# Distributed Multimedia Objects in Peer-to-Peer Overlay Networks

Kenichi Watanabe, Tomoya Enokido, and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University
E-mail {nabe, eno, taki}@takilab.k.dendai.ac.jp

Multimedia objects are distributed and replicated in peer-to-peer (P2P) overlay networks since objects are cached, downloaded, and personalized in local peer computers. An application has to find target peer computers which support enough types and quality of service of objects. Each peer has acquaintance peers which support information on where objects exist and can be manipulated. We discuss how to obtain access rights to detect and manipulate objects with help and cooperation of acquaintances. We discuss a new type of flooding algorithm to find target peers based on charge concept so that areas where target peers are expected to exist are more deeply searched and explosion of messages is prevented.

# P2Pオーバレイネットワーク上に分散したマルチメディアオブジェクト

渡辺 健一　榎戸 智也　滝沢 誠
東京電機大学大学院理工学研究科情報システム工学専攻

P2Pオーバレイネットワークでは、種々のマルチメディアオブジェクトがピア内に分散される。アプリケーションは、十分なQoSを提供するオブジェクトをもつピアを見つける必要がある。各ピアは、オブジェクトの存在位置やその操作方法に関する情報を提供する知人ピアをもつ。本論文では知人ピアを用いたアクセス権の取得方法やオブジェクトの検索方法について議論する。また、オブジェクトを保持するピアコンピュータの検索方法として、新たに電荷という概念をベースとした拡散検索方式を提案する。

## 1 Introduction

According to the development of common frameworks of computers and networks, various types and huge number of computers are now interconnected in *peer-to-peer* (P2P) overlay networks [3]. Service supported by multimedia objects in a P2P overlay network is characterized by quality of service (QoS) like frame rate and number of colours. In multimedia applications, multimedia objects are replicated and distributed in nature on multiple computers since objects are downloaded, cached, and personalized in local computers. In addition, only a part of a multimedia object may be stored in a computer. Furthermore, an object downloaded in computers may support QoS different from the original object. It is critical to support applications with sufficient, possibly necessary QoS in change of services supported by computers and networks. A P2P overlay network is stateless infrastructure where huge number of peer computers 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 peers which can manipulate the object or replica of the object with enough QoS. It is difficult for each peer to perceive what objects are distributed to what peers. Each peer has *acquaintance* peers which the peer perceives what objects with what QoS are stored, perceived, or can be manipulated. A *view* of a peer is a set of the acquaintances. A peer first asks acquaintances in its view to detect objects, obtain access rights, and manipulate objects. If objects which satisfy QoS requirements are not detected in the view, an acquaintance in turn asks its acquaintances. In flooding algorithms [2, 5] to find objects in P2P overlay networks, a peer asks every acquaintance if the peer supports objects and each of the acquaintances furthermore asks its acquaintances. Counters like TTL (time-to-live) [5] and HTL (hops-to-live) [2] are used to prevent indefinite circulation and explosion of request messages. If there are multiple candidate ways to find objects, some way with higher possibility to find the objects should be more deeply searched. A request is assigned some *charge*. The charge of the request is decreased each time the request is passed over a peer. If there are multiple routes from a peer, a request is transmitted to each route with charge. If there is larger possibility to find an object in a route, a request to the route is more charged.

In section 2, we present a system model. In section 3, we discuss distribution of objects in peers. In section 4, we discuss acquaintances of each peer. In section 5, we discuss how to detect objects. In section 6, we evaluate the charge-based flooding algorithm.

## 2 System Model

An object is an instantiation of a *class*. A class is composed of attributes and methods for manipulating its object. An object is an encapsulation of values of attributes and methods. A collection of attribute values are referred to as *state* of an object which is characterized by 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 if the object is manipulated

by 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. Thus, QoS supported by an object depends on QoS of both state and method.

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*. A role is a collection of access rights, which shows a job function in an enterprise.

Suppose a subject $s$ invokes a method $op_1$ on an object $o_1$ where $s$ is granted an access right $\langle o_1, op_1 \rangle$. Then, a method $op_2$ on an object $o_2$ is invoked in the method $op_1$. There are *centralized* and *distributed* approaches to controlling access to objects. In the centralized approach, the subject $s$ invokes the method $op_2$. Here, the subject $s$ is allowed to manipulate the object $o_2$ only if $s$ is granted an access right $\langle o_2, op_2 \rangle$. In the distributed approach, the method $op_2$ is invoked in a method of an object $o_2$ on behalf of the subject $s$. The method $op_2$ is allowed to be performed only if the access right $\langle o_2, op_2 \rangle$ is granted to the invoker object $o_1$. Even if the subject $s$ is not granted the access right $\langle o_2, op_2 \rangle$, the method $op_2$ is invoked by the object $o_2$ if the object $o_2$ is granted the access right $\langle o_2, op_2 \rangle$. The object $o_1$ plays a role of *surrogate* of the subject $s$ to manipulate the object $o_1$.

# 3 Distribution of Objects

## 3.1. Layered structure

There are two layers, *logical* and *physical* ones, in a P2P overlay network [Figure 1]. A logical model is composed of only logical objects which are independent of distribution of objects in peers. Each logical object is identified by an logical object identifier (LOID). An application manipulates logical objects with QoS requirement without being conscious of where the objects exist in networks. A logical request is specified in a form $\langle o, op, Q \rangle$ where $o$ is a logical object, *op* is a method of the object $o$, and $Q$ is QoS requirement.

Each logical object $o$ is distributed to peers at physical layer. A unit of distribution of the logical object $o$ is referred to as *physical object*. A physical model is composed of physical objects. A physical object may be a part of the logical object $o$ and may be one obtained by changing QoS of the logical object $o$. Each physical object is identified by a physical object identifier (POID).

A *directory* shows a mapping information among the logical and physical models. 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 of a P2P overlay network. Each peer can only have a part of the mapping information and the physical model. Peers may be faulty, leave and join the network, and change their physical objects, e.g. downloading and caching. It takes time to propagate the charge to every peer in the network. Hence, some pair of peers have inconsistent mapping information.

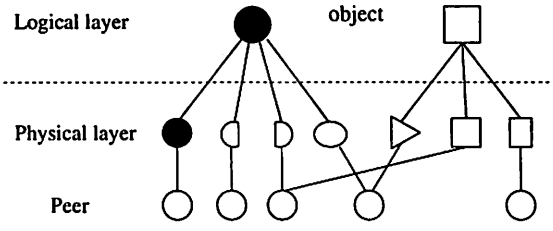


**Figure 1. Layers.**

## 3.2. Physical objects

Let $\rho(o)$ be a collection $\{o_1, \ldots, o_n\}$ of physical objects of a logical object $o$ ($n \geq 1$). A replica of an object is also a physical object. A term "object" means a physical object for simplicity in this paper. Objects are classified with respect to what part of state and methods of a logical object $o$ are supported by the objects with what QoS.

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

Let us consider an example of a *movie* object $o$ which is composed of *plane* and *background* objects [Figure 2]. The *movie* object $o$ supports methods *display* and *delete*. An object $o_1$ is *full* for a logical object $o$ ($o_1 \equiv o$ or $o_1 \equiv^{IEQ} o$). Then, an object $o_2$ is derived by copying the *background* object of the object $o_1$. The object $o_2$ 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 the object $o_1$. $o_3 \subseteq^E o_1$ and $o_3 \equiv^I o_1$. A *display* method is less qualified in the object $o_3$ than *display* 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 the object $o_4$ is more degraded than $o_1$. Here, $o_4$ is less qualified than $o_1$, i.e. $o_4 \subseteq^Q o_1$.

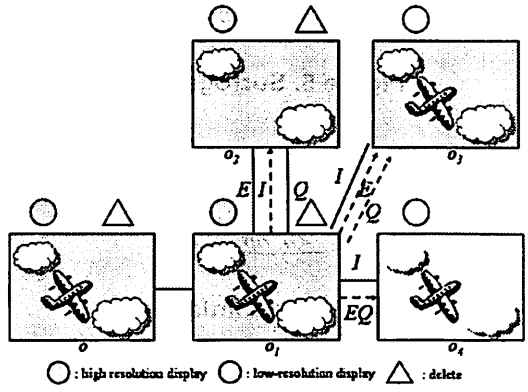


**Figure 2. Relations among Objects.**

Relations among objects are represented in a directed graph named *object graph*. Each node shows an object. A directed edge from a node $o_i$ to another node $o_j$ shows the relations $\equiv$, $\equiv^I$, $\equiv^Q$, $\equiv^E$, $\subseteq^I$, $\subseteq^Q$, and $\subseteq^E$. 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 are following directed straight edges from an object node $o_i$ to another node $o_j$; $o_i - o_j$ if $o_i \equiv o_j$ and $o_i \overset{\alpha}{-} o_j$ if $o_i \equiv^\alpha o_j$ for $\alpha \in \{I, Q, E\}$. 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 are following directed dotted edges from a node $o_i$ to another node $o_j$; $o_i \dashrightarrow o_j$ if $o_i \supseteq o_j$ and $o_i \overset{\alpha}{\dashrightarrow} o_j$ if $o_i \supseteq^\alpha o_j$.

## 4. Acquaintances

We discuss how to find a peer what has multimedia objects which satisfies QoS requirement and manipulate the objects in P2P overlay networks. An application has to find peers which can manipulate objects, maybe replicas of a logical object with QoS which satisfy applications' requirements. Here, a target object shows an object which a peer would like to manipulate. In addition, services supported by peers are dynamically changed. Since types of objects are distributed to a large number of peers and objects are changed time by time, it is not easy to find target peers.

Each peer computer $c$ is composed of an object base $OB$ which is a collection of objects. We discuss what kinds of relations exist among subjects, i.e. peers and objects. First, a peer $c$ is referred to as *serve* an object $o$ (written as $c \mid o$) if the object $o$ is stored in the object base of the peer $c$. The peer $c$ is also referred to as *server* of the object $o$. An object $o$ is manipulated through a method $op$ even by a remote peer.

A peer $c$ can manipulate an object $o$ through a method $op$ ($c \models_{op} o$) if the peer $c$ is granted an access right $\langle o, op \rangle$. 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 $op$. If access to an object $o$ is controlled in a discretional way, access rights granted to a peer $c$ can be further granted to other peers by the peer $c$. A 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 $\langle o, op \rangle$. The peer $c$ can revoke access rights. Here, the peer $c$ is referred to as *granter* of the object $o$. Another peer $c'$ can ask the granter peer $c$ to grant an access right $\langle o, op \rangle$ on the object $o$ to $c'$ if $c'$ is not granted the access right $\langle o, op \rangle$. 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 on the object $o$ 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 holds. A perception relation $\square$ denotes one of the relations $\mid$, $\models$, $\vdash$, and $\triangle$, i.e. $\square \in \{\mid, \models, \vdash, \triangle\}$.

If a peer $c$ knows that another peer $c_i$ serves an object $o$, "$c \rightarrow c_i \mid o$". Similarly, $c \rightarrow c_i \models_{op} o$ if a peer $c$ knows that another peer $c_i$ can manipulate an object $o$ through a method $op$. Here, it is noted that the peer $c_i$ may not be a server of the object $o$. A peer $c$ knows that a peer $c_i$ can manipulate an object $o$ with some method ($c \rightarrow c_i \models o$), i.e. $c \rightarrow c_i \models_{op} o$ for some method $op$. $c \rightarrow c_i \vdash_{op} o$ if a peer $c$ knows that another peer $c_i$ 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 $op$. Knowledge on what a peer $c$ knows is stored in an *acquaintance base* ($AB$) of the peer $c$. Furthermore, $c \rightarrow c_1 \rightarrow c_2 \mid o$ means $c \rightarrow (c_1 \rightarrow (c_2 \mid o))$. That is, a peer $c$ knows that another peer $c_1$ knows that the other peer $c_2$ serves an object $o$ [Figure 4]. A relation $c_1 \rightarrow^* c_2$ holds 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 $c_3$. A peer $c$ may know that another peer $c_i$ knows something about an object $o$ but is not sure which relation $c \rightarrow c_i \mid o$, $c \rightarrow c_i \models o$, or $c \rightarrow c_i \vdash o$ holds. This relation is shown in $c \rightarrow c_i \triangle o$.

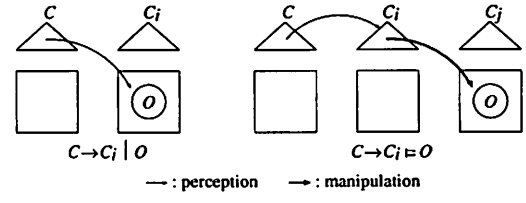


**Figure 3. Acquaintances.**

Each peer $c_i$ has its own *view* which is a subset of peer computers to which the peer $c_i$ can directly perceive in a P2P network. An *acquaintance* of a peer computer $c_i$ is another peer $c_j$ which knows where objects are served and can be manipulated. That is,

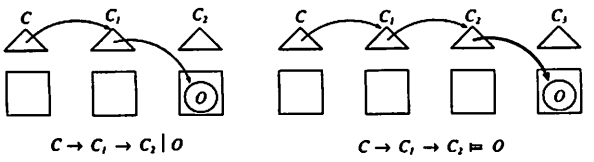


**Figure 4. Acquaintances.**

a peer $c_j$ is an acquaintance of a peer $c_i$ for an object $o$ with respect to a perception relation $\Box$ (written as $c_i \Rightarrow_o^{\Box} c_j$) if $c_i \rightarrow c_j \ \Box^* \ o$ for an object $o$. A peer $c_j$ is an acquaintance of a peer $c_i$ for an object $o$ ($c_i \Rightarrow_o c_j$) if $c_i \rightarrow_o^{\Box} c_j$ for some relation $\Box$. A peer $c_j$ is an *acquaintance* of another peer $c_i$ ($c_i \Rightarrow c_j$) if $c_i \Rightarrow_o c_j$ for some object $o$. Let *view*($c_i$) be a set of acquaintance peers of a peer $c_i$, *view*($c_i$) = { $c_j$ | $c_i \Rightarrow c_j$ }. The acquaintance relation $\Rightarrow$ is reflexive but is neither symmetric nor transitive. Even if a peer $c_i$ thinks another peer $c_j$ to be its acquaintance, the peer $c_j$ may not think the peer $c_i$ to be an acquaintance. A pair of peers $c_i$ and $c_j$ are *friends* iff $c_i \Rightarrow c_j$ and $c_j \Rightarrow c_i$. For a peer $c_i$ and an object $o$, the following sets are defined.

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

Here, $\mathbf{A}(c_i, o) = \mathbf{S}(c_i, o) \cup \mathbf{M}(c_i, o) \cup \mathbf{G}(c_i, o) \cup \mathbf{U}(c_i, o)$.

Suppose a peer $c_i$ would like to manipulate an object $o$. First, a peer $c_i$ asks an acquaintance $c_j$ on an object $o$ in its view *view*($c_i$), $c_j \in \mathbf{A}(c_i, o)$. Suppose that a server $c_j$ of the object $o$ ($c_j$ | $o$) is detected, which satisfies the QoS requirement. Otherwise, the peer $c_i$ has to find another peer $c_k$ which can manipulate an object $o$ ($c_k \models o$) or can grant an access right ($c_k \vdash o$). There are two cases. The peer $c_j$ can make an access to the object $o$ in the server $c_j$. A peer which is granted an access right on an object is a *surrogate* of the object. If the peer $c_i$ has a surrogate $c_j$ of a server $c_k$ of an object $o$, the peer $c_i$ can ask the agent $c_j$ to access to the server $c_k$ [Figure 5]. The surrogate $c_j$ issues requests to the server peer $c_k$ on behalf of the peer $c_i$. In another case, the agent $c_j$ knows that $c_k$ is granted an access right to the object $o$ [Figure 6]. The acquaintance $c_k$ notifies the peer $c_i$ of the granter $c_k$. Here, $c_k$ is now an acquaintance of the peer $c_i$ ($c_i \rightarrow c_k \vdash o$). Then, the peer $c_i$ asks the granter peer $c_k$ to grant an access right on the object $o$ to $c_i$. If granted, the peer $c_i$ manipulates the 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 reliable than $c_k$. Here, the weight $|c \ \Box \ o|$ of a relation $c \ \Box \ o$ for a peer $c$ and an object $o$ is defined to be as follows: $|c \vdash o| = 3$, $|c \models o| = 2$, and $|c \mid o| = 1$. $|c_1 \rightarrow c_2 \ \Box^* \ o|$ is $|c_2 \ \Box^* \ o| + 1$. The trustworthy factor $f_j$ of $c_i \Rightarrow_o c_j$ is defined to be $1 / |c_j \ \Box^* \ o|$.

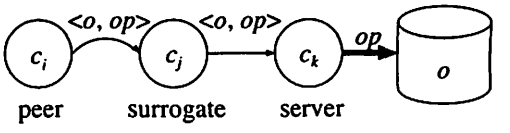


**Figure 5. Surrogate.**

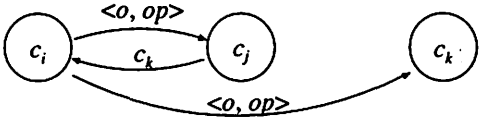


**Figure 6. Acquaintances.**

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

## 5. Detection of Objects

An application issues a 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. We have to find target physical objects of the logical object $o$ supporting the 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 objects is stored in one peer . Every peer first asks the centralized index computer to get location information of an interesting object. Then, a peer is selected in the location information and a request is sent to the peer. In the flooding algorithms [2, 5], 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 counters like TTL (time-to-live) [5] 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 message is thrown away. In the distributed hashing way [4, 6–8], peers are totally ordered, e.g. by their addresses. Peers to which a request is issued are selected by a hashing function. However, it is not easy to adopt the distributed hashing mechanism if objects are *a priori* distributed and objects dynamically join, leave, and change.

In this paper, we discuss a new type of flooding al-

gorithm. Even if there might be bigger possibility to find a solution in one way, some 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 allocated to a request. The charge of a request shows the total amount of communication overheads to find objects. The more a request is charged, the more number of peers can be accessed.

For each request $\langle o, op, Q \rangle$, the application agent tries to find a peer as follows:

1. First, a surrogate peer which is granted an access right $\langle o, op \rangle$ and can support enough QoS is found in an overlay network. If found, the application agent 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 is searched. If found, the application agent negotiates with the granter peer to grant to access right.

Initially, an application agent in a peer computer issues a request $A$ to find an object $o$. The request $A$ is charged for some integer value $V$, $A.charge := V$. The request $A$ is sent to another 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 acquaintance agents $c$ which knows something about objects which $A$ would like to manipulate is found. The request $A$ is *hopeful* on a peer $c$ if $Cand(A, c) \neq \phi$. Otherwise, the request $A$ is *hopeless*. The hopeful request $A$ selects some acquaintances $Target(A, c)$ ($\subseteq Cand(A, c)$). If $|Target(A, c)| > 1$, i.e. $Target(A, c) = \{c_1, \cdots, c_m\}$ ($m > 1$), the request $A$ is split into sub requests $A_1, \cdots, A_m$. Each sub request $A_i$ is sent to a peer $c_i$ ($i = 1, \cdots, m$). Here, the charge is allocated to the sub requests $A_1$, $\cdots$, $A_m$ based on the weight factors. Let $W(c, c_i)$ show the weight of a 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 trust a peer $c_i$ is, the larger amount of charge is allocated to a request $A_i$.

Each request $A$ has some condition $Cond(A)$. The request $A$ tries to find peers which satisfies $Cond(A)$. Suppose the request $A$ finishes manipulating objects in a peer $c$. The request $A$ carries a variable $A.state$ whose initial value is $U$ (unsuccessful). If $Cond(A)$ is satisfied on a peer $c_i$, $A.state = S$ (successful). If $A.state = S$, the request $A$ has so far visited some peer where objects are successfully manipulated.

Suppose a request $A$ finishes manipulating objects in a peer $c$ and $A.charge = \phi$. The request cannot move to other peers due to out-of-charge. The request $A$ backs to the preceding peer from the current peer $c$ if $A.state = S$. Otherwise, the request $A$ is discarded. In another case, the request $A$ is *hopeless*, i.e. $Cand(A, c) = \phi$ but $A.charge > 0$. The request $A$ backs to the preceding peer $c'$. The request $A$ waits for responses other requests. If a sibling request $A'$ backs to the peer $c'$, $A.charge := A.charge + A'.charge$. $A.state := S$ if $A.state = U$ and $A'.state = S$. Suppose all the sibling requests back to the peer $c'$. If $A.charge = 0$, the request $A$ further backs to the preceding peer $c$. $Target(A, c') := Cand(A, c') - Target(A, c')$. If $Target(A, c') \neq \phi$, the request $A$ moves to peers in $Target(A, c')$. A request $A$ is transmitted as follows:

**[Transmission of a request $A$ on a peer $c$]**

1. Initially, an application is initiated on a peer $c$. The application peer issues a request $A$. $A.charge := V$ and $A.state := U$.
2. A set $Cand(A, c)$ of acquaintance peers of $c$ for the condition $Cond(A)$ is obtained for the request $A$. If the request $A$ is hopeless, $A$ backs to the preceding peer from $c$.
3. The request $A$ obtains a set $Target(A, c)$ ($\subseteq Cand(A, c)) = \{c_1, \cdots, c_m\}$ of target peers.
4. The request $A$ is split to sibling requests $A_1, \cdots, A_m$ which move to the target peers $c_1, \cdots, c_m$, respectively. Here, $A_i.charge := A.\text{charge} \cdot \alpha_i$ ($i = 1, \ldots, m$).
5. If a request $A$ moves to a peer $c$, $A.charge := A.charge - 1$. $A$ manipulates objects if the objects can be manipulated in the peer $c$.
6. If a response of sibling request $A'$ backs 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 sibling requests return to the request $A$, other targets $Target(A, c)$ is found for the request $A$. If the target peer is found, **go to** 4. Otherwise, the request $A$ furthermore backs to the preceding peer.

## 6 Evaluation

We evaluate the charge-based flooding algorithm compared with the Gnutella flooding algorithm [5]. Here, peers are interconnected in mesh structure. That is, each peer is physically connected with four neighboring peers. Replicas of an object are randomly distributed to 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 them forwards the request to three neighboring peers. If a same request is received after receiving a request by a peer, the request is discarded. In the full flooding, duplicate requests are not discarded. By using TTL, if some number TTL of peers are hopped by a request, the request message is discarded to avoid the explosion of messages. In the charge-based algorithm, the trustworthy factor is randomly assigned to each peer. Based on the factors of neighboring peers, each peer decides on the charge. Following the figures, the charge-based algorithm sup-

ports smaller communication and higher hit ratio than the flooding algorithm. In the evaluation, 500 × 500 peers are distributed in a mesh. Here, let $\tau$ show how many percentages of the peers have target objects.

Figures 7 and 8 show the total number of messages transmitted to find a target peer for the charge-based and Gnutella algorithms where $\tau$ = 10 [%] and 1 [%], respectively. Figures 9 and 10 show the hit ratio for $\tau$ = 10 [%] and 1 [%], respectively. In the charge-based algorithm, each request is charged with the number of messages which are transmitted by the flooding algorithm for each TTL. For example, 8744 messages are transmitted for TTL = 7. For TTL = 7, a request is charged 8744 in the charge-based algorithm.

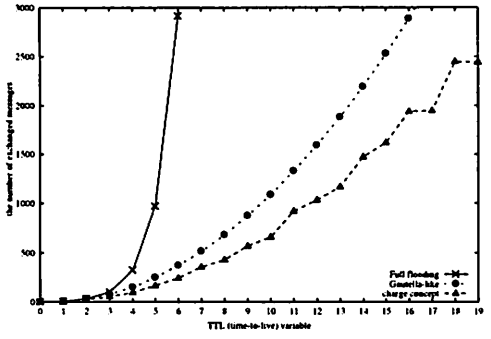

**Figure 7. Number of messages ($\tau$ = 10 [%]).**

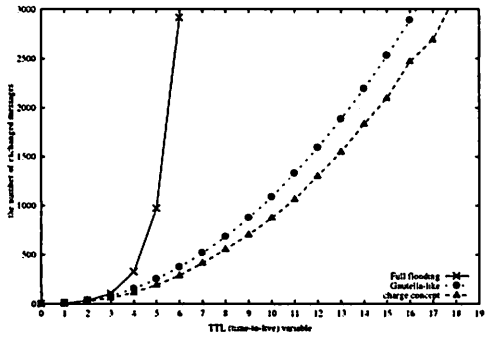

**Figure 8. Number of messages ($\tau$ = 1 [%]).**

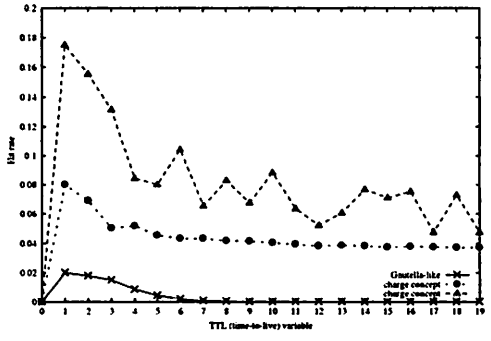

**Figure 9. Hit ratio ($\tau$ = 10 [%]).**

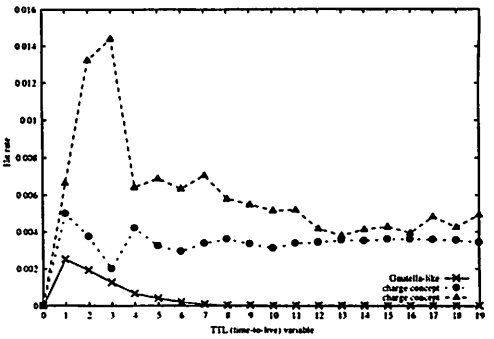

**Figure 10. Hit ratio ($\tau$ = 1 [%]).**

## 7 Concluding Remarks

We discussed how to manipulate multimedia objects distributed in P2P overlay network. The objects are replicated in various ways, fully or partially instantiated, qualified, and equipped in the networks. These relation among objects and replicas are represented in the object graph. Then, we discussed charge-based flooding algorithm to manipulate objects through acquaintances. We evaluated the algorithm for detecting peers compared with other flooding algorithms.

## 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] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.