

A Group Communication Protocol for Autonomic Computing

Tomoya Enokido

Faculty of Business Administration, Rissho University, Japan

Abstract: We discuss a group protocol which supports applications with group communication service when QoS supported by networks or required by applications is changed. An autonomic group protocol is realized by cooperation of multiple autonomous agents. Each agent autonomously takes a class of each protocol function to support only and all QoS required by applications. Classes taken by an agent are required to be consistent with the other agents but might be different from the others. A group is composed of smaller subgroups named views for realizing the scalability. Each view is a subset of the agents in a group, where the agents autonomously take protocol classes consistent with each other. We make clear what combination of classes can be autonomously taken by agents. We also discuss how to autonomously change retransmission classes and evaluate it in terms of delivery time of messages.

Key words: Autonomic group protocol, group protocol, peer-to-peer (P2P) systems, quality of service (QoS), autonomous group (AG) agents

INTRODUCTION

Peer-to-Peer (P2P) systems^[1-2] and grid computing systems^[4] are developed and used in various types of applications. These systems include large number of computers, e.g. millions of computers interconnected in network. In these scalable systems including, centralized architectures cannot be adopted due to difficulty of design, implementation, and maintenance of the systems, and non-centralized computation paradigms, autonomic computing^[4,5] is required to be developed. Multiple peer processes first establish a group and then messages are exchanged among the processes. Group communication protocols^[6-14] support basic communication mechanisms like the causally ordered and atomic delivery of messages to realize cooperation of multiple peer processes. The papers^[9,10] discussed the logical clocks for cooperation of multiple processes in a group. The papers^[6,7,11] discussed the causally ordered delivery of messages in a group which is composed of multiple processes. On the other hand, the papers^[13] proposed the group communication protocols which can support large scale group which is hierarchical structured. The paper^[14] proposed a group communication protocol for multimedia systems. Here, a group protocol is implemented in protocol modules. Each protocol module is composed of protocol functions; multicast/broadcast, receipt confirmation, detection and retransmission of messages lost, ordering of messages received, and membership management. There are various ways to implement each of these function. A class shows a way of implementation of a protocol function. For example, selective and go-back-n retransmissions of messages^[15] are classes for retransmission of lost messages. Distributed and

centralized coordination of multiple processes are classes for coordinating multiple protocol modules.

The complexity and efficiency of implementation of a group protocol depends on what type of service like unicast and multicast and Quality of Service (QoS), i.e. delay time, bandwidth, and loss ratio are supported by underlying networks. Messages sent by a process may be lost and unexpectedly delayed due to congestions and faults in the network. Thus, QoS parameters of networks are dynamically changed. For example, suppose applications require the totally ordered delivery of messages but do not require non-loss property of message and the underlying network is the MAC service of radio network. A group protocol can be easily realized by using the network. However, if the reliable communication is required, messages lost are required to be detected and retransmitted and messages received should be reordered. The higher level of communication function is supported, the larger computation and communication overheads are implied. Hence, the system has to take classes of functions necessary and sufficient to support required service by taking usage of the underlying network service.

The paper^[12] discusses a communication architecture supporting applications with a group of multiple processes, which satisfies application requirements in change of network service. However, a group protocol cannot be dynamically changed among the processes in a group each time QoS supported by the underlying network is changed. In addition, each process in a group has to use the same classes of protocol functions. In ISIS^[7], protocol modules which support service required can be constructed. However, a protocol module of each process cannot be changed in group communication and every process has to take the

same module. It is not easy to change classes of protocol functions in all the processes since a large number of processes are cooperating and some computers like personal computers and mobile computers are not always working correctly due to faults, communication noise, and shortage of battery.

In this paper, we discuss an autonomic group protocol which can support QoS required by applications even if QoS supported by the underlying network is changed. Each protocol module is realized in an autonomous agent. An agent autonomously changes classes, i.e. implementations of each group protocol function so as to support all and only service required by applications in change of network QoS. Here, an agent can take different classes of protocol functions from other agents as long as the classes taken are consistent with the other agents. Agents taking consistent classes can cooperate to support group communication service even if the classes taken by the agents are different. We discuss what combinations of protocol function classes are consistent. If a group is too large for each agent to perceive QoS supported by every other agents and manage the group membership, each agent has a view which is a subset of agents with which the agent can directly exchange messages. In each view, agents exchange messages by using its own consistent classes of protocol functions. Agents which belong to multiple views forward messages from one view to other views. A pair of different views might take different classes of protocol functions. As an example of the autonomic group protocol, we consider retransmission classes. We discuss how and when to change the retransmission classes in each autonomous agent. We also evaluate the autonomic group protocol with change of retransmission classes in terms of delivery time of messages. We show each agent of the autonomic group protocol can be take a retransmission class which can reduce the delivery time in change of QoS of the underlying network.

In section 2, we show a system model. In section 3, we discuss classes of protocol functions. In section 4, we present an agent-based group protocol. In section 5, we discuss how to change retransmission functions.

SYSTEM MODEL

Autonomic group agent: A group of multiple application processes A_1, \dots, A_n ($n \geq 2$) are cooperating by taking usage of group communication service. The group communication service is supported by cooperation of multiple peer autonomous group (AG) agents p_1, \dots, p_n as shown in Fig. 1. For simplicity, a term “agent” means an AG agent in this paper. The underlying network supports a pair of agents with basic communication service which is characterized by QoS parameters; delay time [msec], message loss ratio [%], and bandwidth [bps].

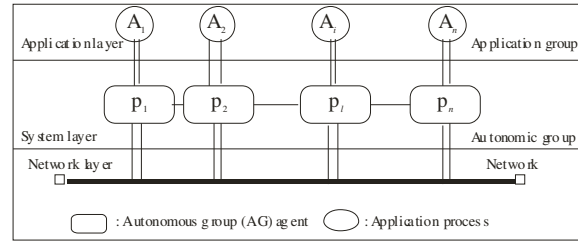


Fig. 1: System model

A group protocol is realized in a collection of protocol functions; transmission, confirmation, retransmission, ordering of message, detection of message lost, coordination schemes, and membership management. There are multiple ways to implement each protocol function. A class of a protocol function shows a way to implement the protocol function. For example, the selective retransmission is a class for the retransmission function and the centralized control is a class for the coordination of multiple agents. The classes are stored in a protocol Class Base (CB) of each agent. Each agent p_i autonomously takes one class for each group protocol function from the protocol CB, which can support an application with necessary and sufficient QoS by taking usage of basic communication service with given QoS supported by the underlying network. Each agent p_i stores QoS information obtained by monitoring the underlying network in a QoS base (QB) of the agent p_i . If enough QoS cannot be supported or too much QoS is supported for the application, the agent p_i reconstructs a combination of protocol function classes by autonomously selecting a class for each protocol function in the class base CB. An agent may take classes different from other agents. In this paper, we newly discuss a consistent relation among combinations of protocol classes. Even if a pair of agents take classes different from one another, the agents can support group communication service if the classes taken by the agents are consistent. We define the consistent relations among the classes in the succeeding section. Here, a pair of agents are consistent if the agents take classes consistent with one another. Each agent negotiates with other agents to make a consensus on which class to take for each protocol function if some pair of agents are inconsistent. In this paper, we discuss how each agent autonomously changes protocol function classes in change of QoS monitored.

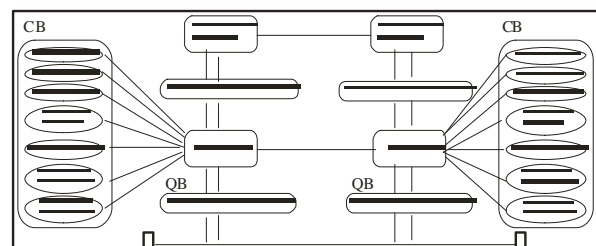


Fig. 2: Autonomic group protocol

A group G is composed of multiple autonomous group agents p_1, \dots, p_n ($n > 1$). An agent is an autonomous peer process which supports an application process with group communication service by exchanging messages with other agents. In a scalable group G including larger number of agents, it is not easy for each agent to maintain complete membership information of the group. Each agent p_i has a view $V(p_i)$ which is a subset of agents to which the agent p_i can deliver messages directly or indirectly via agents. Thus, a view is a subgroup of the group G . In this paper, we assume that for every pair of agents p_i and p_j , p_i is in a view $V(p_j)$ if p_j is in $V(p_i)$. Each agent p_i maintains membership of its view $V(p_i)$. For example, a view can be a collection of agents interconnected in a local network. A pair of different views V_1 and V_2 may include a common gateway agent p_k . A collection of gateway agents which are interconnected in a trunk network is also a view V_3 . In another example, a view is composed of agents which are interconnected in a reliable high-speed network. Here, the agents can reliably exchange messages with high bandwidth. Another view includes agents interconnected in a less-reliable, low-speed network like a mobile network. The agents are required to use additional mechanisms like retransmission of message lost to support reliable communication by using the less-reliable network.

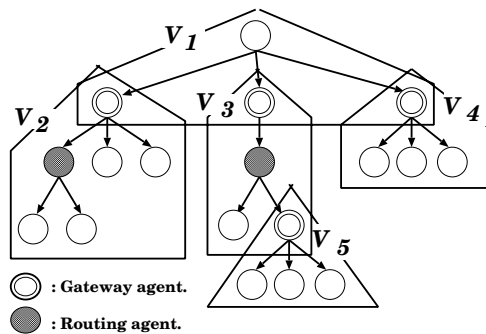


Fig. 3: Group views

An agent p_i which belongs to only one view is a leaf agent. An agent p_i which takes a message m from an application process A_i and sends the message m is referred to as an original sender agent of the message m . If an agent p_j delivers a message m to an application process, the agent p_j is referred to as an original destination agent of the message m . If an agent p_k forwards a message m to another agent in a same view V , p_k is a routing agent. A local sender and destination of a message m are agents which send and receive the message m in a view, respectively.

A view V including all the agents in a group G is referred to as complete. A global view is a complete view in a group G . A view V is partial if $V \subset G$.

In traditional group communications^[7,9-12], groups are complete views. A partial view V is a view which is not complete. A hierarchical group^[13] is composed of partial views. A view is changed if an agent joins and leaves the view V . Here, the view V is dynamic. Otherwise, the view V is static.

FUNCTIONS OF GROUP PROTOCOL

A group protocol is realized in a collection of the protocol functions; coordination of agents, message transmission, receipt confirmation, retransmission, detection of message loss, ordering of messages, and membership management. There are multiple ways to realize each of the protocol functions. A class of a protocol function shows one way to implement the protocol function. In this paper, we consider four protocol functions, coordination, transmission, confirmation, and retransmission functions in the protocol functions, which are the most significant to design and implement a group protocol.

First, there are centralized and distributed classes to coordinate the cooperation of multiple agents in a view. In the centralized coordination, there is one centralized controller in a view V . The centralized controller makes every decision in a group. Most of the traditional systems like teleconferences take the centralized way. On the other hand, each agent makes a decision on correct receipt, delivery order of messages received, and group membership by itself in the distributed coordination class.

Secondly, there are centralized, direct, and indirect classes to multicast a message to multiple agents in a view as shown in Fig. 4. In the centralized transmission, an agent first sends a message to a forwarder agent. Then, the forwarder agent forwards the message to all the destination agents in a view (Fig. 4 (1)). The forwarder agent plays a role of a centralized controller. In the direct transmission, each agent directly not only sends a message to each destination agent but also receives messages from other sender agents in a view V (Fig. 4 (2)). In the indirect transmission, a message is first sent to some agent in a view V . The agent forwards the message to another agent and finally delivers the message to the destination agents in the view V (Fig. 4 (3)). Tree routing^[16] is an example of the indirect transmission.

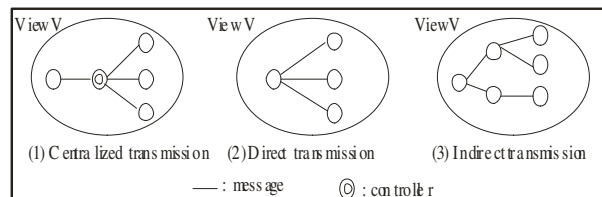


Fig. 4: Transmission classes

Next, there are centralized, direct, indirect, and distributed classes to confirm receipt of a message in a view V . In the centralized confirmation, every agent sends a receipt confirmation message to one confirmation agent in a view V . After receiving confirmation messages from all the destination agents, the confirmation agent sends a receipt confirmation to the local sender agent as shown in Fig. 5 (1). In the direct confirmation, each destination agent p_i in the view V sends a receipt confirmation of a message m to the local sender agent p_i which first sends the message m in the view V (Fig. 5 (2)). In the indirect confirmation, a receipt confirmation of a message m is sent back to a local sender agent p_i in a view V by each agent p_j which has received the message m from the local sender agent p_i (Fig. 5 (3)). In the distributed confirmation, each agent which has received a message m sends a receipt confirmation of the message m to all the other agents in the same view^[11] (Fig. 5 (4)).

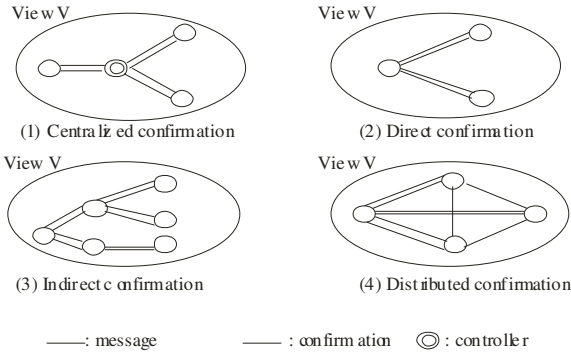


Fig. 5: Confirmation classes

Lastly, there are sender and destination retransmission classes with respect to which agent retransmits a message m lost by a destination agent as shown in Fig. 6. Suppose an agent p_j sends a message m to other agents but one destination agent p_i fails to receive the message m . In the sender retransmission, the local sender agent p_j has first sent the message m in the view V retransmits the message m to the agent p_i . In the destination retransmission, one or more than one destination agent in the view V which have safely received the message m forwards the message m to the agent p_i which fails to receive the message m (Fig. 6 (2)). In the distributed confirmation, each destination agent receiving a message m can know if every other destination agent safely receives the message m . If a destination agent p_j receives no confirmation from another destination agent p_i , the agent p_j detects the agent p_i to lose the message m . Then, the agent p_j can forward the message m to the destination agent p_i .

AUTONOMIC GROUP PROTOCOL

Local protocol instance: In this paper, we discuss protocol functions, coordination, transmission,

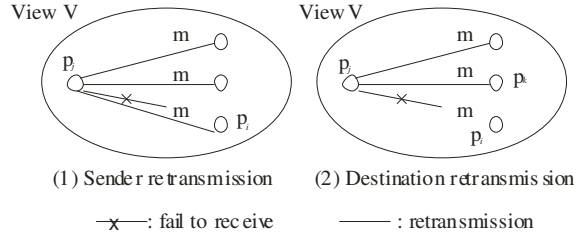


Fig. 6: Retransmission classes

confirmation, and retransmission functions, which are the most significant to design and implement a group protocol. Let F be a set of the significant protocol functions $\{C$ (coordination), T (transmission), CF (confirmation), R (retransmission) $\}$. Let $Cl(F)$ be a set of classes to implement a protocol function F in F . Table 1 summarizes possible classes for the protocol functions, C , T , CF , and R . For example, $Cl(C) = \{C, D\}$ for the coordination function C , which C and D stand for centralized and distributed coordination classes, respectively.

Table 1: Protocol classes

Function F	Protocol classes $Cl(F)$
C	$\{C$ (centralized), D (distributed) $\}$
CF	$\{Cen$ (centralized), Dir (direct), Ind (indirect), Dis (distributed) $\}$
T	$\{C$ (centralized), D (direct) I (indirect) $\}$

We rewrite the function set F to be a set $\{F_1, F_2, F_3, F_4\}$ of protocol functions where each element F_i shows a protocol function, i.e. F_1, F_2, F_3 , and F_4 shows the protocol functions C (coordination), T (transmission), CF (confirmation), and R (retransmission), respectively. A tuple $\langle c_1, c_2, c_3, c_4 \rangle \in Cl(F_1) \times Cl(F_2) \times Cl(F_3) \times Cl(F_4)$ is referred to as a protocol instance, which shows a protocol module realized in an agent. Each agent takes a protocol instance $C = \langle c_1, c_2, c_3, c_4 \rangle$, i.e. a class c_i is taken for each protocol function F_i ($i = 1, 2, 3, 4$). For example, $\langle D(\text{distributed}), D(\text{direct}), Dir(\text{direct}), S(\text{sender}) \rangle$ is a protocol instance, i.e. distributed coordination (D), direct transmission (D), direct confirmation (Dir), and sender retransmission (S).

The destination retransmission class can be taken in coexistence with the distributed confirmation class but not the centralized confirmation class. Thus, each agent cannot take every protocol instance. Only some protocol instances out of possible 48 instances cannot work in an agent.

[Definition] A protocol instance $\langle c_1, c_2, c_3, c_4 \rangle$ is correct iff an agent taking the protocol instance can work with other agents which take the same protocol instance to support group communication service.

A protocol instance which is not correct is referred to as faulty. If an agent takes a faulty protocol instance, the agent cannot work. Thus, only some protocol instances are consistent. An agent can take only a correct protocol instance. A protocol profile is a correct protocol instance. In Table 2, seven possible protocol

profiles are summarized. A protocol profile signature “ $c_1c_2c_3c_4$ ” denotes a protocol profile. For example, *DDDirS* shows a protocol profile $\langle D, D, Dir, S \rangle$ which is composed of distributed coordination (*D*), direct transmission (*D*), direct confirmation (*Dir*), and sender retransmission (*S*) classes. If every agent takes a same protocol profile, a group of the agents can support group communication service. Let $PF(1), PF(2), PF(3), PF(4), PF(5), PF(6)$, and $PF(7)$ show the protocol profiles *CCCSenS*, *DDDirS*, *DDDisS*, *DDDisD*, *DIIndS*, *DIDisS*, and *DIDisD*, respectively, which are shown in Table 2. Let \mathbf{P} be a set $\{PF(i) \mid i = 1, \dots, 7\}$ of all the protocol profiles. In a protocol profile $PF(i)$, i is referred to as the protocol profile number. For example, the protocol profile number of *DDDirS* ($C = PF(2)$) is 2.

Global protocol instance: Suppose autonomous group agents p_1, \dots, p_n are in a view V of a group G . Let C_i be a protocol profile $\langle c_{i1}, \dots, c_{id} \rangle (\in \mathbf{P})$ taken by an agent p_i ($i = 1, \dots, n$). A global protocol instance C for a view $V = \{p_1, \dots, p_n\}$ is a tuple $\langle C_1, \dots, C_n \rangle$ where each element C_i is a protocol profile which an agent p_i takes, $C_i \in \mathbf{P}$. Here, each C_i is referred to as a local protocol instance of an agent p_i ($i = 1, \dots, n$). In traditional protocols, every agent has to take the same local protocol instance, i.e. $C_1 = \dots = C_n$. That is, every protocol module is the same program. A global protocol instance $C = \langle C_1, \dots, C_n \rangle$ is complete if $C_1 = \dots = C_n$. A global protocol instance $C = \langle C_1, \dots, C_n \rangle$ is incomplete if $C_i \neq C_j$ for some pair of agents p_i and p_j . Suppose some agent p_i would like to change a class c_{ik} of a protocol function F_k in a protocol profile C_i with another class c'_{ik} in a complete global protocol instance. All the agents have to be synchronized to make an agreement on a new global protocol instance $\langle C'_1, \dots, C'_n \rangle$ where $C'_1 = \dots = C'_i = \dots = C'_n$. Thus, it takes time to change a protocol profile in every agent. In this paper, we discuss a novel way where each agent can change protocol profiles even if the profiles are not the same as other agents.

A global protocol instance $C = \langle C_1, \dots, C_n \rangle$ is consistent if a collection of agents p_1, \dots, p_n where each agent p_i takes a protocol profile C_i ($i = 1, \dots, n$) can be cooperating to support group communication service. In another word, the local profiles C_1, \dots, C_n are mutually consistent. A global protocol profile is a consistent global protocol instance. It is trivial that a complete global protocol instance is consistent from the definition. In this paper, we discuss a group protocol where a view V of agents p_1, \dots, p_n can take an incomplete but consistent global protocol instance C . First, following types of protocol instances are globally inconsistent.

[Property] A global protocol instance $C = \langle C_1, \dots, C_n \rangle$ of a view V is not consistent if the view V is composed of more than three agents and the global protocol instance C satisfies one of the following conditions:

- * At least one agent in V takes the protocol profile *CCCSenS* (centralized coordination, centralized transmission, centralized confirmation, and sender retransmission) and the global protocol instance C is not complete.
- * At least one agent takes an indirect (I) transmission class in V and at least one other agent takes a direct (*Dir*) confirmation class in V .

We introduce a notation α_I where $I \in 2^{\{1, \dots, 7\}}$ to show a global protocol instance as follows:

- * α_i indicates a global protocol profile where all agents take the same local protocol profile $PF(i)$ ($i = 1, \dots, 7$).
- * Let I be a sequence of protocol profile numbers $i_1 \dots i_l$ ($l \leq 7$). α_I shows a global protocol instance where each agent takes one of the local protocol profiles $PF(i_1), \dots, PF(i_l)$ and each protocol profile $PF(i_k)$ is taken by at least one agent ($k = 1, \dots, l$).

For example, α_2 shows that all the agents take $PF(2) = \textit{DDDirS}$. α_{23} means a global protocol instance where every agent takes $PF(2) = \textit{DDDirS}$ or $PF(3) = \textit{DDDisS}$, at least one agent takes $PF(2) = \textit{DDDirS}$, and at least one agent takes $PF(3) = \textit{DDDisS}$.

[Definition] A global protocol instance α_I can be transitioned to another global protocol instance α_J ($\alpha_I \rightarrow \alpha_J$) in a view V of agents iff

- * if $J = I_i$, the agents can support group communication service even if an agent taking a protocol profile $PF(k)$ where $k \in I$ autonomously takes another protocol profile $PF(i)$ ($i \neq k$).
- * if $I = J_j$, the agents can support group communication service even if an agent taking $PF(j)$ takes $PF(k)$ where $k \in I$.
- * For some global protocol instance α_K , $\alpha_I \rightarrow \alpha_K$ and $\alpha_K \rightarrow \alpha_J$.

According to the definition, the transition relation “ \rightarrow ” is symmetric and transitive. Figure 7 shows a Hasse diagram where a node shows a global protocol profile and a directed edge from α_I to α_J indicates a transition relation “ $\alpha_I \rightarrow \alpha_J$ ”. For example, $\alpha_2 \rightarrow \alpha_{24}$ since an agent can autonomously change a local protocol profile $PF(2) = \textit{DDDirS}$ to $PF(4) = \textit{DDDisD}$.

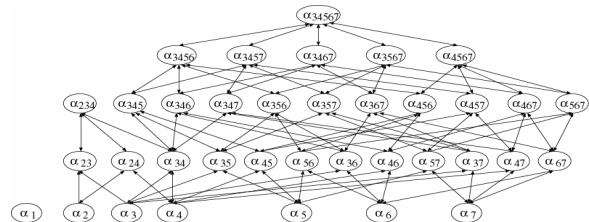


Fig. 7: Hasse diagram

According to change of network QoS and application QoS requirement, each agent autonomously changes the protocol profile. For example, suppose an agent p_3 belongs to a pair of views V_1 and V_2 as shown

Table 2: Protocol profiles

Control	Transmission	Confirmation	Retransmission	Signature
Centralized	Centralized	Centralized	Sender	CCCS
Distributed	Direct	Direct	Sender	DDDirS
		Distributed	Sender	DDDiS
			Destination	DDDisS
	Indirect	Indirect	Sender	DIIndS
		Distributed	Sender	DDiS
			Destination	KIDiS

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

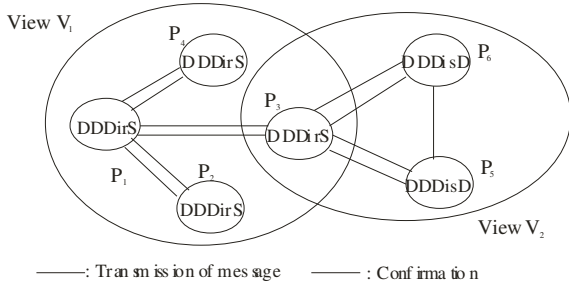


Fig. 8: Change of profiles

RETRANSMISSION

We discuss how an agent can autonomously change the retransmission classes in a group as an example.

Cost model: Suppose there are three agents p_s , p_t , and p_u in a view V . An agent p_s sends a message m to a pair of agents p_t and p_u . Then, the agent p_t receives the message m while the agent p_u fails to receive the message m . The following notations are used to discuss a cost model for the agents p_s and p_t :

- * d_{st} = delay time between agents p_s and p_t [msec],
- * f_{st} = probability that a message is lost,
- * b_{st} = bandwidth [bps].

First, let us consider the sender retransmission. Let lml show the size of a message m [bit]. It takes $(2d_{su} + lml / b_{su})$ [msec] to detect message loss after an agent p_s sends a message m . Then, the sender agent p_s retransmits the message m to an agent p_u which fails to receive the message m . Here, the message m may be lost

again. The expected time ST_{su} and number SN_{su} of messages to be transmitted to deliver a message m to a destination p_u losing m are given as follows:

- * $ST_{su} = (2d_{su} + lml / b_{su}) / (1 - f_{su})$
- * $SN_{su} = 1 / (1 - f_{su})$.

In the destination retransmission, some destination agent p_t forwards the message m to the agent p_u as shown in Fig. 9. The expected time DT_{su} and number DN_{su} of messages to deliver a message m to p_u are given as follows:

- * $DT_{su} = (d_{su} + lml / b_{su} + d_{ut}) + (2d_{ut} + lml / b_{ut}) / (1 - f_{ut})$ if $d_{st} \leq d_{su} + d_{st}$, $(d_{st} + lml / b_{st}) + (2d_{ut} + lml / b_{ut}) / (1 - f_{ut})$ otherwise.
- * $DN_{su} = 1 + 1 / (1 - f_{ut})$.

If $ST_{su} > DT_{su}$, the destination agent p_t can forward the message m to the agent p_u losing the message m because the message m can be delivered earlier.

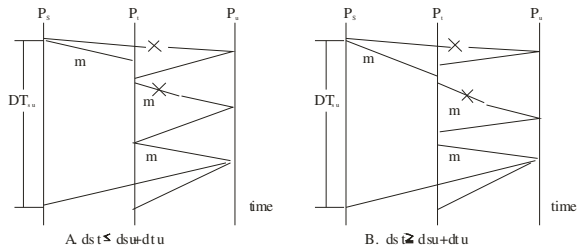


Fig. 9: Destination retransmission

Each agent p_t monitors delay time d_{ut} , bandwidth b_{ut} , and message loss probability f_{ut} for each agent p_u which are stored in the QoS base (QB). For example, an agent obtains the QoS information by periodically sending QoS information messages to all the agents in a view. The agent p_t maintains a vector variable $Q = \langle Q_{1t}, \dots, Q_{nt} \rangle$ in the QoS base QB where $Q_{ut} = \langle b_{ut}, d_{ut}, f_{ut} \rangle$ for $u = 1, \dots, n$. If an agent p_t receives QoS information $\langle b, d, f \rangle$ from an agent p_s , $Q_{su} := \langle b, d, f \rangle$ for $u = 1, \dots, n$.

Change of retransmission class: Suppose an agent p_s sends a message m and every agent p_t takes the sender retransmission class, $C_t = \langle \dots, S \rangle$ in a view. As shown in Fig. 10, an agent p_u fails to receive the message m . According to the change of QoS supported by the underlying network, the sender agent p_s makes a decision to change the retransmission class with the destination one, say an agent p_t forwards the message m

to the agent p_u . However, the agent p_t still takes the sender retransmission. Here, no agent forwards the message m to the agent p_u .

Next, suppose all agents are taking the destination retransmission class. Here, QoS supported by the network is changed and an agent p_t decides to take the sender retransmission class. However, no agent forwards a message m to an agent p_u which fails to receive m since the sender p_s still takes the destination retransmission class. In order to prevent these silent situations, we take a following protocol.

- * A sender agent p_s sends a message m to all the destination agents. Every destination agent sends receipt confirmation not only to the sender agent p_s but also to the other destination agents .
- * If an agent p_t detects that a destination agent p_u has not received the message m , the agent p_t selects a retransmission class which p_t considers to be optimal based on the QoS information Q .
- * If p_t is a destination agent and changes a retransmission class, the agent p_t forwards the message m to the agent p_u and sends *Retx* message to the sender agent p_s .
- * If p_t is a sender of a message m and takes the sender retransmission class, the sender agent p_t retransmits the message m to p_u . If p_t takes a destination retransmission class, p_t waits for *Retx* message from a destination. If p_t does not receive *Retx*, p_t retransmits m to p_u .

It is straightforward for the following theorem to hold:

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

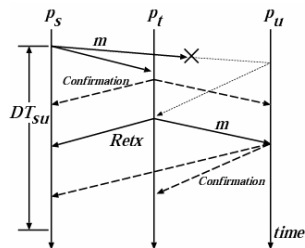


Fig. 10: Retransmission

Evaluation: We evaluate the autonomic group protocol (AGP) in terms of delivery time of a lost message. We make the following assumptions on this evaluation.

- * $d_{st} = d_{ts}$ for every pair of agents p_s and p_t .
- * The protocol processing time of every process is same.
- * No confirmation message is lost.

Let us consider a view $V = \{p_s, p_t, p_u\}$ where every agent takes a profile *DDDisS*, distributed coordination (*D*), direct transmission (*D*), distributed confirmation (*Dis*), and sender retransmission (*S*). Here, suppose that an agent p_s sends a message m to a pair of agents p_t and p_u in the view V . Then, the agent p_t receives the message m while another agent p_u fails to receive the

message m . After the sender agent p_s and destination agent p_t detect that the destination agent p_u fails to receive the message m , each of the agents p_s and p_t autonomously selects the retransmission class based on the QoS information. Here, we measure time to deliver a message m to an agent p_u which fails to receive the message m . In the view V , we assume that bandwidth and message loss ratio between every pair of agents are the same ($b_{st} = b_{su} = b_{ut} = 10\text{Mbps}$) and $f_{st} = f_{su}$ and $f_{ut} = 0$ [%]. Figure 11 shows an agent graph for V where each node denotes an agent and each edge shows a communication channel between agents. A label of the edge indicates delay time.

First, we consider a case $d_{su} \geq d_{st} + d_{ut}$. There are further cases: $d_{st} = d_{ut}$ (Fig. 11 A.1), $d_{st} > d_{ut}$ (Fig. 11 A.2), and $d_{st} < d_{ut}$ (Fig. 11 A.3). Figure 12 shows the expected time DT_{su} for three cases. In Fig. 12, horizontal axis shows a message loss probability of f_{su} and f_{ut} . For case of Fig. 11 A.2, $DT_{su} < ST_{su}$. For case of Fig. 11 A.1, $DT_{su} < ST_{su}$ if $f_{su} > 15\%$ and $f_{ut} > 15\%$. For case of Fig. 11 A.3, $DT_{su} < ST_{su}$ if $f_{su} > 50\%$ and $f_{ut} > 50\%$.

Next, we consider a case $d_{su} \leq d_{st} + d_{ut}$. There are further following cases as shown in Fig. 11:

- * $d_{st} > d_{su}$ and $d_{st} > d_{ut}$: $d_{su} = d_{ut}$ (B.1), $d_{su} > d_{ut}$ (B.2), and $d_{su} < d_{ut}$ (B.3).
- * $d_{ut} > d_{su}$ and $d_{ut} > d_{st}$: $d_{su} = d_{st}$ (C.1), $d_{su} > d_{st}$ (C.2), and $d_{su} < d_{st}$ (C.3).

The expected time DT_{su} (Fig. 11 B and 11 C) is shown for these six cases in Fig. 13 and 14. For Fig. 11 B.1 and B.3, $DT_{su} > ST_{su}$. For Fig. 11 B.2, $DT_{su} < ST_{su}$ if $f_{su} > 20$ and $f_{ut} > 20\%$. For Fig. 11 C, $DT_{su} > ST_{su}$.

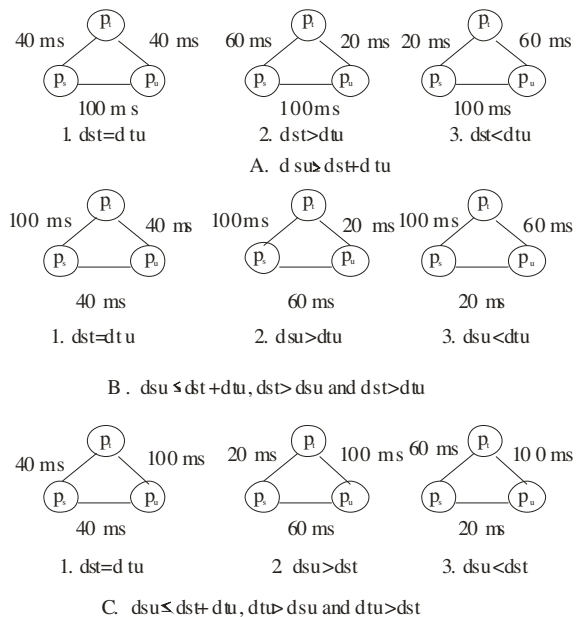


Fig. 11: AG agent graphs

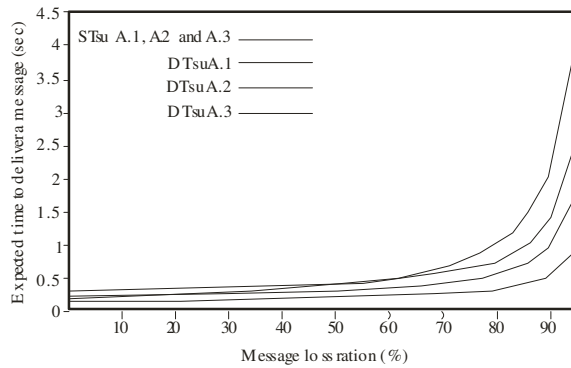


Fig. 12: $d_{su} \geq d_{st} + d_{ut}$

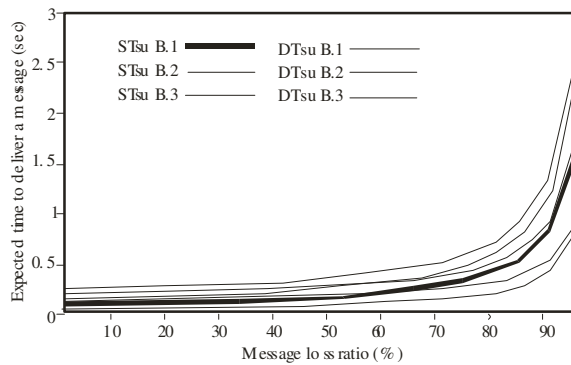


Fig. 13: $d_{su} \leq d_{st} + d_{ut}$, $d_{st} > d_{su}$, and $d_{st} > d_{ut}$

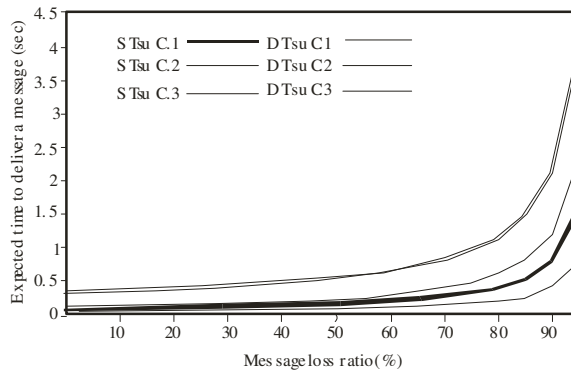


Fig. 14: $d_{su} \leq d_{st} + d_{ut}$, $d_{ut} > d_{su}$, and $d_{ut} > d_{st}$

CONCLUSION

We discussed an agent-based architecture to support distributed applications with autonomic group service in change of network and application QoS. The autonomic group communication service is supported by cooperation of agents. We made clear what classes, i.e. implementation ways of protocol functions to be realized in group communication protocols. Every agent autonomously changes a class of each protocol function which may not be the same as the other agents but are consistent with the other agents in a group. We discussed how to support applications with the

autonomic group service by changing retransmission classes as an example of the autonomic group protocol.

The complexity and efficiency of implementation of a group protocol depends on what type of service and Quality of Service (QoS) are supported by underlying networks. If the higher level of communication function is supported, the larger computation and communication overheads are implied. In addition, in the current information systems, it is not easy to change classes of protocol functions in all the processes since a large number of processes are cooperating and some computers are not always working correctly. Therefore, the system has to take classes of functions necessary and sufficient to support required service by taking usage of the underlying network service. The autonomic group protocol can support these distributed applications. The autonomic group protocol can reduce the complexity and enhance the efficiency of implementation of communication systems for distributed applications.

REFERENCES

1. Agre, P.E., 2003. P2P and the Promise of Internet Equality: Commun. ACM, 46: 39-42.
2. Balakrishnan, H., F. Kaashoek, D. Karger, R. Morris and I. Stoica, 2003. Looking Up Data in P2P Systems: Commun. ACM, 46: 43-48.
3. Foster, I. and C. Kesselman, 1999. The Grid: Blueprint for a New Computing Infrastructure: Morgan Kaufmann Publishers.
4. Bigus, J. P., D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills and Y. Diao, 2002. ABLE: A toolkit for building multiagent autonomic system: IBM System J., 41: 350-371.
5. IBM Corporation, 2002. Autonomic Computing Architecture: A Blueprint for Managing Complex Computing Environments, <http://www-3.ibm.com/autonomic/pdfs/ACwhitepaper1022.pdf>.
6. Ahamad, M., M. Raynal and G. Thia-Kime, 1998. An Adaptive Protocol for Implementing Causally Consistent Distributed Services: Proc. of of IEEE the 18th Intl. Conf. Distributed Computing Systems (ICDCS-18), pp : 86-93.
7. Birman, K., A. Schiper and P. Stephenson, 1991. Lightweight Causal and Atomic Group Multicast: ACM Trans. on Computer Systems, 9: 272-290.
8. Kaashoek, M. F., A. S. Tanenbaum, S. F. Hummel and H. E. Bal, 1989. An Efficient Reliable Broadcast Protocol: ACM Operating Systems Review, 23: 5-19.
9. Lamport, L., 1989. Time, Clocks, and the Ordering of Events in a Distributed System, CACM, 21: 558-565.

10. Mattern, F., 1989. Virtual Time and Global States of Distributed Systems: Parallel and Distributed Algorithms, pp: 215-226.
11. Nakamura, A. and M. Takizawa, 1994. Causally Ordering Broadcast Protocol: Proc. IEEE the 14th Intl. Conf. Distributed Computing Systems (ICDCS-14), pp : 48-55.
12. Renesse, V., K. P. Birman and S. Maffeis, 1996. Horus: A Flexible Group Communication System: CACM, 39: 76-83.
13. Taguchi, K., T. Enokido and M. Takizawa, 2004. Causal Ordered Delivery for a Hierarchical Group: Proc. of IEEE 10th Intl. Conf. Parallel and Distributed Systems (ICPADS2004), pp: 453-460.
14. Tojo, T., T. Enokido and M. Takizawa, 2004. Notification-Based QoS Control Protocol for Multimedia Group Communication in High-Speed Networks: Proc. of IEEE the 24th Intl. Conf. Distributed Computing Systems (ICDCS-2004), pp : 644-651.
15. Kaashoek, M. F. and A. S. Tanenbaum, 1996. An Evaluation of the Amoeba Group Communication System: Proc. of IEEE the 16th Intl. Conf. Distributed Computing Systems (ICDCS-16), pp : 436-447.
16. Garcia-Molina, H. and A. Spaster, 1991. Ordered and Reliable Multicast Communication: ACM Trans. on Computer Systems, 9: 242-271.