ to the application.

The request A is charged with some integer value V, *A.charge* := V. The request A is sent to an acquaintance peer c. Here, *A.charge* is decremented by one, *A.charge* := *A.charge* - 1. If the request A is not satisfiable on manipulating objects in the peer c and is still charged, a set *Cand*(A, c) of candidate acquaintances which knows something about the object o. The request A is *hopeful* on a peer c if *Cand*(A, c) $\neq \phi$. Otherwise, A is *hopeless*. For the hopeful request A, some acquaintances *Target*(A, c) ($\subseteq$ *Cand*(A, c)) are selected. If $|$*Target*(A, c)$| > 1$, i.e. *Target*(A, c) = $\{c_1, \cdots, c_m\}$ ($m \geq 1$), the request A is split into subrequests $A_1, \cdots, A_m$ where each A_i is sent to a peer c_i ($i = 1, \cdots, m$). Here, the charge is allocated to the subrequests $A_1, \cdots, A_m$ based on the weight factors. Let $W(c, c_i)$ show the weight of an acquaintance peer c_i for a peer c. A_i*.charge* := *A.charge* $\cdot\ \alpha_i$ where $\alpha_i = W(c, c_i)$ / $\sum_{j=1}^{m} W(c, c_j)$. That is, the more trustworthy a peer c_i is, the larger amount of charge is allocated to a subrequest A_i.

Some QoS requirement *Q* is specified in each request A. Suppose objects are manipulated in a peer c by the request A. The request A carries a variable *A.state* whose initial value is U (unsuccessful). If *Q* is satisfied on a peer c_i and a request A is performed on the peer c_i, *A.state* is charged with S (successful). After the object o is manipulated by a request A, *A.state* := S. "*A.state* = S" means that the request A has so far visited some peer where objects are successfully manipulated.

If *A.charge* = ϕ, a request A cannot be anymore forwarded to other peers. The response of the request A returns to the preceding peer from the current peer c if *A.state* = S. Otherwise, the request A is discarded. In another case, A is *hopeless*, i.e. *Cand*(A, c) = ϕ but *A.charge* > 0. The response of the request A returns to the preceding peer c'. The request A waits for responses from the other subrequests. If the response of another subrequest A' returns to the peer c', *A.charge* := *A.charge* + A'*.charge*. *A.state* := S if *A.state* = U and A'*.state* = S. Suppose the responses of all the subrequests return to c'. If *A.charge* = 0, the response further returns to the preceding peer c. *Target*(A, c') := *Cand*(A, c') - *Target*(A, c'). If *Target*(A, c') $\neq \phi$, the request A is issued to peers in *Target*(A, c').

[Transmission of a request A on a peer c]

1. Initially, an application is initiated on a peer c. The application issues a request A. *A.charge* := V for some charge V and *A.state* := U.
2. A set *Cand*(A, c) of acquaintance peers of the peer c for the QoS requirement *Q* is obtained for the request A. If the request A is hopeless, the response returns to the preceding peer from the current peer c.
3. A set *Target*(A, c) ($\subseteq$ *Cand*(A, c)) = $\{c_1, \cdots, c_m\}$ of target peers is obtained for the request A.
4. The request A is split to subrequests $A_1, \cdots, A_m$ which are in parallel issued to the target peers $c_1, \cdots, c_m$, respectively. Here, A_i*.charge* := A.charge $\cdot\ \alpha_i$ ($i = 1, \ldots, m$).
5. If a request A is issued to a peer c, *A.charge* := *A.charge* - 1. The peer manipulates objects if the objects can be manipulated in the peer c.
6. If the response of a subrequest A' returns to a peer c, *A.charge* := *A.charge* + A'*.charge*. *A.state* := S if A'*.state* = S. **go to** 2.
7. If responses of all the subrequests return to the request A, other targets in *Target*(A, c) are found. If target peers are found, **go to** 4. Otherwise, the response of A furthermore returns to the preceding peer.

5. Evaluation

We evaluate the charge-based flooding algorithm compared with the Gnutella flooding algorithm [11] in terms of messages transmitted in a P2P overlay network. Here, peers are interconnected in the mesh structure where each peer is connected with four neighboring peers. Replicas of an object are randomly distributed to some number of peers in the network. In the Gnutella flooding algorithm, one peer sends a request to four neighboring peers. If none of the neighboring peers has an object, each of the neighboring peers forwards the request to three neighboring peers. If the same request is received by a peer after receiving a request, the request is discarded. In the full flooding algorithm, duplicate requests are not discarded. By using TTL, if a request hops some number of peers, the request message is discarded. In the charge-based algorithm, the trustworthy factor is randomly assigned to each peer. Based on the weight factors of neighboring peers, each peer decides on the amount of the charge. In the evaluation, 250,000 peers are distributed in a 500×500 mesh. Here, let τ show how many percentages of the peers have target objects in the mesh. For example, τ = 1 [%] means 2,500 peers have replicas of an object.

Figure 5 shows the total number of messages transmitted to find a target peer for the charge-based and Gnutella algorithms where τ = 1 [%]. In the charge-based algorithm, each request is charged with the maximum number of messages which are transmitted by the flooding algorithm for each TTL value. For example, 8744 messages are transmitted in the Gnutella flooding algorithm for TTL = 7. Hence, a request is initially charged 8744 in the charge-based algorithm. The charge-based algorithm implies about 30% fewer number of messages are transmitted than the Gnutella flooding algorithm.

In the flooding algorithms, request messages are distributed to find objects. Request messages which arrive at target peers are *successful* but the others are *unsuccessful*. The ratio of the number of successful messages to the total number of messages is referred to as the *successful ratio*. Figure 6 shows the successful ratios of the algorithms for τ = 1 [%]. The fewer number of target peers implies the larger successful ratio in the charge-based algorithm. The more number of messages can arrive at target peers by the charge-based algorithm than the others.

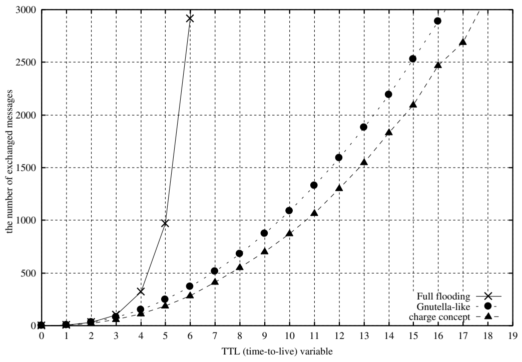

Figure 5. Number of messages (τ = 1 [%]).

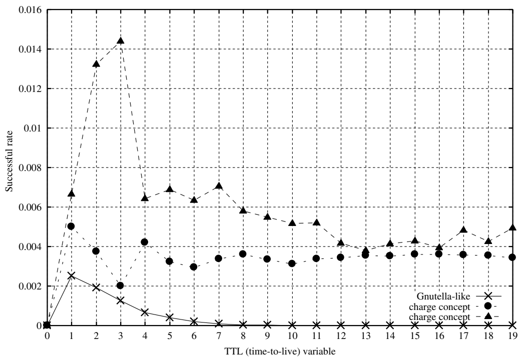

Figure 6. Successful ratio (τ = 1 [%]).

6. Concluding Remarks

We discussed how to detect and manipulate multimedia objects distributed in a P2P overlay network. Multimedia objects are distributed in various ways, fully or partially instantiated, qualified, and equipped in the networks. We discussed how a peer can manipulate objects distributed in peers and grant access rights. Then, we discussed charge-based flooding algorithm to find objects through acquaintances. We evaluated the algorithm for detecting peers compared with other flooding algorithms in terms of the number of messages transmitted in the network. The charge-based algorithm implies the higher hit ratio and fewer number of messages.

References

[1] Napster. http://www.napster.com.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, 2000.

[3] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. *Proc. of the 22nd IEEE International Conference on Distributed Computing System*, pages 23–32, 2002.

[4] F. M. Cuenca-Acuna, R. P. Martin, and T. D. Nguyen. PlaneP: Using Gossiping and Random Replication to Support Reliable Peer-to-Peer Content Search and Retrieval. *Rutgers University technique report*, 2002.

[5] N. Guarino. Formal Ontology and Information Systems. *Proc. of FOIS 98*, pages 3–15, 1998.

[6] Y. Liu, Z. Zhuang, X. Li, and M. N. Lionel. A Distributed Approach to Solving Overlay Mismatching Problem. *Proc. of the 24th IEEE International Conference on Distributed Computing System (ICDCS2004)*, pages 132–139, 2004.

[7] N. Nemoto, K. Tanaka, and M. Takizawa. Quality-Based Concurrency Control for Multimedia Objects. *Proc. of the 7th International Conference on Distributed Multimedia Systems*, pages 218–225, 2001.

[8] OMG Inc. The Common Object Request Broker : Architecture and Specification. *Rev. 2.1*, 1997.

[9] Oracle8i Concepts Vol. 1. 1999. Release 8.1.5.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. *Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, 2001.

[11] M. Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. *Proc. of International Conference on Peer-to-Peer Computing (P2P2001)*, pages 99–100, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[13] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[14] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-resilient Wide-area Location and Routing. *Technical Report UCB//CSD-01-1141*, 2001.