2009 IEEE International Symposium on Parallel and Distributed Processing with Applications

An Efficient Task Allocation Protocol for P2P Multi-Agent Systems

Dayong Ye[1], Quan Bai[1], Minjie Zhang[1] and Khin Than Win[2]
School of Computer Science and Software Engineering[1]
School of Information Systems and Technology[2]
University of Wollongong, NSW 2522 Australia[1,2]
Email: {dy721,quan,minjie,win}@uow.edu.au

Zhiqi Shen
Division of Information Systems
School of Computer Engineering, College of Engineering
Nanyang Technological University, Singapore 639798
zqshen@ntu.edu.sg

***Abstract*—Recently, task allocation in multi-agent systems has been investigated by many researchers. Some researchers suggested to have a central controller which has a global view about the environment to allocate tasks. Although centralized control brings convenience during task allocation processes, it also has some obvious weaknesses. Firstly, a central controller plays an important role in a multi-agent system, but task allocation procedures will break down if the central controller of a system cannot work properly. Secondly, centralized multi-agent architecture is not suitable for distributed working environments. In order to overcome some limitations caused by centralized control, some researchers proposed distributed task allocation protocols. They supposed that each agent has a limited local view about its direct linked neighbors, and can allocate tasks to its neighbors. However, only involving direct linked neighbors could limit resource origins, so that the task allocation efficiency will be greatly reduced. In this paper, we propose an efficient task allocation protocol for P2P multi-agent systems. This protocol allows not only neighboring agents but also indirect linked agents in the system to help with a task if needed. Through this way, agents can achieve more efficient and robust task allocations in loosely coupled distributed environments (e.g. P2P multi-agent systems). A set of experiments are presented in this paper to evaluate the efficiency and adaptability of the protocol. The experiment result shows that the protocol can work efficiently in different situations.**

I. Introduction

Allocating tasks to agents in multi-agent systems is a significant research issue. Task allocation problem can be simply described as that an agent has a task which it cannot complete by itself. The agent then attempts to discover other agents which contain the appropriate resources and assigns the task, or part of the task, to those agents. Recently, many approaches have been proposed for task allocation. Some of them are based on centralized fashion which assumes that there is a central controller to assign tasks to agents in a multi-agent system, e.g. [16]. Centralized fashion can make the allocation process efficient and effective in a small network since the central planner has a global view of the system and it understands which agents are good at which tasks. In that case, communication overhead during allocation processes could be reduced. However, the centralized fashion also has several notable disadvantages. The first one is that in some systems, it is difficult to have such a central controller, such as Peer-to-Peer (P2P) multi-agent systems in which no one agent has a global view but only the local prospect about direct linked neighbors. Secondly, when the central planner is out of order or cracked by some attackers, task allocation will suffer a big trouble in this system. The last drawback is that the scalability in such a system is limited because when too many agents exist, the central controller has to maintain much information to hold the global view and respond plenty of request messages from agents which drastically raises the CPU and memory usage and network bandwidth consumption.

In order to overcome the shortcomings of centralized fashion, some researchers presented distributed task allocation protocols which are based on decentralized control. Compared to centralized fashion, decentralized style is more scalable and robust but the communication overhead ascends. In addition, there are still some limitations in current decentralized task allocation protocols. Some mechanisms, such as [5] and [9], need to form groups of agents before allocating tasks which may result in computation and communication cost soaring up. Some other distributed task allocation protocols, e.g. Greedy Distributed Allocation Protocol (GDAP) [14], only allows agents to request help for tasks to their direct linked neighbors which may increase the possibility of task allocation failure due to limited resources available from the agents' neighbors.

In this paper, we propose an Efficient Task Allocation Protocol (ETAP). This protocol is based on the Contract Net approach, but more suitable for dealing with task allocation problems in P2P multi-agent systems with a decentralized manner. ETAP allows agents to request help from not only neighbors but also other indirect linked agents if needed. The rest of the paper is organized as follows. Section II provides some current related research in this field. After that, ETAP will be depicted in detail in Section III. Section IV demonstrates the experiment and analysis about the quality and performance of ETAP and GDAP. Then, a potential application of ETAP in intrusion detection is illustrated in Section V. Finally, we discuss and conclude our work in Section VI.

II. Related Work

In recent years, many research works about task allocation have been done. Some of them investigate the task allocation problem in a centralized manner. Zheng and Koenig [16] presented reaction functions for task allocation to cooperative agents. The objective is to find a solution with a small team cost and each target to be assigned to the exact number of different agents. This work assumed that there is a central planner to allocate tasks to agents. Kraus et al. [4] proposed

978-0-7695-3747-4/09 $25.00 © 2009 IEEE
DOI 10.1109/ISPA.2009.14

an auction based protocol which enables agents to form coalitions with time constrains. This protocol assumed each agent knows the capabilities of all others, and one manager is responsible for allocating tasks to all coalitions.

The weaknesses of centralized control include hard to realize, single point failure, and bad scalability as described in Section I. To conquer these disadvantages, Task allocation in distributed environments has also been investigated. A classic task allocation protocol for distributed environment is Contract Net Protocol (CNP) [10]. CNP was aimed to cope with problem-solving communication and control for nodes in a distributed problem solver. This protocol facilitates distributed control of cooperative task execution (called *task sharing*) with efficient inter-node communication. *Task sharing* is a process which is carried on between nodes with tasks to be executed and nodes that may be able to execute those tasks. The CNP was then evolved in [11] by adding another concept, *result sharing*. *Result sharing* is a form of cooperation in which individual nodes assist each other by sharing partial results. However, CNP focused only on task allocation process without considering concrete system architectures. According to [5] and [9], the authors developed distributed algorithms with low communication complexity for forming coalitions in large-scale multi-agent systems, while in our approach, it is not necessary to form coalitions among agents before allocating tasks.

Abdallah and Lesser [1] provided a decision theoretic model in order to limit the interactions between agents and mediators. Mediators in this paper mean the agents which receive the task and have connections to other agents. Mediators have to decompose the task into subtasks and negotiate with other agents to obtain commitments to execute these subtasks. However, their work concentrated on modeling the decision process of a single mediator. A scalable and distributed task allocation protocol was presented in [7]. The algorithm adopted in this protocol is based on computation geometry techniques but the prerequisite of this approach is that agents' and tasks' geographical positions are known.

Weerdt et al. [14] proposed a distributed task allocation protocol in social networks. This protocol only allows neighboring agents to help with a task which might result in high probability of abandon of tasks when neighbors can not offer sufficient resources. In this article, ETAP is proposed which enables agents to allocate tasks not only to their neighbors but also to commit unfinished tasks to their neighbors for reallocation. In this way, the agents can have more opportunities to achieve solution of their tasks.

III. Efficient Task Allocation Protocol

To cope with the issue of allocating tasks in a P2P multi-agent system, a decentralized task allocation protocol, ETAP, is elaborated in this section.

A. Problem Description

We formalize the description of task allocation problem in this subsection. Firstly, the definition of *P2P multi-agent network* is given.

- **Definition 1**: A *P2P multi-agent network* is defined as an undirected graph written as $P2P = (A, E)$ where A is the set of agents in the network, namely $A = \{a_1, a_2, ..., a_n\}$ and $E = \{e_{12}, e_{13}, ..., e_{21}, e_{23}, ...\}$ indicates the set of edges which exist between two agents. For example, the edge $e_{ij} \in E$ means there is a connection between the agents a_i and a_j. Therefore, a_i and a_j are neighbors of each other.

Each agent $a \in A$ is defined as a tuple $<AgentID(a), NeigList(a), Resource(a)>$, where $AgentID(a)$ is the identity of the agent, $NeigList(a)$ indicates the neighbors of the agent, and *Resource*(*a*) depicts the resources which the agent contains. Then, the definitions of three terms, i.e. *Initiator*, *Participant*, and *Mediator*, are provided.

- **Definition 2**: Suppose there is a set of tasks $T = \{t_1, t_2, ..., t_n\}$ in a P2P multi-agent system. The agent which requests help for its task(s) is called *Initiator* and the agent which accepts and performs the announced task(s) is called *Participant*. If some tasks of *Initiator* cannot be allocated to its neighbors, the *Initiator* will commit these tasks to its neighbors for reallocation. Therefore, the neighboring agent which accepts the commitment is called *Mediator*.

Each task $t \in T$ is defined as a tuple, namely $<TaskID(t), Resource(t), Benefit(t), Position(t), TTL(t)>$. In this tuple, $TaskID(t)$ is the identity of the task, $Resource(t)$ depicts the resource which is necessary for completing the task, $Benefit(t)$ is the benefit gained when the task is completed successfully, $Position(t)$ demonstrates where the task is from, namely which agent announces this task, and $TTL(t)$ (Time To Live) means how many times this task should be reallocated. $TTL(t)$ would be minus 1 when this task is reallocated once. If a task's TTL value reaches 0, this task will be removed without further reallocating.

In this paper, it is assumed that each task $t \in T$ can be assigned to only one agent to accomplish, as task decomposition is not the concentration of this research. Task allocation, therefore, can be defined as follows.

- **Definition 3**: Given a finite set of tasks, recorded as $T = \{t_1, t_2, ..., t_n\}$, and a finite set of agents, written by $A = \{a_1, a_2, ..., a_m\}$ in a P2P multi-agent system. *Task allocation* in this research means attempting to allocate the n tasks to part or all of the m agents.

A successful task allocation should satisfy the situation that the resources the *Participant* agent has should be more than the resource which is needed for accomplishing the task, namely $Resource(a) \supset Resource(t)$, because one task can only be assigned to one agent. Task allocation would probably be completed if it obeys the constrain that the total number of resources the m agents contain should be more than the entire number of resources required to finish the n tasks, i.e., $\bigcup_{1 \le i \le m} Resource(a_i) \supset \bigcup_{1 \le i \le n} Resource(t_i)$.

The *Initiator* prioritizes the tasks based on the efficiency of each task, and allocate tasks with their efficiency descending. In addition, the *Participant* also chooses the most efficient tasks to offer help every time. The definition of the efficiency of a task, $t \in T$, is described as follows.

- **Definition 4**: The *efficiency* of a task, $effi(t)$, is in the light of the ratio between the benefit gained from

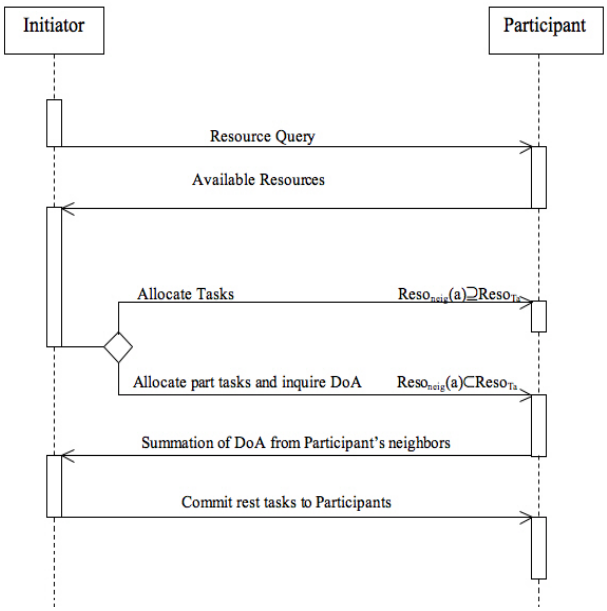


Figure 1. Interaction Process Between *Initiator* and a *Participant*

completing the task and the resources required for accomplishing the task, i.e.

$$effi(t) = \frac{Benefit(t)}{Resource(t)}. \tag{1}$$

B. The Principle of ETAP

In a P2P multi-agent system, there is no agent that has a global view about the system but only the local prospect regarding its neighbors. That means the *Initiator* agent can make contracts only with its neighboring agents instead of indirect linked agents. This research focuses on how to allocate n tasks appropriately to *Participant* agents particularly when the *Initiator* agent's neighbors do not have sufficient resources for tasks. In this paper, it is supposed that the P2P multi-agent system architecture is fixed during task allocation process, because task allocation for dynamic environment is not the contribution of this article. The idea of ETAP is depicted as follows, and Figure 1 briefly depicts the interaction process between an *Initiator* and a *Participant*.

The *Initiator* agent, $a \in A$, attempts to find its neighboring *Participants* to help with its tasks, $t_i \in T_a$ and $T_a \subseteq T$. The *Initiator* agent first sends resource query messages to its neighbors. These neighbors will respond the message with information about the types of resources they contain and the number of resources for each type, the identities of them, and the Degree of Availability (DoA) of them. DoA is defined as follows.

- **Definition 5**: DoA depicts the degree of availability of an agent. Although an agent may have many resources, it might have also stored many requests for tasks from other agents already. Hence, DoA is necessary for evaluating how available an agent is.

$$DoA(a) = \frac{Resource(a) - \bigcup_{t_i \in T_a^P} Resource(t_i)}{Resource(a)}, \tag{2}$$

where T_a^P is the set of tasks that have been allocated to the agent for performing.

The *Initiator* agent then compares the available resources from its neighbors, i.e. $Reso_{neig}(a)$, with the resources required for its tasks, namely $Reso_{T_a}$. They are calculated as $Reso_{neig}(a) = \bigcup_{a_i \in NeigList(a)} Resource(a_i)$ and $Reso_{T_a} = \bigcup_{t_i \in T_a} Resource(t_i)$. This comparison will result in one of the following two cases.

1) **Case One** ($Reso_{neig}(a) \supseteq Reso_{T_a}$): in this situation, the *Initiator*, a, directly requests help for tasks based on the information regarding the available and required resources. It begins with assigning the most efficient task(s). If more than one neighbor can solve one task, the *Initiator* will allocate this task to the one which has the highest DoA. The neighbors receive and store the requests, and select the tasks with the highest efficiency to perform. When the *Initiator* receives the responses from its neighbors, it finally sends contracts for the allocated tasks to the *Participants*. Each contract, $Cont(t)$, consists of a tuple $<ContID(t)$, $TaskID(t)$, $AgentID(a)$, $AgentID(a_i)>$. In this tuple, $ContID(t)$ is the identity of this contract, while $TaskID(t)$ means for which task this contract is, then $AgentID(a)$ and $AgentID(a_i)$ depict the identity of the *Initiator* and *Participant* separately.
2) **Case Two** ($Reso_{neig}(a) \subset Reso_{T_a}$): in this case, the *Initiator*, a, requests help for those tasks, $t_i \in T_a^{'}$, which can be handled by current available resources from its neighbors. The *Initiator* starts with allocating the tasks, $t_i \in T_a^{'}$, also based on the efficiency of these tasks. When finishing assigning tasks, $t_i \in T_a^{'}$, the *Initiator* attempts to commit the rest of the tasks which cannot be dealt with by using the resources of its neighbors, namely $t_j \in (T_a - T_a^{'})$.
The *Initiator* sends request messages to each neighbor to inquire each neighboring agent's neighbors' DoA summation, i.e. $\sum_{a_j \in NeigList(a_i) \wedge (a_j \neq a)} DoA(a_j)$. In this summation, a_i is a neighbor of the *Initiator* a, and a_j is a neighbor of a_i excluding a. After receiving the information about DoA, the *Initiator* partitions the rest of its tasks, $t_j \in (T_a - T_a^{'})$, into several groups. The process of partition depends on the DoA summation from each neighboring agent. The more the DoA value is, the more tasks the relative neighbor will be committed proportionately. Let us suppose an example, in a P2P multi-agent system, the *Initiator*, a, has two neighbors, namely agents a_2 and a_3. The summation DoA of agents a_2 and a_3 are 3 and 4, respectively. We also suppose that a, the *Initiator*, has 14 tasks left. In this situation, the *Initiator* commits 6 tasks to a_2 and 8 tasks to a_3, separately, where 6 is computed from $\frac{14\times3}{3+4}$ and 8 is equal to $\frac{14\times4}{3+4}$. Then, agents, a_2 and a_3, become the *Mediators* and send task request messages to their neighbors for help. This process will

be iterated until all tasks have been allocated or the TTL value of each task reaches 0 or although there are still some unallocated tasks, no more agent can be requested. There might be a condition during the process that one agent is a common neighbor of other two or more agents. In this condition, $TaskID(t)$ is utilized to distinguish different tasks. If this common agent has been requested by one agent, it will reject other request messages with the same $TaskID(t)$.

In the concurrent situation that is one agent has been requested by two or more other agents nearly simultaneously, the agent responds their requests with First-Come-First-Service (FCFS) mechanism, and its available resources announced in each response message will exclude the former announced ones.

IV. Experiment and Analysis

To evaluate the performance of ETAP, we compare ETAP with the Greedy Distributed Allocation Protocol (GDAP) [14]. GDAP is utilized for allocating tasks in a distributed environment. It only allows neighboring agents to help with a task. In this section, we first depict GDAP briefly. Then, the settings of experiment environment and criteria are introduced. Finally, the experiment results and the relevant analysis are illustrated.

A. Greedy Distributed Allocation Protocol

Greedy Distributed Allocation Protocol (GDAP) is employed to handle task allocation problem in agent social networks. An agent social network is defined in [14] as an undirected graph where vertices are agents and each edge indicates the existence of a social connection between two agents. The task allocation process of GDAP is described briefly as follows. All manager agents (like *Initiator* in this paper) try to find neighboring contractors (the same as *Participant* in this paper) to help them with their tasks. They start with offering the most efficient task. Out of all tasks offered, contractors select the task with the highest efficiency, and send a bid to the related manager. A bid includes all the resources the agent is able to supply for this task. If sufficient resources have been offered, the manager selects the required resources and informs all contractors of its choice. When a task is allocated, or when a manager has received offers from all neighbors but still cannot satisfy its task, the task is removed from its task list.

It can be seen that the main shortcoming of GDAP is that it only relies on neighbors which may cause several tasks unallocated due to limited resources, while our research is trying to figure this problem out.

B. Experiment setting

In order to compare the two protocols, ETAP and GDAP, we set an experiment environment for testing them. Two different networks, both of which are P2P architecture, are simulated for evaluating ETAP and GDAP as follows.

1) *Small-world networks* in which most neighbors of an agent are also connected to each other. In this experiment, the generation of a small-world network can follow the approach proposed in [13].
2) *Scale free networks* mean that in a network, a couple of agents have many connections while many other agents have only a small number of connections. The method presented in [2] is borrowed to create a scale free network in this experiment.

There are three different setups used in our experiment.

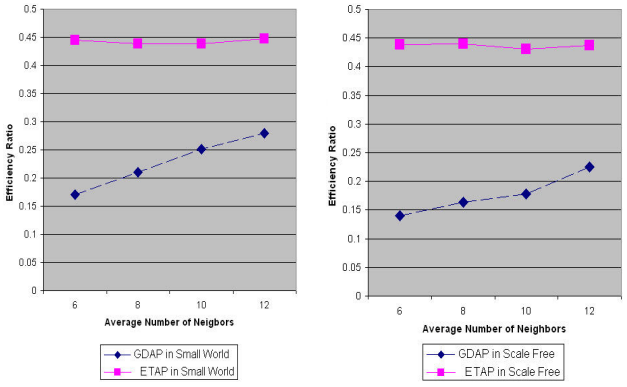
- *Setup 1*: The number of agents and tasks in the P2P network are 50 and 30 separately. The number of types of different resources is 5 and each agent randomly has several of them. The average number of resources for each type is 30 and the average number of resources required by each task is also 30. The *TTL* value for each task is set to 5. In this evaluation, we suppose that each task only needs one type of resources because task decomposition is not considered in this paper (as described in Subsection III-A). The tasks are distributed uniformly on each agent. The exact number of resources of each resource type an agent has and required by a task are both distributed normally. In addition, the average efficiency of tasks is 10 and the exact efficiency of a task also satisfies normal distribution. The only changeable attribute is the average number of neighbors in this setup. This setup is designed to show how different average number of neighbors influences the performance of both ETAP and GDAP.
- *Setup 2*: This setting is similar to *Setup 1* but with a few modifications. The *TTL* value for each task varies from 2 to 10 and the average number of neighbors is fixed at 8. The purpose of this setting is to test the adaptability of ETAP.
- *Setup 3*: In this setting, the average number of neighbors is fixed at 8. The number of agents fluctuates from 100 to 500 and the ratio between the number of agents and tasks is confirmed at $5/3$. The proportion of the number of agents and resources types is set to $10/1$. The *TTL* value for each task transforms from 10 to 50 in order to match the fluctuation of the number of agents. This setup is used for demonstrating the scalability of both GDAP and ETAP in different scale networks with a fixed average number of neighbors.

In this experiment, two criteria are evaluated, i.e. *Efficiency Ratio* and *Run Time*. *Efficiency Ratio* is the proportion of summation efficiency of completed tasks and the expected total efficiency of tasks. *Run Time* is the performing time of ETAP and GDAP in each network under different situations respectively. The unit of *Run Time* is millisecond. For simplicity, we suppose that once a task has been allocated to a *Participant*, the *Participant* would definitely finish this task without failure.

C. Experiment Results

The experiment is performed based on the three aforementioned setups with two different networks for ETAP and GDAP. In order to achieve precise results, each evaluation step is executed 30 times and obtain the average data.

1) Experiment Results from Setup 1: This experiment is done on *Setup 1* as described in IV-B. The purpose of this experiment is to test the influence of different average number of neighbors on both protocols.



(a) GDAP and ETAP in Small World Network

(b) GDAP and ETAP in Scale Free Network

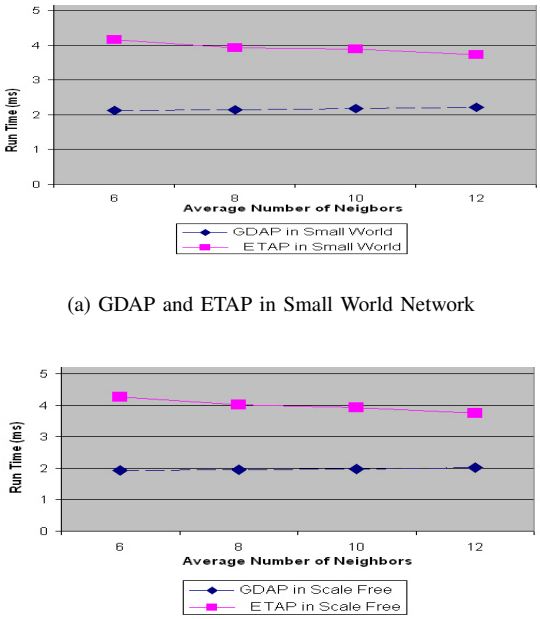
Figure 2. Efficiency Ratio on different average number of neighbors in different networks for GDAP and ETAP

In Figure 2, it is demonstrated that Efficiency Ratio of ETAP in different conditions is much higher and more stable than that of GDAP. This is because task allocation with GDAP is only depending on neighbors of the *Initiator*. Therefore, more neighbors make tasks have more opportunities to be solved. Comparatively, ETAP relies on not only neighbors but also other agents if needed. This feature results in steady performance of ETAP. It is also shown that the performance of GDAP in Small World network is better than that in Scale Free network. This can be explained that in Scale Free network, most agents have very few neighbors which lowers the capability of GDAP. With more average number of neighbors, the performance of GDAP has a trend to converge to that of ETAP. The reason of this situation is that when there are enough neighbors, *Initiator* has more probability to obtain sufficient resources for dealing with its tasks without reallocating tasks further.

Figure 3 depicts the *Run Time* of two protocols in different situations. As ETAP has to reallocate tasks when resources from neighbors are insufficient, the *Run Time* of ETAP is higher than that of GDAP. The worst condition appears in Scale Free network which is owing to most agents have few neighbors that could lead to reallocation steps increasing and more time spending. Oppositely, the presentation of GDAP in this test is relatively steady due to its considering only neighbors which could decrease the time and communication cost during task allocation process. It is also found that with the ascending of average number of neighbors, the time overhead of ETAP is dropping gradually. This is because when *Initiator* has more neighbors, it holds more chances to achieve resources for its tasks directly from neighboring agents. Therefore, the task reallocation steps would decline which could save time consumption.

2) Experiment Results from Setup 2: This experiment is done on the setting described in *Setup 2* which is employed to test the adaptability of ETAP.

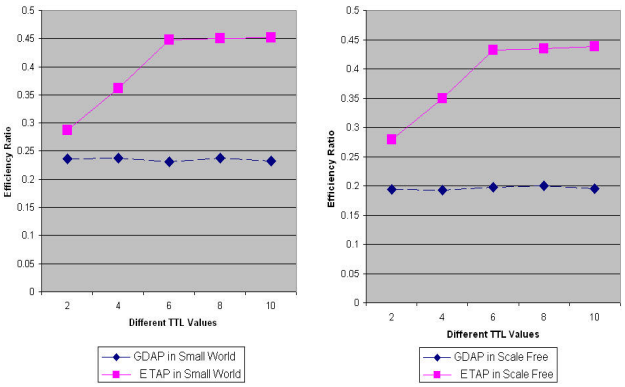
Figure 4 provides that with *TTL* value ascending, the *Efficiency Ratio* of ETAP also soars up. This is because the



(a) GDAP and ETAP in Small World Network

(b) GDAP and ETAP in Scale Free Network

Figure 3. Run Time (ms) on different average number of neighbors in different networks for GDAP and ETAP



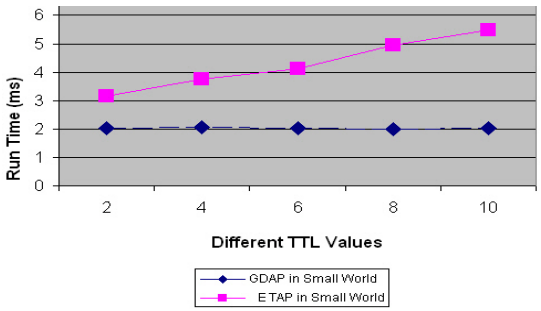
(a) GDAP and ETAP in Small World Network

(b) GDAP and ETAP in Scale Free Network

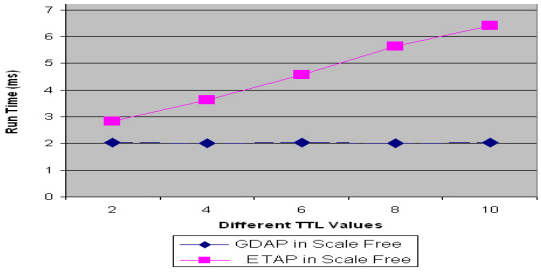
Figure 4. Efficiency Ratio on different *TTL* value in different networks for GDAP and ETAP

higher the *TTL* value is, the more the task reallocation steps are. Therefore, tasks could have more opportunities to be assigned. However, it should also be noticed that when the *TTL* value is more than 6, the *Efficiency Ratio* of ETAP almost keeps steady. This can be explained that when the *TTL* value arrives at a threshold, the reallocation process would nearly traverse the entire network. Hence, it is known that higher *TTL* value cannot bring more *Efficiency Ratio*, if its value exceeds a limitation.

From Figure 5, it is evident that the *Run Time* of ETAP



(a) GDAP and ETAP in Small World Network



(b) GDAP and ETAP in Scale Free Network

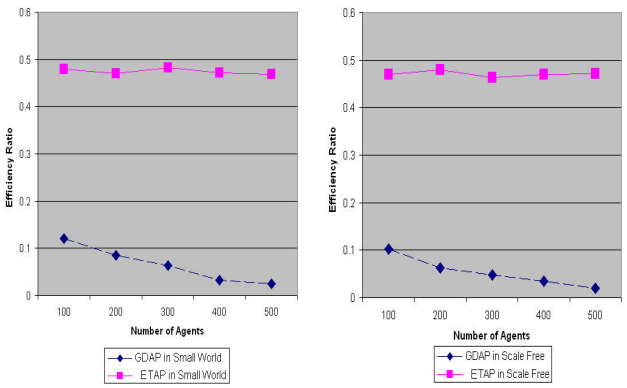
Figure 5. Run Time (ms) on different *TTL* value in different networks for GDAP and ETAP

increases gradually with *TTL* value rising. This is owing to more reallocation steps taking more time and communication cost. Since *TTL* is not related to GDAP, the performance of GDAP keeps firm during the evaluation process.

3) Experiment Results from Setup 3: This experiment is based on *Setup 3* which has been depicted in IV-B. The aim of this experiment is testing the scalability of both GDAP and ETAP in different network scales.

According to Figure 6, we can see that with the increasing of network scale, the *Efficiency Ratio* of GDAP is continually descending while that of ETAP can keep stable and high. This case can be argued that when the network scale soars up, tasks and types of resource also rise proportionally. Although the average number of neighbors is fixed, more tasks and resource types might still lead to tasks unallocated if *Initiator* requesting only neighboring agents. In addition, the condition in Scale Free network is worse than that in Small World network. The reason is the same as the one described in IV-C1 that in Scale Free network, many agents only have a few neighbors which is not good for GDAP. Compared with GDAP, benefited from task reallocation, ETAP can preserve its performance.

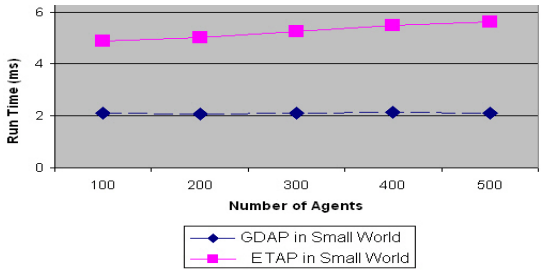
Figure 7 shows the *Run Time* of both GDAP and ETAP in different network scales. ETAP spends more time when there are more agents in the network. This is because the average number of neighbors is confirmed but more large network scale is accompanied by more tasks. Therefore, in order to allocate these tasks, more reallocation steps cannot be



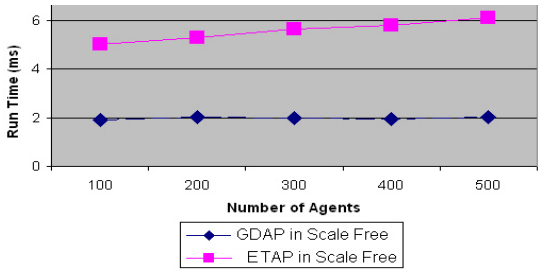
(a) GDAP and ETAP in Small World Network

(b) GDAP and ETAP in Scale Free Network

Figure 6. Efficiency Ratio on different number of agents in different networks for GDAP and ETAP



(a) GDAP and ETAP in Small World Network



(b) GDAP and ETAP in Scale Free Network

Figure 7. Run Time (ms) on different number of agents in different networks for GDAP and ETAP

avoided which results in time and communication overhead rising. On the other hand, the time consumption of GDAP is steady during the entire test process and keeps a lower level than ETAP. This can be apparently explained that GDAP only relies on neighboring agents. Hence, no matter how many agents in the network, if the average number of neighbors is fixed, the cost of time and communication for GDAP is almost settled.

D. Experiment Analysis

From the above description, it is obvious that ETAP can allocate more tasks according to the novel reallocation mechanism, while GDAP needs less time which is benefited from its only relying on neighbors. The efficiency of GDAP is easily affected by the average number of neighbors and specific network styles. Therefore, as depicted in the previous paragraphs and pictures, ETAP is more efficient, adaptable and scalable compared to GDAP. It can also be concluded that the overall performance of ETAP is better than that of GDAP. Although the execution time of ETAP is longer than that of GDAP, this drawback could be improved by simply reducing the *TTL* value. When *TTL* value is set to 1, ETAP approximately degrades to GDAP. That means *TTL* could be a balancer between *Efficiency Ratio* and *Run Time*.

V. An example of ETAP Potential Application

In this section, an intrusion detection example regarding the potential application of ETAP is provided. Intrusion is a set of actions which attempt to compromise the confidentiality, integrity or availability of a resource [6]. In order to prevent information from malicious attackers, Intrusion Detection System (IDS) is used to detect various intrusions in network environment.

There are numerous systems proposed for intrusion detection. However, many of them required a central controller or decision making point although the data collecting might be distributed, such as [3]. There are two common issues in such centralized architectures. The first is single point failure which means a central server or central database is easy to become an outstanding target for attackers. The second is that when a system with a single central server and numerous clients, the scalability is a problem, as the server has to maintain many connections to the clients. In order to overcome the two problems, several P2P intrusion detection architectures were presented, like [15]. In this paper, we will utilize ETAP to demonstrate the detection process against *Doorknob-Rattling* attack [12] in a P2P environment.

In *Doorknob-Rattling* attack, the attacker tries a few username and password combination on some different computers. This results in very few login attempt failures on each computer and the system might not flag such suspicious activities. Therefore, the records about login failures from multiple hosts need to be correlated to check for *Doorknob-Rattling* originated from any remote destination.

The Figure 8 is an example of a P2P network (such as Ad hoc network [8]) which has four agents to be attacked by a remote host simultaneously. In this example, the task can be described as that the detection of *Doorknob-Rattling* attack needs to check failed login attempt records on several agents and this task is a high emergent task. It is also supposed that the number of tasks is 3 and the 3 tasks has the same task tuple. Following the definition about task in Subsection III-A, the description could be matched as the tuple that $<TaskID(t) = Doorknob - Rattling$, $Resource(t) = FailedLoginAttempt > 5$, $Benefit(t) = 10$, $Position(t) = Agent1$, $TTL(t) = 6>$. In this example, *FailedLoginAttempt¿5* is the assumed threshold of failed

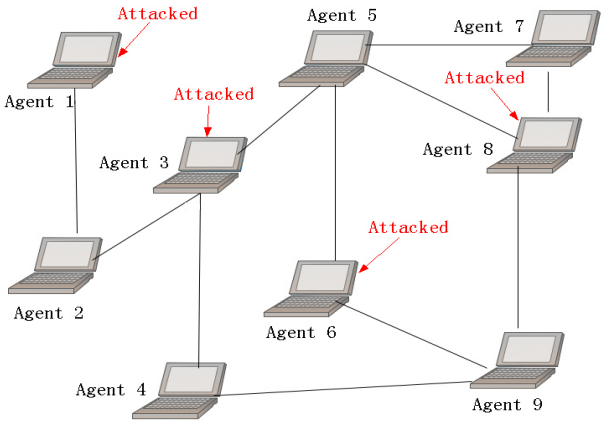


Figure 8. An Example P2P Network which has been attacked

login attempt. That means if the number of failed login attempts is more than 5, this record would be considered as a suspicious one and the agent should initiate a detection process. Furthermore, $Benefit(t)$ can be circumscribed according to emergency of the task and, here, the number 10 is just an instance.

In terms of ETAP, *Agent 1* starts the task allocation process, namely detection process in this example, by first requesting its neighbors for help. However, the only neighbor of *Agent 1* is *Agent 2* which cannot supply enough resource for *Agent 1*. Then, *Agent 1* reallocates its 3 tasks to *Agent 2* following the ETAP approach described in Subsection III-B. *Agent 2* now becomes the *Mediator* and allocates the tasks on behalf of *Agent 1*. *Agent 2*'s neighbor *Agent 3* can handle one of the 3 tasks and *Agent 2* then reallocates the rest of 2 tasks to *Agent 3*. After that, *Agent 3* reallocates the last 2 tasks to *Agent 5* according to communication about each neighbor's DoA. Finally, *Agent 5* allocates the rest 2 tasks to its neighbors *Agent 6* and *Agent 8* one for each. *Agent 6* and *Agent 8* have the required resources for solving the tasks, and, therefore, the allocation process finishes.

Through this example, ETAP shows its potential capability of handling a real case. However, people might argue that in this example *Agent 3*, *Agent 6* and *Agent 8* may also initiate the detection process synchronizingly with *Agent 1*. This problem is called task duplication which may increase the redundant time and communication overhead. Agent negotiation mechanism could be borrowed to solving this issue in some extent. Nevertheless, the aim of this example is just providing an overview of ETAP potential application. Detailed discussion of application is out of the scope of this article. In addition, other task allocation protocols, if applied in real cases, may also encounter some problems.

VI. Conclusion and Future Work

In this paper, we proposed an efficient task allocation protocol for P2P multi-agent systems and demonstrated the potential application for intrusion detection in a P2P network. Although the protocol overcomes several problems, which exist in some current related works, due to its de-

centralization and reallocation features, it still has several drawbacks, which have to be faced in future research, as follows.

- Since tasks are allowed to be reallocated in this protocol, there is a delay of communication during task allocation process. However, compared to the advantages benefited from our presented protocol, delay is a trivial problem. If time would definitely be a significant factor in some cases, the TTL could be set to a lower value to alleviate this condition. In addition, other related works which exploited distributed allocation approaches also encounter this issue more or less.
- The reallocation mechanism of ETAP is based on DoA. DoA can only provide information about the available resources ratio of each agent, instead of specific resources. In this way, unfinished tasks of an *Initiator* might be committed to its neighbors whose neighboring agents have higher available resource ratios rather required resources. This is why the *Efficiency Ratio* of ETAP is not high enough even though the allocation process nearly traverses the entire network. Improvement of reallocation mechanism is the next step of our research.
- A trade-off has to be made about how much historical data, e.g. request and response messages, should be stored in each agent. Too much data occupies large disk space and memory in the agent, while too few data leads to the possibility of missing some traces about previous contracts. Nevertheless, this problem is not unique to our protocol but common to other task allocation protocols.
- During task allocation process, an attribute, *deadline*, would be better to setup because *Initiator* cannot always wait for the responses of request messages from other agents. If the predefined deadline of a request message is advent, the *Initiator* should ignore that request message and resends a new one. However, setting deadline will make the task allocation protocol extremely complex because each step of communication may face this embarrassment. In addition, since clock synchronization in a P2P multi-agent system is very hard to realize, setting deadline might not achieve its expected effects.
- Furthermore, for simplicity, we supply several assumptions. In real world, these assumptions might be difficult to satisfy. However, this research focuses on initial theoretical attempt instead of real applications and we will perfect this protocol further in the future.

In conclusion, several issues are worth future research. In this paper, we mainly studied how to reallocate tasks if resources from neighboring agents are not sufficient. An interesting way which could possibly improve the performance and adaptability of ETAP is refining the reallocation mechanism and utilizing a common interaction language to represent the communication messages. We will continue to reinforce this protocol.

REFERENCES

[1] S. Abdallah and V. Lesser. Modeling task allocation using a decision theoretic model. In *Proceedings of 4th Automous Agents and Multiagent Systems (AAMAS 2005)*, pages 719–726, Utrecht, Netherlands, July 2005.

[2] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] D. Dasgupta, F. Gonzales, K. Yallapu, J. Gomez, and R. Yarramsetti. Cids: An agent-based intrusion detection system. *Computer & Security*, 24(5):382–398, Aug. 2005.

[4] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of 2th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, pages 1–8, Melbourne, Australia, July 2003.

[5] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *Proceedings of 4th International Conference on Multi-Agent Systems*, pages 167–174, Boston, Massachusetts, USA, July 2000. IEEE Computer Society.

[6] P. Ning, S. Jajodia, and X. Wang. Abstraction-based intrusion detection in distributed environment. *ACM Trans. Information and System Security*, 4(4):407–452, Nov. 2001.

[7] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 1191–1198, Bologna, Italy, July 2002.

[8] P. SANTI. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37(2):164–194, June 2005.

[9] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, May 1998.

[10] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, C-29(12):1104–1113, Dec. 1980.

[11] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Trans. Syst., Man, and Cyber.*, SMC-11(12):61–70, Jan. 1981.

[12] S. Snapp and J. Brentano. Dids (distributed intrusion detection system): Motivation, architecture and an early prototype. In *Proceedings of the 14th NIST-NCSC National Conference on Computer Security*, Washintong, D.C., Oct., 1991.

[13] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[14] M. D. Weerdt, Y. Zhang, and T. Klos. Distributed task allocation in social networks. In *Proceedings of 6th Automous Agents and Multiagent Systems (AAMAS 2007)*, pages 500–507, Honolulu, Hawaii, USA, May 2007.

[15] D. Ye, Q. Bai, and M. Zhang. P2p distributed intrusion detections by using mobile agents. In *Proceedings of the 7th IEEE/ACIS International Conference on Computer and Information Science*, pages 259–265, Portland, Oregon, USA, May 2008.

[16] X. Zheng and S. Koenig. Reaction functions for task allocation to cooperative agents. In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 559–566, Estoril, Portugal, May 2008.