A Client-Server System for Ubiquitous Video Service

[1]Ronit Nossenson, [2]Orit Yudilevich, [2]Omer Marlowitz
[1]*Faculty of Computer Science,* Jerusalem College of Technology, Jerusalem, Israel
[2]*School of Computer Science,* The Interdisciplinary Center, Jerusalem, Israel

***Abstract* — In this work we introduce a simple client-server system architecture and algorithms for ubiquitous live video and VOD service support. The main features of the system are: efficient usage of network resources, emphasis on user personalization, and ease of implementation. The system supports many continuous service requirements such as QoS provision, user mobility between networks and between different communication devices, and simultaneous usage of a device by a number of users.**

***Keywords* — Seamless Content Delivery, Ubiquitous Multimedia Service, Personal Multimedia Delivery, Live Video Transmission, VOD Transmission.**

I. Introduction

A new generation of distributed services ranging from entertainment services such as live video streaming, video on demand and on-line gaming, to life- saving applications such as medical services and monitoring, are being deployed in heterogeneous and ubiquitous environments. To be accepted by both users and network operators, these ubiquitous services must deliver continuous service, as well as adaptive and satisfactory Quality-of-Service with a minimum overhead of network resources. Providing ubiquitous services entails a number of complex issues, such as supporting the required QoS during a session, seamless handovers between different radio access technologies (RATs), supporting user mobility, etc.

In this article we introduce a simple client-server architecture and algorithms for live video and Video On Demand (VOD) ubiquitous services. To achieve satisfactory continuous service with minimum overhead, collaboration and coordination between small number of agents uses several communication methods including wireless or cellular connections. This article is an extended version of our previous results [13].

The main features of the architecture are as follows:

1. Efficient usage of network resources complying with the required/preferred QoS.
2. User- driven architecture which enables easy personalization.
3. Ease of implementation.

Previous work on ubiquitous multimedia services has focused on middleware solutions (see, for example, [1-7, 9, 10]). These ideas are good and effective but they require the cooperation of network operators. Since the conventional business model is defined only between the content provider and its content consumers, the readiness of operators to deploy such solutions is limited.

As illustrated in Fig. 1, our novel architecture relies on both the user's communication devices (for example, smart phone, PDA or laptop) and the continuous service/content provider system; no changes or extensions are needed in the operator network. The system can completely handle a range of continuous service requirements: QoS provision, user mobility between RATs and between different communication devices, simple user interface and personalization, simultaneous usage of the same device by a number of users (while protecting privacy) and so on. The description focuses on ubiquitous live video and VOD services, but the system can be easily extended to other services as well.

The architecture is based on the “best k” algorithm to ensure efficient use of network resources [8]. This algorithm provides high quality live-video transmission by using few agents and proposes ways to minimize the usage of network resources. Experimental results show that by using the best-k algorithm, high quality video can be delivered with an overhead factor of 1.65%.

This paper is organized as follows. In the next section the

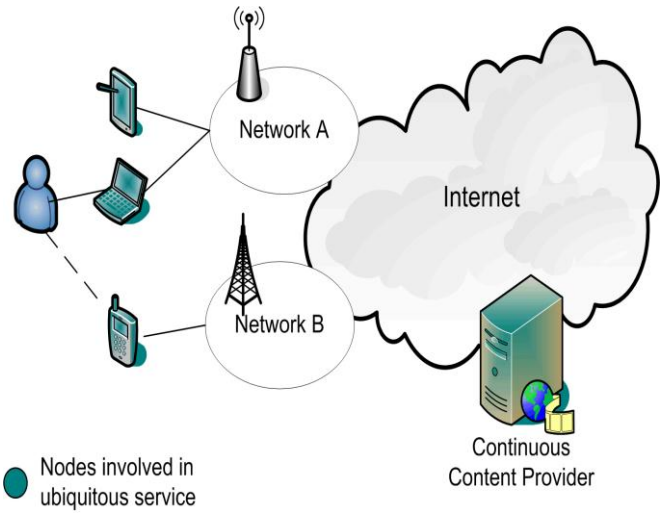


Fig. 1. High level description of a client-server system for ubiquitous service. The subscriber can receive a service via laptop, smart phone, etc.

DOI: 10.9781/ijimai.2012.174

ubiquitous system requirements are outlined. In section III, we describe the specifications of the new system by topic. These include the system building blocks, state machines and essential procedures. Next, some experimental results are described in Section IV. Finally, we suggest future research directions.

II. System Requirements

Providing ubiquitous service raises a number of issues that must be addressed. In this section we specify the technical requirements for a ubiquitous service system. First we define the terms used in this work. Next, we list the requirements according to their functionality: general, usability, QoS, multiple users' and privacy.

A. Definitions

Content provider functions as the server side. Its responsibilities include user management, QoS management and content provision. *Service* is a video service such as live video or VOD. *User* is a person who subscribes to the service. *Client* is the software which provides a special service on a user device. A user has one client on each device, per service. For example, a user who subscribes to 3 services via 4 devices has 12 clients. The client is responsible for communication between the user and the system, measuring and reporting QoS, agent management and combining the received data when necessary.

Agent is the software that is responsible for receiving and transmitting data for the service via a specific communication interface/technology. For example, a device with cellular, WiFi and BT connections has one client and up to three agents per service and user (see Fig. 2). Two or more users can use the same service via one device simultaneously (watching a movie together, for example). In this case the first user who activated the service is the *primary* user and the other users are referred to as *secondary* users. The primary user and secondary users together are referred to as the *service users' group on a device*. The primary user manages the service users' group that is using the service.

B. General Requirements

G1. The system supports ubiquitous service for live video transmission and video on demand (VOD) applications.

G2. The system attempts to provide QoS as close as possible to the preferred quality (see requirement Q1 below), with minimal user intervention and with minimal overhead for network resources.

G3. Scalability requirements: The system supports up to A potential agents per user. The system supports up to B activations of service per minute. The system supports up to C simultaneous active services. A user may have up to D active services and up to E paused services simultaneously. The parameters A, B, C, D and E can be extended by simple hardware extension.

C. Electronic Image Files (Optional)

U1. Service is supplied via one client at a specific time.

U2. The system provides a convenient user interface.

U3. The user needs to configure a set of agents for each service for each client (device).

U4. For a service, at least one device (client) and one agent must be registered.

U5. The user can alter its set of agents at any time using a convenient interface.

U6. For each service specified in requirement G1 the system supports the following operations: Start, Stop, Pause, and Resume.

U7. The "Start" operation is used for service activation the first time as well as for service re-activation after a "Stop" action. It can be used for inactive service only.

U8. The operation "Stop" is used for termination of the active or paused service.

U9. The operation "Pause" is used for temporary halts of an active service, up to a predefined timeout. The timeout can be interrupted by user operation ("Resume"). Otherwise the service is terminated.

U10. The "Resume" operation is used to continue the service after the "Pause" operation, depending on the type of service, assuming that the timeout has not expired.

U11. The outcome of the "Resume" operation for VOD

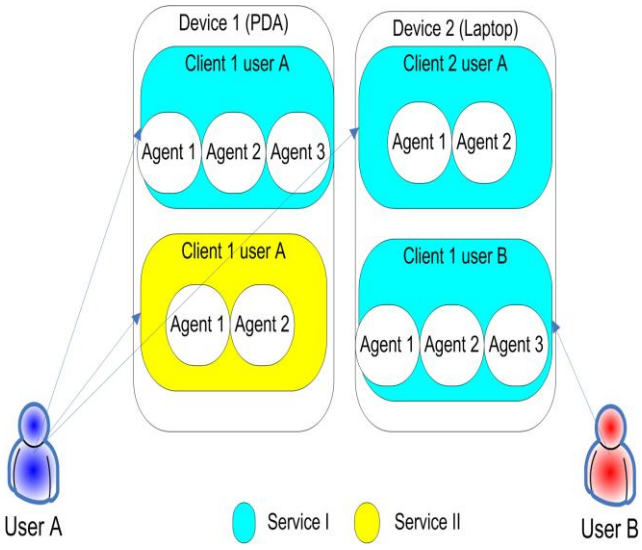


Fig. 2. User-Device-Client-Agent-Relationship. A user can have several devices and can be subscribed to several services. For each (service, user, device) triple there is one client. Each client can manage several agents. Two users can share a device, for the same service or for different services.

application is continuation of the video transmission from the point where it was paused. For the Live Video application it corresponds to resumption of the on-line video at the current time.

U12. For an active service, at least one agent is active.

U13. Changes of agents, without a change in client (switching, adding and subtract agents) are done transparently without the user's intervention.

U14. A paused service can be resumed from any subscribed client (device continuously).

U15. There can be only one active service per device.

U16. A user can have several active services, on several devices.

U17. New service activation on a device that already has another service active on it, is subject to the approval of the primary user of the active service and is equivalent to pausing/ termination of the previous service.

D. QoS requirements

Q1. The user can define *Preferred QoS parameters* and *Required QoS parameters* per service. In addition, the system has default values for these parameters per service.

Q2. The system aims to provide the user with QoS as close as possible to the preferred QoS, with minimal user intervention. If this is impossible the system aims to provide QoS above the required QoS. If the QoS falls below the required level the user is informed and the service is terminated.

Q3. The system provides continuous service for the user as long as there is at least one active client and active agent capable of providing QoS above the required level.

Q4. The QoS parameters are defined for each service separately, including the following characteristics as a minimum: bandwidth, delay and jitter.

Q5. When the QoS parameters are higher than the required level, but below the preferred thresholds, the system attempts to improve the service in the following way: (1) if the preferred QoS can be achieved using the current client (by changing agent/s), the change is performed transparently; (2) if the preferred QoS can be achieved only by another client (device), the transition can be performed, subject to the user's approval; (3) if the preferred QoS cannot be achieved using any other client (device), the system provides the best QoS possible via the current client (device).

Q6. If the QoS is below the required threshold, the system tries to improve the QoS via the current client. If this is impossible, the user is advised to move to another device. If the required QoS cannot be met the service is terminated.

Q7. For each user and active service, the system maintains a set of potential agents as a function of QoS parameters, environmental changes, user preferences, and agent availability.

E. Multiple User Requirements

M1. An active service has one primary user.

M2. There can be several secondary users for an active service.

M3. A user can join a service on a specific device which is managed by a different primary user, subject to both users' approval.

M4. A secondary user can disjoin a service; this action is equivalent to pausing or stopping the service to the specific user.

M5. A primary user can be replaced by another user, subject to both users' approval. The previous primary user becomes a secondary user in this case.

F. Privacy Requirement

P1. A user cannot access information on services that another user is subscribed to, even if they co-exist on the same device (see Fig. 2), unless the user is a primary user who is aware of the secondary users of the same service.

III. System Specifications

In this section we elaborate on the technical problems and provide specifications and algorithms for the system. The listed specifications provide a feasible solution for the pre-defined requirements. This section is divided into the following sub-sections: service state machine and QoS specification, user interface specification, agent state machine, multi user specification and user mobility issues are discussed in the final sub section.

A. Service State Machine and QoS Specifications

According to the requirements, each service can be in one of three states per user: Not Active, Active (A, B, C and D sub-states) or Paused. Fig. 3 depicts the transitions between the states, showing all the valid transitions, their triggers and actions.

1) From *"Not Active Service" to "Active Service"*: The actions that take place in this case are: (i) the system establishes a connection with the agent that sent the "Start" command before it starts to transmit the content; (ii) A handshake procedure is performed with each user's agents, and a list of available agents is generated. The handshake procedure and the management of the available agent list are described in the agent state machine below.

2) From *"Active Service" to "Not Active Service"*: This transition can occur in two cases: upon a user "Stop" command or if the QoS falls below the required level (see requirements Q2, Q6). In these cases the data flow to/from the client is terminated, a termination message is sent to all

available agents, so they can move to the "not active" state, all session data are dropped on both server and client sides.

3) From *"Paused Service" to "Not Active Service"*: A service can go from "Paused" to "Not Active" in two cases: by a user "Stop" command or by a pause timeout expiration, see requirement U9. In these cases, the system sends a termination message to all clients to turn the available agents to "not active". Service- session related information is dropped from both the server and client sides.

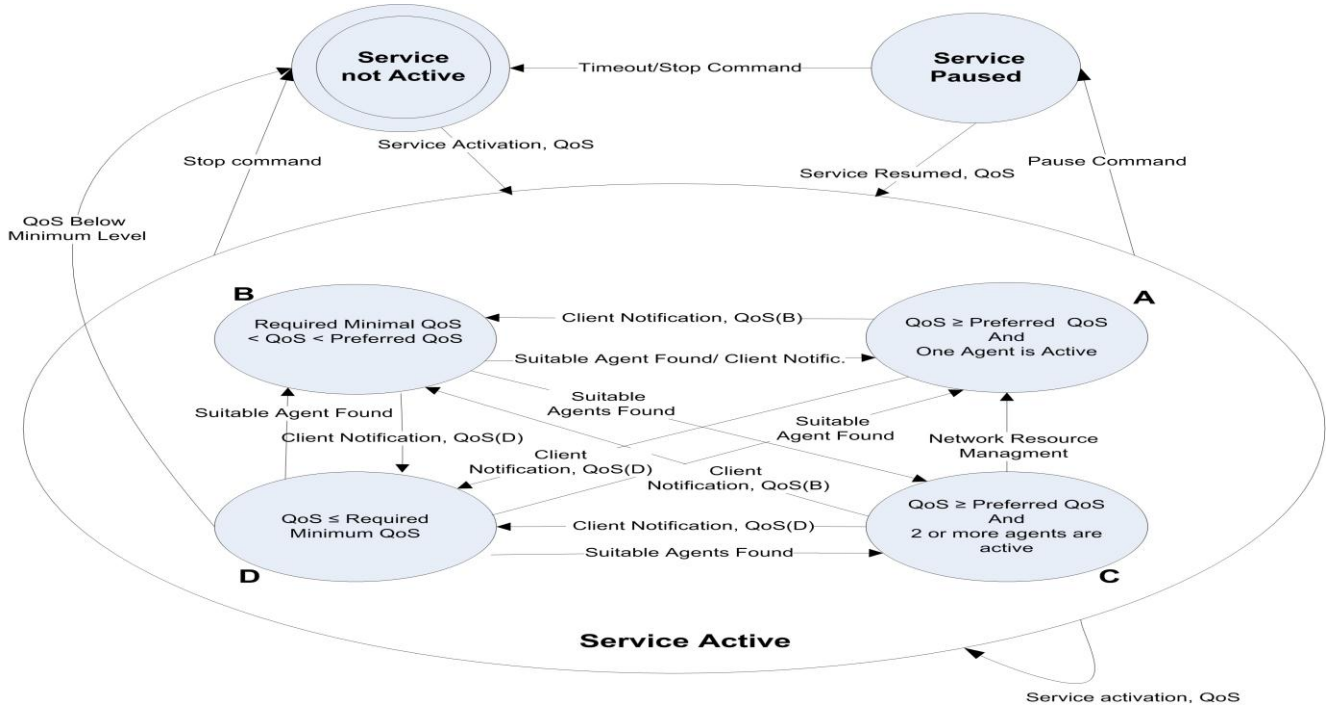


Fig. 3. Service State Machine and QoS management state machine for active service.

4) From *"Active Service" to "Paused Service"*: The user can implement this transition in several ways: by sending pause command, activation of another service on the same device, approval of client swapping proposed by the system and switching from primary user to secondary user in the multiple user service mode (see requirements U9, U17, M3). In these cases all the service session data are saved in both server and client. Service paused messages are sent to the active agents, so they can go into the "available" state. Data transmission is stopped. On the server side, additional data are stored, such as last active client, pointer to the last transmitted I frame, last packet sequence number, file offset (the location in the movie for VOD service), and last decoding format in use. In addition, the timer for maximum paused time is activated.

5) From *"Paused Service" to "Active Service"*: In case the service is resumed on the same client it was paused on, all the data exist on both the client and the server sides and the service is simply resumed. Otherwise, if the service is resumed on a different client (device) a format adaptation is performed if needed. The last I frame is sent to the client together with the following P frames and a specific notification for the video player film offset. This enables VOD service to resume from the same point it was paused on the previous device. For live video service, the service is resumed according to the current time.

6) From *"Active Service" to "Active Service"*: If the "resume" command is initiated on the active client, the command is discarded, otherwise the command is equivalent to pausing the service on a current client and resuming it on a new one.

Other state transitions are server internal and are related to QoS provision. Specifically, it meets requirements G2, U12, U13, Q2, Q3, Q5, Q6 and Q7 above.

The essential server functions are:

- Ensuring an acceptable QoS level via agent management.
- Providing maximum transparency to the user.
- Ensuring efficient usage and minimum overhead of the network resources.

The essential client functions are:

- Providing the user interface to the system.
- Monitoring the QoS parameters for active services and informing the server if needed.

- Combining and synchronizing the data, using [8] or a similar algorithm.

The following information is stored on the server per active user of a specific service:

- The current QoS parameters.
- The current active client and its set of active agents.
- A list of available clients with their available agents.
- Session associated information.
- List of clients and agents used in the last time interval.
- List of forbidden client transitions.

During active service, the server runs the state machine as presented in the "Active Service State" in Fig. 3. In sub-state A the user receives its preferred QoS by one agent; thus the system does not have to improve its QoS or to reduce network resource overhead associated with it.

Sub-state B is characterized by the QoS between the preferred and required levels, thus the service can be continued along with system efforts to improve the QoS according to requirements G2 and Q5. The requirements define the following priorities: QoS, minimum user intervention and network resources (see requirements G2, U13, Q5, Q6); hence, in state B, the following procedure is performed periodically:

```
StateBProc(PreferredQoS, CurrentClient,
      AllAvaliableClients):
Begin
  Bool IsPreferredQoSPossible = false;
  Bool IsPreferredQoSReached = false;
  IsPreferredQoSPossible =
    BestQoS(PreferredQoS, CurrentDevice,
    AllAvaliableClients);
  If (IsPreferredQoSPossible)Then
    Bool IsPreferredQoSReached =
    SwapAgentbyQoS (PreferredQoS, QoSList);
  End
  If (Not IsPreferredQoSReached)Then
       ImproveCurrentClientQoS(CurrentQoS,
             CurrentDevice);
  End
End
```

The function `BestQoS` is described below, it is responsible for generating the list of respective agents and possible QoS by staging a competition between the agents (for details regarding agent competition see [8]). The QoS list includes: (i) Best QoS that can be reached by one agent of the current client (device) (`BestQoSSingleAgentCurrentClient`) and corresponding agent; (ii) Best QoS that can be reached by two or more agents on the current client (device) (`BestQoSMultiAgentCurrentClient`) and corresponding agents; (iii) Similar data about other available clients and agents (iv) Best possible QoS for each client achieved by one or more agents. The inputs to the function are: requested QoS threshold, current device, full list of

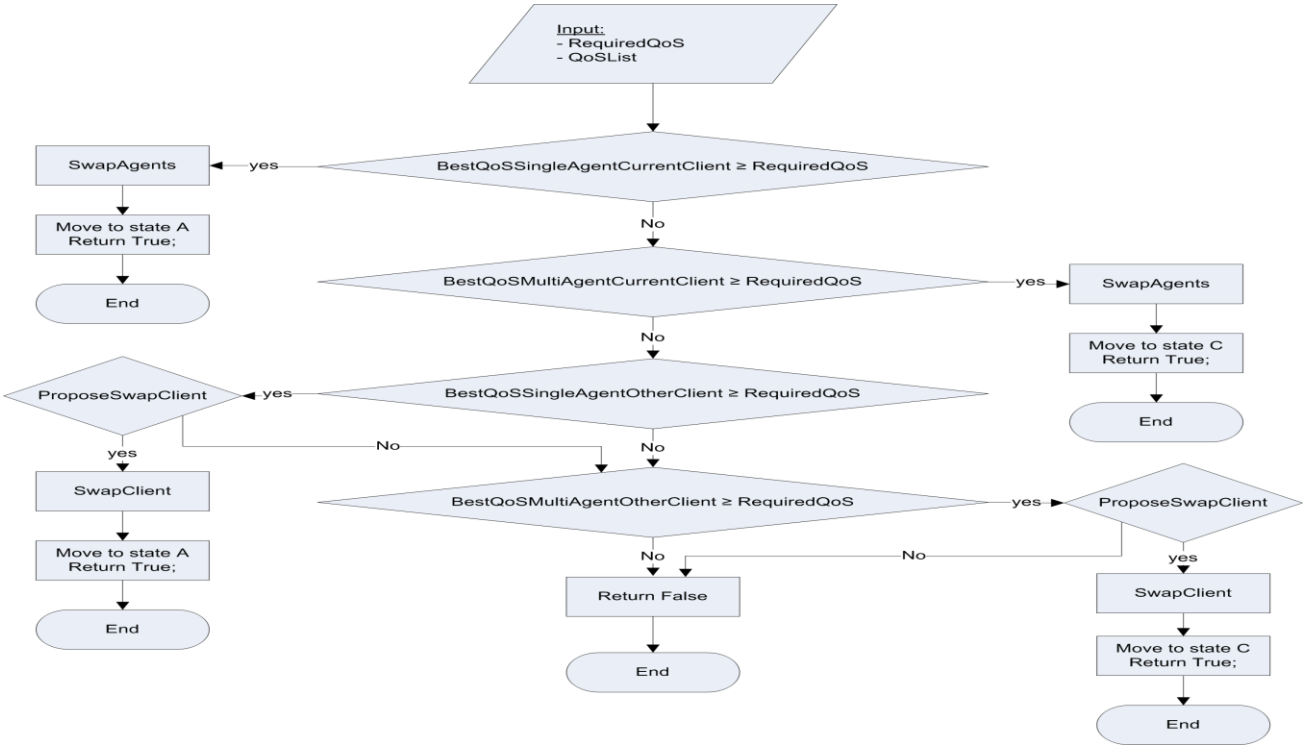


Fig. 4. Swap agent by QoS procedure flow chart

devices and agents.

The pseudo code of the function `BestQoS` is as follows:

```
BestQoS(QoSThreshold,CurrentClient,
                  AllAvaliableClients):
Begin
Competition(Current Client's Agents);
Set BestQoSSingleAgentCurrentClient, Agent;
Set BestQoSMultiAgentCurrentClient, Agents;
If((BestQoSSingleAgentCurrentClient ≥
  QoSThreshold)Or
  (BestQoSMultiAgentCurrentClient ≥
  QoSThreshold)) Then return true;
/* Needed QoS can't be reached by current
client, rest of the clients are checked */
Competition(Available Clients);
Set BestQoSSingleAgentOtherClient, AgentList;
Set BestQoSMultiAgentOtherClient,
AgentMatrix;
maxQoS = max(BestQoSSingleAgentOtherClient,
BestQoSMultiAgentOtherClient);
return (maxQos  ≥ QoSThreshold)
End
```

When agent/s that supply the required QoS is/are found, the system attempts to swap to this/these agent/s. This is performed by the `SwapAgentbyQoS()` procedure. The flow chart of this procedure is presented in Fig. 4. This procedure scans the possible agents according to a pre-defined order (requirements G2, U13, Q5, Q6). The first choice is one agent on the current client, then, multiple agents on the current clients, finally single and multiple agents of other clients. This procedure updates the QoS state machine, as required.

The system should introduce the user to the full list of other clients that can provide the preferred QoS. Client switching is always subject to user approval. In addition, the system should maintain a list of clients whose transitions are forbidden (see user interface specification for details), to avoid undesirable proposals to the user (requirement G2, Q5).

If the preferred QoS cannot be reached the system should try to improve the QoS on the current client (requirement Q5), to the best possible QoS. This is performed by the `ImproveCurrentClientQoS()` procedure. It uses the current QoS and current client as inputs. This procedure does not affect the QoS state machine, since after its termination the QoS level is still between the preferred and the required thresholds. The pseudo code of this procedure is as follows:

```
  ImproveCurrentClientQoS(CurrentQoS,
     CurrentClient)
  Begin
  MaxQoS
=max(BestQoSSingleAgentCurrentClient,

BestQoSMultiAgentCurrentClient);
   If  ((MaxQoS > CurrentQoS) And
  (MaxQoS==BestQoSSingleAgentCurrentClient))
  Then
    SwapAgents(CurrentAgents, NewAgent);
  Else if (MaxQoS > CurrentQoS)
  Then
    SwapAgents(CurrentAgents, NewAgents[]);
  End
```

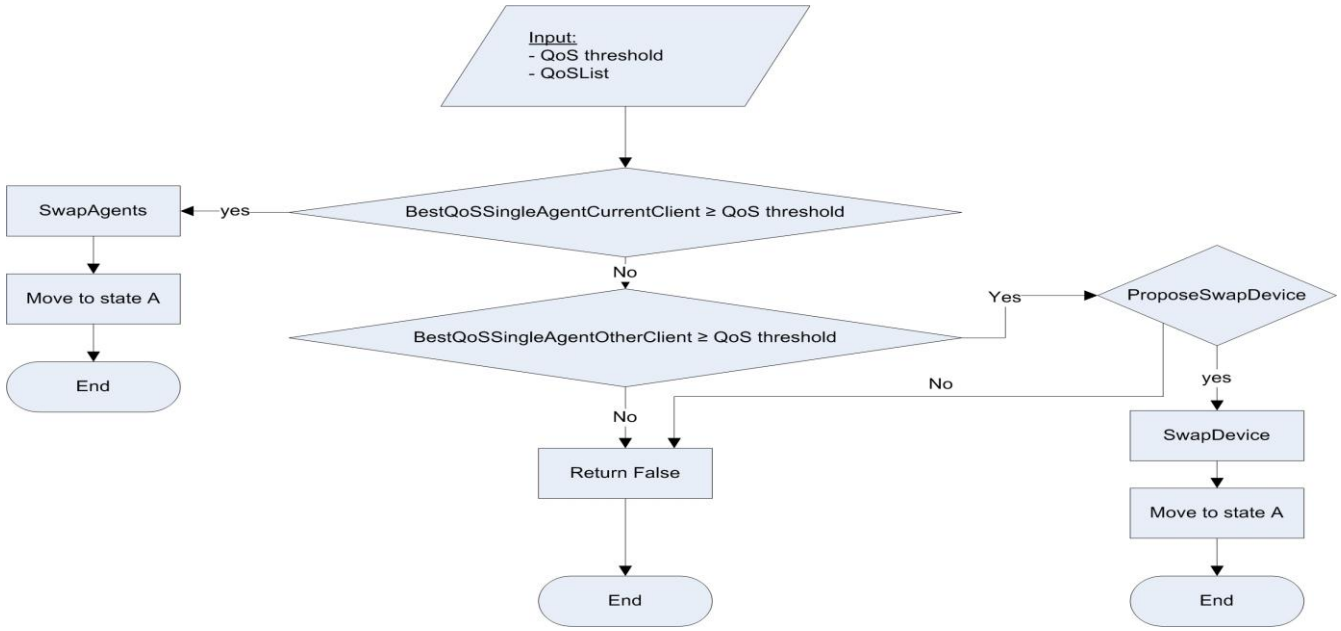


Fig. 5. Swap agent by overhead procedure Flow Chart

In sub-state C the QoS is equal or higher than the preferred QoS, but is supplied by more than one agent. In this case the system should try to reduce network resource overhead, by attempting to supply the preferred QoS using one agent if possible (requirement G2). The system should try to reduce the number of active agents. This is done by the `SwapAgentbyOverhead()` procedure that runs periodically. The flow chart for this procedure is presented in Fig. 5. The procedure updates the QoS state machine accordingly.

In sub-state D the QoS is below the required threshold. In this case the system attempts to improve the QoS; if this attempt fails the service is terminated (see Q2 and Q6.). The pseudo code of this procedure is as follows:

```
  StateDProc():
  Begin
  Bool IsPreferredQoS, IsRequiredQoS = false;
  IsPreferredQoS=SwapAgentbyQoS(PreferredQos)
;
  If (PreferredQos) Then
      Move to State A;
      Return;
  IsRequiredQoS                              =
SwapAgentbyQoS(RequiredQos);
  If (!IsRequiredQoS) Then
      Notify user of service termination;
      Exit Active Service State Machine;
      Service state = Not Active;
      Terminate service;
  Else
    ImproveQoSCurrentClient;
    Move to State B;
  End
```

B. Agent State Machine

In this sub-section we introduce the agent state machine (Fig. 6). As mentioned earlier, for active service, one or more agents can be used for service supply (they are in the "active" state), while the other agents for this service are in the "available" state. Upon service activation the system sends activation messages to all user clients (for the current service). The clients instruct their agents to move to the "*available state*". The clients and agents must respond to the activation message. This process is referred to as the "handshake procedure". The outcome of this "handshake procedure" is a list of all available agents for the current service. In order to keep the available agents list updated, all agents must send (by client) keep-alive messages. When a client recognizes that agent/s becomes available after unavailability (for example device turn on) a notification is sent to the system. If the service is active, an activation command to the agent/s is sent. Similarly, when agents become unavailable for active or paused service, they should inform the system if possible. Once an agent does not send a keep-alive message for a specific period of time it is removed from the available agents list ("*not active*" state). Agent that has completed the handshake procedure is regarded as in *available* or *active* state until service termination.

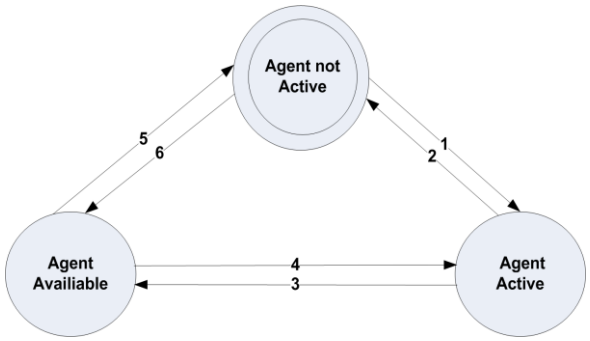


Fig. 6. Agent State Machine

Below are the detailed agent state transitions of Fig. 6.

1) From "*Not Active" to "Active"*. This transition occurs only upon service activation on the specific device.

2) From *"Active" to "Not Active"*. This transition takes place when the agent was one of the agents that provided the service and the service is terminated due to a user's command or due poor QoS (see sub-section QoS specification above).

3) From *"Active" to "Available"*, this transition occurs in the case of agents/client switching for QoS reasons (see QoS specification sub-section above) or a service state change from active to paused (see service state machine sub-section above).

4) From *"Available" to "Active"*, this can occur if the agent is selected by the system for service transmission for QoS reasons or when the service state moves from "Paused" to "Active" by command from this agent.

5) From *"Available" to "Not Active"*, this transition takes place for available agents of a service when the service state is changed to "Not Active" (from "Active" or "Paused" states).

6) From *"Not Active" to "Available"*, this transition takes place for all agents upon service activation command.

In "Not Active" state the agents have no connection to the system. In the "Available" state the agents are connected to the system and keep-alive messages are exchanged as described above. An agent is "Active" if it is one of the agents that supply the service.

C. User Interface Specifications

User interface has to meet requirements G3, U3, U4, U5, U6, Q1 and P1. In order to meet privacy requirements (P1) the interface to a service should be a remote web page and it should be password protected, for example.

The actions that can be performed by the user interface are:

1) *Join the service:* First, the user has to register and to accept a username and password (requirement P1) to access the system. The next step is setting the preferred and required QoS (requirement Q1). The user can choose the default values. The use of these values is described in the QoS specification sub-section above. Then, the user should register the devices; this process is described below.

2) *Client and agent management:* Client and agent management is divided into three sub-processes: client/agent addition, client/agent subtraction and client transition management. As a guideline the user does not have to address agents. The user chooses to alter the client setting; the system requests the data (list of connection interfaces) from the client and presents the options to the user. These procedures can be implemented at any time according to requirement U5.

(i) *Client/Agent addition*: When subscribing a device or agent to a service, a user can choose to connect from the current device ("Add this device") or to identify the subject device by a unique identification, an IP address for instance. After the device is chosen the system queries the device about possible agents (connection interfaces). All possible agents are presented to the user and the user chooses which agents to install. In case of agent addition the user should choose "modify client settings" and the system will return all the options. After the user confirms the choice the profile at the client side is updated. If the number of potential agents exceeds the A parameter from requirement G3, the system displays the message to the user and client/agent addition is aborted.

(ii) *Client/Agent subtraction*: The procedure can be performed from any device. In agent subtraction, as mentioned above, user chooses "modify client settings" and the system shows all agents. In the case of client subtraction the system presents the user list of all of its clients (devices) and the user chooses the client/s to subtract. The system does not allow subtraction of the last client/agent (requirement U4). After the user confirms the choice the profile at both server and client sides is updated.

3) *Client transition management:* In the QoS specifications sub-section, it was noted that in case of insufficient QoS or inefficient use of network resources the system can suggest switching clients to the user. As mentioned earlier the system should try to minimize the use of this option, to meet the requirement for minimal user intervention (G2). For this reason there should be a minimum period of G1 minutes between two successive proposals to switch devices. Additionally, the system should store recently (for G2 minutes) used agents and clients, in order to avoid "ping-pong" transitions between clients. For the user's convenience the system should avoid proposing invalid transitions. For this reason the system stores a list of forbidden client transitions, per user. The user should manage this list. This list can be updated in two ways: by the user interface or when the system displays the list of optional clients for transition, the user can assign "Do not propose this transition again" to one or more clients. The system also should not propose a switch to a device in active service (of any user) to comply with requirement U15. In summary, client transition can be proposed to a user if: this is the first proposal for G1 minutes and there exists a potential device for transition that has not been used in the last G2 by this user and the service, does not appear on the forbidden transitions list, does not run an ubiquitous service to any user and meets the QoS threshold. Upon the user's approval of the transition client, a swapping procedure is performed. Client swapping is similar to pausing and resuming the service on a new client; the only difference is that the "pausing" is triggered by the user's approval of the client switching and not by the pause command. Pausing and resuming service is described in the service state machine sub-section above. Further work could be done to elaborate the list of proposed clients for transition, by studying users' preferred transitions, for example.

4) *Activation, stopping, pausing and resumption of a service:* These actions are described in the service state machine and the QoS sub-sections. For privacy (requirement P1) the activation and resumption of a service should be password protected operations.

5) *Multiple user service management:* The interface is described in the multiple user service management sub-section below.

6) *Disjoin the service:* The user should choose to unsubscribe to the service. After the user confirms the choice, all associated session information on both the client and server sides are removed.

D.Multiple User Specifications

One aspect of ubiquitous service is that several users may start to consume a service together and then want to continue to consume it separately at a different time and place. The requirements for multiple users (M1 – M5) specify this case.

In order to support the multiple user mode the system and the client need to support the following actions:

- Add user to service user group.
- Remove user from service user group.
- Primary user substitution.

All of the above only apply to active service.

1) *Add user to service user group:* To add a user to a service group, the primary user should choose the "Add user to the current service" option and specify the user. The system asks for the new user's confirmation, according to

requirement M2, by requesting a password. Once the subject user is approved to join the service, the system performs the following steps: the user is added to the service's user group, a connection is established with all the users' clients to move agents to the "available" state and to generate a list of available agents. The service state for the new user is "paused". Note that the user must be subscribed to the service, but the device can only be subscribed by the primary user.

2) *Remove user from service group:* Confirmation by the subject user is performed as above. If the primary user wants to leave the service a new primary user should be chosen according to requirement M5 (below). Then the departing user should chose to stop or to pause the service (requirement M3).

3) *Primary user substitution:* The following steps should be performed: (i) upon user request to change primary user, the system should generate a list of potential primary users. The new primary user can be a user who is in the service's user group and subscribed the current device to the service. If there is no such user, the primary user change cannot be done. (ii) The list of potential primary users is displayed to the current primary user and he/she should choose the new primary user. (iii) The last step is to stage the competition between agents of the new primary user on the current client, and to choose active agent/s for the service [8]. The previous primary user becomes the secondary user and stays in the service's user group.

E. User Mobility

User mobility is a focal issue in ubiquitous service. In this sub-section we show that the mobility issue is resolved in our system. User mobility can have two negative effects: the user needs to switch devices and/or the QoS degrades due to coverage or load issues. If the user needs to switch devices this should be done by pausing the service on the old client and resuming it on the new one. The case of QoS degradation is discussed above. Thus our specifications resolve the user mobility issue without having to take any location associated actions, by enabling a high level of customization and addressing QoS as a general issue that is not related solely to mobility.

IV. Performance Evaluation

In the performance evaluation of our method we consider the following additional competing methods: single transmission (that is, the way live video is transmitted today) and simple (not controlled or coordinated) multiple transmission of 2-5 agents. In simple multiple transmissions, the agents transmit the video streams without coordination with the server and the client joins the streams using the simple join function of the minimal arrival time. For every packet sequence number, the client considers the first instance to arrive. That is, the resulting arrival times are the minimum arrivals times of every packet.

Due to the short distance between the agents, we cannot assume that they are statistically independent. Therefore, our method for evaluating the suggested solution is by measurements of real traffic, rather than theoretical analysis. First, we transmit video using several agents in various conditions in order to collect the data. The transmission of the agents was done using LU60 of LiveU [11], using one to five cellular modems connected to three different cellular networks. Each agent has a different connection to the internet. Next, a feasible solution for splitting and joining is implemented. We record the received data with LiveU's server (LU1000) [11] and also using 'Wireshark' software [12]. We collect data which is relevant to parameters such as delay, jitter and retransmission ratio. Therefore, we record for each packet in each transmission from each agent the

TABLE III
Packet Loss Ratio

Process	Average	Worst case
1 agent	2.56%	17.0%
2 agents	0.04%	1.22%

TABLE IV
Average Percentages of Jitter Condition Violation

Cond.	Best (A)	Best (B)	Best (C)	1 agt.	2 agt.	3 agt.	4 agt.	5 agt.
>0	66%	79%	83%	80%	73%	70%	69%	69%
>1	61%	75%	79%	76%	69%	66%	64%	64%
>2	58%	71%	75%	75%	67%	63%	61%	60%
>3	55%	67%	71%	73%	65%	60%	57%	56%
>4	45%	49%	51%	69%	56%	48%	41%	37%
>5	42%	44%	46%	67%	54%	44%	37%	31%
>6	37%	37%	38%	65%	50%	39%	31%	24%
>7	35%	35%	35%	63%	48%	37%	28%	22%
>8	33%	33%	33%	60%	45%	34%	26%	20%
>9	26%	27%	28%	53%	38%	29%	22%	17%
>10	13%	14%	16%	39%	26%	19%	15%	12%
>11	7%	8%	11%	32%	20%	14%	11%	9%
>12	6%	7%	10%	30%	18%	13%	10%	8%
>13	5%	7%	9%	29%	17%	12%	9%	8%
>14	5%	7%	9%	28%	17%	12%	9%	8%
>15	5%	6%	9%	28%	16%	11%	9%	7%
>16	5%	6%	8%	27%	16%	11%	8%	7%
>17	4%	6%	8%	26%	15%	10%	8%	6%
>18	4%	5%	7%	25%	14%	10%	7%	6%
>19	4%	4%	6%	21%	12%	8%	6%	5%
>20	3%	3%	4%	16%	9%	6%	5%	4%
>25	2%	2%	2%	11%	6%	4%	3%	3%
>30	2%	1%	1%	9%	5%	4%	3%	3%
>35	2%	1%	1%	8%	4%	3%	3%	2%
>40	2%	1%	1%	8%	4%	3%	3%	2%
>45	2%	1%	1%	7%	4%	3%	3%	2%
>50	2%	1%	1%	6%	3%	2%	2%	2%

Packet Sequence Number and Time of Arrival. Finally, we evaluate the method's potential performance using the data collected at the beginning.

The recording is done throughout the day including both peak (busy hour) and off-peak hours. Each experiment consists of 5 samples of video transmissions using one to five simultaneous agents. The experiment is repeated 9 times with long video files (about 15 minutes, 30,000-65,000 packets each). In addition, the experiment is repeated twice with short video files (five minutes) to validate that the observed statistic behavior also fit short transmissions. Overall, the recording trace includes statistics of ~ 6 million real packets.

The analysis was performed three times, with jitter requirements of 13 msec. ("Condition A"), 25 msec. ("Condition B"), and with jitter requirement of 40 msec. ("Condition C"). The considered performance parameters are overhead factor, packet loss ratio and jitter.

Regarding the average overhead, processes with one agent naturally have no overhead (factor 1), processes with two agents have overhead of factor 2 (every packet is transmitted twice), and so on. Table I summarizes the overhead factor of our method relative to a single agent process. Each line in the table specifies the average overhead factor, the observed minimum overhead factor and the observed maximum overhead factor. The best k process with parameter 13 has an average overhead factor of 3.08, the best k process with parameter 25 has an average overhead factor of 1.91 and the best k process with parameter 40 has an average overhead factor of 1.65. The differences in the overhead factors are due to the fact that fewer competitions are generated when the requirement from the jitter is less demanding. In a competition all the potential 5 agents transmit two segments, thus, the overhead increases with the number of competitions. Interestingly, the observed minimum overhead factor of the best k process with parameter 40 is 1.09 which is an excellent result. In this observation, only 46 competitions were generated by the algorithm (92 segments) out of 4000 segments in the total transmission and all other 3908 segments were transmitted by a single agent only (97.7%).

To understand the source of the overhead results, Table II plots the percentages of segments transmitted by a number of agents in each algorithm. For example, when using the best k process with parameter 40, an average of 83.7% of the segments were transmitted by only one agent, 0.1% of the segments were transmitted by exactly two agents and 16.2% of the segments were transmitted by all 5 agents (during competitions). Table II illustrates that the best k algorithms with parameters 25 and 40 chose most of the time to use only one transmitting agent, but kept replacing it when its performance decreased. These insights imply that the overhead can be reduced significantly by developing a different mechanism to replace/select the transmitting agents other than a competition.

TABLE I
OVERHEAD FACTOR

Process	Average	Minimum observed	Maximum observed
Best k (A)	3.08	2.05	3.93
Best k (B)	1.91	1.10	2.79
Best k (C)	1.65	1.09	2.72

TABLE II
AVERAGE PERCENTAGES OF TRANSMITTING AGENTS

Process	% using 1 agt.	% using 2 agt.	% using 3 agt.	% using 4 agt.	% using 5 agt.
Best k (A)	45.5%	3.2%	0.0%	0.0%	51.3%
Best k (B)	77.0%	0.4%	0.0%	0.0%	22.6%
Best k (C)	83.7%	0.1%	0.0%	0.0%	16.2%

Regarding the packet loss ratio, all best k processes and all multiple transmission processes with 3 agents or more have 0.0% average packet loss ratio. The measurements of the processes with one and two agents are presented in Table III. Generally, the networks are reliable and usually the packet loss ratio is very low. However, a very high packet loss ratio of up to 17% packet loss ratio was observed for a single agent in some cases. Naturally, using additional agent reduces the packet loss ratio dramatically, and using more agents or more sophisticated algorithms reduce the packet loss ratio to 0.0.

In order to evaluate the impact of the statistics on the actual user experience we study the function 1-CDF (Cumulative Distribution Function). It represents the average percentages of times that the arrival process violates the corresponding jitter condition. Table IV presents these jitter statistic of the competing methods. Each line describes the average percentages of times that the arrival processes violate the corresponding jitter condition. That is, the packets inter-arrival time is larger than the specified threshold. For example, in line number twelve, the jitter condition is "smaller than 11", and the process "best k with parameter 25" violates this condition in 8% of the samples on average while the process that uses simple multiple transmissions of three agents violates this condition 14% on average. As can be seen from this table, starting from jitter condition "smaller than 11" the best k processes with parameters 13 and 25 outperform the other processes with a significant small number of condition violation. The best k process with parameter 40 behaves very similar to the process with five multiple agents starting from jitter condition "smaller than 13". Note that all best k processes perform at least three times better than single transmission.

All above mentioned results imply that there is no need to require a strong performance condition to improve the

performance significantly. The performance of the algorithm under different performance requirements is similar. However, the overhead increases with the performance condition strength. Thus, selecting normal to weak performance condition is recommended.

V. Further Work and Conclusion

This article described a novel approach to ubiquitous multimedia service - client/server architecture. The system incorporates detailed requirements, specifications and algorithms. We address all known issues related to ubiquitous service: QoS management, efficient usage of network resources, limited overhead, simultaneously usage of a device by a number of users, user mobility and user interface. An additional advantage of our system is that it is network independent, and thus can use any RAT technology. Obviously, it can coexist with other ubiquitous service architectures.

References

[1] P. Bellavista, A. Corradi, and L. Foschini, "MUM: a middleware for the provisioning of continuous services to mobile users", In IEEE Sym.. on Computers and Communications, Vol. 1, pp. 498-505, 2004.

[2] P. Bellavista, A. Corradi, and L. Foschini, "Context-aware handoff middleware for transparent service continuity in wireless networks", Pervasive Mob. Comput. vol. 3(4), pp. 439-466, August 2007.

[3] P. Bellavista, A. Corradi, and C. Giannelli, "Mobility-aware management of internet connectivity in always best served wireless scenarios", In Mobile Networks and Applications, Vol. 14, pp. 18-34, Kluwer Academic Publishers, 2009.

[4] Y. Cui, K. Nahrstedt, and D. Xu, "Seamless User-Level Handoff in Ubiquitous Multimedia Service Delivery" Multimedia Tools Appl. vol. 22(2), pp. 137-170, February 2004.

[5] R. Ferrus, O. Sallent, and R. Agusti, "Interworking in heterogeneous wireless networks: comprehensive framework and future trends", IEEE Wireless Communications, vol. 17(2), pp. 22-31, April 2010.

[6] Yong-Ju Lee, Choon-Seo Park, Jin-Whan Jeong, Hag-Young Kim and Cheol-Hoon Lee, "UMOST : Ubiquitous Multimedia Framework for Context-Aware Session Mobility", Int. Conf. on Multimedia and Ubiquitous Engineering (MUE 2008), pp. 3-8, April 2008.

[7] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-aware middleware for ubiquitous and heterogeneous environments", IEEE Comm. Magazine, vol 39(11), pp. 140-148, November 2001.

[8] R. Nossenson and O. Markowitz, "Using coordinated agents to improve live media contents transmissions", Int. Conf. on Systems and Networks Communications (ICSNC 2011), pp. 167-170, Oct. 2011.

[9] M. Takemoto, T. Oh-ishi, T. Iwata, Y. Yamato, Y. Tanaka, K. Shinno, S. Tokumoto, and N. Shimamoto, "A service-composition and service-emergence framework for ubiquitous-computing environments", International Symposium on Applications and the Internet Workshops, pp. 313-318, 2004.

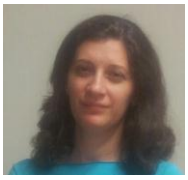
[10] M. Takemoto, H. Sunaga, K. Tanaka, H. Matsumura, and E. Shinohara, "The Ubiquitous Service-Oriented Network (USON) - An Approach for a Ubiquitous World Based on P2P Technology", in Proc. Peer-to-Peer Computing, pp.17-24, 2002.

[11] LiveU web site: http://www.liveu.tv/ accessed: January 2012.

[12] Wireshark web site: http://www.wireshark.org/ accessed: January 2012.

[13] R. Nossenson, O. Yudilevich and O. Markowitz, "Client-Server Architecture and Algorithms for Ubiquitous Video Service", The 6th International Conference on Multimedia and Ubiquitous Engineering (MUE 2012), Madrid, Spain, July 2012.

Ronit Nossenson holds a PhD on stochastic models for web servers and an M.Sc. on incremental connectivity of a graph, both from the computer science department at the Technion, Israel institute of technology. She has over 12 years of experience in modeling and analysis of web traffic. Since 2005, she specializes in optimization and trouble shooting of data traffic over cellular network, with tier-one operators' real traffic analysis. Currently, Ronit is the head of the Communication M.Sc. program in the Computer Science department of the Jerusalem College of Technology.



Orit Yudilevich received the BA degree in computer science and mathematics from the Tel-Aviv University and has recently (2011) received her M.Sc. degree in computer science from the Interdisciplinary Center Herzliya. She is now with the Israel Defense Force.

Omer Markowitz received the BA degree in computer science from the Tel aviv-Yaffo Academic College in 2004 and has recently (2011) received his M.Sc. degree also in Computer science from the Interdisciplinary Center Herzliya.