in 5th. Workshop on Applications and Services in Wireless Networks,
June 29th - July 1st, 2005, Paris;

Application-level Policies for Automatic Switching between Content Redistributors

Malohat Ibrohimovna Kamilova[1], Cristian Hesselman[1,2], Ing Widya[1], and Erik Huizer[3]
[1]Centre for Telematics and Information Technology, University of Twente, The Netherlands,
[2]Telematica Instituut, The Netherlands
[3]Institute of Information and Computing Sciences, Utrecht University, The Netherlands
{m.kamilova, i.a.widya}@utwente.nl, cristian.hesselman@telin.nl, huizer@cs.uu.nl

***Abstract*—We consider the distribution of live and scheduled multimedia content (e.g., radio or TV broadcasts) through multiple aggregators. An aggregator is an access-controlled redistributor that operates a pool of proxy servers to aggregate content from source content providers and forward it to mobile hosts. Aggregators package content into channels (e.g., BBC 9 o'clock news) and offer them at various quality levels (e.g., using different encodings) and prices. Mobile hosts receive channels via a 'beyond 3G' wireless Internet, which consists of different types of wireless networks (e.g., 802.11 and UMTS networks).**

In this paper, we investigate how a mobile host can switch to another aggregator and an associated quality level while it is receiving a channel. This situation can for instance occur when a mobile host roams into a (different type of) wireless network through which a new aggregator is available, or when the mobile host roams out of a network while its current aggregator is only available through that network. We use an earlier developed application-level protocol that enables mobile hosts to discover accessible aggregators and to switch to one of them. We propose a complementary control component that automatically matches the user's preferences (e.g., regarding cost or quality) with the aggregators' available quality levels and the capabilities of the wireless networks the mobile host can connect to. The control component uses policies to decide when to invoke the protocol to hunt for alternative aggregators, how to rate their quality levels based on the user's preferences, and when and how to switch to another aggregator. Together, the protocol and the control component ensure service continuity without bothering the user. We present the design of the system and its prototype deployed in a small-scale test bed.

I. Introduction

IN In the near future, the fringes of the Internet will consist of different types of wired and wireless networks that are operated by different administrative authorities [1]. In addition, the growing number of application-level Internet services (e.g., streaming) enable a mobile host to choose a service from many alternative service providers and receive a selected service anywhere, via a chosen wireless network. A mobile host may even (temporarily) receive a certain service simultaneously via different interfaces.

In this paper, we consider a content distribution environment in which so-called "content aggregators" receive live and scheduled multimedia content (e.g., a TV broadcast) from content origins (e.g., cnn.com) and subsequently redistribute it to mobile hosts via the wireless Internet. Each aggregator offers its content in various configurations (e.g., in different encodings and qualities) to serve different types of wireless networks and mobile hosts and streams content off a pool of proxy servers.

The distribution of the same content through multiple aggregators enables mobile hosts to switch from one aggregator to another while they receive the content. These switches can for instance occur when a mobile host roams into a wireless network through which it can reach an aggregator that was previously unavailable and that can deliver the content in a configuration that better matches the user's preferences. The target configuration may differ from the old one, which means that switching may also involve adaptation of quality.

If a switch is necessary, mobile hosts should be switched between aggregators in a transparent manner to hide the complexity of the infrastructure from end-users [2]. The selection of a target aggregator is influenced by the user's preferences so that the user's experience of always receiving a channel at the right quality is maximized, even during roaming.

We have developed an application-level protocol that enables mobile hosts to switch between aggregators. The protocol allows mobile hosts to discover the configurations in which they can receive a particular piece of content from a set of available aggregators and to bind to one of these aggregators to actually receive the content in the 'best' available configuration [3].

This paper discusses a control component that automatically switches a mobile host to the aggregator that provides a certain piece of content in the best configuration. The control component runs on mobile hosts and aims to keep the stream of content alive despite changes in the mobile host's environment (e.g., due to roaming) and changes in the user's preferences (e.g., prefer cheap low quality configurations over expensive high quality ones). In particular, the control component decides when and how to invoke the previously mentioned application-level protocol to discover aggregator configurations and to switch between aggregators. This paper

therefore focuses on the control part of aggregator switching. Issues related to the encoding and rendering of streams are outside the scope of the paper.

The control component is based on policies, which are basically if-condition-then-action rules. Policy-based systems are considered flexible in managing changes in the environment of systems [4-10].

The design of our policy-based control component is based on the IETF's policy model [11,12]. Although this policy model was originally developed for network management purposes, we apply it at the application-layer level to control the mobile host's switching behavior.

This paper is structured as follows: Section II describes the live-multimedia content delivery model. Section III gives a high-level overview of the architecture of the mobile hosts that operate in the environment discussed in Section II. Section IV describes the policy-based control component, in particular the policies that it uses, its architecture, and its behavior. Section V explains the prototype of the system. Section VI discusses related work and Section VII provides conclusions and future work.

II. Content Delivery Model

We consider an environment in which mobile hosts receive live multimedia content in the form of channels. We define a channel as a user-level label (e.g., 'BBC 9 o'clock news') that indicates what content is carried by a set of streams (typically RTP streams [13]).

The key characteristic of the environment is that mobile hosts can receive the same channel from multiple alternative aggregators [3]. Figure 1 shows an example in which mobile user Bob can receive channel 'BBC 9 o'clock news' from aggregators media-forward.nl and stream-it.com.

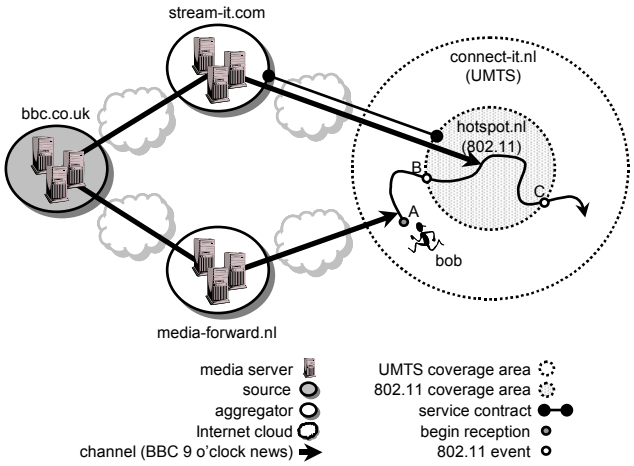

Fig. 1. Streaming via multiple aggregators.

To be able to serve different types of wireless networks and mobile hosts, each aggregator offers its channels in various configurations. A configuration is a set of streams with specific compression and packetization parameters (e.g., codec type, compression ratio, and sampling rate). Different configurations of the same channel typically require different levels of resources (e.g., network bandwidth or processing power) and provide different perceptual quality levels to the end-user at different prices. To actually receive a channel in a certain configuration, a mobile host connects to a proxy server (e.g., to a server of stream-it.com) that can stream the channel in that particular configuration.

A mobile host can exploit the availability of multiple aggregators by dynamically switching to the aggregator that provides a channel in the 'best' configuration, where 'best' is determined by the user's preferences (e.g., regarding costs and quality). As a result, the mobile host continues to receive the channel, but it receives the channel from a different aggregator, possibly in a different configuration.

One of the reasons to switch to another aggregator is to ensure continuity of the service when a mobile host's current best aggregator disappears. This typically happens when a mobile host roams into a network area where its current aggregator is unavailable. As an example, consider the network environment of Figure 1, which consists of an UMTS network operator (connect-it.nl) and an 802.11 operator (hotspot.nl). In this specific example, aggregator stream-it.com is bound to hotspot.nl (e.g., as a result of a service contract), which means that it is only available in hotspot.nl's 802.11 network. As a result, Bob can receive the BBC 9 o'clock news from stream-it.com via the 802.11 network between points B and C, but his host will have to switch back to media-forward.nl when Bob moves out of hotspot.nl's coverage area at point C. Observe that media-forward.nl is not bound to any of the network operators in Figure 1, which means that it is available through any of their networks.

Other reasons for aggregator switching are the appearance of a new aggregator that offers the channel the user is receiving in 'better' configurations, the mobile host's battery power dropping to a level at which it can no longer use the current configuration, and so on.

Observe that aggregators authenticate users and that they determine in which configurations a particular user can receive a particular channel (authorization). Aggregators set up so-called application-level roaming agreements to serve each other's users (cf. traditional network-level roaming agreements between cellular operators), which means that users only have to subscribe to a few aggregators (typically one). Authentication and authorization are however outside the scope of this paper. We refer to [14] for more details on the authentication of the users of mobile hosts.

III. Mobile Host Architecture

Figure 2 shows the high-level architecture of a mobile host that makes use of the aggregator infrastructure of Section II.

In this paper, we concentrate on the ALIVE components (ALIVE stands for Aggregator Switching System for Mobile Receivers of Live Multimedia Streams), which are application-level components that together control and perform the execution of switches between aggregators (see

the top left part of Figure 2). The ALIVE controller is responsible for controlling the execution of a switch, whereas the ALIVE protocol entity provides the functions that (support) the actual execution of the switch. The protocol entity is the result of earlier work [3,15].

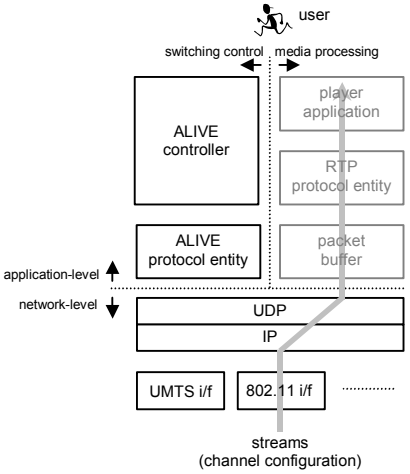

Fig. 2. Architecture of mobile hosts.

Mobile hosts also contain application-level components that are responsible for processing and rendering the multimedia packets of a stream (top right part of Figure 2) [16]. The network-level part of a mobile host (below the horizontal dotted line) consists of UDP and everything below it (e.g., different types of network interfaces). These parts of the mobile host are however outside the scope of this paper.

A. ALIVE Controller

The task of the ALIVE controller is to automatically invoke the ALIVE protocol, which it does using policies. We will discuss the ALIVE controller and its policies in detail in Section IV.

B. ALIVE Protocol Entity

The ALIVE protocol entity is the client part of the ALIVE protocol, which is a simple application-level protocol that enables mobile hosts to (1) discover in which configurations the individual aggregators in a set of aggregators can deliver a certain channel, and (2) execute a switch by disconnecting from the old aggregator and connecting to a new one.

Figure 3 shows the typical behavior of the ALIVE protocol, in this case at point B of Figure 1. The discovery part of the protocol begins with the transmission of one or more configuration requests to the aggregators the mobile host can reach at its current location (stream-it.com and media-forward.nl in Figure 1). Each request contains the name of the channel that the user is receiving (BBC 9 o'clock news in Bob's case). A mobile host can transmit a configuration request to an aggregator via any of the networks through which it can reach that aggregator (e.g., at point B, Bob's host can transmit a configuration request to media-forward.nl via UMTS or the via the 802.11 network).

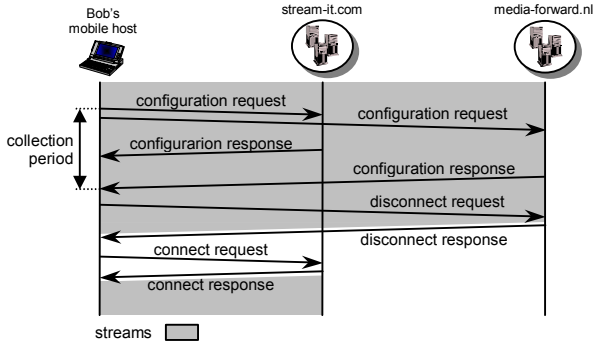

Fig. 3. Typical behavior of the ALIVE protocol.

Aggregators react to a configuration request with a configuration response. This message contains descriptions of the aggregator's configurations in which the mobile user can receive the channel listed in the configuration request.

The mobile host collects configuration responses for a certain period of time, which we call the collection period. The collection period either ends when a mobile host has received a response for each request that it sent, or when it reaches a certain maximum, whichever happens first. The end of the collection period also marks the end of the discovery phase.

To actually switch to another aggregator, the mobile host first sends a disconnect request to its current aggregator (e.g., to media-forward.nl) and then a connect request to the new one (e.g., to stream-it.com). Alternatively, the mobile host could also transmit these messages the other way around.

The connect message contains a description of the configuration in which the mobile host wants to receive the channel from the target aggregator. Aggregators acknowledge both types of messages with a response.

The ALIVE protocol makes use of UDP. The primary reason is that UDP enables the ALIVE protocol to send a message right away without having to wait for the establishment of a connection (as is the case with TCP), which enables the switching process to complete more quickly.

We implemented the ALIVE protocol as a thin layer on top of the Session Initiation Protocol (SIP) [17], which is a reliable application-level signaling protocol. SIP is typically used in combination with the Session Description Protocol (SDP) [18], which we used to specify configurations [3,15].

In [14], we heuristically analyzed the delay overhead of the SIP-based implementation of the ALIVE protocol in a contemporary hotspot environment (i.e., consisting of an 802.11 network with an overlaying UMTS/GPRS network). Using an optimized network and default SIP parameters, we found that the ALIVE protocol provides an additional delay overhead for discovery and switching of about 32% compared to a bare IP-level handoff. The actual overhead, however, highly depends on the deployment scenario, for instance on the authentication procedures used, if a network operator uses a 'fast' version of DHCP, and so on.

C. Media Processing Components

The media processing part of the mobile host consists of a

playout buffer, a depacketizer, and a player. The playout buffer receives multimedia packets from the network and temporarily stores them to ensure continuous playout when packets are lost [19] or delayed [20]. The depacketizer gets packets from the buffer and assembles them into data units that can be decompressed and rendered by the player. The typical (de)packetizer in the Internet is the Real-time Transport Protocol (RTP) [13].

The media processing components on the mobile host are outside the focus of this paper, so we will not consider them any further.

D. Other Components

Mobile hosts run two additional components that Figure 2 does not show. The first one is an Electronic Program Guide (EPG) like the Mbone tool sdr [21] that enables users to select the channel they want to receive. In this paper, we assume that users are already receiving a channel, so we will not consider EPGs nor how they interact with aggregators.

The other component is an IP connectivity handler, which is responsible for providing the mobile host with IP-level connectivity to the Internet via a certain network operator. The IP connectivity handler for instance enables a host to discover aggregators that are bound to a certain network (e.g., by putting the aggregator's URI in the SIP option for DHCP [22]), which is a step that precedes the discovery of channel configurations (see Figure 3).

IV. Policy-based Controller

The goal of the ALIVE controller (see Figure 2) is to automatically switch a mobile host between aggregators to receive a channel in the best possible configuration. One of the controller's tasks is to somehow decide when it should invoke the ALIVE protocol (e.g., when signal strength is low) and how to invoke it (e.g., first connect to the new aggregator and then disconnect from the old one).

The ALIVE controller uses policies to control this behavior. The main advantages of using policies is that they can change the controller's switching behavior without halting the mobile host ('always on') and that policies can be retrieved from a central point. The latter simplifies the management of policies and keeps the behavior of mobile hosts similar (e.g., the devices of a single user, or the pool of mobile devices of a company).

In this paper, we assume that the ALIVE protocol entity is the only entity under control of the ALIVE controller. Other components that the ALIVE controller might be in charge of (e.g., the IP connectivity handler of Section III) are outside the scope of this paper.

A. ALIVE Policies

ALIVE policies consist of the usual condition and action parts [12,23] augmented with a clause that specifies the goal of the policy [24]. The condition indicates when the policy fires, the action indicates which part of the ALIVE protocol should be invoked when the policy fires (i.e., initiate discovery, connect to an aggregator, or disconnect from an aggregator), and the goal represents what the policy is trying to accomplish in terms of switching behavior (e.g., smooth switching). An illustration of an ALIVE policy would be this one:

```
goal: highly smooth switching
      if ((on change in signal strength) &&
          (receiving through this network) &&
          (signal strength ≤ threshold))
      then <discover alternative configurations>
```

The goal of this policy is to switch to another aggregator in a highly smooth manner (i.e., before the playout buffer of Figure 2 empties) when the mobile host roams out of the network through which it receives a channel (e.g., at point C in Figure 1). To accomplish this, the policy could for instance use a relatively low threshold value in its if part. As a result, the ALIVE controller will proactively start to discover alternative configurations. This behavior increases the probability that the ALIVE controller will be able to switch the mobile host to another aggregator before the mobile host looses connectivity with its current network, which might result in an empty playout buffer and thus in perceptual glitches.

A similar policy with a 'medium' or 'low' smoothness goal will typically use a larger value for the threshold. This increases the probability of perceptual glitches, but makes the host stick with its current network longer (cf. [25]). This might be advantageous if the network is an 802.11 network, which is typically cheaper to use than a network like UMTS.

A potential problem occurs when a certain context parameter fluctuates around a threshold value (e.g., the signal strength of a base station) and that this causes the ALIVE controller to switch the mobile host back and forth between aggregators, for instance along the edges of wireless networks. To avoid such problems, the ALIVE policies should use a combination of environment parameters, contain some sort of hysteresis [25], or use back-off techniques in the conditions of cost-benefit policies. A similar problem exists at the network-level where mobile hosts handoff between networks [26].

The ALIVE controller uses the ALIVE policies to decide on the moment to invoke the ALIVE protocol. The ALIVE protocol entity must execute the actions of a policy when it fires, which means that the ALIVE policies are obligation policies [27]. Other policy types (e.g., prohibition or authorization policies [6,24]) are outside the scope of this paper. Similarly, the use of policy specification languages (e.g., Ponder [28]) is not in our scope,, because our objective is to investigate the use of policies for application-level control and we only apply simple policies to control (light weight) mobile hosts.

B. High-level Behavior

Figure 4 shows the desired high-level behavior of the ALIVE controller as a state machine in which the transitions are controlled by policies. The main states of the state

machine relevant for this paper are: monitoring, discovery, and switching.

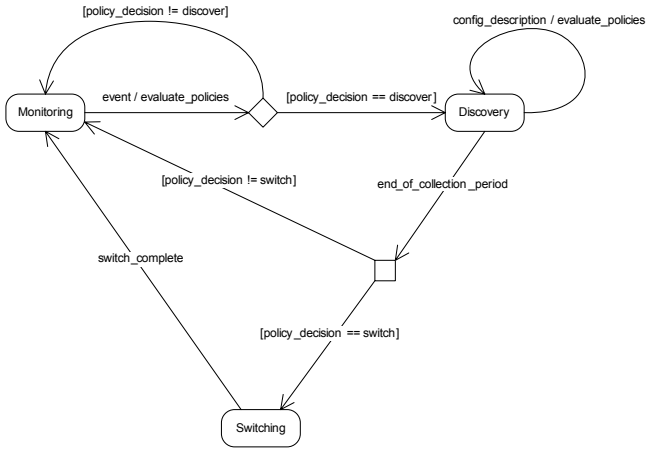

Fig. 4. High-level behavior of the controller.

When a mobile host is receiving a channel, the controller starts in the monitoring state to monitor environment changes (e.g., the appearance of a new wireless network or the current network becoming unreachable) that might require a switch to another aggregator. If such an event occurs, the controller uses so-called discovery policies to decide if it should enter the discovery state, which represents the phase in which the controller discovers configurations (see Figure 3). Discovery policies also determine the length of the collection period (see Figure 3).

In the discovery state, the controller collects the configuration descriptions that the ALIVE protocol entity receives. The controller uses switching policies to decide which of these configurations is the best one. The controller enforces the policy decision at the end of the collection period in order to have the complete overview of available alternatives. If another aggregator provides the channel in a better configuration, the controller transitions to the switching state to execute the switch. If the current aggregator provides the channel in the best configuration, then the controller simply transitions back to the monitoring state.

In the switching state the controller controls the actual execution of a switch by the ALIVE protocol entity. The switching policies determine if the controller should execute the switch in a make-before-break fashion (i.e., first connect to the target aggregator and then disconnect from the old one) or in a break-before make fashion (first disconnect, then connect). The first variant might be particularly useful if the mobile host can simultaneously connect to different types of networks (e.g., to UMTS and to 802.11). In this case, the host can temporarily receive a channel through multiple network interfaces, which facilitates smooth switching.

The controller transitions to the monitoring state when it has completed the control of the switch.

We note that there exist other types of policies that control the execution of the ALIVE protocol (e.g., policies that determine through which interfaces configuration request messages should be sent), but they are outside the scope of this paper.

C. Architecture

Figure 5 shows the architecture of the ALIVE controller. It is inspired by the IETF policy model [11] for network management, but applied at the application-level for switching control.

The main components of the IETF's policy model are a policy decision point (PDP) and a policy enforcement point (PEP). A PDP is a controlling entity that governs the behavior of a PEP (the controlled entity).

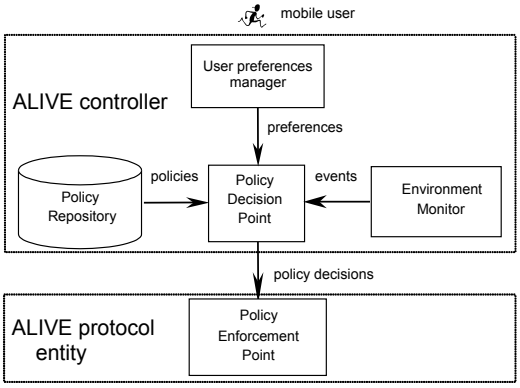

Fig. 5. Architecture of the ALIVE controller.

Figure 5 shows that the heart of the ALIVE controller consists of a PDP and that the ALIVE protocol entity forms the PEP. The other components are a policy repository for storing ALIVE policies, an environment monitor for catching events (e.g., the appearance of a new network) and a user preferences manager to enable users to change their preferences.

1) Policy Decision Point

In general, a PDP encapsulates a set of policies that it uses to control the behavior of a PEP. When a policy fires, the PDP forwards the action part of that policy to the PEP in the form of a policy decision. Policy decisions align the behavior of the entire system (PDP plus PEP) with the goals of the policies in the PDP.

The policy decisions of the ALIVE PDP are either "discover" or "switch". In the first case, the policy decision also specifies the maximum length of the collection period. In the second case, it also contains the order in which the protocol entity should execute the switch (i.e., break-before-make or make-before-break) and a description of the best configuration.

The ALIVE PDP uses information from the environment monitor and from the user preferences manager (see below) to evaluate its policies.

Observe that the ALIVE PDP could convey its decisions to the ALIVE protocol entity through a protocol like COPS [29] if either of them would not reside on the mobile host (e.g., when the PDP would be located in the aggregator infrastructure).

2) Policy Enforcement Point

In general, a PEP is the controlled entity upon which policy decisions are being enforced (by the PDP). The PEP receives directives from the PDP in the form of policy decisions (see PDP).

In our system, the ALIVE protocol entity of Figure 3 embodies the PEP because it executes the policy decisions "discover" and "switch".

3) Environment Monitor

The main task of the environment monitor is to monitor the availability of resources and to generate an event in case of a change. The environment monitor for instance keeps track of resources such as the state of the host's network interfaces (up or down), their IP addresses, current signal levels (e.g., of 802.11), the host's velocity, and so on.

The events that the monitor produces for instance contain a resource type (e.g., the name of a network interface) and a parameter-value pair (e.g., signal strength-10dB).

The environment monitor also serves requests, for instance for information on the host's current battery power level. The environment monitor is an external information source from the PDP's point of view [11].

4) User Preferences Manager

The user preferences manager allows the user to specify his preferences, for instance what he considers the 'best' configuration (e.g., the cheapest one or the one that provides the highest quality) and if he minds glitches in the playout of the channel. The latter for instance determines which policies the PDP has to use. For example, if the user does not want to notice glitches, the PDP will have to use policies that realize a highly smooth switching goal (see Section IV.A).

The preferences manager also generates an event that when the user makes a change (e.g., when he redefines 'best'). The PDP uses such an event to check if it should switch the mobile host to another aggregator.

5) Policy Repository

The policy repository is a storage facility for descriptions of ALIVE policies. The policies in the repository are inactive in the sense that they are not actively used by the PDP. The PDP can however retrieve them at run-time (i.e., without halting the mobile host), thus flexibly changing the switching behavior of mobile hosts. If the repository is centralized, the same ALIVE policies can be retrieved by several PDPs, which provides the mobile hosts with the same control behavior.

Figure 6 illustrates what the ALIVE discovery policies might look like if they are expressed in XML. The first policy has *highly smooth switching* as a goal and initiates discovery when the average packet loss increases beyond 20%. The second policy has *moderately smooth switching* as a goal and invokes discovery when the average packet loss level hits 50%.

```
<?xml version="1.0" encoding="UTF-8"?>
<policies
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="policy_repository.xsd">
   <policy>DISCOVERY
      <goal>
         <smoothness>HIGH</smoothness>
      </goal>
      <condition>
         <receiving_interface>WLAN</receiving_interface>
         <packet_loss>20
            <operator>GREATER_THAN</operator>
         </packet_loss>
      </condition>
      <action>
         <collection_period>SHORT</collection_period>
         <discover_alternatives/>
      </action>
   </policy>
   <policy>DISCOVERY
      <goal>
         <smoothness>MODERATE</smoothness>
      </goal>
      <condition>
         <receiving_interface>WLAN</receiving_interface>
         <packet_loss>50
            <operator>GREATER_THAN</operator>
         </packet loss>
      </condition>
      <action>
         <collection period>DEFAULT</collection_period>
         <discover_alternatives/>
      </action>
   </policy>
</policies>
```

Fig. 6. Discovery policies in XML.

In general, a PDP can retrieve policies from a repository based on several criteria under predefined circumstances. In this paper, we simplify the retrieval process and retrieve policies based on their goals (e.g., retrieving policies that realize a highly smooth switching). We further simplify the retrieval process by expressing the user's preferences in the same terms as the policies' goals (e.g., switching smoothness and price). This means that the user preferences directly determine which policies the PDP will retrieve. For example, if the user does not mind perceptual glitches, then the PDP will for instance retrieve policies that realize a moderately smooth switching (for both discovery and switching policies). This example also shows that a change of user preferences can trigger the PDP to retrieve new policies from the policy repository.

D. Interactions

Figure 7 shows the typical behavior of the ALIVE controller, in this case at point B in the scenario of Figure 1. At point B, the environment monitor detects the availability of the 802.11 network of hotspot.nl and reports this event to the PDP. The event triggers the evaluation of the discovery policies that reside in the PDP. In this example, the result is that the PDP decides to instruct the ALIVE protocol entity to transmit configuration requests to stream-it.com and media-forward.nl and passes the configuration descriptions in the responses back to the PDP. The PDP's switching policies evaluate the responses and decide that stream-it.com provides BBC 9 o'clock news in the best configuration and control the mobile host to switch in a make-before-break manner (e.g., as part of a highly smooth switching). The PEP first connects the mobile host to stream-it.com and then disconnects it from media-forward.nl. As a result, the mobile host then receives

BBC news from stream-it.com via the 802.11 network of hotspot.nl.

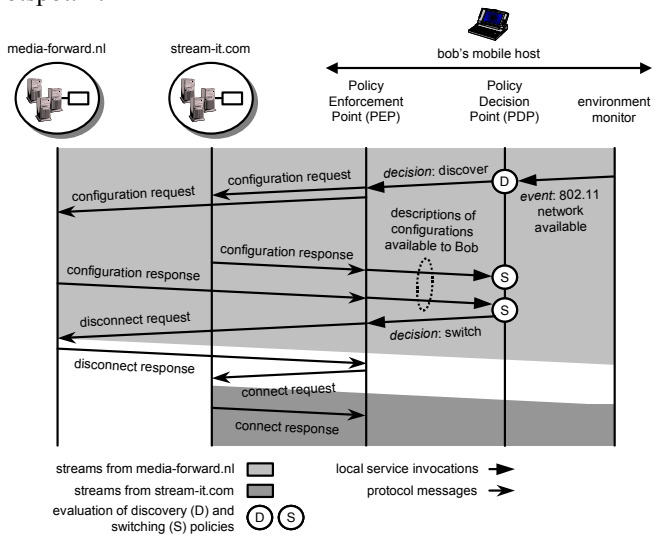

Fig. 7. Typical system behavior

V. PROTOTYPE

To demonstrate the concept of policies and flexibility of policy mechanisms, we implemented the ALIVE controller (Section V.A) and deployed it in a small-scale test bed (Section V.B).

A. Implementation

Figure 8 shows the high-level architecture of our implementation of the ALIVE controller and its relation to the architecture of Figure 5. The controller's PDP consists of an event handler, a policy evaluator, a configuration selector, and a policy manager. The environment monitor is made up of a mobility manager [30] and an RTCP protocol entity [13]. The user preferences and the ALIVE protocol entity map one-to-one to the corresponding component in the controller's architecture.

We implemented the ALIVE controller in C, except for the user interaction part, which we wrote in Tcl/Tk.

1) Mobility Manager

The mobility manager [30] is a system-level service that keeps track of the state of the mobile host's interfaces (e.g., if they are up or down and if they have an IP address or not), the signal strength of the 802.11 network to which the host attaches (if any), of other available 802.11 networks, and so on. The mobility manager reports changes (e.g., the mobile host detaching from a network or a change in 802.11 signal strength) to the event handler.

2) RTCP Entity

The Real-Time Control Protocol (RTCP) is the control part of the Real-Time Transport Protocol (RTP) [13], which is a standardized protocol that multimedia applications can use to packetize compressed multimedia information (e.g., MPEG-2 elementary streams) for transmission over the Internet.

The RTCP entity we used is part of VIC [31], a well-known Internet application for video conferencing. We used VIC in receiver mode to render multimedia streams (see the top right part of Figure 2) and made a small extension to its RTCP code so that it reports current and average packet losses to the event handler.

3) User Preferences Manager

The user preferences manager stores the user's preferences, which for instance consist of a preferred price, a preferred quality level, and a preferred goal for switching between aggregators (e.g., highly smooth switching). The preferences manager loads a user's default preferences from a text file. Its GUI enables users to change the default preferences, for instance by selecting another goal (e.g., from moderately smooth switching to highly smooth switching).

4) Event Handler

The event handler essentially runs the finite state machine of Figure 4. It catches events from the mobility manager, the RTCP entity, and the user preferences manager, and requests the policy evaluator to make a policy decision before it transitions to another state. A policy request includes the information that the event handler received in the event (e.g., a new signal strength level or the new state of the host's 802.11 interface). The event handler enforces decisions on the ALIVE protocol entity by calling its functions.

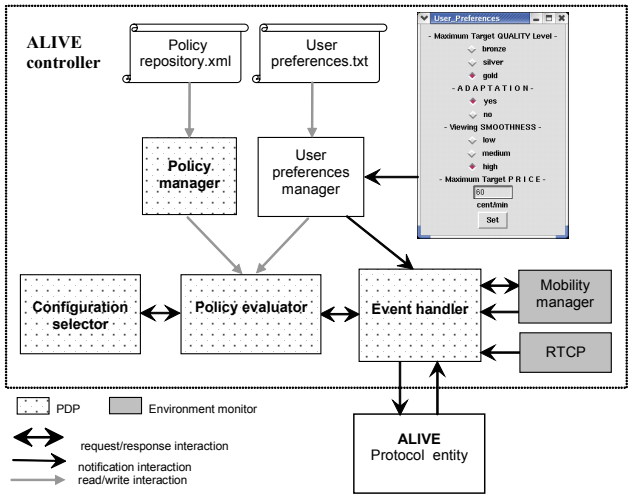

Figure 8. Implementation architecture.

5) Policy Evaluator

The policy evaluator is the policy engine that implements the evaluate_policies call in the finite state machine of Figure 4. The policy evaluator interprets policies by examining a linked list that contains the memory representation (C structs) of the policies that the policy manager (see below) has retrieved from the policy repository. The downside of interpreting policies is that the policy system must be recompiled when new language elements are added to the policy descriptions. An alternative approach would have been to implement policies as runnable code (e.g., as Java scripts) [9,10].

The policy evaluator receives policy requests from the event handler and returns policy decisions. To accomplish this, the policy evaluator checks the information it received from the event handler against the linked list of policies. If the information from the event handler causes the condition of one of the policies to evaluate to true, then the evaluator returns that policy's action as a policy decision to the event handler. A policy decision for instance includes the collection period to use (discovery policy), the best configuration, and how to switch to another aggregator (switching policy).

Observe that the policy evaluator returns immediately when a policy condition becomes true, which implies that at most one policy will fire at a time. This simplifies the ALIVE PDP as it does not have to deal with policy conflicts.

6) Configuration Selector

The configuration selector represents an algorithm to select the best configuration according to the user preferences, available configurations and networks. We separated it from the policy evaluator because of its different computational nature.

7) Policy Manager

The policy manager retrieves XML policy descriptions from a repository file and parses them into the linked list of policies (see Policy Evaluator). The policy manager matches the goals of the policies with the preferences of the user to retrieve the appropriate policy descriptions so that the policy evaluator can use them.

The run time parsing of the XML policies is not yet integrated into the testbed (Section V.B) and remains as a future work.

8) ALIVE Protocol Entity

ALIVE protocol entity is the client-side of the ALIVE protocol (see Section III). It is a thin layer on top of Open SIP [32], an open source SIP stack written in C. Open SIP also contains a parser for the Session Description Protocol (SDP) [18], which is an IETF standard that can for instance be used to describe configurations.

B. Testbed

We set up a small-scale testbed to test our implementation and to experience the flexibility of the ALIVE policies. The idea behind the setup is to emulate the scenario of Figure 1. Since we did not have access to a wide-area wireless network that could carry video streams (typically UMTS), we used an 802.11 wireless LAN to emulate the wide area network and an Ethernet to emulate the hotspot. This enabled us to emulate roaming by plugging the Ethernet cable in and out of the mobile host.

Figure 9 shows the set up. The main components are a laptop, an aggregator server, an 802.11 access point and a fixed Ethernet. The laptop represents the mobile host of Figure 1. It is equipped with an Orinoco 802.11b Gold card and an Ethernet card. The 802.11 base station and the Ethernet LAN form two separate subnets representing two separate network operators. The aggregator server hosts the two aggregator domains of Figure 1 in the form of two processes (see below). The laptop and the aggregator server both run Linux.

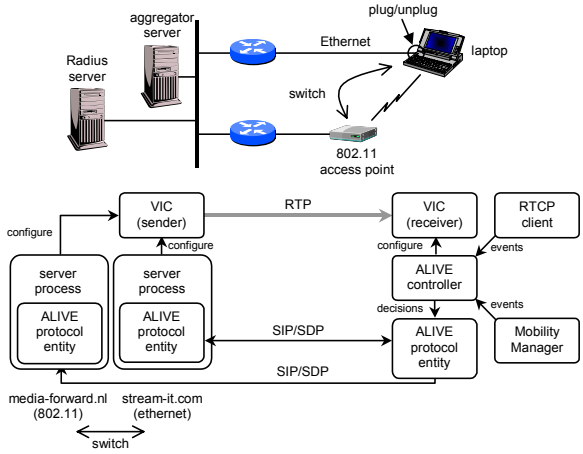

Fig. 9. Testbed

The ALIVE controller, the ALIVE protocol entity run on the laptop. The laptop also runs VIC in receiver mode to receive and render multimedia streams (see Section V.A). The aggregator server runs two processes that execute the server-side of the ALIVE protocol. One process represents aggregator media-forward.nl, the another stream-it.com. The aggregator processes authenticate users with a Radius server.

The aggregator server runs VIC in sender mode to emulate the streaming servers of the two aggregators. The aggregators thus share a pool of media servers in this scenario, which is something that will not happen in practice.

We have extended the VIC software on the aggregator server so that it can interpret configuration descriptions and use that information to configure VIC's framerate, quality, and bitrate (kbps). The extension can also dynamically configure VIC to stream to another IP address, for instance to the IP address of the laptop's Ethernet interface instead of to its 802.11 interface.

As an example scenario, assume that the laptop receives a multimedia stream from 'media-forward.nl' via the 802.11 'overlay network'. If we plug in the Ethernet cable (i.e., enter the 'hotspot'), the ALIVE controller will go through its list of policies and will decide to run the discovery part of the ALIVE protocol. In the process, it will also set the maximum length of the collection period (e.g., 50 milliseconds for highly smooth switching and 1 second for moderately smooth switching).

Next, the ALIVE controller collects the responses from 'media-forward.nl' and from 'stream-it.com' and decides to switch the mobile host to stream-it.com. To accomplish this, the ALIVE controller informs the ALIVE protocol of the decision, which then sends a connect message to stream-it.com. This connect message includes a description of one of

stream-it.com's configurations (the best one).

After the switch, the laptop receives the channel from stream-it.com via its Ethernet interface. The configuration in which it will receive the channel will typically be of a higher quality because the Ethernet has a higher capacity than the 802.11 network.

Figure 10 shows a screenshot of the laptop before and after the switch. The window at the bottom shows a few status lines that the controller prints during the switch.

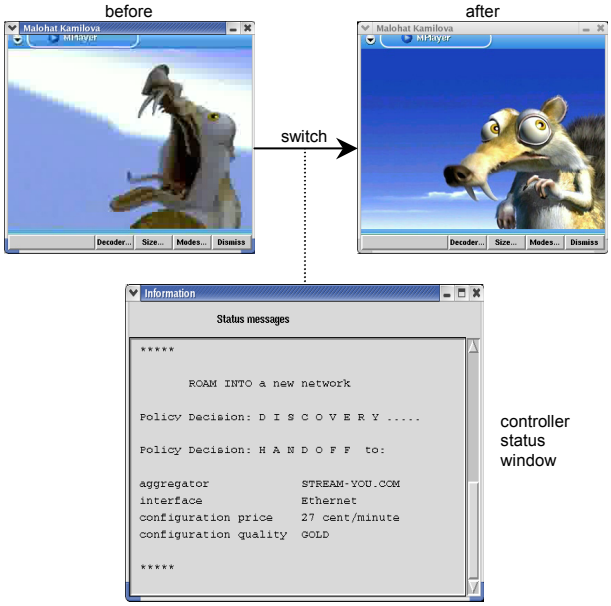

Fig. 10. Screenshots

Another example scenario would be modifying the user preferences while the mobile host is receiving a stream of content. In this case, the ALIVE controller may select and apply a new set of policies to control the behavior of the mobile host in accordance with the new user preferences.

These scenarios demonstrate the flexibility of a policy-based control system, in particular for controlling switches based on the user's preferences and the conditions of aggregators and wireless networks (i.e., the mobile host's context). They also demonstrate the ability of policies to change the behavior of the system on the fly.

VI. Related work

Policies and policy-based systems become popular in designing flexible and adaptive systems [7]. Known policy-based systems for Internet service control typically use network-level policies rather than application-level policies and focus on determining which network (operator) provides the best service [5,33,34]. Wang et al. [33] consider policy-enabled handoffs between different networks. However, we also consider switches between application-level content providers. Furthermore, their policies are based on user preferences such as price, power consumption and quality, whereas our policies are also based on the changes in the environment, such as packet losses, signal strength, and so on. Murray et al. [5] discuss the selection of a best network for a mobile host according to the current load on the networks. The selection in their system is controlled by policy decision logic that sits in the infrastructure, while ours only sits on mobile hosts. Lee et al. [34] take a different approach to determine the best service, which uses prediction functions rather than policies. In their approach the decisions on the best network for the applications running on the user devices are done at a separate device, called a Personal Router, while in our case we applied a distributed approach, so that the control of switching is in the mobile hosts.

VII. Conclusions and future work

In this paper, we propose a policy-based control component that can dynamically switch mobile hosts to the best redistributor (i.e., aggregator) of a particular multimedia channel (e.g., a TV broadcast). The control component runs on mobile hosts and makes policy decisions based on the user's preferences and information about the mobile host's context (e.g., the current signal strength level of a particular base station). Mobile hosts receive the multimedia content via a heterogeneous wireless Internet.

We discuss the architecture of the control component as well as its prototype. The prototype implementation demonstrates that switching between content redistributors using application-level policies is a valid approach. We have also shown the viability of applying the IETF's policy model at the application level for controlling the switching behavior of the mobile host across alternative aggregators.

We are currently experimenting with the performance of the prototype system. In particular, we are investigating its performance in terms of the delays incurred by SIP packets across an unreliable transport (UDP and 802.11). The short-term future work is integrating the XML parser into the code of the policy manager, which will make it possible to modify and download policies into the ALIVE controller at run time.

Acknowledgment

Hans Zandbelt, Remco Poortinga, and Arjan Peddemors (Telematica Instituut) helped with implementing the ALIVE controller.

This work was sponsored by the Awareness project (http://awareness.freeband.nl), which is part of the Freeband Communications Research Program.

References

[1] M. Haardt and W. Mohr, "The Complete Solution for Third-Generation Wireless Communications: Two Modes on Air, One Winning Strategy", IEEE Personal Communications, December 2000.

[2] L. Kleinrock, "An Internet Vision: the Invisible Global Instrastructure", AdHoc Networks Journal, Vol. 1, No. 1, July 2003, pp. 3-11.

[3] C. Hesselman, H. Eertink, I. Widya, and E. Huizer, "A Mobility-aware Broadcasting Infrastructure for a Wireless Internet with Hotspots", In Proc. WMASH'03, San Diego, USA, September 2003.

[4] P. Vidales, R. Chakravorty, and C. Policroniades , "PROTON: A Policy-based Solution for Future 4G devices", In Proc. POLICY 2004, NY, June, 2004, pp. 219-223.

[5] K. Murray, R. Mathur, and D. Pesch, “Intelligent Access and Mobility Management in Heterogeneous Wireless Networks using Policy”, In Proc. the 1st international symposium on ICT, Dublin, Ireland, 2003, pp 181-186.
[6] S. Calo and M. Sloman, “Policy-Based Management of Networks and Services”, Journal of Network and Systems Management, Vol.11, No.3, September 2003, pp. 249-252.
[7] H. Lutfiyya, G. Molenkamp, M. Katchabaw and M. Bauer, “Issues on managing soft QoS requirements in distributed systems using a policy-based framework”, In Proc. POLICY’ 2001, Berlin Heidelberg, 2001, pp. 185-201.
[8] W. Zhuang, Y.S. Gan, K.J. Loh and K.C.Chua, “Policy-based QoS management architecture in an integrated UMTS and WLAN environment”, IEEE Communications Magazine, November 2003, pp. 118-125.
[9] P. Martinez, M. Brunner, J. Quittek, F. Strauss, J. Schoenwaelder, S. Mertens and T. Klie, “Using the Script MIB for Policy-based Configuration Management”, In Proc. NOMS’2002, Florence, April 2002.
[10] R. Montanari, E. Lupu and C. Stefanelli, “Policy-Based Dynamic Reconfiguration of Mobile-Code Applications”, Computer, magazine published by IEEE Computer Society, 2004, pp.73-80.
[11] “A Framework for Policy-based Admission Control”, RFC 2753, January 2000.
[12] “Policy Core Information Model”, RFC 3060, 2001.
[13] “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, January 1996.
[14] C. Hesselman, “Distribution of Multimedia Streams to Mobile Internet Users”, Ph.D. thesis, University of Twente, May 2005, http://www.lab.telin.nl/~hesselma/thesis/thesis_cr_final.pdf.
[15] C. Hesselman, H. Eertink, I. Widya, and E. Huizer, “Delivering Live Multimedia Streams to Mobile Hosts in a Wireless Internet with Multiple Content Aggregators”, to appear in Mobile Networks and Applications journal (Kluwer Wireless), Summer 2005.
[16] C. Hesselman, I. Widya, A. van Halteren, and B. Nieuwenhuis, “Middleware Support for Media Streaming Establishment Driven by User-oriented QoS Requirements”, In Proc. IDMS'00, The Netherlands, October 2000.
[17] “SIP: Session Initiation Protocol”, RFC 3261, June 2002.
[18] “SDP: Session Description Protocol”, RFC 2327, April 1998.
[19] R. Karrer and T. Gross, “Dynamic Handoff of Multimedia Streams”, In Proc. NOSSDAV ‘01, Danfords on the Sound, New York, June 2001, pp. 125-133.
[20] M. Li, M. Claypool, and M. Rinicki, “MediaPlayer™ versus RealPlayer™ - A Comparison of Network Turbulence”, Proc. ACM SIGCOMM Internet Measurements Workshop, Marseille, France, November 2002.
[21] SDR, http://www-mice.cs.ucl.ac.uk/multimedia/software
[22] “Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers”, RFC 3361, August 2002.
[23] “Terminology for Policy-Based Management”, RFC 3198, November 2001.
[24] M. Cox and R. Davison, “Concepts, Activities and Issues of Policy-based Communications Management”, BT Technology Journal, V 17, Issue 3, July 1999, pp. 155-169.
[25] C. Hesselman, H. Eertink, and A. Peddemors, “Multimedia QoS Adaptation for Inter-tech Roaming”, Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC’01), Hammamet, Tunisia, July 2001.
[26] G. Pollini, “Trends in Handover Design”, IEEE Communications Magazine, March 1996, also available through IEEE Communications Surveys, http://www.comsoc.org/pubs/surveys/pollini/pollini-org.html.
[27] P. Linington, Z. Milosevic, and K. Raymond “Policies in Communities: Extending the ODP Enterprise Viewpoint”, In Proc. EDOC'98, San Diego, USA, page 11, November 1998.
[28] N. Damianou, N. Dalay, E. Lupu and M. Sloman, “Ponder: A language for specifying security and management policies for distributed systems”, language specification (version 2.1), Imperial College Research Report DOC 2000/01, Imperial College of Science, Technology and Medicine, London, England, April 2000.
[29] “The COPS (Common Open Policy Service) Protocol”, RFC 2748, January 2000.
[30] A. Peddemors, H. Zandbelt, and M. Bargh, "A Mechanism for Host Mobility Management supporting Application Awareness " , In Proc. MobiSys'04, June 2004.
[31] VIC, http://www-mice.cs.ucl.ac.uk/multimedia/software/vic
[32] oSIP webpage, http://www.gnu.org/software/osip/
[33] H.Wang, R. Katz, and J.Giese, “Policy-Enabled Handoffs Across Heterogeneous Wireless Networks”, In Proc. WMCSA 1999, New Orleans, USA, February 1999.
[34] G. Lee, P. Faratin, S, Bauer and J. Wroslawski, “Automatic Service Selection in Dynamic Wireless Network Environments”, poster MobiCom’03, SanDiego, USA, September 2003.