2003－DPS－113 (17)
2003／6／20

Autonomic Group Communication Protocol

Tomoya Enokido and Makoto Takizawa
Tokyo Denki University
E-mail {eno, taki}@takilab.k.dendai.ac.jp

Abstract

Multiple peer processes are exchanging multimedia messages with each other in an underlying network. A group protocol supports applications with enough quality of service (QoS) in change of QoS supported by the network. An autonomic group service is supported for applications by cooperation of multiple autonomous agents. Each agent autonomously takes a class of each protocol function like retransmission. Classes taken by an agent are required to be consistent with, but might be different from the others. A group is composed of views in each of which agents autonomously take protocol classes consistent with them.

自律的なグループ通信プロトコル

榎戸 智也　　滝沢 誠
東京電機大学理工学部情報システム工学科

複数の対等なプロセス間での多対多の通信を行うためのグループ通信プロトコルは、メッセージの送受信、紛失の検出、紛失したメッセージの再送信機能などの複数の機能により構成され、この機能を用いることで、様々なサービスをアプリケーションに提供する。しかし、ネットワークが提供するサービスの品質 (QoS) は、輻輳や機器の障害により動的に変化する。そのため、従来のグループ通信プロトコルが提供する固定の機能を用いたサービス方式では、常にアプリケーションの要求を満足することができない。更に、常に高機能なグループ通信機能を用いてサービスを提供し続けた場合、その通信と処理にかかるオーバヘッドが増大するという問題も発生する。本研究では、ネットワークが提供する QoS の動的変化に対して、各計算機上のグループ通信エージェントが、アプリケーションの要求を満たすよう最適なグループ通信機能を選択し、かつ他のグループ通信エージェントと交渉しながら柔軟に対応できる「自律的なグループ通信プロトコル」を提案する。

1. Introduction

Peer-to-Peer (P2P) systems [6] are getting widely used like grid computing [4] and autonomic computing [1]. Group communication is required to realize cooperation of multiple peer processes. In group communications, multiple peer processes first establish a *group* and then messages are exchanged among the processes [2, 7, 8, 10, 11]. There are group protocols which support multiple peer processes with the ordered delivery of messages [2,7,8,10,11]. A group protocol is realized by protocol functions; multicast/broadcast, receipt confirmation, detection and retransmission of messages lost, ordering of messages received, and membership management. There are various ways to realize each of these functions like selective and go-back-n retransmissions.

The complexity and efficiency of implementation of group protocol depends on what types and quality of service (QoS) are supported by the underlying network. QoS parameters like bandwidth are dynamically changed due to congestions and faults. Furthermore, there are various types of networks each of which is characterized by QoS parameters. The higher level of communication function is supported, the larger computation and communication overheads are implied. Hence, the system has to take only classes of functions necessary and sufficient to support service required by applications by taking usage of the underlying network service. The paper [11] discusses an architecture to design a protocol supporting a group of multiple processes which satisfies application requirements. However, the protocol cannot be dynamically changed each time QoS supported by the underlying network is changed. In addition, each process in a group has to use the same group protocol functions. In peer-to-peer applications [6], it is not easy to change protocol functions in all the processes since a large number of processes are cooperating and some computers are not always working well. Each process has a *view* which is a subset of processes to which the process can directly send messages. If a group is too large for each process to perceive QoS supported by other processes and manage the group membership, the group is decomposed into views.

In this paper, we discuss an *autonomic* group protocol which can support types and quality (QoS) of service required by applications even if QoS supported by the underlying network is changed. Each group protocol module is realized in an autonomous agent. An agent autonomously changes implementation of each group protocol function depending network QoS monitored. In addition, an agent negotiates with the other agents in the view so that the classes of protocol functions are consistent with, not necessarily same as the other agents.

In section 2, we show a system model. In section 3, we discuss how each process perceives other processes in a group. In section 4, we discuss classes of protocol functions. In section 5, we present an agent-based architecture to support the autonomic group service. In section 6, we discuss how to change retransmission functions.

2. System Model

A group of multiple *application processes* A_1, ..., A_n ($n \geq 2$) are cooperating by taking usage of group communication service. The group communication service is supported by cooperation of multiple *system processes* p_1, ..., p_n through exchanging messages by using underlying network service. In this paper, a term "*process*" means a system process.

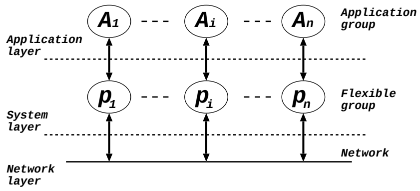

Figure 1. System model.

The underlying network is modeled to be a collection of bidirectional channels among processes. Each channel supports some quality of service (QoS). QoS is characterized by parameters; delay time [msec], message loss ratio [%], and bandwidth [bps]. QoS parameters supported by each channel are changed due to congestions and faults in the network.

A group of multiple processes p_1, ..., p_n (n>1) are exchanging messages in the network. Let $s_i(m)$ denote a sending event of a message *m* in a process p_i. A message m_1 *causally precedes* another message m_2 ($m_1 \rightarrow m_2$) if and only if (iff) $s_i(m)$ *happens before* $r_j(m)$ [2, 7]. m_1 is *causally concurrent* with m_2 ($m_1 \parallel m_2$) if neither $m_1 \rightarrow m_2$ nor $m_2 \rightarrow m_1$. A pair of messages m_1 and m_2 are *causally delivered* iff m_1 is delivered before m_2 in every common destination of m_1 and m_2 if $m_1 \rightarrow m_2$. Here, a pair of causally concurrent messages can be delivered in any order. In the *totally ordered* delivery, all the messages are delivered to every common destination of the messages in the same order.

3. Views in Group

A $group$ G is composed of multiple peer processes p_1, ..., p_n ($n > 1$). In a group G including larger number of processes, it is not easy for each process to deliver messages to all the processes and maintain membership information on all the member processes. Each process p_i has a view $V(p_i)$ which is a subset of processes to which the process p_i can directly send messages or deliver messages via processes. Thus, a view V is a subgroup of G. Each process p_i maintains membership of its view $V(p_i)$. For every pair of processes p_i and p_j, p_i in $V(p_j)$ iff p_j in $V(p_i)$. A pair of different views V_1 and V_2 may include a common process p_k. The process p_k plays a role of a $gateway$ process between processes in V_1 and V_2. If a process p_i belongs to only one view, p_i is referred to as $leaf$ process.

A process p_i in a view V which takes a message m from an application process A_i and sends the message m to processes in the view V is a $sender$ process of m. If a process p_j in a view V delivers a message m to an application process A_j, the process p_j is a $destination$ process of m. If a process p_k receives a message m in a view V and forwards the message m to a process p_l in another view V', the process p_k is a $gateway$ process. Here, if the process p_k forwards the message m to another process in the same view V, the process p_k is a $routing$ process. Let $src(m)$ be an original source process and $dst(m)$ be a set of original destination processes. A $local$ sender and destination processes of a message m are processes which send and receive the message m in a view, respectively.

A view V which includes all the processes in a group G is referred to as $complete$. A $global$ view is a complete view including all the processes in a group G. If $V \subset G$, V is $partial$. A partial view V is changed if a system process joins and leaves the view V. If a view $V(p_i)$ is changed, $V(p_i)$ is $dynamic$. If $V(p_i)$ is invariant, $V(p_i)$ is $static$.

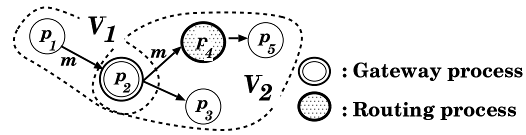

Figure 2. Group views.

Suppose a process p_i in a view V sends a message m to another process p_j. If p_j is in a same view V as p_i, p_i sends m to p_j, possibly m is routed to p_j in the view V. Otherwise, p_i sends m to a gateway process p_k in V. Then, the process p_k forwards m to another view V'. Here, if the destination process p_j is in the view V', the gateway process p_k forwards the message m to the destination process p_j. Otherwise, the gateway process p_k forwards the message m to another gateway process in the view V'. For example, suppose that $dst(m)$ = $\{p_3, p_5\}$ and a sender process p_1 of a message m belongs to a view V_1 = $\{p_1, p_2\}$. Then, p_1 sends a message m to the gateway process p_2 in the view V_1 [Figure 2]. The process p_2 is a member of a view V_2 = $\{p_2, p_3, p_4, p_5\}$. Therefore, the process p_2 delivers the message m to a pair of processes p_3 and p_4 in the same view V_2 In addition, the message m is delivered to the destination process p_5 via the process p_4. Here, p_2 is a local sender process and p_4 is a routing process.

4. Functions of Group Protocol

4.1. Protocol functions

A group protocol among multiple processes p_1, ..., p_n is realized by following protocol functions:

1. Coordination of the processes.
2. Message transmission.
3. Receipt confirmation.
4. Retransmission.
5. Detection of message loss.
6. Ordering of messages.
7. Membership management.

There are multiple ways to realize each of these functions. A $class$ of protocol function shows one way of implementation of the protocol function. We discuss what classes exist for each protocol function in this section.

4.2. Coordination

There are $centralized$ and $distributed$ approaches to coordinating cooperation of processes in a view. In the centralized control, there is one centralized controller in a view V. On the other hand, there is no centralized controller in the distributed control scheme. Each process makes a decision on correct receipt and delivery order of messages received by itself.

4.3. Transmission

There are *centralized*, *direct*, and *indirect* approaches of a process p_i to transmitting a message to multiple processes in a view [Figure 3]. In the centralized transmission, there is one forwarder process in a view V [Figure 3 (1)]. Each process has to exchange messages through the forwarder process in the view V. The forwarder process plays a role of a controller which makes a decision on the delivery order of messages and manages membership in the view V.

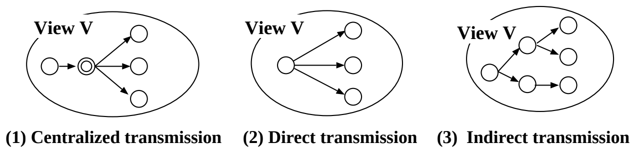

Figure 3. Transmission schemes.

In the direct transmission, each process directly not only sends a message to each destination process but also receives messages from other sender processes in a view V [Figure 3 (2)]. That is, there is no centralized forwarder process.

In the indirect transmission, messages are first sent to some process in a view V. The process forwards the message to another process in the view V and finally delivers the message to the destination processes in the view V [Figure 3 (3)]. The tree routing [3, 5] is an example.

4.4. Confirmation

There are *centralized*, *direct*, *indirect*, and *distributed* schemes to confirm receipt of a message in a view V of a group G. In the centralized scheme, every process sends a receipt confirmation message to one *confirmation* process in a view V. After receiving confirmation from all the destination processes, the confirmation process sends a receipt confirmation of the message to the local sender process [Figure 4 (1)].

In the *direct* confirmation, each destination process p_i in the view V sends a receipt confirmation of a message m to the local sender process p_i which first sends the message m in the view V [Figure 4 (2)].

In the *indirect* confirmation, a receipt confirmation of a message m is sent back to a local sender process p_i in a view V by each process p_j which has received the message m from the local sender process p_i [Figure 4 (3)]. Finally, the local sender process of the message m in the view V receives receipt confirmation messages. This means "every destination process in the view V has received the message m".

In the *distributed* confirmation [10], each process which has received a message m sends a receipt confirmation of the message m to all the other processes in the same view [Figure 4 (4)]. Each process in a same view V can know whether or not all the other processes in V have received a same message m by using the distributed confirmation scheme.

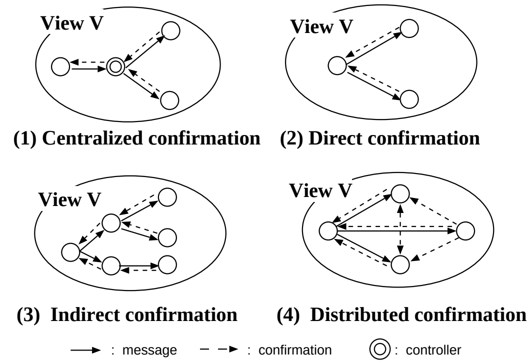

Figure 4. Confirmation schemes.

4.5. Ordering of messages

Messages received are ordered by each process in the distributed approach. In order to causally deliver messages, real time clock with NTP (network time protocol) [9], linear clock [7], and vector clock [8] are used. The realtime clock can be used in a personal area network. However the realtime clock cannot be used in a large area due to larger delay time.

The process can causally deliver messages by using the the linear clock or the vector clock under an assumption that the underlying network is reliable. No message gap can be detected by using the these clocks. Nakamura and Takizawa [10] discuss a vector of message sequence numbers to detect message loss and causally order messages. Each message m sent by a process p_i is assigned a sequence number $m.seq$. The sequence number seq is incremented by one each time p_i sends a message. The process p_i manipulates variables rsq_1, ..., rsq_n to. Each variable rsq_j shows a sequence number seq of message which p_i expects to receive next from another process p_j (j = 1, ..., n). A message m sent by p_i carries the receipt confirmation $m.rsq_j$ (= rsq_j) (j = 1, ..., n).

Suppose a process p_i receives a message m from another process p_j. If $rsq_j = m.seq$, the process p_i accepts the message m. Otherwise, there is some message m' from p_j where $rsq_j \leq m'.seq < m.seq$, i.e. p_j fails to receive m'. If p_i accepts a message m from a process p_j, the receipt confirmation information carried by the message m is stored in a matrix Ack, where $Ack[j, k] := m.rsq_k$ (k = 1, ..., n). A message m_1 *causally precedes* another message m_2 ($m_1 \rightarrow m_2$) iff $m_1.rsq < m_2.rsq$ [10]. A message m received from a process p_j is referred to as *pre-acknowledged* by a process p_i if $m.seq < min(Ack[1, j]$, ..., $Ack[n, j])$. Here, the process p_i is sure that the message m is received by every process. However, there might be still another process p_k where m is not pre-acknowledged, i.e. p_k does not know if some process p_l has received the message m. The process p_k may not reject the message m due to timeout because p_k does not receive the receipt confirmation form p_l. Hence, the process p_k cannot deliver the message m. A message m from a process p_j is referred to as *acknowledged* iff m is pre-acknowledged and there is one pre-acknowledged message m_k from every process p_k

where $m \rightarrow m_k$. That is, the process p_i is sure that m is pre-acknowledged in every process.

4.6. Detection of message loss

Messages are lost due to buffer overrun, unexpected delay, and congestions in the network. Message loss can be detected by checking sequence numbers as presented in the preceding subsection. On receipt of a message m from another process p_j, a process p_i accepts m if rsq_j = $m.seq$. Then, rsq_j is incremented by one. Otherwise, the process p_i finds there is some message m' from p_j where $rsq_j \leq m'.seq < m.seq$. Now suppose that a process p_i sends a message m. The message m carries a sequence number $m.seq$ and receipt confirmation $m.rsq$ (=$\langle$ $m.rsq_1$, ..., $m.rsq_n$ $\rangle$).

4.7. Retransmission

There are $sender$ and $destination$ retransmission schemes with respect to which process retransmits the message m lost [Figure 5]. Suppose a process p_j sends a message m to processes and one destination process p_i fails to receive m. In the $sender\ retransmission$, the local sender process p_j which first sent the message m in the view V retransmits the message m to p_i. In the $destination\ retransmission$, one or more than one destination process in the view V which has safely received the message m forwards m to the process p_i which fails to receive m [Figure 5 (2)]. In the distributed confirmation, not only a sender process but also every destination process in V receive receipt confirmation of a message m from every other destination process in V. Hence, each process can know if every other destination process safely receives a message m.

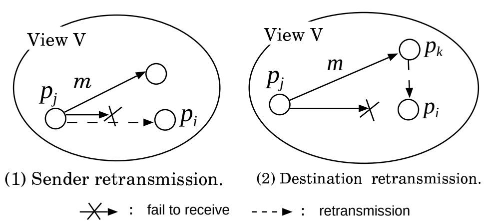

Figure 5. Retransmission scheme.

4.8 Membership management

In the $centralized$ way, one membership manager communicates with all the member processes to obtain their states. In the $distributed$ way, each process obtains the states of the other processes by communicating with other processes.

5. Autonomic Group Protocol

5.1. Architecture

Group communication service is supported by cooperation of multiple peer processes. The cooperation is coordinated by a group protocol. A system process means a protocol module which is realized in an $agent$ named $autonomic\ group$ (AG) $agent$. The classes for each protocol functions are stored in a protocol class library (PCL). The group communication service is realized by cooperation of multiple AG agents. Each application process A_i takes group communication service through an AG agent p_i. Each AG agent p_i autonomously takes one class for each group communication function from the PCL, which can support an application with necessary and sufficient QoS by taking usage of basic communication service supported by the underlying network. Each AG agent p_i monitors QoS supported by the underlying network. The network QoS information monitored is stored in a QoS base (QB) of p_i. If enough QoS cannot be supported or too much QoS is supported for the application, the AG agent p_i reconstructs a collection of group protocol function classes which are consistent with the other AG agents by selecting a class for each protocol function in the PCL. Here, each AG agent negotiates with other AG agents to make a consensus on which class to take.

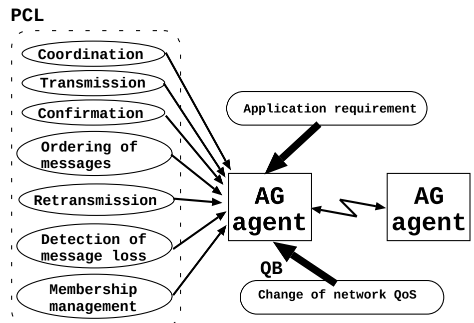

Figure 6. Autonomic group protocol.

5.2. Consistent combination of classes

Each AG agent takes a collection of classes for protocol functions. In this paper, we consider significant functions, coordination, transmission, confirmation, and retransmission function in the protocol functions. Let F be a set of the protocol functions, i.e. $\{C$(coordination), T(transmission), CF(confirmation), R(retransmission)$\}$. For each protocol function f in F, $Cl(f)$ shows a set of classes each of which shows implementation of the protocol function. $Cl(C)$ = $\{C(centralized)$, $D(distributed)\}$, $Cl(CF)$ = $\{Cen(centralized)$, $Dir(direct)$, $Ind(indirect)$, $Dis(distributed)\}$, $Cl(T)$ = $\{C(centralized)$, $D(direct)$, $I(indirect)\}$, and $Cl(R)$ = $\{S(sender)$, $D(destination)\}$. Let F be a set $\{f_1, f_2, f_3, f_4\}$ of protocol functions where $\langle f_1, f_2, f_3, f_4 \rangle$ = $\langle C, T, CF, R \rangle$. A tuple $\langle C_1, C_2, C_3, C_4 \rangle \in Cl(f_1) \times Cl(f_2) \times Cl(f_3) \times Cl(f_4)$ shows a $protocol\ instance$. Each AG agent takes a protocol instance $\langle C_1, C_2, C_3, C_4 \rangle$, i.e. a class C_i is taken for a protocol function f_i (i = 1, 2, 3, 4).

As discussed in the preceding section, the destination retransmission scheme can be taken in the distributed confirmation scheme but not in the centralized one. Thus, only some protocol instances of function classes are consistent. An agent can take only a consistent protocol instance. If an AG agent takes an inconsistent protocol instance, the agent cannot work. Table 1 summarizes possible, consistent protocol instances. A $profile$ C_1 C_2 C_3 C_4 shows a consistent protocol instance $\langle C_1, C_2, C_3, C_4 \rangle$ which each AG agent can take. Each profile is identified as shown in Table 1. Let

Table 1. Consistent protocol classes.

Control	Transmission	Confirmation	Retransmission	Profile
Centralized control	Centralized transmission	Centralized confirmation	Sender retransmission	CCCenS
Distributed control	Direct transmission	Direct confirmation	Sender retransmission	DDDirS
		Distributed confirmation	Sender retransmission	DDDisS
			Destination retransmission	DDDisD
	Indirect transmission	Direct confirmation	Sender retransmission	DIDirS
		Indirect confirmation	Sender retransmission	DIIndS
		Distributed confirmation	Sender retransmission	DIDisS
			Destination retransmission	DIDisD

P be a set of consistent protocol instances which are shown in Table 1.

5.3. Consistent set of profiles

Suppose AG agents p_1, ..., p_n are in a view V of a group G. Let C_i show a consistent protocol instance taken by an agent p_i, C_i = $\langle$ C_{i1}, ..., C_{i4} $\rangle \in P$. A $global$ protocol instance C for a view V = $\{p_1$, ..., $p_n\}$ is a tuple $\langle$ C_1, ..., C_n $\rangle$. Here, each C_i is referred to as $local$ protocol instance of an agent p_i (i = 1, ..., n). In traditional group protocols, every process has the same local protocol instance, i.e. C_1 = $\cdots$ = C_n. Hence, if some AG agent p_i would like to change a class C_{ik} of a protocol function f_k, all the AG agents have to be synchronized to make consensus on a new protocol instance. A global protocol instance C = $\langle$ C_1, ..., C_n $\rangle$ is referred to as $complete$ if C_1 = $\cdots$ = C_n. In this paper, we discuss a protocol where a view of AG agents p_1, ..., p_n can take an $incomplete$ instance C = $\langle$ C_1, ..., C_n $\rangle$ where $C_i \neq C_j$ for some pair of AG agents p_i and p_j. First, suppose that a global protocol instance C = $\langle$ C_1, ..., C_m $\rangle$ is complete and some AG agent p_i changes a local protocol instance C_i with another one C'_i. We discuss whether or not $\langle$ C_1, ..., C_{i-1}, C'_i, C_{i+1}, ..., C_n $\rangle$ is consistent, i.e. agents p_1, ..., p_n can cooperate even if $C'_i \neq C_j$ for some AG agent p_j.

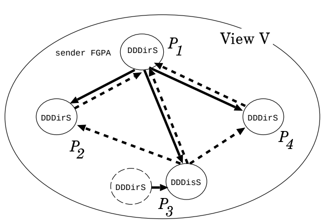

Figure 7. Change of profiles.

According to change of network QoS and application requirement, each AG agent autonomously changes the profile. For example, suppose that every AG agent takes the profile DDDirS in a view V including four processes p_1, p_2, p_3, and p_4 [Figure 7]. Here, suppose that a global protocol instance C is $\langle$ DDDirS, DDDirS, DDDirS, DDDirS $\rangle$ and some AG agent p_3 which takes Dir (Direct confirmation) would like to change with Dis (Distributed confirmation). We discuss whether or not a global protocol instance $\langle$ DDDirS, DDDirS, DDDisS, DDDirS $\rangle$ in consistent. While other AG agents take DDDirS, p_3 takes DDDisS. On receipt of a message m from p_1, the agent p_3 sends the receipt confirmation of m to not only the sender AG agent p_1 but also other destination AG agents. The sender AG agent p_1 receives the receipt confirmation of the message m from all the destination AG agents p_2, p_3, and p_4. The AG agents p_2 and p_4 receive the confirmation from p_3. The agent p_2 receives the confirmation from only p_3 but neither p_1 nor p_4. The confirmation from p_3 implies the confirmation from p_3 and p_4. Hence, the agent p_2 knows that all the other agents p_3 and p_4 receive the message. Thus, DDDirS can be changed to DDDisS. Here, if an AG agent p_3 takes DDDisS, the sender AG agent p_1 with DDDirS can receive the receipt confirmation message of a message m from all destination AG agents. Therefore, the sender AG agent p_1 and a pair of AG agents p_2 and p_4 do not need to change the profile.

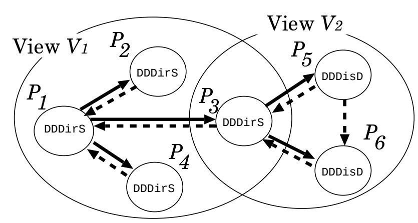

Figure 8. Change of profiles.

Next, suppose an AG agent p_3 belongs to a pair of views V_1 and V_2 [Figure 8]. In the view V_1 where all of the AG agents take DDDirS, an AG agent p_1 sends a message m to all the other AG agents. On receipt of the message m, an AG agent p_3 with DDDirS forwards the message m to the other AG agents p_5 and p_6 which belong to another view V_2 with DDDisD. Here, the AG agent p_3 can receive the receipt confirmation of the message m from a pair of AG agents p_5 and p_6 in the view V_2. In addition, the AG agent p_3 sends back the receipt confirmation of the message m to the original sender AG agent p_1. Here, the original sender AG agent p_1 can receive the receipt confirmation from all the destination AG agents in the view V_1. Therefore, the AG agent p_3 does not need to change the profile since the AG agent p_3 can forward the message m to another AG agent in the view V_2.

6. Retransmission

6.1 Cost model

Suppose an autonomic group (AG) agent p_s sends a message m to three AG agents p_t, p_u, and p_v in a view V.

Then, a pair of AG agents p_t and p_u receive the message m while another AG agent p_v fails to receive m. Here, let d_{ij} be delay time of channel C_{ij} between AG agents p_i and p_j[msec]. Let f_{ij} shows probability that a message is lost in a channel C_{ij}. and b_{ij} indicate bandwidth of the channel C_{ij} [bps]. $|m|$ shows size of message m [bit].

First, let us consider the sender retransmission. It takes ($2d_{sv}$+ $|m|$ / b_{sv}) [msec] to detect message loss after the AG agent p_s sends the message m. Then, the AG agent p_s retransmits m to p_v. Here, the message m may be lost again. The expected time ST_{sv} and expected number SN_{sv} of messages to be transmitted to deliver a message m to a destination p_v are given as follows:

1. ST_{sv} = ($2d_{sv}$ + $|m|$ / b_{sv}) / ($1 - f_{sv}$).
2. SN_{sv} = 1 / ($1 - f_{sv}$).

In the destination retransmission, some destination AG agent p_t forwards the message m to the AG agent p_v [Figure 9]. The expected time DT_{sv} and expected number DN_{sv} of messages to deliver a message m to p_v are given as follows:

1. DT_{sv} = (d_{st} + $|m|$ / b_{st}) + ($2d_{tv}$ + $|m|$ / b_{tv}) / ($1 - f_{tv}$).
2. DN_{sv} = ($2 - f_{tv}$) / ($1 - f_{tv}$).

If $ST_{sv} > DT_{sv}$, the destination AG agent p_t can forward the message m to the AG agent p_v because the message lost can be delivered earlier.

Each AG agent p_t monitors delay time d_{tu}, bandwidth b_{tu}, and message loss probability f_{tu} for each AG agent p_u which are received in the QoS base (QB). For example, the AG agent p_t obtains the QoS information by periodically sending $ping$ messages to all the AG agents in the group. The AG agent p_t maintains the quality of service (QoS) information in a variable Q of QB where Q_{tu} = $\langle$ b_{tu}, d_{tu}, f_{tu} $\rangle$ for u = 1, ..., n. If the AG agent p_t receives QoS information from another AG agent p_s, Q_{su} = $\langle$ b_{su}, d_{su}, f_{su} $\rangle$ for u = 1, ..., n.

6.2 Change of retransmission scheme

Let us consider an example. Suppose a sender AG agent p_s sends a message m and all the AG agents take the sender retransmission. An AG agent p_v fails to receive the message m [Figure 9]. According to the change of QoS supported by the underlying network, the sender p_s makes a decision to change the retransmission scheme with the destination one. However, the AG agent p_t still takes the sender retransmission. Here, no AG agent forwards the message m to p_v. In order to prevent these silent situations, we take a following protocol:

1. Sender AG agent p_s sends a message m to all the destination AG agents. Every destination AG agent sends receipt confirmation not only to the sender AG agent p_s but also to the other destination AG agents [Figure 9].
2. If an AG agent p_i detects that a destination AG agent p_v has not received the message m, p_i selects a retransmission scheme which p_i considers to be optimal based on the QoS information Q.
   2.1 If p_i is a destination AG agent and changes a retransmission scheme, p_i forwards m to p_v and sends $Retx$ message to the sender AG agent p_s.
   2.2 If p_i is a sender of a message m and takes a sender retransmission scheme, p_i retransmits m to p_v. If p_i takes a destination retransmission scheme, p_i waits for $Retx$ message from a destination. If p_i does not receive $Retx$, p_i retransmits m to p_v.

[**Theorem**] At least one AG agent forwards a message m to an AG agent which fails to receive the message m. □

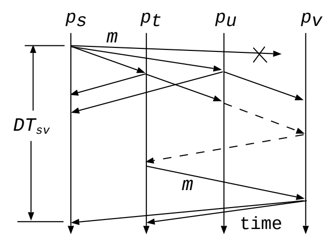

Figure 9. Destination retransmission.

7. Concluding Remarks

In this paper, we discussed an agent-based architecture to support distributed applications with autonomic group service in change of network and application QoS. We made clear what classes of functions to be realized in group communication protocols. Every autonomic group (AG) agent autonomously changes implementation of each protocol function which may not be the same as but are consistent with the other agents in a group. We discussed how to support applications with the autonomic group service by changing retransmission schemes as an example.

References

[1] Autonomic computing architecture : A blueprint for managing complex computing environments. 2002. http://www-3.ibm.com/autonomic/pdfs/ACwhitepaper1022.pdf.

[2] K. Birman. Lightweight causal and atomic group multicast. *ACM Trans. on Computer Systems*, pages 272–290, 1991.

[3] S. Deering. Host groups: A multicast extension to the internet protocol. *RFC 966*, 1985.

[4] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[5] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM Trans. on Cumputer Systems*, 9(3):242–271, 1991.

[6] L. Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, 1978.

[8] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[9] D. L. Mills. Network time protocol. *RFC 1305*, 1992.

[10] A. Nakamura and M. Takizawa. Causally ordering broadcast protocol. *Proc. of IEEE ICDCS-14*, pages 48–55, 1994.

[11] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *CACM*, 39(4):76–83, 1996.