

Accepted Manuscript

A belief-desire-intention agent architecture for partner selection in peer-to-peer live video streaming applications

Suleyman Yildirim, Muge Sayit, Geylani Kardas



DOI: [10.1111/exsy.12086](https://doi.org/10.1111/exsy.12086)

To appear in: *Expert Systems*

Published online: 21 August 2014

Please cite this article as: Suleyman Yildirim, Muge Sayit, Geylani Kardas, A belief-desire intention agent architecture for partner selection in peer-to-peer live video streaming applications, *Expert Systems*, doi: [10.1111/exsy.12086](https://doi.org/10.1111/exsy.12086)

This is a PDF file of an unedited manuscript that has been accepted for publication. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

A Belief-Desire-Intention Agent Architecture for Partner Selection in Peer-to-Peer Live Video Streaming Applications

Suleyman Yildirim^{1,2}, Muge Sayit², and Geylani Kardas^{2,*}

¹TU.School for Technological Design, Stan Ackermans Institute, Eindhoven University of Technology, NL-5600 MB, Eindhoven, the Netherlands

s.yildirim@tue.nl

²International Computer Institute, Ege University, 35100, Bornova, Izmir, Turkey

muge.fesci@ege.edu.tr, geylani.kardas@ege.edu.tr

Abstract

In peer-to-peer (P2P) video streaming systems, one of the most challenging parts is to schedule video data dissemination, i.e. each peer should carefully select the partner(s) it receives video from and the partner(s) it sends data to. We believe that an agent-based partner selection approach may improve the quality of streaming by taking both autonomy and dynamic plan selection into account in a goal-oriented manner. In this study, a Belief-Desire-Intention (BDI) agent architecture for partner selection in P2P video streaming systems is introduced. The major concern of our study is to exhibit how to select the best partner during video streaming session while maximizing the quality of video and minimizing delay and hop count. The effects and comparative results of executing proposed agent behaviors are evaluated in the study. The proposed autonomous agent-based approach also provides an infrastructure in which the best plan for the achievement of optimum streaming goal can be dynamically determined and executed at runtime. Experimental results of the implementation have revealed us that both of the partner selection methods (with or without agents) manage to increase the video quality. However, the agent-based approach performs better in terms of received bitrate, delay and hop count during streaming.

Keywords: Agent, Belief-Desire-Intention (BDI), JACK, partner selection, peer-to-peer, video streaming.

1. Introduction

Development of software systems based on intelligent agents maintains its supremacy over both artificial intelligence and software engineering research areas. According to a widely accepted definition, an agent is a computer system that is situated in some environment, and

* Corresponding author. Tel: +90-232-3113223 Fax: +90-232-3887230

that is capable of autonomous action in this environment in order to meet its design objectives (Wooldridge, 2002). Considering autonomous, responsive and proactive characteristics of the agents, they bring a promising approach for various domains including live video streaming.

Nowadays, many of the network resources are consumed by video streaming applications running over the Internet. Peer-to-peer (P2P) video streaming systems use a method enabling video data exchange between peers. This approach reduces the overload of the servers during the utilization of the network resources and enables a large number of peers to enjoy video streaming (Xie et al., 2007; Liu et al., 2010; PPLive, 2012; PPStream, 2012; Sayit et al., 2012). In those systems, one of the most challenging parts is to schedule a video data dissemination, i.e. each peer should carefully select the peer(s) it receives video from and the peer(s) it sends data to. This mechanism is also known as partner selection. There are systems in which the partner selection is made by examining the buffer levels of partners (Hei et al., 2007), by considering upload bandwidth of partners (Zhang et al., 2005), by the distance between the partners and the source node (Li et al., 2009), or considering the past behaviors of the peers (Le Blond et al., 2012).

Recently, the software agents have been used in video streaming systems. Various studies incorporate the software agent technology with P2P live video streaming applications (Molina et al., 2009; Pournaras et al., 2009; Chen et al., 2010; Carrera and Iglesias, 2011; Orynczak and Kotulski, 2011). However, these noteworthy studies mainly focus on the different aspects of the streaming and they do not directly address the partner selection according to QoE parameters (e.g. bitrate, delay) with the agent-based approach, which is crucial for P2P live video streaming.

In this study, a novel Belief-Desire-Intention (BDI) agent architecture for partner selection is presented. We propose an agent-based software architecture in which bitrate, delay, and hop count knowledge from source to destination nodes are given as environment facts to the agents who represent peers (nodes) in underlying network called as overlay network. Considering the aforementioned facts, agents execute their plans and decide which of the partners in the overlay network provide the optimal streaming. In addition, the proposed method is compared with a non-agentified partner selection method. We believe that such an agent-based partner selection may improve the quality of streaming by taking both autonomy and dynamic plan selection into account in a goal-oriented manner. The major concern of our

study is to show how to select the best partner during video streaming session while maximizing the quality of video and minimizing delay and hop count. Furthermore, the proposed algorithm considers the buffer level of a peer. Each peer in a P2P video streaming system waits for a period to fill its buffer after starting the streaming application. A buffering mechanism is required since a peer may need to consume data from its buffer if the bitrate of the received video is not adequate to play the video file in a proper manner. A peer's buffer may also be filled during streaming session if one or more of the partners' available upload bandwidth is higher than the required bitrate of the video. In our proposed system, agent of a peer considers the peer's buffer level while making partner selection. Apart from these contributions, the influence of non-agentified and agent-based partner selection methods on this highly dynamic environment was compared and evaluated.

The rest of the paper is organized as follows: Section 2 presents a brief description of BDI architecture that we use during the design and implementation of our system. Section 3 describes P2P live video streaming system essentials. In Section 4, we introduce our BDI agent architecture for partner selection in P2P live video streaming. Section 5 discusses the implementation details of the system with JACK framework (AOS, 2012a). Section 6 includes the gained experiences and achieved results. Section 7 discusses the previous studies which make use of software agents in P2P video streaming applications. Finally, we conclude and describe future work in Section 8.

2. Belief-Desire-Intention (BDI) Architecture

Originating from Bratman's view on humans and folk psychology (Bratman, 1987), BDI architecture (Rao and Georgeff, 1995) brings a powerful abstraction mechanism to overcome complex problems and has been used in many agent-based systems. An illustration of the BDI architecture is shown in Figure 1. The core components are Beliefs, Desires, Intentions and Plans. Beliefs represent the knowledgebase and assumptions that an agent has about its environment. Desires stand for goals or objectives while intentions are deliberative attitudes of the agent. Intentions are the subset of the desires that an agent has committed to. Plans, which constitute the fundamental for the BDI architecture (Wooldridge 2002), are the collection of steps to achieve the intentions of an agent. An agent has a set of pre-compiled plans which is designed by the agent programmer. The decision as to which plan is employed during the execution of the agent is defined by the context condition of each plan. After the context condition is satisfied, the plan is deemed as applicable. Apart from these core

components, there is an additional construct called as event which represents the data input received by sensors or generated internally in order to trigger plans during the agent's execution.

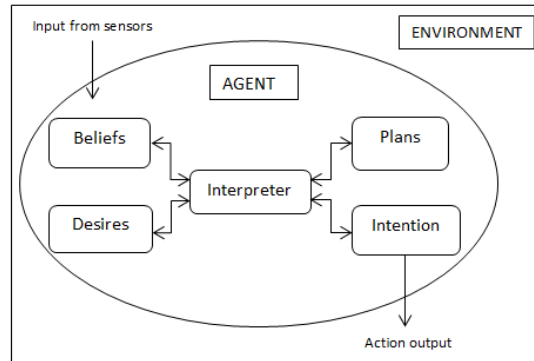


Figure 1: BDI agent architecture (taken from (Wooldridge, 2002))

Figure 2 provides a legend for tracing the execution of the proposed system according to BDI notions throughout this paper. The graphical symbols used to build agents and related components in our system are listed in this figure. Those symbols are adapted from the design palette of JACK Development Environment (JDE) (AOS, 2012b), which is used as the BDI design and implementation tool in our study.



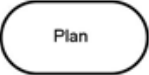
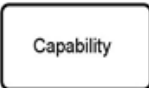

Symbol	Meaning
 Agent	Agent in a Multi-agent System (MAS)
 Event	Event that an agent handles or posts
 Plan	Plan that an agent uses
 Capability	Capability that an agent uses
 Beliefset (Beliefset)	Beliefset that an agent uses, reads or modifies

Figure 2: The graphical symbols and their meanings used throughout the paper

3. Application Domain

With an increasing demand of video streaming over the Internet, applications which enable the nodes in a video streaming session to send video data to each other have gained popularity since these applications help reduce the load of servers. A network which consists of nodes sending data to each other is called as P2P network. In video streaming applications running on a P2P network, latency and bandwidth constraints are more stringent than for file sharing systems running on the same type of network. Although there is no time constraint for a packet to be received in a classical file sharing system, each packet must reach to receiver before its play out time in a video streaming system. Therefore, video streaming multicast systems running on a P2P network must be designed according to timing constraints based on the play out time of video packets.

One of the most challenging parts of video streaming systems running on a P2P network is to cope with dynamic network conditions and unpredictable node behaviors. Thus, an application layer overlay network is created in order to adjust according to unpredictable node behaviors or to minimize packet transmission corruption due to changing network conditions. If an increase in packet loss rate is detected, new partner(s) may be chosen, since high packet loss rate causes decrease in video quality or disruption in display. Selecting and/or changing partners which video packets are received from changes this overlay architecture dynamically during streaming session. Since this is one of the most crucial parts of a streaming system, there are remarkable partner selection algorithms proposed in the literature. For instance, partner selection may be performed by considering the packets belonging to the candidate partners (Xie et al., 2007), by considering the available bandwidth of candidate partners (Zhang et al., 2005) or by considering the distance between candidate partners and source (Li et al., 2009). There are also some studies on partner selection according to their stability (Yu et al., 2006; Wang et al., 2010a). In these systems, if a node stays longer than the others in the system, then it is more preferable to be selected as a partner. In this paper, we consider three most important parameters of a streaming system, bitrate, delay and hop count; and we propose an agent-based software system that combines and evaluates all these parameters for a well-planned and efficient partner selection process.

Delay, hop count and bitrate parameters from source to destination nodes are given to the software agents as environment facts. Agents use their planning mechanism according to these facts and finally make the decision of partner selection.

As can be seen in Figure 3, the system architecture is designed in a layered manner. According to this architecture, there is an overlay network at the bottom level and a Multi-agent system (MAS) comprised of intelligent agents at the top level. Communication between each node at the bottom level and an agent representing that node is handled by TCP/IP communication which is described in detail in Section 4 of this paper. In the proposed architecture, each peer is represented by one agent. The reason that we prefer to use one-to-one mapping between agents and peers is the real-time operation delay constraint in P2P video streaming system. We observed that when one agent is responsible for a group of peers, the delay introduced by the communication between a peer and its representative agent causes the peer to take action at a slow pace during partner selection. Hence, by defining one agent running on top of each peer, the communication delay between the peer and its agent becomes zero, and the system performance does not degrade due to the communication cost.

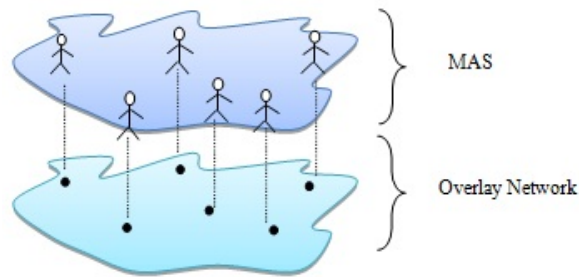


Figure 3: Layered system architecture

Agents need to take the following parameters into account in order to decide which partner to be chosen:

Bitrate: When a node takes part in a video streaming session and queries the parent nodes, the first parameter to be considered is the bitrate received from source to parent node. Bitrate is the most significant parameter that influences the partner selection for node taking part in video streaming session. When the bitrate decreases below a threshold, it directly affects the quality of streaming, or causes disruption in display. It is important to state that we assume the threshold at issue as 500 kbps since it is an acceptable video bitrate preferred in P2P live video streaming systems (Xie et al., 2007).

Delay: The most significant feature that distinguishes video streaming applications from file sharing is delay. In order to continue video streaming without interruption, packets must arrive before the playback time. Moreover, delay time from source to destination must also be short in live streaming applications.

Hop count: Hop count is a parameter that affects packet loss and delay. It is expected that considering hop count when deciding which partner to choose provides an advantage in terms of both delay and packet loss.

4. Proposed Agent Architecture for Partner Selection

The architecture of the agents employed during partner selection in P2P video streaming will be discussed in this section. It is worth noting that the discussion presented here mainly covers the agent internals for partner selection and the effects of applying such an agent-based partner selection during streaming. Whole software development process of the complete MAS is out of the scope of this paper. However, interested readers may refer to our companion work (Teket et al., 2014) for both the analysis and design of the MAS in terms of a well-defined software methodology.

In order to gather functionalities of agents into cluster and hence cover an agent's events, plans and beliefs (knowledgebase), we prefer to use *Capability* structures (Padgham and Winikoff, 2004). This approach enables higher level of abstraction and encapsulation than object oriented systems, the simplification of system design, and it allows code reuse between agents or other capabilities within the same agent (Busetta et al., 2000). Such capability abstraction also paves the way for the easy implementation of our BDI design in various MAS development frameworks such as JADEX (Pokahr et al., 2007) and JACK (AOS, 2012a) since those frameworks include agent capability descriptions as first-class entities. The capability description is mostly considered as the highest abstraction level inside Prometheus methodology (Padgham and Winikoff, 2004) and also used during the architectural design phase of Tropos methodology (Bresciani et al., 2004) while creating agents from actors. However, the derivation of agent architecture given in this paper is not directly bound to those agent-oriented software engineering (AOSE) methodologies, and hence the related BDI design can also be realized without using the capability abstraction. As will be discussed in the following subsections, building blocks in the introduced BDI architecture are events, beliefs, plans, etc. regardless of the capability abstraction. In subsection 4.1, a brief

description about the capabilities is given and then the details of each capability are discussed in subsections 4.2 and 4.3.

4.1. Overview

The agents attending P2P video streaming session are called as *Node* agents. Figure 4 illustrates the capability types of a *Node* agent. *TCPConnection* capability is related to the interaction with the agent's environment. It fetches the data coming from overlay network and adds this information to the agent's beliefset. On the other hand, *PartnerSelection* capability is related to the agents' proactive behavior on achieving the goal of selecting appropriate partners. It takes the advantage of sophisticated plans, elaborate decision-making and plan reuse. To summarize, the execution mechanism of the system is as follows: First, an environmental data is fetched by *TCPConnection* capability. Then, proactive behavior of the agent carries out the partner selection according to that fetched data.

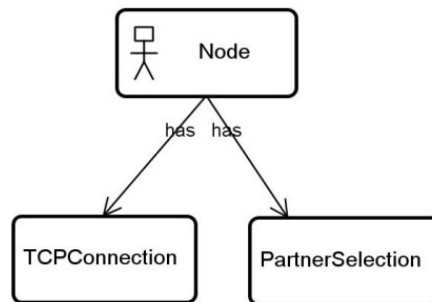


Figure 4: Agent Capability Overview

4.2. *TCPConnection* Capability

TCPConnection capability is responsible for communication with overlay network. It can be regarded as an interface between the agent and its environment. Figure 5 shows the execution cycle in *TCPConnection*. Agent first receives *DataStreamInput* event from the environment. This event triggers *SocketConnection* plan which establishes a TCP/IP connection in order to retrieve information from overlay network. "handles" statements are used in Figures 5 and 6 to represent agent plan executions when a specific event arises.

After the agent gets the data associated to each node from overlay network, it adds these information to its knowledgebase. In order to achieve this, *SocketConnection* plan posts *AddToBeliefset* event. When *AddToBeliefset* event is posted, it triggers its *InitBeliefset* plan which adds the environmental data to the *Partners* beliefset. With each "posts" statement in

Figure 5 and 6, we describe an event that the agent can post. Posting an event means that an agent creates an instance of the event and posts it internally (i.e. sends the event to itself).

There are three sections in Partners beliefset. The first part stores the other nodes from which Node agent actively gathers video packets. This part usually consists of three nodes, i.e. the number of partners of nodes equals to 3. Apart from the partners, there are two node lists for a node. One of these lists is called as membership table and the nodes exchange one or more of its partners by selecting a node from the membership table. The other node list is called as partnership table and the nodes in this table are selected if a suitable partner cannot be found in the membership table. "uses" statements in Figure 5 and 6 represent such utilization of beliefsets by the agent plans. Each "uses" statement means that an agent's plan reads or modifies a beliefset.

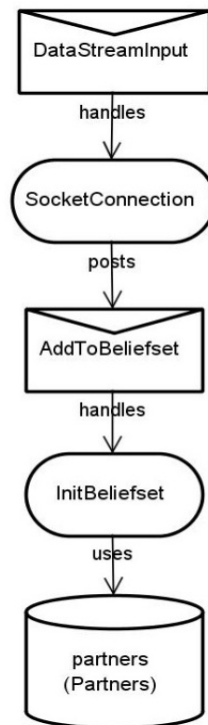


Figure 5: *TCPConnection* capability

4.3. *PartnerSelection* Capability

Figure 6 illustrates the execution cycle of *PartnerSelection* capability. *PartnerSelection* capability begins with the creation of *SelectPartner* event. After then, this event triggers *MonitorChanges* plan. After *MonitorChanges* plan calculates the buffer level and determines the node that has minimum upload bandwidth, the agent creates *BufferLevel* event in order to determine which node to be removed and which of the partner is the most suitable to receive

video data. This event conveys the information of buffer level and the node which is going to be changed. There are three plans that may be executed when *BufferLevel* event is generated by the *Node* agent. These plans are *IncreaseBitrate*, *IncreaseBufferLevel* and *ChangeByDelayAndHopcount*. The selection criterion is determined by the context condition of each plan. At this point, it is worth discussing the details of these plans.

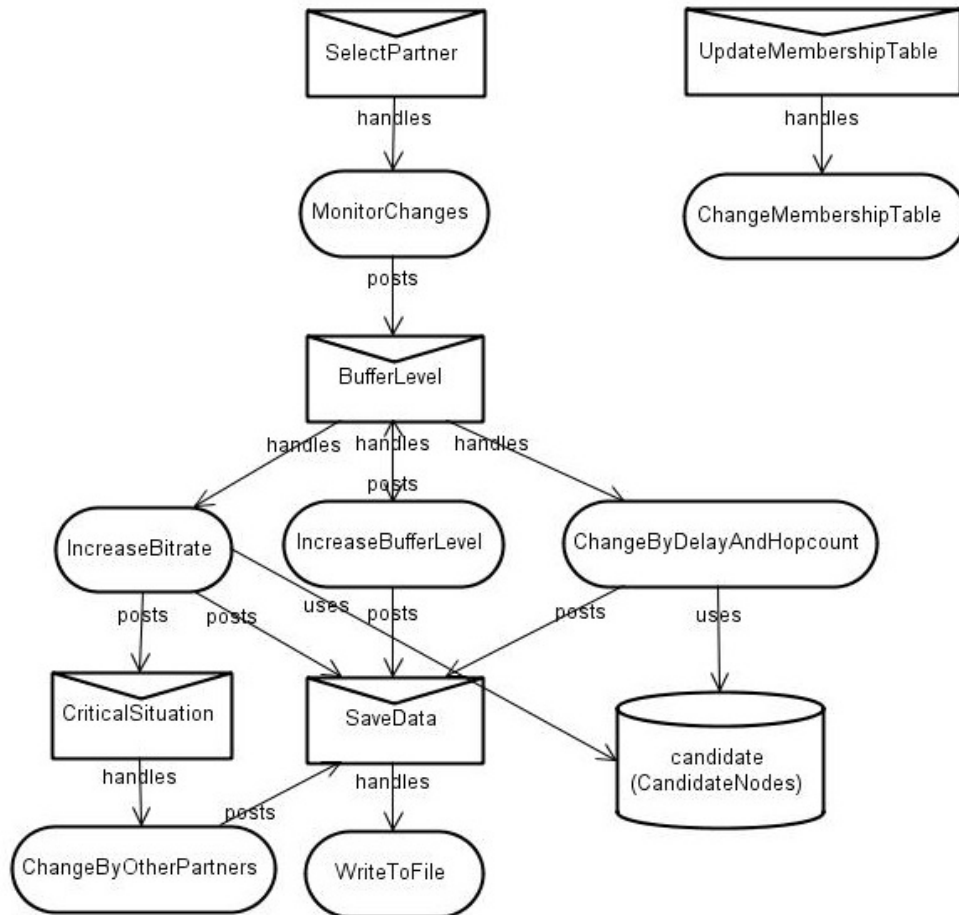


Figure 6: *PartnerSelection* Capability

IncreaseBitrate plan is responsible for increasing the total received bitrate and making it greater than the threshold. The context condition of *IncreaseBitrate* plan is satisfied if buffer level is less than or equal to initial buffer level which is set between 2 and 10 before starting the system. However, it does not take the total received bitrate into account. The execution of the plan is as follows: It determines candidate nodes by searching membership table in *Partners* beliefset. If the upload bandwidth of the node is greater than the node, which will be removed from *Partners* beliefset, it is added to *CandidateNodes* beliefset. This process goes on until the agent reaches to the end of the membership table. If it finds candidate nodes, the agent probes the best node to be selected from the *CandidateNodes* beliefset. Then, it

compares the delay of each node according to the following behavior model: If the delay difference of two nodes is less than the threshold, the agent selects the node that has minimum hop count. If the hop counts are equal, the agent chooses the node that has minimum delay. Finally, if the delay difference of two nodes is greater than or equal to the threshold, the agent again chooses the node that has minimum delay.

After determining the best candidate, the agent updates its *Partners* beliefset and checks whether it managed to increase the total received bitrate above the threshold or not. If the agent believes that it has succeeded, it calculates the average of the received bitrate, delay and hop count of the first three partners and saves the data (see the trigger of *WriteToFile* by sending *SaveData* event in Figure 6). When received bitrate goes down below the threshold, it means that the agent cannot perform streaming video properly. Hence, the agent tries to find a new way to increase the upload bandwidth above the threshold. Thus, a new task is created by generating *CriticalSituation* event to search for a suitable partner (Figure 6).

When *CriticalSituation* arises within the *Node* agent, *ChangeByOtherPartners* plan is executed as its context condition is true in all circumstances. We should note that *CriticalSituation* event conveys the data of two nodes. One of the data is related to the node that is going to be altered. In other words, this node is removed from the agent's partners. The other data is related to the node that is previously selected by *IncreaseBitrate* plan. Previously selected node is compared with the node selected by *ChangeByOtherPartners* plan which follows the similar selection procedure of *IncreaseBitrate* plan. However, it does not try to establish *CandidateNodes* beliefset as *IncreaseBitrate* does. When the agent selects a partner from partnership table, it also checks whether the selected node's upload bandwidth is greater than the node that will be removed. If the selected node's bitrate is greater than bitrate of the node that is previously selected by *IncreaseBitrate* plan, *ChangeByOtherPartners* plan updates the agent's *Partners* beliefset. The agent calculates the average of the received bitrate, measured delay and hop count values of the partners and saves the data. When the selected node's upload bandwidth is less than the upload bandwidth of the node that is selected by *IncreaseBitrate* plan, *ChangeByOtherPartners* plan does not select any partner. Therefore, it saves the selection of *IncreaseBitrate* plan.

On the other hand, *IncreaseBufferLevel* plan is responsible for increasing the buffer level as its name depicts. The context condition of this plan is satisfied when buffer level is between 2

and 4 seconds. In that case, the total received bitrate might be less or greater than the threshold. In order to increase the buffer level, the agent must either increase the total received bitrate of the partners or decrease the threshold. Since the threshold is a user defined static variable, the only way to increase the buffer level is to raise the total received bitrate. Although *IncreaseBitrate* plan is responsible for increasing the bitrate, we choose to use another mechanism when the amount of the buffer level is different from the context condition of *IncreaseBitrate* plan. To that end, the agent searches for the nodes in membership table. The algorithm given in Listing 1 is executed for selecting nodes in *IncreaseBufferLevel* plan.

```

1   boolean nodeFound = false;
2   for each node N in membership table do
3       if (b(Ni) >= b(Nk) and d(Ni) <= d(Nk) and h(Ni) <= h(Nk)) then
4           select(Ni);
5           nodeFound = true;
6       end
7   end
8   if (nodeFound) then
9       AddToBeliefset(Ni);
10      RecalculateBufferLevel();
11      for each node M in active partners do
12          select(Min{b(Mi), b(Mk)});
13      end
14      postBufferLevelEvent();
15  end
16  else
17      CalculateReceivedBitrate();
18      SaveData();
19      false; //causes plan to terminate
20  end

```

Listing 1: Node selection algorithm in *IncreaseBufferLevel* plan

The execution mechanism is as follows: First, agent determines the best node which has higher upload bandwidth, smaller delay and smaller hop count than the node that will be changed (Lines 2 -7). If the agent finds a suitable node, it updates *Partners* beliefset (Line 9). Second, the agent re-calculates the buffer level since the total received bitrate of the partners have changed (Line 10). Third, it searches the partners and finds the node having a minimum upload bandwidth (Lines 11 -13). At last, it posts the *BufferLevel* event again (Line 14). Note that *BufferLevel* conveys the updated buffer level and node information to be removed from the agent's beliefset. This process continues until membership table is empty or buffer level reaches above 4 seconds. In other words, the agent exhibits a goal oriented behavior by insisting on increasing buffer level. If the membership table is empty and *IncreaseBufferLevel* plan cannot find a node that increases the buffer level above 4 seconds, it saves the last

selection (Lines 16 -19). Then, it finishes its execution. If *IncreaseBufferLevel* plan succeeds to increase buffer level above 4 seconds and if the total received bitrate of the partners is greater than or equal to the threshold, *ChangeByDelayAndHopcount* plan's context condition is satisfied and the related plan is started.

The purpose of *ChangeByDelayAndHopcount* plan is to select better partner as it has already received satisfactory bitrate. Therefore, the agent tries to minimize hop count and delay simultaneously. Within the framework of this plan, *CandidateNodes* beliefset is updated according to the agent's inference on partners' delay, hop count and received bitrate.

As the last to say, the agent checks the cumulative received bitrates of the nodes in membership table in a constant manner. In order to observe the changes, the agent sends itself an *UpdateMembershipTable* event from time to time. *UpdateMembershipTable* event triggers the *ChangeByMembershipTable* plan which removes all nodes that have inefficient amount of cumulative bitrate.

5. Implementation of Agent-based Partner Selection

In order to implement the BDI agent architecture proposed in this paper, the JACK framework (AOS, 2012a) was chosen. There also exist various agent platforms other than JACK for implementing agent systems such as JADE (Bellifemine et al., 2001), JADEX (Pokhahr et al., 2005; Pokahr et al., 2007), JASON (Bordini et al., 2007), 2APL (Dastani, 2008) and GOAL (Hindriks, 2009). Furthermore, there is also an active work for extending the capabilities of BDI systems by presenting expressive goals, planning and real-time execution (Sardina and Padgham, 2011). However, the following features of JACK framework caused us to prefer JACK during the implementation.

JACK is an agent development framework to create autonomous systems which is built on the top of Java programming language, and which is based on the BDI architecture (Winikoff, 2005). JACK Agent Language is the main component of JACK that extends Java programming language in both syntactic and semantic way. It has three constructions: base classes, declarations and reasoning method statements. Base classes are *Agent*, *Beliefset*, *Event*, *Plan* and *Capability* that enable programming according to the agent-oriented paradigm. Declarations specify the relationships and dependencies between the base classes.

Reasoning method statements describe the actions that the agent can perform to exhibit an intelligent behavior.

The agents, designed according to the architecture presented in Section 4 of this paper, were implemented by using the JACK BDI Application Programming Interface (API). Similar to our MAS design process, we also used JACK Development Environment (JDE) (AOS, 2012b) during the system implementation step. JACK codes can easily be achieved from the graphically modeled MAS designs inside JDE. It is not possible to discuss the full implementation in the paper due to space limitations. However, in order to give some flavor of the implementation, *IncreaseBitrate* plan of proactive behavior is discussed here with reference to the related event and the beliefset declarations.

BufferLevelProactive event, which is received by the *IncreaseBitrate* plan of the agent, was implemented as the extension of a JACK *BDIGoalEvent* base class (see Appendix A). Furthermore, *Partners* beliefset of *Node* agents was implemented as JACK *ClosedWorld* (see Appendix B). Each beliefset should be inherited from either *ClosedWorld* or *OpenWorld* base class in JACK and it consists of a key field, value fields and queries. Open World semantics is used where a belief is true, false or unknown. On the other hand, a belief is either true or false in Closed World. Interested readers may refer to (AOS, 2012a) for further information on these two beliefset types.

IncreaseBitrate plan was created as the subclass of JACK *Plan* class as expected. It covers all declarations required for the relations between other JACK Agent Language types such as events, beliefsets and/or interfaces the plan uses (see Appendix C). Agent reasoning and actions required for the execution of *IncreaseBitratePlan* were coded inside the *body()* method of the related JACK Plan class. By means of this method, the agent performs tasks in order to achieve the goal of increasing the total received bitrate above the threshold (see Appendix D).

6. Results

In this section, we discuss the results of executing the agent system that is constructed according to the proposed architecture. The upload bandwidth, delay and hop count values of the nodes in the membership tables of the agents are generated according to Beta distribution (Beta Distribution, 2012). We choose the Beta distribution as it can generate the distributions

representing different scenarios in real systems (Wang et al., 2010b). Based on the parameters generated from the distribution model, both agent-based and non-agentified partner selections were observed. In other words, the agent's behavior was compared to that of the non-agentified partner selection in which agents are not employed and the system is constructed as a classical P2P video streaming system like (PPLive, 2012) or (PPStream, 2012). For this classical and non-agentified system, peer software was designed and implemented to perform the partner selection in which a peer selects the partners having the maximum upload bandwidth. Similar to ordinary P2P streaming applications, only the instant values of delay, hop count or bitrate properties are taken into account for the determination of best partners in our non-agentified implementation. On the other hand, the agents in our agent-based implementation consider the combination of all these properties, apply in the above discussed planning mechanism and make selection in a proactive manner.

While the non-agentified approach does not employ sophisticated planning mechanism for the partner decision, our agent-based system uses goal oriented approach when selecting the best partner. According to the results of the experiment, it is observed that the non-agentified system makes decisions with greedy approach, whereas proactive structure of the implemented agent-based system applies more elaborate selection mechanisms.

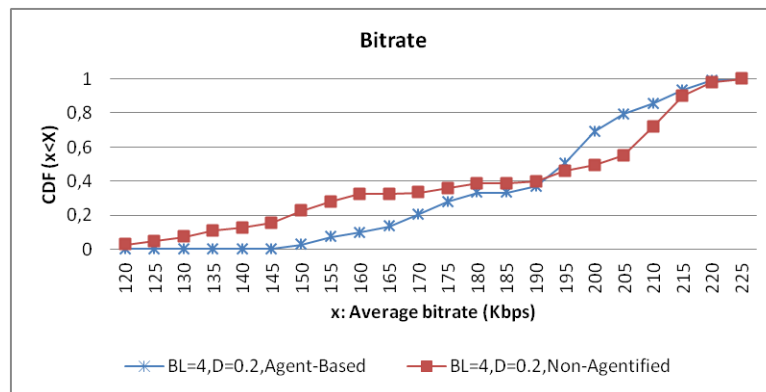
Before getting into the details of graph explanations, let us introduce the abbreviations and terms used in the graphs. For each line graph, there are two abbreviations: *BL* and *D*. While the former stands for initial buffer level of the agent, the latter stands for the delay difference between the two nodes. The graphs have also two terms, respectively *Agent-Based* and *Non-Agentified*. As their names already denotes, *Agent-Based* represents the agent-based partner selection while *Non-Agentified* represents the ordinary partner selection where the software agents are not employed. The agent's behavior is examined with 9 different initial buffer levels which takes off from 2 seconds and end in 10 seconds. Note that the initial buffer level determines the initial waiting time, i.e. the time elapsed until the playout time after streaming session begins. During the execution of the system, the buffer level continuously changes. In addition, for each initial buffer level, three different delay difference values are set as 0.2, 0.5 and 1 second to examine the effect of the delay difference. It is preferred to decrease the delay difference between the partner nodes in order to decrease the playback lag between partners. Note that, all graphs show the cumulative distribution of the selections. Due to space limitations, the results are presented for only the initial buffer level of 4 seconds, which is an

acceptable initial waiting time for P2P video streaming applications. However, the comparison of the selections of the agent-based and the non-agentified partner selection can be seen overall at the end of this section (in Table 1).

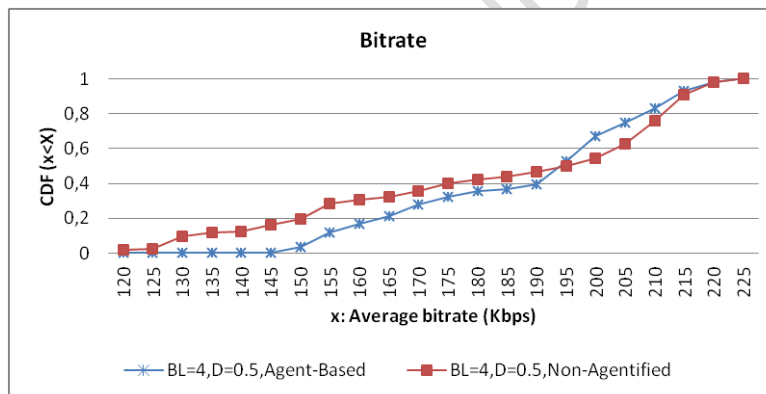
Figure 7 illustrates the average of the received bitrates for various delay differences after the node selection is over. As can be seen, the percentage of selecting the nodes that have higher bitrate is more likely in agent-based partner selection for all delay difference values. To give an example, in Figure 7(a), the percentage of the average of the received bitrate that is smaller than 150 kbps is almost zero in an agent-based partner selection, while it is 20% in non-agentified method. This means that the agent does not select the nodes that have a low bitrate. Moreover, this trend continues until the average of the received bitrate which is lower than 190 kbps. After that value, the probability of selecting the nodes that have higher bitrate increases. This is crucial for P2P video streaming systems in which the higher bitrate a node receives, the better quality it watches the video. For example, if the delay difference is set to 0.2 seconds, the percentage of the average of the received bitrate that is lower than 200 kbps is around 70% in agent-based partner selection, while it is around 50% in non-agentified partner selection (Figure 7(a)). It is also observed that the best selections are made by agent if the delay difference is set to 0.2 seconds. If the delay difference is set to 0.5 and 1 second, we observe that agent also outperforms the non-agentified method. For instance, the percentage of the average of the received bitrate that is smaller than 200 kbps is almost 80% in agent behavior while it is 60% in non-agentified method (Figure 7(b)). Similarly, the percentage of the average of the received bitrate that is smaller than 205 kbps is 70% in agent-based partner selection while it is 60% in non-agentified method (Figure 7(c)). As it is seen, the results show that the proactive structure of the agents succeeds to choose the nodes which have higher upload bandwidth.

When it comes to the measured delay, the agent always makes better decisions. As can be seen from Figure 8, the probability of selecting the nodes that have smaller delay is higher in agent-based partner selection than non-agentified method for all the delay difference values. For example, in Figure 8(a), if the delay difference is set to 0.2 seconds, the percentage of the measured delay that is smaller than 1 second is almost 60%, while it is 20% in non-agentified method. In other words, the probability of the measured delay that is smaller than 1 second is almost three times greater in the agent-based partner selection. The trend goes on until the delay value is less than 2 seconds. After that value, the selections of both the agent-based and

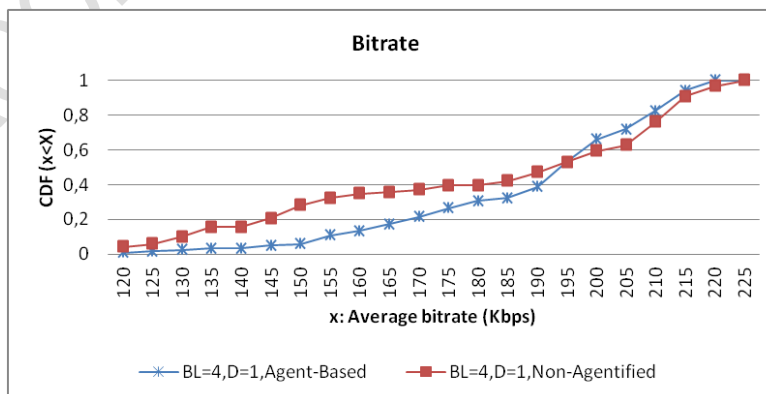
the non-agentified methods are identical. When the delay difference is set to 0.5 seconds (Figure 8(b)), it is observed that the selections of both mechanisms do not alter dramatically compared to Figure 8(a). If the delay difference is set to 1 second (Figure 8(c)), the percentage of selecting the nodes that have small delay slightly decreases or remains stable in both mechanisms. However, the agent-based partner selection outperforms the non-agentified method for each measured delay values.



(a)

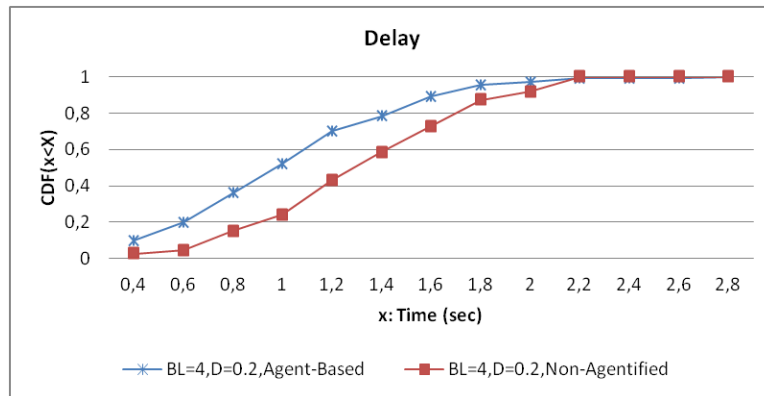


(b)

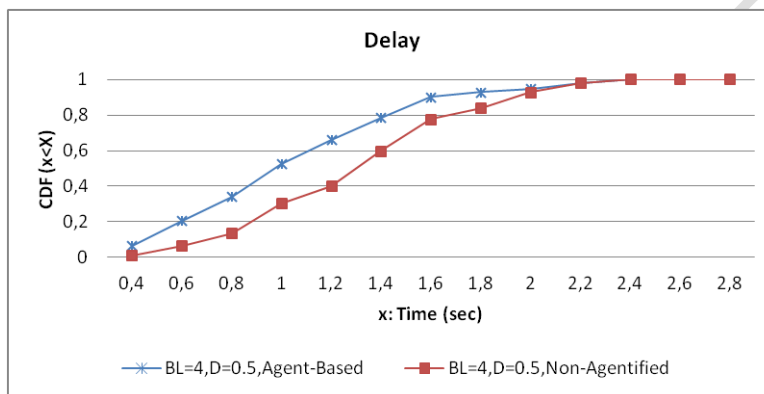


(c)

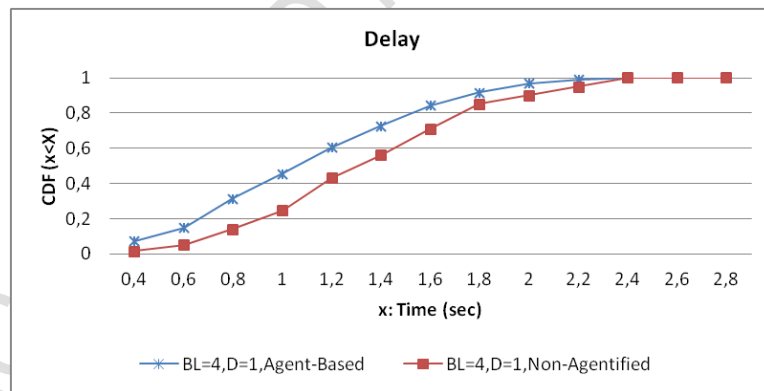
Figure 7: Cumulative distribution of the average received bitrate (a) Delay difference is 0.2 seconds; (b) Delay difference is 0.5 seconds; (c) Delay difference is 1 second



(a)

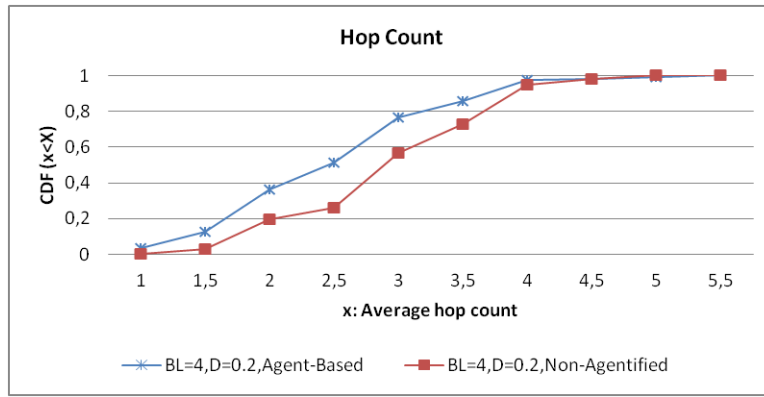


(b)

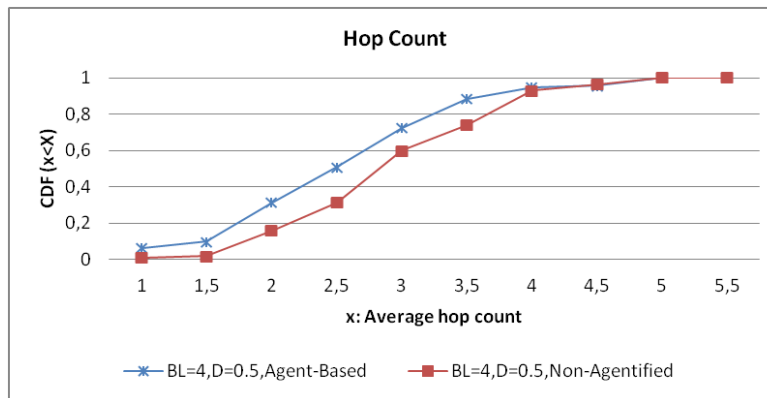


(c)

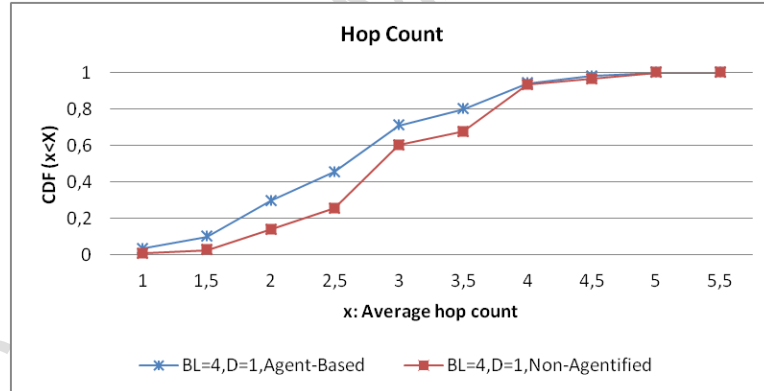
Figure 8: Cumulative distribution of delay. (a) Delay difference is 0.2 seconds; (b) Delay difference is 0.5 seconds; (c) Delay difference is 1 second



(a)



(b)



(c)

Figure 9: Cumulative distribution of hop count. (a) Delay difference is 0.2 seconds; (b) Delay difference is 0.5 seconds; (c) Delay difference is 1 second

Last, the agent-based partner selection also outperforms the non-agentified method when we take the measured hop count value into account. As can be seen from Figure 9, the probability of selecting the nodes that have smaller hop count is higher in agent-based partner selection. The best behavior of the agent is observed if the delay difference is set to 0.2 seconds. For instance, in Figure 9(a), the percentage of the hop count that is smaller than 2 is almost 40% in the agent-based partner selection, while it is 20% in the non-agentified method. Another

example is that if delay difference is set to 0.5 seconds (Figure 9(b)), the percentage of the hop count that is smaller than 2.5 is 50% in the agent-based partner selection while it is 30% in the non-agentified method. On the other hand, the selection results of both of the mechanisms are identical after the cumulative hop count value is greater than or equal to 4 seconds for all delay difference values. However, if delay difference is set to 1, the agent-based approach also makes a better selection compared to the non-agentified method (Figure 9(c)).

Table 1 shows all results of the agent-based and the non-agentified partner selections. It is clear from the table that both of the mechanisms manage to increase the video quality by keeping the average of the received bitrate value above the threshold. As mentioned in Section 3, in order to watch the video without any interruption, a peer must exceed a certain threshold which is defined as 500 kbps in our system. Moreover, for each initial buffer level and corresponding delay difference parameters, the agent manages to decrease delay and hop count better than the non-agentified method. For instance, if the initial buffer level and the delay difference are set to 3 seconds and 0.5 seconds respectively, the average of the measured delays in agent-based partner selection is 1.31 seconds while it is 1.52 seconds in the non-agentified selection. Similarly, when it comes to the hop count, the average of the measured hop counts in the agent-based partner selection is 2.9, while it is 3.2 in the non-agentified selection. Furthermore, when the initial buffer level is less than 6 seconds, the agent always makes better selection in terms of total received bitrate, delay and hop count. For example, if the initial buffer level and the delay difference are set to 4 seconds and 1 second respectively, the average of the received bitrates in the agent-based partner selection is 193 kbps while it is 184 kbps in the non-agentified selection. In a similar vein, the average of measured delays and hop counts are 1.28 seconds and 3.0 in agent-based partner selection respectively, while they are 1.51 seconds and 3.3 in the non-agentified selection. We also observe that the agent's selection in terms of bitrate is steadier than the non-agentified selection. There is a correlation between delay difference parameters and the agent's selection. The overall trend shows that when delay difference is set to 0.5 seconds, the average received bitrate generally reaches a peak when the results are compared to the other delay difference settings (i.e. 0.2 and 1 seconds). However, we cannot infer the attitude of the non-agentified method.

On the other hand, the agent's decisions in terms of the total received bitrate may not be better than the non-agentified method's decisions when the initial buffer level is greater than or equal to 6 seconds. For instance, if the initial buffer level is set to 8 seconds and the delay difference is set to 0.2 seconds, the average received bitrate in agent-based partner selection is 192 kbps while the average received bitrate in the non-agentified selection is 196 kbps. The reason might be that if the initial buffer level is too high (in our case it is greater than or equal to 6 seconds), the agent may not pay attention to the total received bitrate as it is already buffered a sufficient amount of video data in its buffer. For this reason, the agent tries to minimize the delay and hop count. Besides, it is not preferred to choose the buffer level too high in real live video streaming applications since it causes longer initial waiting time.

Table 1: Comparison of agent-based and non-agentified partner selection

Parameters		Agent-based partner selection			Non-agentified partner selection		
Initial Buffer Level (second)	Delay Difference (second)	Average Bitrate (kbps)	Average Delay (second)	Average Hop Count (hop)	Average Bitrate (kbps)	Average Delay (second)	Average Hop Count (hop)
2	0.2	206	1,3	3,0	194	1,53	3,4
2	0.5	209	1,3	3,0	195	1,57	3,4
2	1	208	1,3	3,0	191	1,52	3,3
3	0.2	209	1,31	2,9	188	1,47	3,2
3	0.5	210	1,31	2,9	185	1,52	3,3
3	1	209	1,28	2,9	187	1,48	3,2
4	0.2	193	1,21	2,8	188	1,50	3,3
4	0.5	192	1,23	2,9	186	1,48	3,3
4	1	193	1,28	3,0	184	1,51	3,3
5	0.2	193	1,33	3,0	190	1,53	3,2
5	0.5	197	1,38	3,1	197	1,51	3,2
5	1	195	1,25	2,9	191	1,42	3,1
6	0.2	192	1,32	3,0	194	1,45	3,1
6	0.5	193	1,34	3,0	196	1,49	3,2
6	1	198	1,36	3,1	196	1,53	3,3
7	0.2	192	1,40	3,2	194	1,54	3,3
7	0.5	194	1,46	3,3	195	1,53	3,3
7	1	191	1,33	3,1	193	1,53	3,3
8	0.2	192	1,29	3,0	196	1,47	3,2
8	0.5	198	1,29	3,0	193	1,37	3,0
8	1	192	1,31	3,0	197	1,46	3,1
9	0.2	192	1,31	3,0	190	1,48	3,2
9	0.5	194	1,33	3,1	197	1,56	3,3
9	1	192	1,38	3,1	193	1,52	3,3
10	0.2	194	1,27	2,9	196	1,44	3,1
10	0.5	193	1,31	3,0	197	1,50	3,2
10	1	191	1,30	3,0	193	1,48	3,2

7. Related Work

Recently, a number of agent-based systems have been proposed to tackle with certain problems of P2P live video streaming. Pournaras et al. (Pournaras et al., 2009) propose adaptive virtual organization model to build robust tree topologies. In this work, each software agent is responsible for a node that resides in underlying network infrastructure which is called 'overlay network'. The nodes are controlled by agents when a failure occurs in overlay network. In the tree-based hierarchical structures, the removal of a node may hamper the efficiency of streaming quality. Thus, this work addresses the effect of failures of nodes in tree overlay and how agents can be used to heal the possible failures in the tree overlay. In (Carrera and Iglesias, 2011), a multi-agent architecture for automatic diagnosis of multimedia streaming faults in uncertain situations is presented. The proposed system is evaluated in P2P streaming scenario in which a multimedia provider and a multimedia consumer take part. The authors propose two modules in order to accomplish a network diagnosis: hypothesis generation and hypothesis confirmation. Hypothesis generation uses Bayesian inference engine to infer the source of fault by analyzing network symptoms. The output of this phase is a hypothesis conveyed to second phase in which the deliberation of hypothesis confirmation takes place.

Chen et al. (Chen et al., 2010) present evolutionary games for cooperative P2P video streaming. They discuss the real-time P2P video streaming among groups of software agents to reduce traverse links and increase streaming performance. Peers, which are interested in joining a live video streaming session, form a group and cooperate with each other to increase streaming performance. According to this approach, each peer group has upload and download capacity. Peers choose k number of agents to download streaming data from other groups. Then, the agents distribute the data to the peers within the group. The problem is to determine how many peers should be chosen as agents by group members. The authors state that the probability of real-time streaming is higher compared to the non-cooperative P2P approaches. Molina et al. (Molina et al. 2009) propose an architecture for mobile transient networks in which a number of mobile devices deliver multimedia content to each other. The agent-based negotiation model is used inside the ad-hoc network to reach an agreement as to whether to deliver the content or not. In each mobile network, there is a coordinator agent which proposes to download multimedia content. After the coordinator agent proposes a content to download, the other agents start to negotiate and reach an agreement as to whether

to download the content or not. Then, the subset of agents which agree to download the content retrieves the specific part of the multimedia content. Orynczak and Kotulski (Orynczak and Kotulski, 2011) present an agent-based approach for real-time applications in terms of quality of service and security aspects. The system comprises agents which represent each node over an underlying network, and communicate with each other to build a dynamic routing table enabling more affecting routing. Each agent continuously observes different parameters given by other agents to control the quality of transmissions by analyzing bandwidth, lost packets and time difference between packets. The agents use these data to establish dynamic routing tables which are continuously updated whenever a change is noticed. Menkovski and Liotta (Menkovski and Liotta, 2013) discuss the design of an agent for adaptive video streaming systems. The agent examines the traffic and makes decisions based on the reinforcement learning. Taking the achieved quality of experience into consideration, the agent learns an optimal control strategy.

In order to position our approach inside the current picture of the agent-based video streaming literature, Table 2 gives a qualitative comparison of the related work by taking into consideration the following main features of such agent-based P2P video streaming systems: agent-peer mapping, the structure of the overlay network and the purpose of employing agents. Enabling adaptation, selection of overlay nodes, network layer routing and providing fault tolerance are the most encountered reasons for the use of the agents inside the P2P video streaming.

In (Pournaras et al., 2009), although the proposed system considers QoS parameters, it focuses on constructing resilient tree-based overlay networks while our proposed system primarily focuses on mesh-based overlay networks. A node receives video packets from one parent in the tree based system; on the other hand, a node in the mesh-based overlay may receive video from one or more parents. In (Carrera and Iglesias, 2011), the authors focus on streaming video between the two nodes, rather than designing a streaming protocol running over a network of large amount of nodes. Chen et al. (Chen et al., 2010) use agents to cluster nodes in a P2P network according to their bandwidth capacity without considering distance from source or packet loss rate. In (Molina et al., 2009), the coordinator agent chooses the peer couples, i.e. partners, according to their potential contribution, but coordinator agent does not consider available bandwidth or packet loss between peer couples. In (Orynczak and Kotulski, 2011), an agent may change the streaming path by changing network level routing

but this work does not consider node selection for the overlay network, which is an important parameter that provides seamless streaming in live video streaming systems. The focus of the work presented in (Menkovski and Liotta, 2013) does not involve the partner selection since the structure of the network is end-to-end. As a result, our BDI architecture differs from those studies by considering the overlay node (partner) selection in mesh-based streaming systems according to bitrate, delay and hop count at the same time.

Table 2: A qualitative comparison of agent-based P2P video streaming approaches

	Molina et al., 2009	Pournaras et al., 2009	Chen et al., 2010	Carrera and Iglesias, 2011	Orynczak and Kotulski, 2011	Menkovski and Liotta, 2013	Our BDI architecture
Agent-peer mapping	m-to-n	one-to-one	m-to-n	one-to-one	one-to-one	one-to-one	one-to-one
Structure of the overlay network	mesh	tree	mesh	end-to-end (unicast)	mesh	end-to-end (unicast)	mesh
Adaptation	-	-	-	-	-	+	-
Selection of overlay nodes	+	+	+	-	-	-	+
Network layer routing	-	-	-	-	+	-	-
Fault tolerance	+	+	+	+	+	-	+

There are also various studies which make use of agent technology including BDI implementations for different application domains (Munroe et al., 2006). Initial attempts to apply BDI approach to real world applications have started with space shuttle missions (Georgeff and Ingrand, 1989), continued with military simulations for combat pilots (Murray et al., 1995) and followed by unmanned air vehicles (Lucas et al., 2003). BDI notion is also used in manufacturing systems. In (Fletcher et al., 2003), an agent-based holonic control system was successfully developed. In this study, a specific type of agent is assigned for each control unit in manufacturing system. Other real world applications are in the area of

meteorological forecasting (Dance et al., 2003; Mathieson et al., 2004) in which Australian Bureau of Meteorology aimed to enhance the forecasting and alerting capabilities by using the multi-agent based solution. The recent studies show that BDI paradigm is still applicable to many problems. For instance, Nordbo (Nordbo, 2011) uses an agent technology in the context of robotics domain in which two robots exchange information in order to achieve their goals. An agent-based decision support system is introduced in (Sokolova and Fernandez-Caballero, 2009) for evaluating an environmental impact on human health. Data mining techniques for knowledge discovery are used by the intelligent agents as a foundation for decision making and recommendation generation. Similar to our work, the design environment of JACK is used during the implementation of the proposed system. Gascuena et al. (Gascuena et al., 2011) proposed a computational agent model which is applied to surveillance systems. Similarly, Gomez-Romero et al. (Gomez-Romero et al., 2011) presented an ontological knowledge representation by using BDI agent architecture for tracking moving objects. Design of a moving robot application for the detection and following of humans based on MAS approach is discussed in (Gascuena and Fernandez-Caballero, 2011). The implementation of the proposed system is again realized by using JACK. The last example (Kardas et al., 2012) can be given from the stock market domain in which BDI agents are employed during the stock trading. Although the above mentioned studies present significant application of BDI in autonomous system realizations, none of them considers the domain of P2P video streaming and brings an agent-based solution to the problem of optimal partner selection.

8. Conclusion and Future Work

A BDI agent architecture has been presented for partner selection in P2P video streaming system. To the best of our knowledge, the work described in this paper is the first attempt to propose an agent-based partner selection for mesh-based streaming systems considering the combination of bitrate, delay and hop count. Both required algorithms that should be executed inside the agent plans and the implementation details of the proposed system were discussed in the paper. Effects and comparative results of the agent-based and the non-agentified partner selection were evaluated.

Agents execute their plans and decide which of the nodes in the overlay network should be selected as partners for streaming. Furthermore, an autonomous agent-based approach, brought by this study, also provides an infrastructure in which the best plan to achieve the

optimum streaming or any other goals can be dynamically determined and executed at runtime.

Finally, as has already been discussed in the paper, experimental results of the implementation showed us that both of the partner selection methods (with or without agents) manage to increase the video quality by keeping the average received bitrate value above the threshold. However, the agent-based approach performs better in terms of received bitrate, delay and hop count during the streaming.

For future work, we will take the incentive mechanisms into account. The incentive mechanism allows us to construct a fair system, in which a peer contributing more, i.e. reserving more upload bandwidth to the system, receives video at higher bitrate. In the next phase of our study, we plan to implement an incentive P2P system with agents.

Acknowledgement

This work is funded by the Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) under grant 111E022.

Appendix A: Implementation of *BufferLevelProactive* event in JACK

```

1   public event BufferLevelProactive extends BDIGoalEvent {
2       public double bufferLevel;
3       public int changeNode;
4       public int changeBitrate;
5       public double changeDelay;
6       public int changeHopcount;
7       #posted as
8       change(double bufferLevel,int changeNode,int changeBitrate,double
9           changeDelay,int changeHopcount) {
10          this.bufferLevel = bufferLevel;
11          this.changeNode = changeNode;
12          this.changeDelay = changeDelay;
13          this.changeBitrate = changeBitrate;
14          this.changeHopcount = changeHopcount;
15      }

```

BufferLevelProactive event conveys buffer level and the node information that will be removed from the agent's beliefset (Lines 2 - 6). The information is conveyed by event's *change()* posting method (Line 8 - 14) which is declared by *#posted as* JACK specific method declaration (Line 7).

Appendix B: Implementation of *Partners* beliefset in JACK

```

1 public beliefset Partners extends ClosedWorld {
2   #key field int $nodeNumber;
3   #value field int $bitRate;
4   #value field double $delay;
5   #value field int $hopCount;
6   #indexed query getAllTuples(logical int $nodeNumber, logical int
      $bitRate, logical double $delay, logical int $hopCount);
7   #function query boolean checkTotalBitRate() {...}
8   #function query int getBitRateAverage() {...}
9   #function query double getDelayAverage() {...}
10  #function query double getHopCountAverage() {...}
11 }

```

Beliefset is expressed as a relation and consists of a key field, value fields and queries (Lines 2 - 10). In our implementation, the *Node* agent stores the information of the other nodes' as tuples. Each tuple represents a node which has an id, bitrate, delay and hop count. Key field is analogous to primary key and represents the "id" of a node (Line 2). Value fields are analogous to the data associated with key field, and represent the bitrate, delay and hop count (Lines 3- 5). In addition, when the agent needs to retrieve a tuple in its relation, it makes use of the user defined queries such as *getAllTuples*, *getBitRateAverage*, *getDelayAverage* and *getHopCountAverage* (Lines 6 -10).

Appendix C: Implementation of *IncreaseBitrate* plan in JACK (Part I)

```

1 public plan IncreaseBitrate extends Plan {
2   #handles event BufferLevel value;
3   #posts event CriticalSituation criticalSituation;
4   #posts event SaveProactiveData printResult;
5   #uses interface Node self;
6   #uses data Partners partners;
7   #uses data CandidateNodes candidate;
8   static boolean relevant(BufferLevel value) {
9     return (value.bufferLevel <= INITIALBUFFER);
10  }
11  context() {
12    partners.getTotalBitRate() < BITRATETHRESHOLD;
13  }
14  #reasoning method
15  body()
16  {
17    (...) //will be given in the next appendix
18  }
19 }

```

Between lines 2-7, the declarations are given for describing the relationship between events, beliefsets and/or interfaces. For instance, the events that are received and posted by this plan are declared to the agent by using *#handles event* (Line 2) or *#posts event* (Lines 3 - 4).

Similarly, the interfaces and beliefsets that are used are declared with *#uses interface* (Line 5) and *#uses data* (Line 6 - 7) respectively. Second, when *BufferLevel* event is received, the agent first checks whether *IncreaseBitrate* plan is relevant to the event via *relevant()* function (Lines 8 -10). Therefore, it checks the incoming data from event by comparing event's *bufferLevel* parameter. If the *bufferLevel* is less than or equal to 2 seconds, relevant function returns true and the second filtering occurs in *context()* method which is related to the agent's knowledgebase (Lines 11 -13). It uses *Partners* beliefset's *getTotalBitrate()* function to check the state of the world if the total received bitrate is less than the threshold. If it is true, *context()* returns true and the agent starts the top level reasoning method called as *body()* of the plan (Lines 15 -18).

Appendix D: Implementation of *IncreaseBitrate* plan in JACK (Part II)

```

1 #reasoning method
2 body()
3 {
4   (...)
5   for(Cursor c = partners.getAllTuples($nodeNumber,$bitRate,$delay,$shopCount);c.next()){
6     if ((i >= 3) && ($bitRate.getValue() >= value.changeBitrate) && (i < 10))
7       candidate.add($nodeNumber.getValue(),$bitRate.getValue(),$delay.getValue(),$shopCount.getValue());
8     i++;
9   } //end for
10  if (candidate.getAllTuples($nodeNumber,$bitRate,$delay,$shopCount).next()){
11    for (Cursor c = candidate.getAllTuples ($nodeNumber,$bitRate,$delay,$shopCount); c.next(); ) {
12      if (j == 0) {
13        selectNode = $nodeNumber.getValue();
14        selectBitrate = $bitRate.getValue();
15        selectDelay = $delay.getValue();
16        selectHopcount = $shopCount.getValue();
17      }
18      if ((j > 0) && (Math.abs($delay.getValue() - selectDelay)) < self.DELAY_DIFFERENCE) {
19        if (selectHopcount > $shopCount.getValue()) {
20          selectNode = $nodeNumber.getValue();
21          selectBitrate = $bitRate.getValue();
22          selectDelay = $delay.getValue();
23          selectHopcount = $shopCount.getValue();
24        }
25        if (selectHopcount == $shopCount.getValue() && selectDelay > $delay.getValue()) {
26          selectNode = $nodeNumber.getValue();
27          selectBitrate = $bitRate.getValue();
28          selectDelay = $delay.getValue();
29          selectHopcount = $shopCount.getValue();
30        }
31      }
32      if ((j > 0) && (Math.abs($delay.getValue() - selectDelay)) >= self.DELAY_DIFFERENCE
33        && selectDelay > $delay.getValue()) {
34        selectNode = $nodeNumber.getValue();
35        selectBitrate = $bitRate.getValue();
36        selectDelay = $delay.getValue();
37        selectHopcount = $shopCount.getValue();
38      }
39      j++;

```

```

40     } // end for
41     partners.remove(value.changeNode,value.changeBitrate,value.changeDelay,value.changeHopcount);
42     partners.add(value.changeNode,selectBitrate,selectDelay,selectHopcount);
43     if (partners.checkTotalBitRate()){
44         @post(printResult.saveAverageData(partners.getBitRateAverage(),partners.getDelayAverage(),
45             partners.getHopCountAverage()));
46     } else {
47         calculateMinUploadBandwidth();
48         if(!@subtask(criticalSituation.select(changeNode, changeBitrate, changeDelay, changeHopcount,
49             selectBitrate, selectDelay, selectHopcount));) {
50             @post(printResult.saveAverageData(partners.getBitRateAverage(),partners.getDelayAverage(),
51                 partners.getHopCountAverage()));
52         } //end if
53     } //end inner else
54 } else {
55     calculateMinUploadBandwidth();
56     if(!@subtask(criticalSituation.select(changeNode, changeBitrate, changeDelay, changeHopcount,
57         selectBitrate, selectDelay, selectHopcount));) {
58         @post(printResult.saveAverageData(partners.getBitRateAverage(),partners.getDelayAverage(),
59             partners.getHopCountAverage()));
60     } //end if
61 } //end outer else
62 } //end body()

```

The agent first retrieves all tuples from *Partners* beliefset by querying via *getAllTuples()* and assign the result to JACK *Cursor* (Line 5). JACK *Cursor* structure is similar to the relational database cursor which returns the multiple results according to the user's query. In addition to cursors, there are JACK specific logical members which are passed as an argument to the *getAllTuples()* function. They represent the unknown variables until an assignment to a specific variable is done. After all tuples are retrieved, the agent iterates through membership table and adds the tuples to candidate beliefset via *add()* base function of beliefset (Lines 5 - 9). After then, the agent checks the candidate beliefset via *getAllTuples()* (Line 10). If there is at least one tuple in candidate beliefset, the agent performs the node selection and removes the node that is going to be replaced with the selected node by using *remove()* base function of beliefset (Line 41). Then, the selected node is inserted into the beliefset with the index of removed node by *add()* method (Line 42). The next step of the plan is to check whether the agent has achieved the goal of increasing the total received bitrate above the threshold. Therefore, the agent checks the *Partners* beliefset with *checkTotalBitRate()* function which returns a boolean value (Line 43). If returned value is true, meaning that the total bitrate is above the threshold, the agent posts itself *printResult* event via *@post* reasoning method to save the data (Line 44). Otherwise, it re-calculates the tuple which has a minimum upload bandwidth (Line 47) and posts itself *criticalSituation* event to choose a partner from partnership table via *@subtask* reasoning method (Line 48). Note that *@subtask* is different from *@post* reasoning method since the agent waits for the completion of a plan triggered by

the posted event. If a plan, triggered by *criticalSituation* event, successfully chooses a node, *@subtask* method is also regarded as successful and plan continues its execution. Otherwise, *@subtask* statement returns false and plan is forced to save the last selection by posting itself *printResult* event via *@post* reasoning method (Line 50). Eventually, if the agent cannot find any node in its candidate beliefset, it again re-calculates the tuple which has a minimum upload bandwidth (Line 55) and posts itself *criticalSituation* event to choose a partner from partnership table via *@subtask* reasoning method (Line 56). If the method fails, *@subtask* method returns false and then the plan is forced to save the last selection by posting itself *printResult* event via *@post* reasoning method (Line 58).

References

(AOS, 2012a) Agent Oriented Software Group (2012) "JACK Intelligent Agents", available at: http://www.aosgrp.com/documentation/jack/Agent_Manual.pdf (last access: January 2014).

(AOS, 2012b) Agent Oriented Software Group (2012) "JACK Intelligent Agents Development Environment", available at: http://www.aosgrp.com/documentation/jack/JDE_Manual.pdf (last access: January 2014).

(Bellifemine et al., 2001) Bellifemine F., Poggi A, and Rimassa G. (2001) "Developing Multi-Agent Systems with a FIPA-compliant Agent Framework", *Software: Practice and Experience*, Vol. 31, Issue 2, pp. 103-128.

(Beta Distribution, 2012) Beta Distribution (2012) available at: <http://www.mathworld.wolfram.com/BetaDistribution.html> (last access: January 2014)

(Bordini et al., 2007) Bordini, R.H., Hubner, J.F., and Wooldridge, M. (2007) "Programming Multi-Agent Systems in AgentSpeak using Jason", John Wiley & Sons.

(Bratman, 1987) Bratman, M. (1987) "Intention, Plans, and Practical Reason", Harvard University Press, Cambridge, MA.

(Bresciani et al., 2004) Bresciani P, Perini A, Giorgini P, Giunchiglia F, and Mylopoulos J. "Tropos: An agent-oriented software development methodology", *Autonomous Agents and Multi-Agent Systems*, Vol. 8, Issue 3, pp. 203-236.

(Busetta et al., 2000) Busetta, P., Howden, N., Ronnquist, R., and Hodgson, A. (2000) "Structuring BDI agents in functional clusters", *Lecture Notes in Computer Science*, Vol. 1757, pp. 277–289.

(Carrera and Iglesias, 2011) Carrera, A. and Iglesias, C.A. (2011) "Multi-agent Architecture for Heterogeneous Reasoning under Uncertainty Combining MSBN and Ontologies in Distributed Network Diagnosis", In proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2011), pp.159-162.

- (Chen et al., 2010) Chen, Y., Wang, B., Lin, W.S., Wu, Y., and Liu, K.J.R. (2010) "Evolutionary games for cooperative P2P video streaming", In proceedings of the 17th IEEE International Conference on Image Processing (ICIP 2010), pp. 4453-4456.
- (Dance et al., 2003) Dance, S., Gorman, M., Padgham, L., Winikoff, M. (2003) "A deployed multi agent system for meteorological alerts". In proceedings of Deployed Applications of Autonomous Agents and Multiagent Systems Workshop, held in AAMAS'03, Melbourne, Australia.
- (Dastani, 2008) Dastani, M. (2008) "2APL: a practical agent programming language", Autonomous Agents and Multi-Agent Systems, Vol. 16, Issue 3, pp. 214-248.
- (Fletcher et al., 2003) Fletcher, M., Lucas, A., Jarvis, D., Jarvis, J., Ronnquist, R. R., McFarlane, D.C., Brusey, J., and Thorne, A. (2003) "JACK-based holonic control of a gift box packing cell", Technical Report, Distributed Information and Automation Laboratory, Cambridge, UK.
- (Gascuena and Fernandez-Caballero, 2011) Gascuena, J.M. and Fernandez-Caballero, A. (2011) "Agent-oriented modeling and development of a person-following mobile robot", Expert Systems with Applications, Vol. 38, Issue 4, pp. 4280-4290.
- (Gascuena et al., 2011) Gascuena, J.M., Fernandez-Caballero, A., Lopez, M.T., and Delgado, A.E. (2011) "Knowledge modeling through computational agents: application to surveillance systems", Expert Systems, Vol. 28, Issue 4, pp. 306-323.
- (Georgeff and Ingrand, 1989) Georgeff, M.P. and Ingrand, F.F. (1989) "Monitoring and control of spacecraft systems using procedural reasoning", In proceedings of the Space Operations Automation and Robotics Workshop.
- (Gomez-Romero et al., 2011) Gomez-Romero, J., Patricio, M.A., Garcia, J., and Molina, J.M. (2011) "Communication in distributed tracking systems: an ontology-based approach to improve cooperation", Expert Systems, Vol. 28, Issue 4, pp. 288-305.
- (Hei et al., 2007) Hei, X., Liu, Y., and Ross, K.W. (2007) "Inferring Network-Wide Quality in P2P Live Streaming Systems", IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 9, pp. 1640-1654.
- (Hindriks, 2009) Hindriks, K. (2009) "Programming rational agents in goal", In Bordini et al. (Eds): Multi-Agent Programming: Languages, Tools and Applications. Springer, pp. 119-157.
- (Kardas et al., 2012) Kardas, G., Challenger, M., Yildirim, S., and Yamuc, A. (2012) "Design and implementation of a multiagent stock trading system", Software: Practice and Experience, Vol. 42, Issue 10, pp. 1247-1273.
- (Le Blond et al., 2012) Le Blond, S., Le Fessant, F., and Le Merrer, E. (2012) "Choosing partners based on availability in P2P networks", ACM Transactions on Autonomous and Adaptive Systems, Vol. 7, Issue 2, Article 25, pp. 1-14.

- (Li et al., 2009) Li, Y.-T.H., Ren, D., Chan, S.-H.G., and Begen, A.C. (2009) "Low-delay mesh with peer churns for peer-to-peer streaming", In proceedings of the International Conference on Multimedia and Expo (ICME 2009), NJ, USA, IEEE Press, pp. 1546-1547.
- (Liu et al., 2010) Liu, Z., Wu, C., Li, B., and Zhao, S. (2010) "UUSee: large-scale operational on-demand streaming with random network coding", In proceedings of the 29th Conference on Information communications (INFOCOM 2010), NJ, USA, IEEE Press, pp. 2070-2078.
- (Lucas et al., 2003) Lucas, A., Ronnquist, R., Ljungberg, M., Howden, N., Corke, P., and Sikka, P. (2003) "Teamed UAVs - a new approach with intelligent agents", In proceedings of the 2nd conference on AIAA Unmanned Unlimited Conference, San Diego, CA, USA.
- (Mathieson et al., 2004) Mathieson, I., Dance, S., Padgham, L., Gorman, M., and Winikoff, M. (2004) "An open meteorological alerting system: issues and solutions", In proceedings of the 27th Australasian Computer Science Conference, Dunedin, New Zealand, pp. 351-358.
- (Menkovski and Liotta, 2013) Menkovski, V., and Liotta, A. (2013) "Intelligent control for adaptive video streaming", In proceedings of IEEE International Conference on Consumer Electronics (ICCE 2013), Las Vegas, Nevada, USA, IEEE Press, pp. 127-128.
- (Molina et al., 2009) Molina, B., Pileggi, S.F., Esteve, M., and Palau, C.E. (2009) "A negotiation framework for content distribution in mobile transient networks", *Journal of Network and Computer Applications*, Vol. 32, Issue 5, pp. 1000-1011.
- (Munroe et al., 2006) Munroe, S., Miller, T., Belecheanu, R.A., Pechoucek, M., McBurney, P., and Luck, M. (2006) "Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents", *The Knowledge Engineering Review*, Vol. 21, Issue 4, pp. 345-392.
- (Murray et al., 1995) Murray, G., Steuart, S., Appl, D., McIlroy, D., Heinze, C., Cross, M., Chandran, A., Raszka, R., Rao, A.S., Pegler, A., Morley, D., and Busetta, P. (1995) "The challenge of whole air mission modeling", In proceedings of the Australian Joint Conference on Artificial Intelligence (AI '95), Melbourne, Australia.
- (Nordbo, 2011) Nordbo, E. (2011) "Inter-Agent Communication in Multi-Agent Systems", Master's Thesis, Norwegian University of Science and Technology.
- (Orynczak and Kotulski, 2011) Orynczak, G. and Kotulski, Z. (2011) "Agent based infrastructure for real-time applications", *Annales UMCS, Informatica*, Vol. 11, Issue 4, pp. 33-47.
- (Padgham and Winikoff, 2004) Padgham, L. and Winikoff, M. (2004) "Developing Intelligent Agent Systems - A practical guide", John Wiley & Sons.
- (Pokahr et al., 2005) Pokahr, A., Braubach, L., and Lamersdorf, W. (2005) "Jadex: A BDI Reasoning Engine", In Bordini et al. (Eds): *Multi-Agent Programming Languages, Platforms and Applications*, Springer, pp. 149-174.

- (Pokahr et al., 2007) Pokahr, A., Braubach, L., Walczak, A., and Lamersdorf, W. (2007) "Jadex - Engineering Goal-Oriented Agents", In Bellifemine et al. (Eds): *Developing Multi-Agent Systems with JADE*, Wiley Publishing, pp. 254-258.
- (Pournaras et al., 2009) Pournaras, E., Warnier, M., and Brazier, F. (2009) "Adaptive Agent-Based Self-Organization for Robust Hierarchical Topologies", In proceedings of the 2009 International Conference on Adaptive and Intelligent Systems (ICAIS 2009), Washington, DC, USA, IEEE Computer Society, pp. 69-76.
- (PPLive, 2012) PPLive (2012), available at: <http://www.pplive.com> (last access: January 2014).
- (PPStream, 2012) PPStream (2012), available at: <http://www.ppstream.com> (last access: January 2014).
- (Rao and Georgeff, 1995) Rao, A. and Georgeff, M. (1995) "BDI Agents: From Theory to Practice", In proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, pp. 312-319.
- (Sardina and Padgham, 2011) Sardina, S. and Padgham, L. (2011) "A BDI agent programming language with failure recovery, declarative goals, and planning", *Autonomous Agents and Multi-Agent Systems*, Vol. 23, Issue 1, pp. 18-70.
- (Sayit et al., 2012) Sayit, M.F., Tunali, E.T., and Tekalp, A.M. (2012) "Resilient peer-to-peer streaming of scalable video over hierarchical multicast trees with backup parent pools", *Signal Processing: Image Communication*, Vol. 27, Issue 2, pp. 113-125.
- (Sokolova and Fernandez-Caballero, 2009) Sokolova, M.V. and Fernandez-Caballero, A. (2009) "Modeling and implementing an agent-based environmental health impact decision support system", *Expert Systems with Applications*, Vol. 36, Issue 2, pp. 2603-2614.
- (Teket et al., 2014) Teket, K.D., Sayit, M. and Kardas, G. (2014) "Software agents for peer-to-peer video streaming", *IET Software*, DOI: 10.1049/iet-sen.2013.0181.
- (Wang et al., 2010a) Wang F., Xiong Y., and Liu J. (2010) "mTreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, Issue 3, pp. 379-392.
- (Wang et al., 2010b) Wang, M., Xu, L., Ramamurthy, B. (2010) "Comparing multi-channel Peer-to-Peer video streaming system designs", In proceedings of the 17th IEEE Workshop on Local and Metropolitan Area Networks, NE, USA, pp. 1-6.
- (Winikoff, 2005) Winikoff, M. (2005) "JACK Intelligent Agents: An Industrial Strength Platform", In Bordini et al. (Eds): *Multi-Agent Programming Languages, Platforms and Applications*, Springer, pp. 175-193.
- (Wooldridge, 2002) Wooldridge, M. (2002) "An Introduction to Multi-agent Systems", John Wiley & Sons.

(Xie et al., 2007) Xie, S., Li, B., Keung, G.Y., and Zhang, X. (2007) "Coolstreaming: Design, Theory, and Practice", IEEE Transactions on Multimedia, Vol. 9, Issue 8, pp. 1661-1671.

(Yu et al., 2006) Yu H., Zheng D., Zhao B.Y., and Zheng W. (2006) "Understanding user behavior in large-scale video-on demand systems", ACM SIGOPS Operating Systems Review, Vol. 40, Issue 4, pp. 333–344.

(Zhang et al., 2005) Zhang, X., Liu, J., Li, B., and Yum, Y.-S.P. (2005) "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming", In proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005), Miami, FL, USA, pp. 2102-2111.

The Authors

Suleyman Yildirim received his B.Sc. degree in Applied Mathematics in 2008 and M.Sc. degree in Information Technologies in 2012 both from Ege University, Turkey. His main research area covers design and implementation of autonomous agent systems for various application domains such as intelligent stock trading and effective live video streaming. He is currently with Stan Ackermans Institute, Eindhoven University of Technology, the Netherlands and pursuing a PDEng degree in Software Technology.

Muge Sayit received Ph.D. and M.Sc. degrees in Information Technologies from International Computer Institute, Ege University Turkey, in 2011 and 2005, respectively and received her B.Sc. degree in Mathematics from Ege University in 1999. She has been working as an assistant professor at International Computer Institute, Turkey for two years. Her research interests include P2P networks, live streaming and video-on-demand.

Geylani Kardas received his B.Sc. degree in Computer Engineering from Ege University, Turkey, in 2001, and received his M.Sc. and Ph.D. degrees in Information Technologies from International Computer Institute, Ege University, Turkey, in 2003 and 2008 respectively. Since 2009, he is an assistant professor with International Computer Institute, Turkey. His main research interests cover autonomous and intelligent agent systems, agent-oriented software engineering and model driven development of software systems.