Available online at www.sciencedirect.com
ScienceDirect
Journal of Network and Computer Applications 30 (2007) 1428–1444
ELSEVIER
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

Design and analysis of a stable set-up protocol for transcoding multicast trees in active networks

Bart Duysburgh*, Thijs Lambrecht, Filip De Turck, Bart Dhoedt, Piet Demeester

Department of Information Technology (INTEC), Ghent University, St. Pietersnieuwstraat 41, B-9000 Ghent, Belgium

Received 11 November 2003; accepted 8 June 2006

Abstract

Based on active networking, an advanced streaming service was designed to offer different formats of the same stream using a single multicast tree. To that end, the initial format of the stream that is sent into the tree is transcoded to the other requested formats in the nodes of the tree, based on application level functionality residing in these network nodes. To set up such a multicast tree (including the necessary forwarding state in the nodes) and to install the necessary functionality in the nodes, a number of tree set-up procedures were designed. In this paper, performance aspects of these procedures are investigated: the stability and consistency of the forwarding state during the set-up procedures and the influence of dynamic user behaviour on the multicast tree. This performance assessment is based on a thorough analysis of the different set-up procedures and on simulations of the procedures. Based on the analysis, it is seen that great care must be taken during the set-up procedures in order to avoid interference with the existing streams. Therefore, extensions for the procedures are proposed. Furthermore, the influence of dynamic user behaviour on session state and performance attributes is analysed.

Keywords: Multicast tree; Media transcoding; Active networks; Set-up procedures; Analysis of stability

*Corresponding author. Tel.: +32 9 264 99 70; fax: +32 9 264 99 60
E-mail addresses: bart.duysburgh@intec.ugent.be (B. Duysburgh), thijs.lambrecht@intec.ugent.be (T. Lambrecht), filip.deturck@intec.ugent.be (F. De Turck), bart.dhoedt@intec.ugent.be (B. Dhoedt), piet.demeester@intec.ugent.be (P. Demeester).

1084-8045/$ - see front matter © 2006 Published by Elsevier Ltd.
doi:10.1016/j.jnca.2006.06.003

1. Introduction

The concept of active networking (Tennenhouse et al., 1997; Smith et al., 1999; Calvert et al., 1998; Psounis, 1999) provides an advanced networking infrastructure with increased flexibility and additional functionality, which is partly hosted by the network nodes. This way, the network can support advanced services with application level functionality residing in the network nodes. This is achieved by allowing each packet to carry its own forwarding routine that is executed on each node. This forwarding routine can specify a complex behaviour based on the active networking API provided by the nodes. This API allows to handle packets in the nodes, possibly including complex processing and modification of the payload, and to maintain application state in an information cache. Additionally, state information can be retrieved from the node environment and additional software modules can be called from node extensions. Consequently, each packet carries a program that defines its behaviour in the nodes and allows to provide advanced application functionality in the network.

This infrastructure is used to implement an advanced media streaming service. Here, transcoding functionality is implemented in network nodes, allowing delivery of multimedia streams to terminals with heterogeneous capabilities (e.g. in terms of access bandwidth or processing power) while optimising bandwidth usage. Instead of building several multicast trees, each carrying a specific version of the same streaming content, only one version of the digital content is sent by the source. Transcodings performed at strategic locations in the network allow to deliver the content with a customised quality to each client. The streamed data are sent through the network in *DATA* packets that find their way towards the users that joined the session based on the tree forwarding state that is set up in the network nodes. This tree forwarding state is maintained per session on each node in the information cache and contains information on the basic stream that is received and the different versions that are requested by the downstream users. More specifically, it is specified on which downstream link a specific version of the stream has to be sent and whether or not a transformation is needed from the basic stream.

The set-up of the multicast tree is triggered by the *JOIN* messages from the users, indicating the streaming session of their interest (content selection) and the required version (bitrate, codec, etc.). While travelling through the network these messages will modify the forwarding state in some nodes, according to the carried request. In the same way, the state will be modified by *LEAVE* messages from users wanting to leave the session.

Two particular problems have to be considered when applying this dynamic tree set-up procedure. First, when a streaming session has started, *DATA* packets are sent through the multicast tree, finding their way towards the joined users based on the forwarding information in the nodes. When during this session some users want to join or leave the session, *JOIN* or *LEAVE* messages will be sent in the network, triggering changes in the forwarding state. Since this state is used at the same time for routing the *DATA* packets, great care has to be taken. The set-up procedures have to be carefully designed to avoid interrupting the existing data streams by changes in the forwarding state. Second, in some cases the multicast group of the session may be characterised by a large degree of dynamic behaviour, with users leaving or joining frequently and unexpectedly. This way, the original multicast tree set up at the start of the session may gradually change to a totally different tree. Due to this gradual change, the tree topology is influenced by the existing

state in the nodes and the resulting tree for the final multicast group will differ from the more optimal tree resulting from the same group joining the session at once. The differences between both multicast trees will not have a considerable influence on the users, but they are important for the performance of the network and the usage of resources in the network.

Consequently, we investigate the following two aspects: the stability and consistency of the multicast forwarding state during the set-up procedures and the performance of the resulting multicast tree after a number of changes in the multicast group. The former is discussed in Section 3, while the latter is discussed in Section 4. First of all, a description of the advanced multicast service and the set-up procedures are described in Section 2.

2. An advanced multicast service with transcoding capabilities

Multicast technology has always been considered the preferred technique to support the distribution of media streams to a large group of users simultaneously. In the traditional application of media streaming one specific version, in terms of encoding or bitrate, is distributed to all users. This approach however lacks a lot of flexibility and does not allow any customisation of the service by the users. Therefore, it could be favourable to provide multiple versions of the streamed media. In this way, customers can choose a version according to the capabilities of their terminal and their network connection. They can customise the service according to their own needs and wishes, while considering the available support for different codecs on their terminal or the technology and bandwidth of their connection.

In order to support this advanced feature of multicast streaming in the traditional network, separate multicast trees are needed for each version of the stream. This results in a significant increase of the load on both the server and the network. However, in an active network this advanced feature can be supported with a considerable reduction in network and server load. Only one multicast tree is used, sending only one version of the media stream and performing transformations of the original stream to the requested versions in intermediate nodes of the tree. In Fig. 1, a simple example is given of such a multicast session.

For the discussions in the following sections, we focus on a general media distribution service providing five versions of the data with different bandwidths. These versions are referred to as codecs A, B, C, D and E, listed in descending order of used bandwidth. Transformations between these codecs are only performed from a higher to a lower bandwidth. After all, the requested codec with the highest bandwidth is initially sent in the tree, providing the data at the best quality, and lower bandwidth version can be deduced from it at the cost of a reduced quality.

To distribute the appropriate versions of the streamed media to the different users of the session, a multicast tree has to be set up, incorporating the specific requests of all users. The state information for this multicast tree is maintained per session in the information cache of each active networking node. This state contains a reference to the corresponding session and consists of pointers to the next hops to which the stream has to be forwarded. For each pointer the requested version of the stream in that part of the tree is specified. When the pointer indicates the local node as the next hop, the specified version has to be delivered to the local user of that node. This way, a multicast tree as shown in Fig. 1 can be specified. In each node of the tree, a table with the corresponding tree state information is

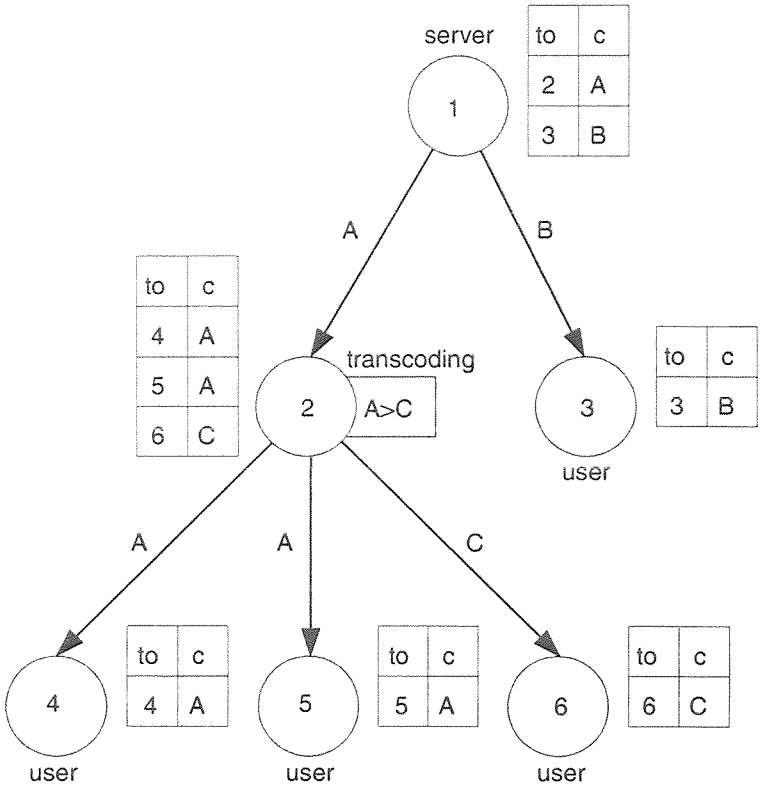

Fig. 1. Example of the tree state information set up in the nodes for a simple tree with transcoding capabilities.

set up. This table contains the pointers towards the next hop (in the to-field of each entry), as well as the requested codec for each hop (in the *c*-field of each table entry).

The media stream is transported in *DATA* capsules through the multicast tree. The forwarding routine, carried in the *DATA* capsule, specifies how the capsule and its payload travel through the network. The forwarding routine is excuted in each node, and uses the tree state information for this particular session available at this particular node. In node 2 of Fig. 1, the arriving *DATA* capsules carry the codec with the highest bandwidth specified in that node's tree state, which is called the upstream codec. This codec A version of the stream has to be forwarded to nodes 4 and 5, while codec C is required in node 6. Therefore, at the arrival of the *DATA* capsule carrying codec A, it is duplicated and forwarded towards nodes 4 and 5. For a third instance of the same capsule, the payload is first transcoded to the requested codec C and the capsule is then forwarded to node 6. Further on, this basic example will be extended to more complex situations, where the processing load of the nodes is taken into account and the simple set-up is not sufficient. For these complex set-up procedures, additional tree state will be necessary.

2.1. Basic tree set-up procedures

The set-up of the tree forwarding state in the nodes is triggered by the *JOIN* messages from the users, carrying requests to join a session and to receive a specific version. These messages are transported in specific capsules, which in their dedicated forwarding routine also carry the intelligence to set up the node's tree state. Consequently, the capsules will set

up the tree state on their way through the network while their forwarding routine is executed on each node that is passed.

With these capsules, a shortest path tree can be set up, based on the unicast routing information. Taking into account the possible asymmetric character of the network's routes, different multicast trees may be set up depending on the direction of the set-up procedure: from users to server (upstream) or from server to users (downstream). In the first case, the tree state is set up in the nodes by the *JOIN* capsule travelling from the users to the server. In the second case, the *JOIN* capsule has to travel to the server and an acknowledgement in the form of a *J-ACK* capsule is sent back to the users, setting up the tree state along its way. The downstream set-up will result in a more optimal shortest path tree, with the streamed packets travelling along the shortest path from the server to the users. However, the upstream set-up will be faster and reduces the amount of signalling messages in the network since the *JOIN* messages only travel upstream until an existing part of the tree is reached, immediately setting up the tree state in the nodes that are passed.

Fig. 2 illustrates the difference between both set-up procedures. It is assumed that the unicast routes between nodes 5 and 1 are different in both directions. The route from nodes 5 to 1 is via node 2, whereas the route in the opposite direction is via node 3. In this example the user in node 4 joins the session first with codec A and later the user in node 5 joins with codec B. In the upstream set-up of Fig. 2(a), the *JOIN* capsule from user 4 travels to node 1 setting up the tree state in nodes 4, 2 and 1 (step (1)–(2)), and the stream of *DATA* capsules with codec A starts when the *JOIN* capsule is processed in the server at node 1. The *JOIN* capsule from user 5 only travels to node 2 where the present tree state is inspected, finding that codec A is delivered to that node (step (3)). Consequently, additional state is added, specifying that a transcoding is needed from codec A to B and that the latter should be forwarded to node 5. At the next arrival of a *DATA* capsule with codec A, the stream towards node 5 will start. In the downstream set-up of Fig. 2(b), the *JOIN* capsules are travelling to the server where they trigger the set-up of tree state in the

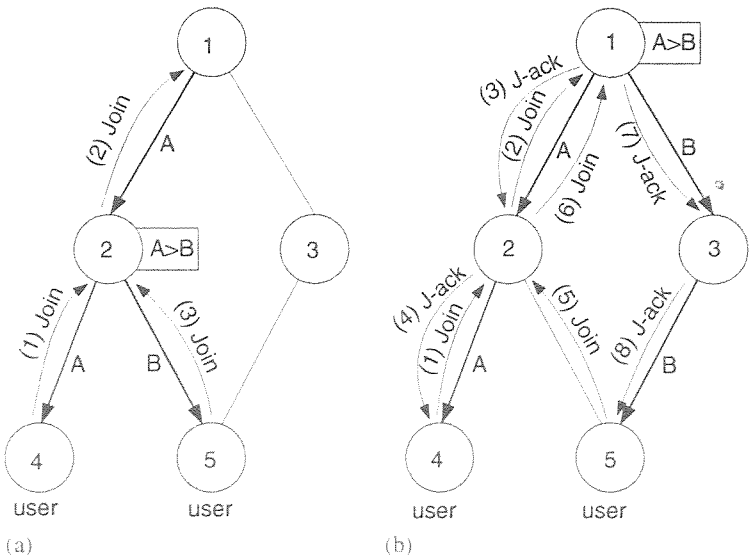

Fig. 2. Illustration of the difference between the tree set-up procedures in upstream (a) and downstream (b) direction.

server node (steps (1)–(2) and (5)–(6)). Next, the *J-ACK* capsule is sent, setting up the tree state in nodes 2 and 4 (step (3)–(4)) and in nodes 3 and 5 (step (7)–(8)). These capsules are immediately followed by the *DATA* capsules, starting the media stream. In this case, a transcoding from codec A to B is set up in node 1 and the codecs use separate paths. From this discussion, it is clear that the two procedures result in different trees when asymmetric unicast routes are involved, while in the case of symmetric routes identical trees are set up. In Duysburgh et al. (2004), these set-up procedures are described more extensively and a performance analysis is made. Based on its much faster response times and its reduced complexity of the set-up procedure the upstream procedure is preferred over the downstream procedure. Furthermore, the incidence rate of asymmetric routes is rather low in current networks, reducing the number of cases where the downstream procedure could have an advantage. Therefore, further discussions will focus on the upstream set-up procedure.

2.2. *Set-up procedures in nodes with insufficient resources*

This general and relatively simple set-up procedure discussed in the previous section can only be used when the network nodes are not overloaded and additional transcoding processes can still be installed. However, when a transcoding process is needed on a node with insufficient resources, more complex procedures are used to overcome the problem. The general approach to these procedures is to relocate the transcoding process to a node with sufficient resources and to forward the stream accordingly. Three different strategies have been proposed. The first two strategies attempt to maintain the session's traffic on the existing tree by pushing the transcoding process along the currently used links of the tree in the upstream or downstream direction. The process is relocated to the nearest node of the tree in that specific direction capable of accommodating the requested transcoding. These strategies are called *Push Upstream* and *Push Downstream*, respectively. A third strategy tries to find the closest node in the entire network capable of accommodating the transcoding process. This node will then function as a proxy node for the transcoding and the strategy is called *Push to Proxy*. A suitable proxy node is determined, based on the unicast routing table and the load information of the nodes, which is distributed over the network. First, the neighbours of the overloaded node are examined whether they have sufficient means to support the relocated transcoding process. If so, the node with the most means available is selected. Otherwise, if no suitable node is found, the nodes that are 2 hops away are examined in the same way. In subsequent steps, groups of nodes that are even further away are examined and hence the closest node with sufficient resources is selected.

The three strategies are illustrated in Fig. 3 with node 2 being overloaded. Consequently, the transcoding process from codec A to B, which is needed to deliver codec B to node 4, cannot be installed on node 2. In the *Push Upstream* procedure, the transcoding is relocated to node 1 and codec A and B are streamed simultaneously from nodes 1 to 2. In node 2, these streams are forwarded in the appropriate direction. In the case of *Push Downstream*, codec A is sent from nodes 2 to 4, where it is transcoded to codec B. In the *Push to Proxy* strategy, codec A is sent from node 2 to the transcoding proxy node 5. After transcoding the stream, it is returned from nodes 5 to 2 and forwarded to node 4.

To maintain these more complex transcoding trees in the network, additional tree state information has to be kept in the tree nodes. Therefore, two extra fields are introduced in

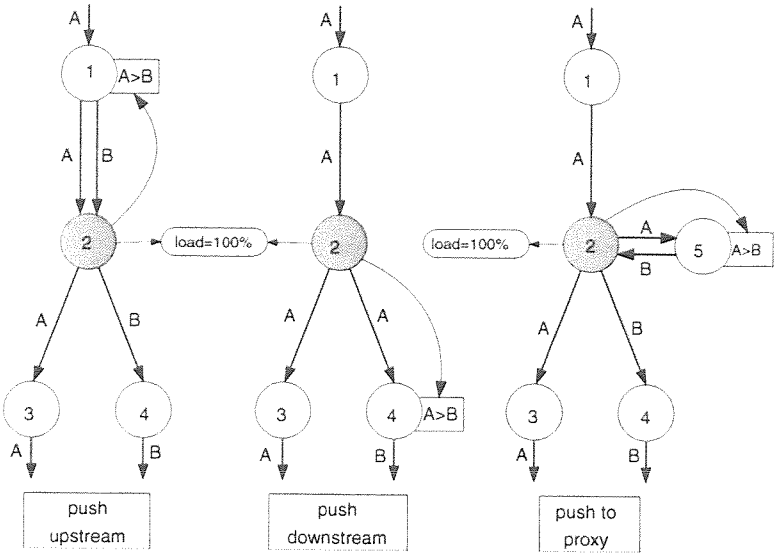

Fig. 3. Strategies to overcome processing power scarcity in nodes where transcodings are needed.

the tree state entries in addition to those illustrated in Fig. 1: the *tc*-field and the *psh*-field. The *tc*-field indicates whether or not the requested codec in the entry should be transcoded from the upstream codec. When $tc = no$ is specified for an entry, the codec may not be transcoded from the uplink codec. The stream is only forwarded over the link specified in the entry when the specified codec explicitly arrives at the node. The *psh*-field is used in the entries of a transcoded codec in the path between the proxy node or the upstream node where the transcoding was pushed to and the overloaded node. The entries of such a codec that is either transported in parallel with the actual upstream codec or on a specific branch between the proxy and the overloaded node are marked with $psh = yes$. This way, the *psh*-field indicates that these entries should not be modified since they are required to direct the transcoded stream to the originating node along a particular path.

The use of this tree state is illustrated in Fig. 4. In Fig. 4(a), the transcoding from codec A to B needed on node 5 is pushed upstream to node 2. Consequently, nodes 2 and 3 contain tree state that forwards codec A and B simultaneously to node 5. The entries for codec B in these nodes are therefore marked with $psh = yes$, indicating that this is pushed traffic. In nodes 3 and 5, codec B should not be transcoded from codec A, so the entries for codec B are marked with $tc = no$. In node 4, the regular tree state for a node with sufficient resources is shown, with one transcoding from codec B to C. In Fig. 4(b), a tree is shown with a transcoding process pushed to a proxy node. The transcoding from codec A to B, required to deliver codec B to node 6, is relocated to node 5. Codec A is forwarded to the proxy node as if node 5 were just one of the user nodes of the tree. The stream from nodes 5 to 2 carrying codec B is indicated as pushed traffic ($psh = yes$). In Fig. 4(c), it can be seen that in trees with transcodings pushed to downstream nodes, the *tc*- and *psh*-fields are not really needed to specify the tree.

To set up the tree state in these *Push* procedures, additional capsules are used. In case of the *Push Upstream* or *Downstream* procedure, the *Push* capsule is simply sent in the respective direction, travelling until a node is found capable of accommodating the required transcoding and modifying the tree state on its way. For the *Push to Proxy* procedure, a *PUSH* and a *P-ACK* capsule are needed to set up the tree state in both

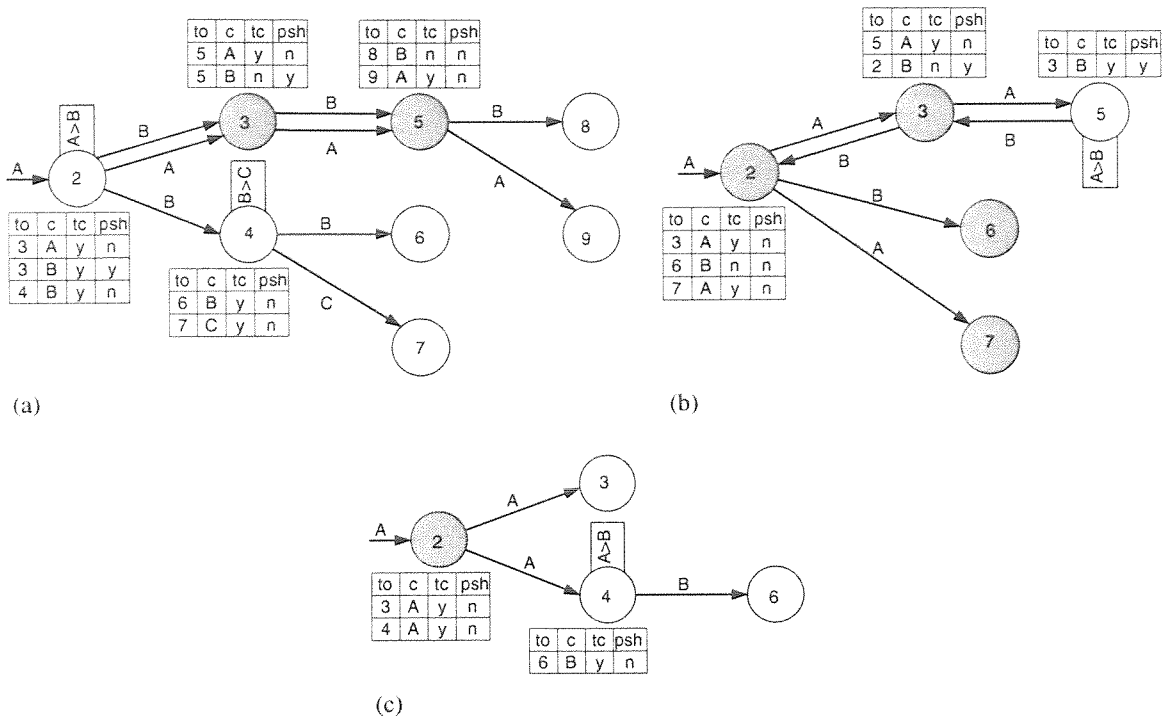

Fig. 4. Illustration of the tree state information in complex trees with pushed transcoding processes.

directions between the overloaded node and the transcoding proxy node. Because the original codec should be delivered to the proxy node and the transcoded codec has to be sent back, this procedure is more complex than the other two. More specifically, four special cases can occur, which are illustrated in Fig. 5. In Fig. 5(a), the general set-up of traffic to and from the proxy node is shown. When the *JOIN* capsule requesting codec B arrives at node 2, it finds that codec A is delivered to the node but insufficient resources are present to perform a transcoding locally (step (1)). When node 4 is determined as a suitable proxy node, a *PUSH* capsule is sent to node 4, specifying that codec C should be transcoded from codec A and should be delivered to node 2 (steps (2)–(3)). When the capsule arrives in node 4, the necessary tree state is set up and a *P-ACK* capsule is sent back to node 2, setting up the necessary tree state for the traffic in both directions along its way (steps (4)–(5)). When the state is also configured in node 2, the media stream finds its way to and from the proxy node and to node 6.

During the set-up procedure, the *PUSH* and *P-ACK* capsules both perform some checks on their way between the overloaded and the proxy node. The *PUSH* capsule determines whether the transcoded codec is available in the intermediate nodes. This is illustrated in Fig. 5(b), where the transcoded codec B is encountered in node 3 and it is simply forwarded to node 2. This way no transcoding process has to be set up on node 4. Consequently, the necessary state is installed on node 3 and a *P-ACK* capsule is sent back to set up the tree state on its way to node 2. The *P-ACK* capsule checks if a codec with a higher bandwidth than required is present in the nodes on its way from the proxy node to the overloaded node. In both Figs. 5(c) and (d), such codec is encountered in node 3, so it is not necessary to send the original codec A from the overloaded node 2 towards the proxy node 4. Instead the stream can simply be forwarded from nodes 3 to 4. Since the *P-ACK* capsule by default

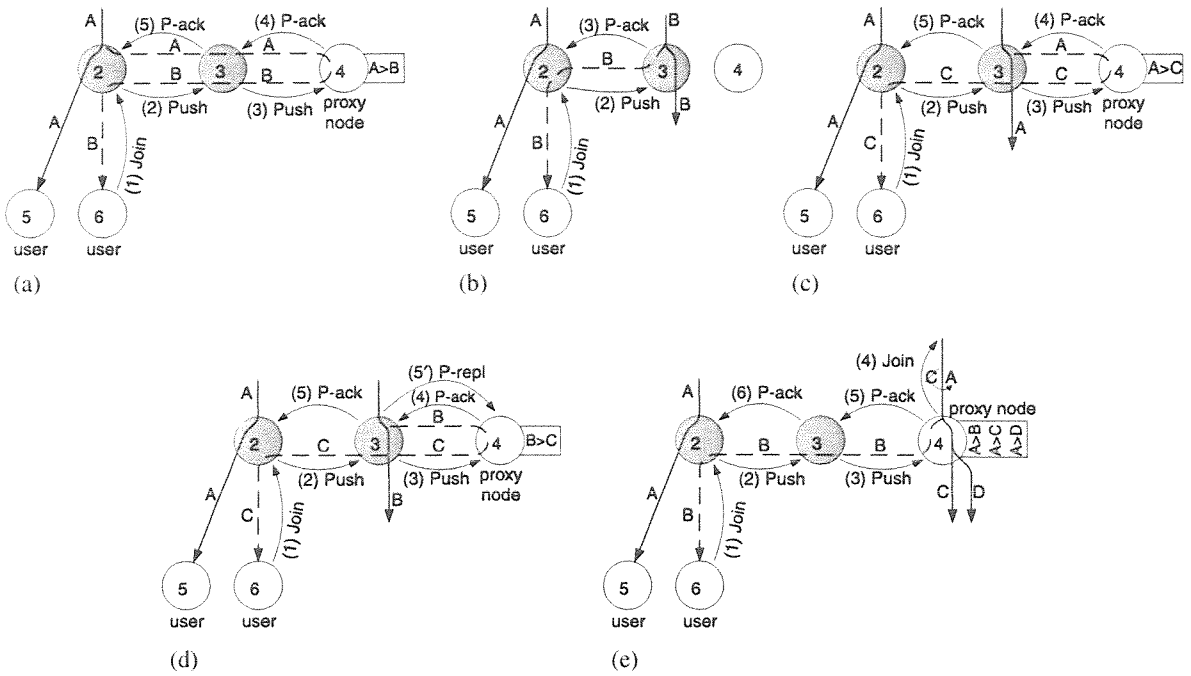

Fig. 5. Set-up of traffic between an overloaded node and a transcoding proxy node.

sets up the state to forward the original codec A towards the proxy node, it is necessary for the case of Fig. 5(d) to replace this information with the correct state to forward codec B to the proxy node and to transcode it there to codec C. Therefore, a *P-REPL* capsule is sent back to the proxy node, replacing the default state with the correct one. In the situation of Fig. 5(c), the original codec is found in node 3, so the default state for codec A can be used. Further on, the *P-ACK* capsule proceeds its way from node 3 to the overloaded node, while setting up only the tree state necessary to deliver codec B to node 2. Another specific case for the *P-ACK* capsule is shown in Fig. 5(e). When on the proxy node a codec with a sufficiently high bandwidth is found, only the transcoding and the return path for the transcoded codec to the overloaded node has to be set up. However, when a codec with lower bandwidth is present in the proxy node, this upstream codec C should be replaced with the original codec A to allow the transcoding to codec B. Therefore, a *JOIN* capsule has to be sent upstream to replace codec C with codec A (step (4)) and afterwards the return path to the overloaded node is set up (steps (5)–(6)).

3. Stability and consistency of tree state during set-up procedures

An important aspect of the set-up procedures described in previous section is the stability and consistency of the tree state during the procedures. When for example new users join or leave an active session, the existing tree state in some nodes may be modified while it is at the same time used for routing *DATA* capsules in the tree and to deliver the media stream to the users. This way, existing streams may be interrupted for a short time during these procedures, degrading the service of the users that already joined the session.

Table 1
Different types of capsules used in the multicast service

Capsule	Influence on tree state	Purpose
JOIN	Adds new or overwrites existing entries	Add branch
DATA	Uses tree state to forward streamed data	Transport stream
PUSH	Adds new or overwrites existing entries	Relocate transcoding
P-ACK	Adds new entries	Relocate transcoding
P-REPL	Overwrites existing entries, which are not yet used	Clean up tree
LEAVE	Removes entries which are not needed anymore	Remove branch

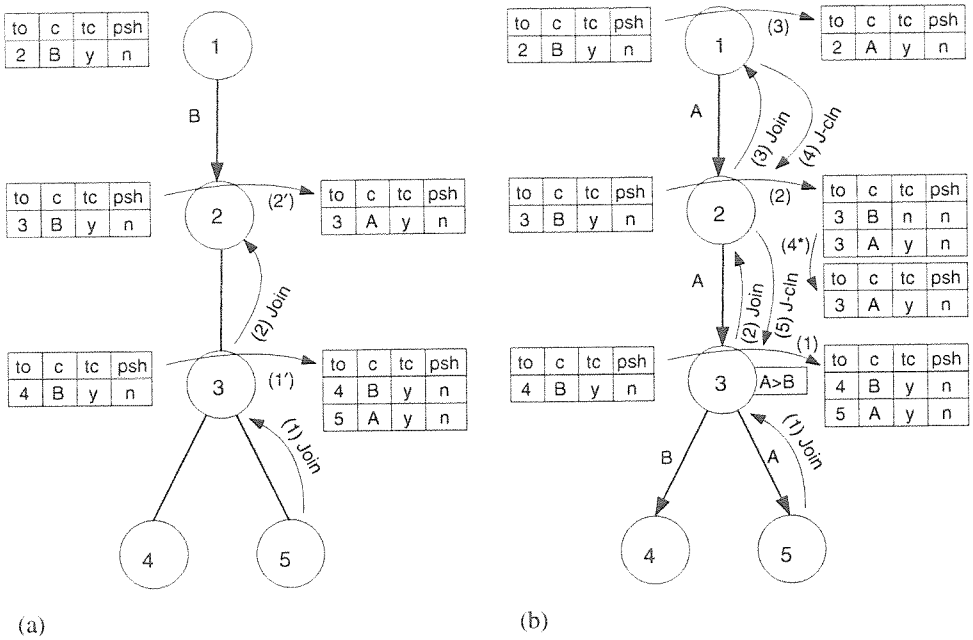

Fig. 6. Influence of the tree set-up procedure on the existing streams and their tree state. In Fig. 6(a), the existing stream with codec B from nodes 1 to 4 is interrupted when in node 2 the entry for codec B is overwritten with codec A. In Fig. 6(b), this interruption is avoided by setting up state for codec A in parallel with the existing state for codec B. Afterwards, the state for codec B is cleaned up.

Consequently, great care must be taken during the procedures to preserve the existing streams.

In Table 1, an overview is given of the different capsules involved in the multicast service and their respective influence on the tree state in the nodes. It is clear that instability and inconsistency of the tree state for the existing streams can be caused by the *JOIN* and the *PUSH* capsules, since they modify existing tree state in the nodes. In Fig. 6(a), a set-up procedure is illustrated where no precautions are taken to preserve the existing streams. User 4 has joined the session and a stream with codec B is already set up, while user 5 issues a request for codec A. In node 3 a transcoding from codec A to B is installed (step (1′)) and a *JOIN* capsule is sent upstream to set up a path to deliver codec A to node 3. In node 2 the entry for codec B is overwritten with codec A (step (2′)), since codec B will be transcoded from codec A. However, at this moment *DATA* capsules carrying codec B are

still sent from nodes 1 to 2. These capsules are not forwarded to nodes 3 and 4 since only an entry for codec A exists. The stream towards node 4 is therefore interrupted and will only resume when the full path for codec A is set up. In Fig. 6(b), this problem is avoided by adding a new entry for codec A and modifying the codec B entry ($tc = no$). This ensures that codec B will still be forwarded during the set-up. However, when codec A is finally delivered in node 2, the state for codec B will not cause a transcoding from codec A to B and codec A will just be forwarded. The state for codec B then becomes useless and therefore *J-CLN* capsules are sent to clean up this state (step (4′)). These capsules are sent back along the path of the *JOIN* capsule, which was recorded on its way upstream. The *J-CLN* capsules are only sent after a certain delay in order to make sure that all capsules carrying codec B have reached the appropriate users and none are using the transient codec B state anymore. In order to avoid that other procedures take into account the information of the temporarily remaining entries for codec B, they are explicitly marked to be transient (by $*$ in Fig. 6(b)).

In case of the *Push* procedures, similar mechanisms are used to avoid interrupting the existing streams. In Fig. 7(a), it is first shown that the *Push Upstream* procedure has no problems since it comes down to adding new entries for the pushed codec and modifying the attributes of the existing entries ($tc = no$). In Figs. 7(b) and (c), the *Push Downstream* and *Push to Proxy* procedures are illustrated. In both cases, a transcoding from codec A to B is needed in the overloaded node 3, which has to be pushed to another node, and codec A has to be installed in the upstream links replacing codec B. To keep the procedure under control, first the path for the pushed codec is set up and second the new codec is installed in the upstream links. For the first step, a *PUSH* capsule is sent in the appropriate direction and a *P-ACK* capsule returns to the overloaded node to confirm that the relocation of the transcoding is completed. However, in the *Push Downstream* procedure, some transient state similar to the one in Fig. 6(b) is left in the downstream path, which is needed until the upstream path is also completely set up. The second part starts after the arrival of the *P-ACK* capsule and a *JOIN* capsule is sent upstream to set up the state for codec A. The transient state left in the nodes by the *JOIN* capsule is cleaned up by the *J-CLN* capsule (step (8′)). Only then the upstream set-up is completed and the *J-CLN* capsule will proceed from node 3 along the downstream path to clean up the other transient state left by the earlier *Push* procedure (steps (9′)–(10′)).

From this discussion it is clear that the set-up of new streams can be made compatible with preserving existing streams at the expense of a more complex set-up procedure. This includes extra messages being sent in the network and a larger response time for the new users.

4. Stability of the multicast tree with changing multicast groups

In the preferred upstream set-up procedure, the *JOIN* capsules travel towards the server until an existing part of the multicast tree is encountered and then a new branch is attached. Consequently, the configuration of the new tree will depend on the configuration of the existing tree in the network. This way, when a multicast group is not static and users are joining and leaving the session regularly, the tree configuration will evolve with each user leaving or joining. Since each new tree configuration is based on the previous one, the resulting trees may be different from the optimal tree, which would result from the final multicast group joining the session at once. In this section, the final tree resulting after a

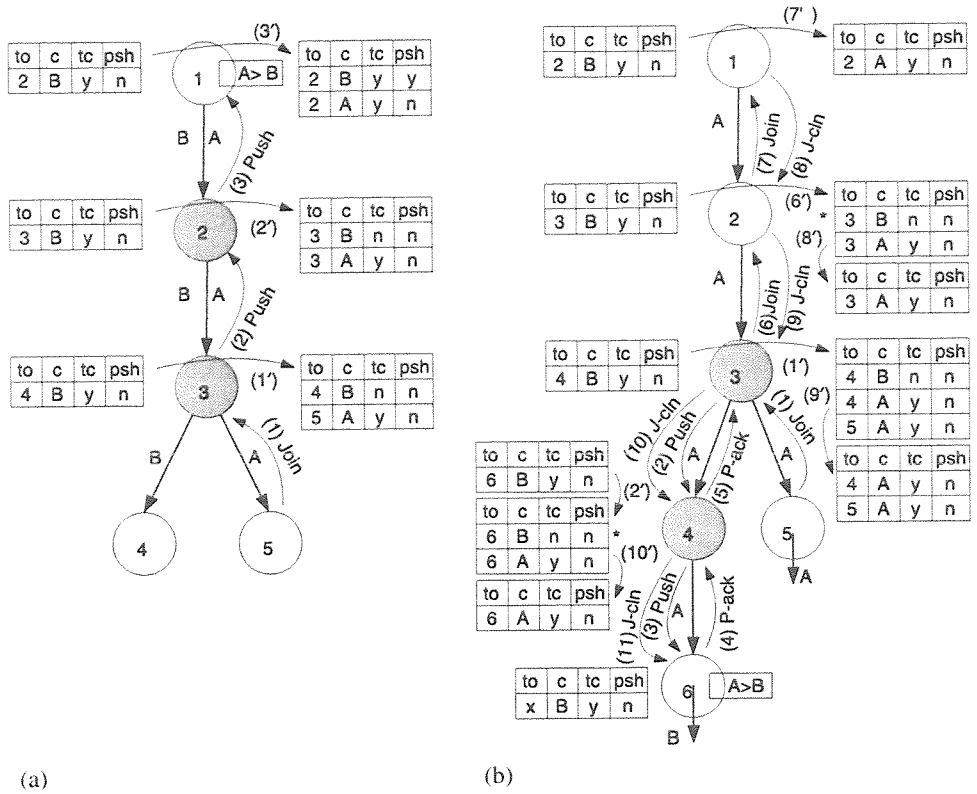

Fig. 7. Influence of the push procedures on the existing streams and their tree state. The *Push Upstream* procedure (a) has no significant influence, whereas the *Push Downstream* (b) and the *Push to Proxy* procedure (c) will interfere with the existing streams. Therefore, specific measures are required to preserve the existing streams.

number of changes in the multicast group is compared to the best possible tree that can be achieved for that specific set-up. This discussion is based on an analysis of the *Leave* procedures and on simulations.

Table 2
Codecs provided in the multicast service

	Codec	Bitrate (kbit/s)	Reference
A	PCM 16 bit	128	
B	G.711 μ-law	64	ITU-T (1972)
C	G.726-7 4 bit	32	ITU-T (1990a,b)
D	G.728	16	ITU-T (1992)
E	G.729	8	ITU-T (1996)

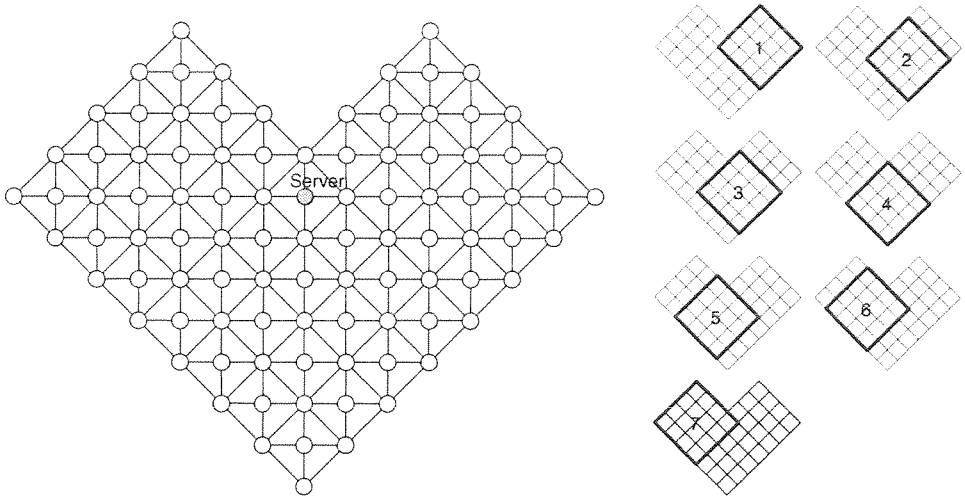

Fig. 8. The grid network used for the simulations and its division into different zones.

The simulations were performed in the ns2 network simulator (UCB/LBNL, 1997) for a voice streaming service offering the voice codecs listed in Table 2. In the grid network depicted in Fig. 8(a), a server node and seven different zones are defined. These zones, which are used to determine the moving multicast groups, are shown in Fig. 8(b). In the simulations, an initial and a final multicast group are defined, with the initial group always located in zone 1 and the final group in one of the seven zones. During the simulation, the users of the initial group join the session first. Then, a number of users will join or leave the existing session in a random order. This will finally result in the final group having joined the session. In another run of the simulation, the users of the final group join the session at once. The resulting trees of both runs are compared to determine the performance of both approaches, thereby assessing the influence of dynamic user behaviour in the resulting multicast tree.

Below, it is shown that the performance of the multicast tree is mainly determined by the way the existing tree state of an existing multicast tree is affected by users leaving the session. Therefore, the possible behaviour of the *LEAVE* capsules is described. In the simplest approach to the *Leave* procedure, the tree state that is no longer useful after a user has left, is removed. The *LEAVE* capsule travels upstream along the tree, removing the entries that were only used to deliver the stream to the leaving user. All other tree state

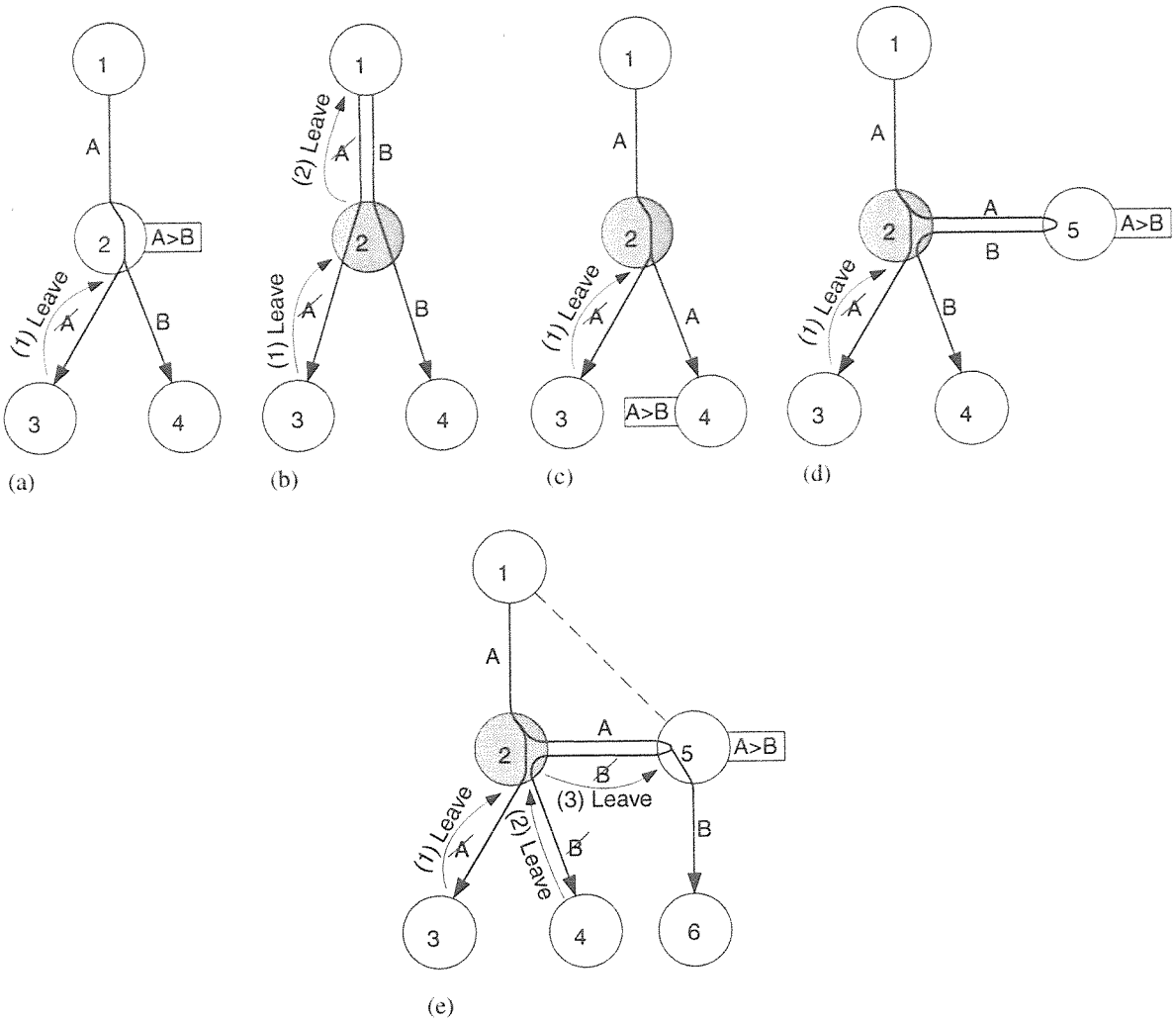

Fig. 9. Illustration of the remaining tree state exhibiting reduced performance after a user leaves the session. In (a), it is shown that this can be the case in a normal situation without overloaded nodes. In (b), it is shown that the *Push Upstream* procedure exhibits no reduced performance. However, the *Push Downstream* and *Push to Proxy* procedure can exhibit reduced performance as shown in (c)–(e).

information is left unchanged. However, a certain part of this remaining state may also be used to deliver the stream to the leaving user. The remaining tree will then partly be influenced by the previous users that have already left the session. In Fig. 9(a), a simple example shows how the remaining tree may have a reduced performance due to the influence on the tree state of the leaving user. When user 3 leaves the session, a *LEAVE* capsule is sent upstream, removing the state for codec A in node 2. The stream between the server and node 2 remains unchanged since it is needed to deliver codec B to node 4. The remaining tree is now suboptimal with codec A delivered to node 2 instead of codec B. Similar situations involving the three *Push* strategies are also shown in Fig. 9. In case of *Push Upstream* in Fig. 9(b), the parallel state for codec A can be removed in both node 2 and the server, resulting in an optimal tree. However, in case of *Push Downstream* or *Push to Proxy* (Fig. 9(c) and (d)), the procedure results in a tree with reduced performance.

Due to the fact that during the evolution between the initial and the final multicast group the tree configuration comprises a number of situations similar to Fig. 9, the resource usage in the tree is expected to increase as users join and leave the tree. The final tree could also exhibit reduced performance since each new tree builds on the state of the existing tree. To avoid this, a more complex *Leave* procedure is needed where a clean-up of the existing tree state is triggered each time a user leaves a session. In this case, each intermediate tree will still be the best possible solution for that situation, except for the *Push to Proxy* procedure. When in the *Push to Proxy* procedure a new branch is attached on a part of the tree that consists of pushed traffic, the path from the user to the server will not be the shortest path, which is illustrated in Fig. 9(e). When user 6 joins the session, codec B is directly delivered from the proxy node 5. When users 3 and 4 leave the session, node 6 now remains connected to the server via the suboptimal path over nodes 1, 2, 5 and 6 instead of the optimal path over nodes 1, 5 and 6. So, even when a clean-up of the tree state is triggered and only codec B is sent to node 6, this path will remain suboptimal.

From this discussion, the following conclusions can be drawn considering the performance of the multicast tree with changing user groups. For the *Push Upstream* and *Push Downstream* procedure, the final multicast tree exhibits similar topology and performance as if the final group joins the session at once if the *Leave* procedure triggers a clean-up of the remaining tree state. For the *Push to Proxy* procedure, the final tree will differ significantly from this situation even after a clean-up of the tree state, resulting in additional bandwidth usage and extra transcodings in the tree. Compared to these cases, the simple approach to the *Leave* procedure, which was described first, can be considered as the worst case scenario, since no precautions are taken to avoid suboptimal trees. Consequently, an analysis of the differences between the final tree and the best possible tree in this approach gives a good idea of the worst case scenario. Furthermore, the measured values for the additional bandwidth usage and the number of extra transcodings in the tree can be considered as upper limits for all other practical cases.

The values shown in Fig. 10 are obtained from simulations of the worst case approach, in which the bandwidth usage and the number of transcodings is observed, for both trees. The average increase of bandwidth usage and the number of transcodings in the tree is shown as a function of the network load, which is expressed as the fraction of overloaded nodes in the network. The simulations were done for seven pairs of an initial and a final zone, defining the part of the network where the initial and the final multicast group, containing 10 users, are located. For each of these seven pairs, the initial zone is zone one and the final zone is one of the seven zones, as indicated in the legend.

First, it should be noticed that the three approaches give the same results for an unloaded network, since no transcodings are relocated and a simple shortest path tree with transcoding nodes is set up for each case. Second, for a completely loaded network, the final tree is identical to the best possible solution for the three approaches. After all, in this case transcodings are only relocated to the edge of the network, either to the server or the user nodes. Consequently, no suboptimal trees can arise when users leave the session. Third, the main conclusion should be that the differences in bandwidth usage and number of transcodings are limited to about 2.5%. Depending on the different locations of the initial and the final multicast group, this may be even less. This is mainly be determined by the overlap between the original and the final zone and the location of both with regard to the server node.

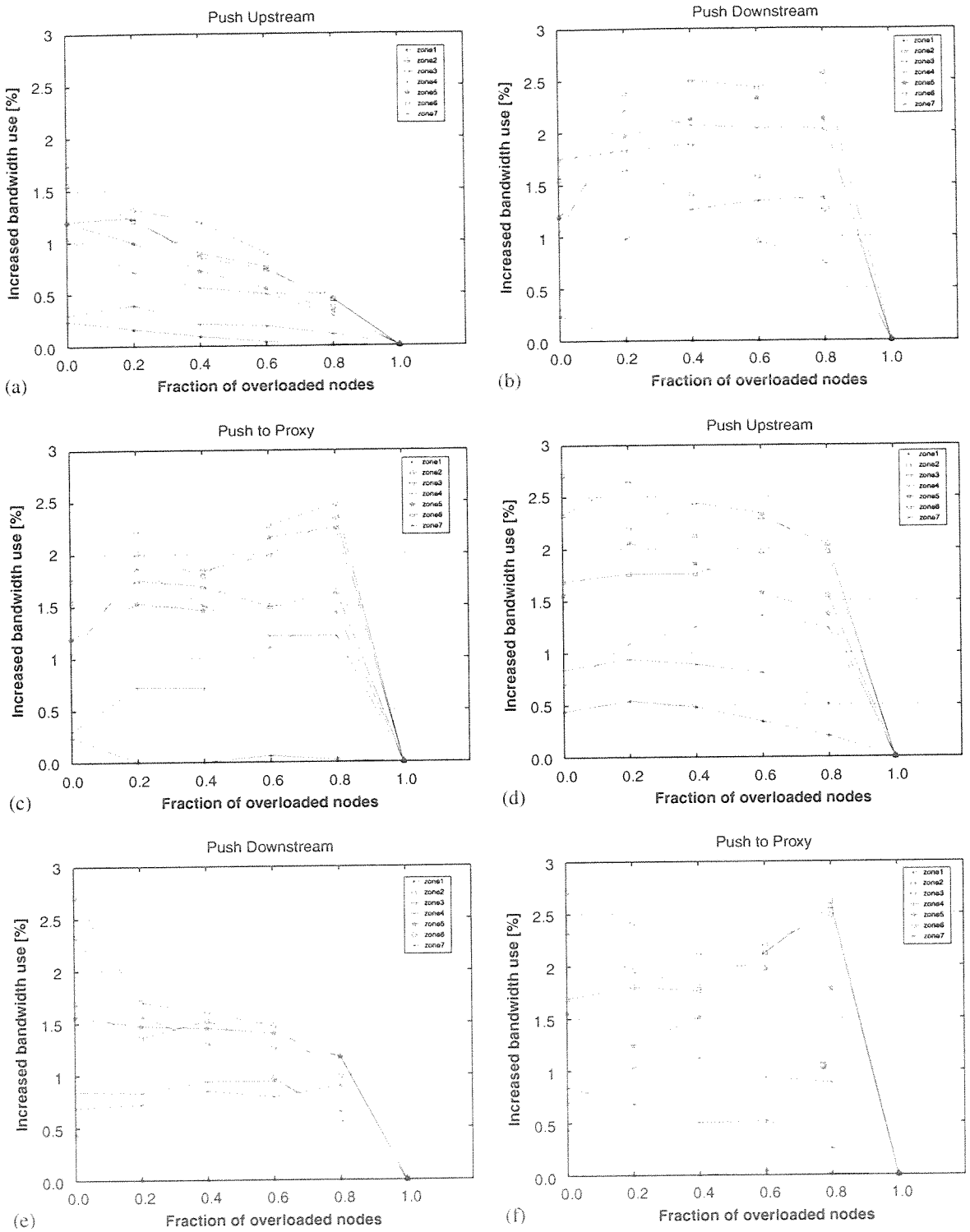

Fig. 10. Influence of a changing user group on the use of network resources in the tree.

5. Conclusion

In this paper, we presented an advanced streaming service based on active networks and offering different formats of a stream via the same multicast tree. We made a detailed analysis of the performance of the different set-up procedures used to install the

forwarding state for the service in the network nodes. In the analysis we focussed on two aspects of the performance. First, we investigated the stability and consistency of the forwarding state during the set-up procedures. It was shown that in some cases a set-up procedure may interfere with the existing streams, causing interrupts or bad performance during this set-up. Therefore, great care must be taken for the design of these set-up procedures. A number of solutions were presented to prevent the procedures from interfering with the streams. Second, the influence of dynamic user behaviour on the performance of the multicast tree was investigated. The procedures that are used to remove branches from the tree when a user leaves the session are for reasons of complexity and performance not optimised to remove all redundant state in the multicast tree. Hence, the resulting tree after a user left the session may exhibit reduced performance and over time this performance may be reduced even more. This was shown based on an analysis of the *Leave* procedure and on simulations. From these simulations it was seen that the reduction of performance in terms of used bandwidth and number of transcodings in the tree is limited to 2.5%.

Acknowledgement

Part of this work has been funded by the Ghent University GOA project "PAN" (Programmable and Active Networks).

References

Calvert KL, Bhattacharjee S, Zegura E, Sterbenz J. Directions in active networks. IEEE Commun Mag 1998;36:72–8.

Duysburgh B, Lambrecht T, De Turck F, Dhoedt B, Demeester P. An active networking based service for media transcoding in multicast sessions, IEEE Trans Syst Man Cybern 2004;34(1):19–31.

ITU-T, G.711: Pulse code modulation (PCM) of voice frequencies, ITU-T, 1972.

ITU-T, G.726: 40, 32, 24, 16 kbit/s adaptive differential pulse code modulation (ADPCM), ITU-T, 1990a.

ITU-T, G.727: 5-, 4-, 3-, and 2-bits Sample embedded adaptive differential pulse code modulation, ITU-T, 1990b.

ITU-T, G.728: Coding of speech at 16 kbit/s using low-delay code excited linear prediction, ITU-T, 1992.

ITU-T, G.729: Coding of speech at 8 kbit/s using conjugate structure algebraic-code-excited linear-prediction (CS-ACELP), ITU-T, 1996.

Psounis K. Active networks: applications, security, safety, and architectures. IEEE Commun Surv 1999;2(1):2–16.

Smith JM, Calvert KL, Murphy SL, Orman HK, Peterson LL. Activating networks: a progress report. IEEE Comput Mag 1999:32–41.

Tennenhouse D, et al. A survey of active network research. IEEE Commun Mag 1997;35:80–6.

UCB/LBNL, Network simulator, ns, version 2, 1997. (http://www.isi.edu/nsnam/ns).