An Agent-Inspired Active Network Resource Trading Model Applied to Congestion Control

Lidia Yamamoto, Guy Leduc

Research Unit in Networking, Institut Montefiore, B28, B-4000 Liège, Belgium
yamamoto@run.montefiore.ulg.ac.be
WWW home page: http://www-run.montefiore.ulg.ac.be

Abstract. In order to accommodate fluctuations in network conditions, adaptive applications need to obtain information about resource availability. Using active networks, new models for adaptive applications can be envisaged, which can benefit from the possibility to send mobile code to the network nodes. We describe a model for trading resources inside an active network node, based on the interaction between capsules as reactive user agents, and resource manager agents which reside in the network nodes. We apply the model to the case of a many-to-one audio application with congestion control, which trades off link resources against memory when there is congestion at the outgoing interface towards the destination. Our simulation results indicate that the application makes effective use of the available resources, and it also allows resources to be shared according to user preferences.

1 Introduction

Adaptive applications can tolerate fluctuations in resource availability, and are becoming increasingly important in a strongly decentralised and heterogeneous environment such as the Internet today. In such a complex network, different network technologies and user terminals are interconnected together, and a multitude of applications and services must coexist.

An adaptive application must be able to make optimal use of the available resources, and be able to adapt itself to fluctuations in resource availability. Code mobility as provided by active networks can become a useful tool to help in the adaptation process, since it becomes possible to inject customised computations at optimal points in the network.

A considerable amount of research has been dedicated to algorithms inspired on optimisation and economy theories to control resource usage in networks. In the context of adaptive applications assisted by active networks, the benefit of such techniques is two-fold: on one side, optimal resource sharing configurations can be achieved in a decentralised way; on the other side, it becomes easier to quantify heterogeneity in terms of resource availability, to offer the users the opportunity to trade one type of resource for another.

However, relatively few results have been shown which directly apply such artificial economy models to the specifics of active networks, with special attention to highly adaptive applications. We address this issue in this article by

Second International Workshop on Mobile Agents for Telecommunication Applications (MATA 2000)
Eric Horlait (Ed.): LNCS 1931, pp. 151-169. Paris, France, September 2000.

providing a simple model which allows the active applications to make decisions about the amount of resources to use, according to the network conditions found in the active nodes. Using such a model, an audio mixer is developed as an instance of adaptive active network application, which is able to trade bandwidth for memory according to the available prices of each resource.

2 Background

2.1 Active Networks

Active networks (AN) allow the network managers or users to program the network nodes according to their needs, offering a great amount of flexibility. The nodes of an active network [21] are capable not only of forwarding packets as usual but also of loading and executing mobile code. The code can be transported within specialised signalling channels (programmable networks) or within special packets called "capsules" (active networks). Capsules might contain the code itself (such as in [10]) or a reference to it, such that it can be downloaded when the first capsule containing the reference arrives at a given node (such as in [23]). If the distinction between active and programmable networks seemed at some point in time clear [6], the tendency today seems to be towards an integration of the two concepts, since both are forms of achieving open programmability in networks [5], and special flavours in between or combining both approaches are also possible [11].

A framework for an active node architecture is being proposed in [3]. It includes a supporting operating system (the NodeOS), one or more execution environments (EE), and the active applications (AA). The NodeOS is responsible for managing local resources such as CPU processing time, link bandwidth and memory storage. On top of the NodeOS, a number of EEs can be installed. On top of each EE, various AAs can be dynamically loaded and executed. The EE is responsible for controlling the access from the AAs to local resources, and limiting resource usage depending on specified policies.

The NodeOS plays a crucial role in providing access to local node resources, as well as information about resource availability. A NodeOS API is currently being defined [18]. At the moment this API treats four types of resources: computation, memory, communication, and persistent storage. The communication resource is handled through the channel abstraction, which when ready should include QoS support, as well as access to link information such as bandwidth, queue length, and other properties and statistics.

When not all the network nodes are active, it is necessary to discover resources outside an active node. For this purpose, complementary efforts such as CMU Remos [15] could be used. The CMU Remos interface enables network-aware applications to obtain network properties such as topology, latency and bandwidth. Another interesting approach appears in [20], where an *equivalent link* abstraction is proposed, such that from an AN point of view it is possible to look at a set of non active nodes as a single link, with some mechanisms needed to discover the (possibly changing) properties of such a virtual link.

2.2 Mobile Agents and Active Networks

Mobile agents are autonomous pieces of mobile code that travel through the network acting on behalf of their owners. The intersection between mobile agent technology and active network technology is the use of mobile code. The capsules of an active network can be seen as subclasses of mobile agents, specialised for network-related operations.

However, in practice many differences subsist: mobile agents concentrate on application-level duties, and can generally accomplish much more complex tasks than what is generally allowed to capsules. The architectures for mobile agent platforms and active network platforms differ in the kind of support for code mobility that is offered. Mobile agent platforms tend to concentrate on application level services and active network platforms are optimised for transport rather than processing of information.

2.3 Resource management

One of the main difficulties encountered in classical adaptation approaches is how to obtain the required information about resource availability, mainly when this information is hidden in a blackbox network and has to be inferred using only some indirect indications that are observed at the end systems. Using active networks, new models for adaptive applications could be envisaged, which can benefit from the possibility to send capsules or agents to certain elements inside the network. These agents can be in charge of collecting information about network conditions, without having to rely on indirect indications or on heavy signalling protocols. Indeed, the idea of sending small pieces of code directly to where the data needs to be treated, instead of exchanging a large amount of data, is one of the main motivations of mobile agent technology, and it can also be applied to mobile code in the case of active networks.

Actually many adaptation mechanisms come from the world of mobile agents. Some examples are: In [2] the problem of budget planning for mobile agents is addressed, such that they can successfully complete their tasks given their limited budget constraints. In [22] an open resource allocation scheme based on market models is applied to the case of memory allocation for mobile code. In [12] an adaptive QoS scheme for MPEG client-server video applications is described. It is based on intelligent agents that reserve network bandwidth and local CPU cycles, and adjust the video stream appropriately. In [24] a market model to allocate QoS is applied to a conferencing tool targeted at casual meetings where sudden variations in bandwidth availability require an adaptive QoS control strategy.

A lot of work has been done in applying optimisation and economy theories to the distributed allocation of resources in networks [7]. The algorithms derived from such studies are very promising since they converge towards a global optimum in a decentralised way. Several authors (see [14] for an overview) have applied such theories to the problem of end-to-end congestion control, i.e. where

bandwidth is the main scarce resource. The results shown are promising since they are generic enough to be adapted to a wide variety of applications.

Active networks can more easily benefit from such algorithms compared to classical networks, since their code can be dynamically deployed. However, the current AN architectures still offer little support for such algorithms to be implemented in a straightforward manner. The main difficulty is with respect to the interfaces to local information concerning resource availability, which need to be exported to the active applications. In this matter, we can learn from the mobile agents field in order to model the interactions between resource manager agents and user agents. Software agent communication paradigms such as Agent Communication Languages can be helpful, but still need to be specialised to the AN context.

A cost model for active networks is proposed in [17], which expresses the trade-off between different types of resources in a quantitative way. However, the recursive approach adopted makes its use more appropriate in the context of reservation-based applications, instead of highly adaptive ones.

3 Resource Trading Model

In previous work [25] we studied the potential and limitations of active networks in the context of adaptive applications, with a case study on a layered multicast application. These first results led us to propose a protocol that we called Active Layered Multicast Adaptation (ALMA) [26]. As a side effect of designing ALMA, a slightly different model of an active node popped up, in which the intelligence is distributed among the active applications representing the user interests, and the active resource managers representing the network provider interests. Market-based techniques are naturally suitable for such a model. In the present paper we push the idea forward by analysing the impact of its various facets on the domain of adaptive applications over active networks.

Our model aims at offering a generic communication abstraction between active network agents, such that different adaptive applications and different resource management policies can be implemented. We are mainly concerned with adaptation to available resources when resources cannot be reserved in advance, either because the router itself does not support reservations, or because the network is heterogeneous and some of the routers along the path offer no QoS support. The idea is to enable auto-configurable applications and resource managers, such that the code from both types of agents can be dynamically loaded, in order to make them evolve to adapt to new conditions.

Figure 1 shows the model as it could be implemented over the DARPA AN architecture.

In the model, two types of agents communicate to seek an equilibrium. Each agent tries to optimise its own benefits: on one side resource manager agents have the goal of maximising resource usage while maintaining a good performance level. On the other side, user agents try to obtain a better quality/price relation for the resources consumed, and to efficiently manage their own budgets avoiding

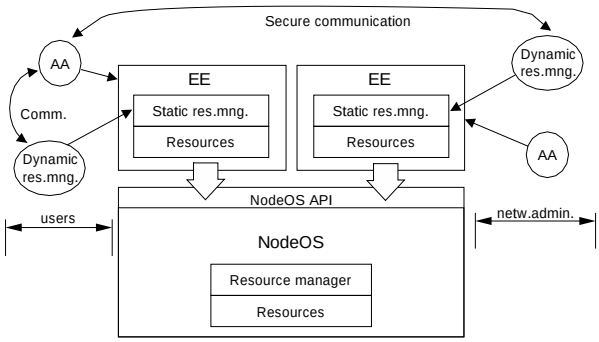

Fig. 1. Resource managers and active applications. Hypothetic placement over the DARPA AN architecture

waste. Both types of agents are implemented as AAs with different privileges, and they communicate such that the resource managers can "sell" resources to the user agents at a price that varies as a function of the demand for the resource.

A currency is introduced into the system to allow for trading of different resource types. This is the basic requirement for artificial economy models such as [2, 7, 22] to develop in active nodes, and is also an essential feedback parameter for most algorithms based on optimisation (e.g. [14]).

Such a model is per se not entirely new, and is in fact a mere simplification of existing artificial economy models which have been mainly applied to the agents domain. For example, in [9] a similar although more complex model is presented, and applied to a circuit switching set-up of paths (reservation-based). The main difference with respect to our model is that we try to adapt it to the specifics of adaptive applications over active networks (no resource reservations), in which reaction time is critical, therefore precluding the use of complex transactions.

3.1 Resource Manager Agents

Resource managers export resource prices which are a function of the resource utilisation. The utilisation is related to the load, and to the demand for a resource. The function or algorithm used to calculate prices can be shaped to implement desired policies, such as to achieve high utilisation, but also to offer good quality to the users. Resource managers may contain dynamic and/or static code. Some lower level functions which are especially time-critical might be implemented using static code (or even in hardware) while the mobile part would be used to implement more complex policies and to select from a set of pre-existent lower level functions. It is necessary to have the possibility to use dynamic code in order to be able to improve strategies, that is, make them evolve over time, in an active network. Here is one of the places where the alliance between AN and mobile agents can become a must: resource managers

can be deployed using mobile agents that are sent by the network manager in order to install new policies. Classical mobile agents for network management can be used for this purpose.

Resource managers are implemented as AAs injected by the network provider, which are executed with network administrator's privileges. There is one class of resource manager for each type of resource concerned. The most relevant classes are: link managers, CPU managers, and memory managers. We will give some more attention to link managers, since they play a crucial role in network congestion control.

A link manager is a resource manager responsible for the resources of a link, such as bandwidth and outgoing queue space. It exports a link price which is a function of the current load on the link. The price works as a congestion indication [14]. Since the link managers are controlled by the network provider, as the other resource managers, the shape of the function is a provider's choice, and can be updated to achieve a desired effect, such as to encourage or discourage bursty traffic, to achieve a certain level of utilisation, to converge faster or smoother, etc. For example, a binary feedback scheme such as the one provided by RED active queue management [8] can be regarded as a special case of pricing indication for TCP/IP networks. A RED gateway avoids bias against bursty traffic by dropping or marking packets with a probability that is a function of the average queue size. In [13] a marking scheme called REM is proposed, which is based on optimisation results and is shown to achieve better throughput and fairness than RED, while at the same time presenting no packet losses. In [1] it is shown that faster convergence to the optimum bandwidth shares can be achieved by only changing the price adjustment algorithm from the simple gradient projection used in [14] to a scaled gradient projection using an approximate Newton method.

Link manager agents also allow abstractions to be made which enable the user applications to adapt to a wide variety of environments in a transparent way. By exporting prices instead of the link internal state information directly, it is possible to hide the specific details of link characteristics while at the same time offering a proper congestion indication to adaptive applications. For example, the price function for a classical point-to-point link would be different from that of a wireless link, where the local interface load is not a good indication of the actual link utilisation.

An important feature of the link manager price abstraction is that it allows the active applications to deal with non-active nodes in a transparent way. For example, a link manager that implements the equivalent link abstraction [20] could export a price which would be a function of the estimated average rate, average delay and packet loss probability, which are changing properties in the case of an Internet virtual link. This type of abstraction is crucial for the success of active networks, since the deployment of active nodes will depend on their ability to complement and interwork with existing technologies.

A lot of research is still to be done on how to adjust prices in artificial agent-based economies. An interesting analysis can be found in [16]. For this

paper however this will not be our focus. We will rather concentrate on the user side, assuming that the resource manager agents in the active nodes are able to implement suitable pricing algorithms.

3.2 Capsules as Reactive User Agents

The second type of agents in the model are the active packets themselves, modelled as simple reactive mobile agents. Actually capsules can be regarded as a specialised subclass of mobile agents. They travel to network nodes where they decide when to continue or stop the trip (e.g. stop due to congestion), when to fork new capsules (e.g. in a multicast branch), and which amount of resources to use at each node in order to complete their (generally simplified) tasks.

Capsules have the properties of autonomy and mobility, but need to be simple enough to be executed at the network layer, where performance is often critical. They need to take fast decisions using little data and reasoning, therefore they must be extremely reactive. They can therefore be classified in the reactive agents category. This does not exclude from the model the possibility of using more intelligent mobile agents, jointly with capsules. However, in this paper we focus only on capsules.

Capsules generally represent the user interests, with the goal of attaining the best possible quality at the lowest possible costs. This means that a capsule must be able to make rational decisions on how to spend its limited budget, after consulting resource manager agents for information about prices of the various resources need.

Capsules carry a budget that allows them to afford resources in the active nodes, as the resource limit field in ANTS [23]. It looks like a TTL (Time-To-Live) field which is decremented by n units for the consumption of a certain amount of resources. When it reaches zero the capsule is discarded. As explained in [23], the budget field must be protected such that the capsules themselves cannot modify it. Again, since capsules are reactive agents, the transactions must be kept simple enough. Instead of the auction mechanisms frequently used by agents [7], the capsules will most of the time either accept or refuse to pay a given price. Refusing might imply that the capsule simply disappears from the system, because it does not have enough budget to proceed its journey to its destination. Again, we do not preclude the usage of more complex mechanisms implemented by intelligent mobile agents. These agents can also have access to the resource manager interfaces, therefore they can also benefit from the model, however they are not our focus as pointed out earlier in this section.

At the end systems (hosts) conventional programs or intelligent agents can be used to spawn capsules. Since these hosts can dedicate significant amount of resources to information processing, sophisticated strategies can be used to decide which capsules to spawn and when, and how much of the total user's income can be assigned to each capsule (planning). These tasks are then not delegated to the capsules themselves, but can be delegated to more generic mobile agents such as the ones described in [2].

Here we face the problem of how the budget should be distributed between capsules for one user, and among different users, and also of how to make purchase decisions according to budget constraints. This can be done with the help of utility functions, which quantify the level of user satisfaction for receiving a certain amount of a good, or more generally a combination of goods (market basket) [19].

According to economics, a typical behaviour of a rational consumer agent is to try to maximise its utility subject to the budget constraints given by their limited income.

For an example where only two goods are involved, the user optimisation problem is then typically expressed as:

$$Maximise\ U(x,y) \tag{1}$$

subject to:

$$p_x x + p_y y = I \tag{2}$$

where $U(x,y)$ is the utility function, x and y are the quantities of two goods in their respective units, p_x is the price per unit of x, p_y the price per unit of y, and I is the user's income.

Such a maximisation process will lead to an equilibrium if the utility function satisfies some properties such as being strictly concave increasing, i.e. the increase in satisfaction is smaller the more items of one good are consumed. The increase in satisfaction that a user obtains from consuming one more item of a given good is called the marginal utility. Within the rational consumer assumption, the marginal utility is a decreasing function of the number of items of a given good. This is called the diminishing marginal utility assumption, and it is directly related to the demand function of a given user for a given good. It means that the more one has from something, the less it is willing to pay to obtain more of it.

A typical utility function is the Cobb-Douglas utility function, given by:

$$U(x,y) = a \cdot log(x) + (1-a)log(y) \tag{3}$$

where $0 \leq\ a \leq 1$ is a constant which represents the importance the user assigns to x with respect to y.

Applying the method of Lagrange multipliers, the solution of the user optimisation problem with the Cobb-Douglas utility function is given by [19]:

$$x(p_x) = \frac{a \cdot I}{p_x} \tag{4}$$

$$y(p_x) = \frac{(1-a)I}{p_y} \tag{5}$$

The functions $x(p_x)$ and $y(p_x)$ are the demand functions for goods x and y given their current prices per unit p_x and p_y respectively. With demand functions shaped like these, it is possible to chose the quantity of a given resource to

consume in order to maximise user satisfaction, given the current price for the resource and the agent's income. Since the price information for each type of resource is available in the active nodes, it is possible for an agent to calculate the amount of resources it can consume at each node, given only an upper bound on the budget per hop it has planned to spend. In the case of capsules, the planning decisions to calculate this upper bound could be performed either at the hosts or by more intelligent mobile agents that would be sent to the active nodes less frequently than capsules. The techniques for such planning are out of scope of this article, and the interested reader can refer to [2] for more information.

The same reasoning can be easily generalised to an arbitrary number of resources [7]. However, for the purpose of our study this will be sufficient, since we will restrict ourselves to link and memory resources.

Another important characteristic of the Cobb-Douglas function is that it allows us to easily quantify the preferences of different consumers towards one good. For example, given two user agents u_1 and u_2 with the same income I, if agent u_1 has the weight a_1 on its utility function for good x, and agent u_2 has weight $a_2 = n \cdot a_1$ for the same good, we have:

$$x_1(p_x) = \frac{a_1 \cdot I}{p_x} \tag{6}$$

$$x_2(p_x) = \frac{a_2 \cdot I}{p_x} = \frac{n \cdot a_1 \cdot I}{p_x} = n x_1(p_x) \tag{7}$$

Thus for a given price p_x we have:

$$a_2 = n \cdot a_1 \Rightarrow x_2 = n \cdot x_1 \tag{8}$$

It means that if we know that agent u_2 values resource x twice as much as agent u_1, then when faced with the same price, u_2 will get twice as much of x as u_1. This is an interesting tool for computer network applications, since it provides a quantitative way to provide service differentiation according to the preferences of users. This capability is already known from literature, e.g. [7, 14]. For example, when trading bandwidth for memory storage, a time-constrained application such as an audio-conference will certainly prefer bandwidth to storage if prices are the same, while a bulk transfer application would probably prefer to store as much information as possible when the links are congested, in order to avoid losses and retransmissions.

In real world economies it is difficult to quantify utilities, but in artificial economies this might be less difficult. Without having to relate artificial currencies to real ones, we could imagine that the maximum budget per unit of time is controlled by a policy server from the network provider that the user is subscribed to, in order to guarantee that users will employ such budget rationally and prevent malicious users from grabbing most of the resources by marking all their capsules with a high budget. This can be compared to the IETF diffserv policies. However, in diffserv only a few predefined classes of service are available, while with such artificial economies, a whole range of classes could appear (and eventually die out), defined by their particular utility characteristics.

4 Congestion Control for a Concast Audio Mixer

We illustrate the use of the trading model through a congestion control scheme for a many-to-one (concast) service. The concast example shows capsules that trade bandwidth for memory when there is congestion.

The term "concast" has been defined in [4] as a many-to-one service, in opposition to multicast (one-to-many). Figure 2 illustrates this concept. While multicast copies information from one source to many destinations, concast merges information from several sources to one destination. A concast service can be used, for instance, to aggregate feedback in a reliable multicast service, to transmit reception statistics in a multimedia session, to merge information coming from several sources in an auction or tele-voting application, or to combine several real-time streams into one, e.g. to perform audio mixing from several audio sources.

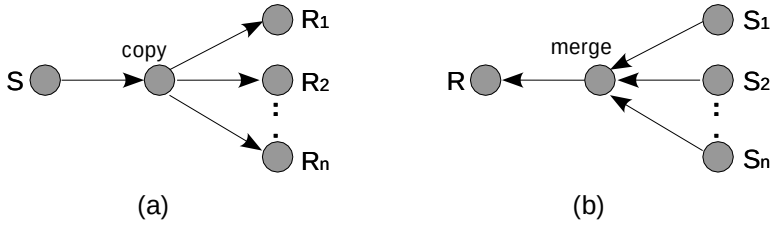

Fig. 2. Multicast (a) versus Concast (b) service abstractions.

The concast service is faced with the feedback implosion problem, that is, multiple simultaneous sources might congest the path to the single destination, if no congestion control is performed. This problem is aggravated by the fact that many concast flows are used as signalling to support a more robust protocol such as the aggregation of NACK feedback messages for a reliable multicast protocol. Flow control for such signalling messages is often neglected or oversimplified, since the signalling traffic is assumed to be kept small enough when compared to the data traffic. This might lead to poor performance when there is congestion in such a signalling path. We argue that for future sophisticated active services congestion control will be equally important on the signalling and data paths, since the distinction between both tends to become naturally blurred as we approach new network service composition frameworks which are not simply stack-based as the classical OSI model.

For the purpose of this study, we focus on the case of an audio mixing application. However, the same ideas are applicable to any application making use of the concast service, differing only in the way that packets are combined (merge semantics).

In our audio mixing application, several sources generate audio streams in a session. The streams are collected at a single node, which can either record them, play them back or redistribute them to a multicast group. If the receiving node collects all the data and mixes it locally, it might end up with the implosion problem. It is possible to perform mixing operations at every active node, so that the receiver gets a single stream already mixed. However this might cost too much processing and/or memory space in the active routers, so it might become too expensive. Also note that mixing streams delays them, since it is necessary to wait until a minimum number of packets arrive in order to sum up their audio payloads.

The mixing operation is a physical sum of audio samples. Therefore the resulting audio payload has the same number of bytes as each of the original payloads. We assume that there is a maximum amount of signals that can be added up without saturation. Another assumption is that a constant bit rate codec is used, with no silence suppression, such that the signals are generated at a constant rate from the beginning to the end of the session. This initial rate will be altered along the path, as capsules are mixed according to the network conditions.

Note that the mixing operation requires a list of addresses, in order to identify the list of sources already mixed. One might argue that such a list might also occupy bandwidth, since the packet size has to increase in order to accommodate it. But if an estimation of the group size is available, a fixed-size bitstream can be used to hold the list of sources. Adding or removing a source becomes as simple as setting or resetting a bit in the bitstream. The intersection and union operations can also be implemented as AND and OR binary operations respectively.

We propose to trade bandwidth for processing and memory space to achieve a compromise solution which uses resources efficiently and therefore is able to control congestion. The idea is that when there is congestion at an outgoing interface, the consequently high bandwidth prices will push the users to save bandwidth by performing mixing of data. When the congestion clears up, the users can again benefit from the available bandwidth to avoid the extra delays imposed by the mixing operation.

This application needs only one type of capsule: the audio data capsule, which carries the audio payload from a source, or a list of sources, to the destination. The capsule consults the link managers and memory managers in the active nodes along the path, in order to decide whether to mix with other capsules of the same session, or to proceed its journey to the destination. For simplification purposes, the CPU manager is not involved. We assume that CPU costs are either negligible or can be combined with memory costs.

When a capsule arrives at a node, the first thing it does is to check whether another payload coming from the same source is already buffered. In this case, the arriving capsule cannot mix its payload to the buffered one, which carries earlier samples. Note that this check is in fact an intersection operation to check whether one of the sources mixed in the current capsule is already buffered. If that is the case, the arriving capsule immediately dispatches the buffered payload by

creating a new data capsule and injecting it into the local execution environment. This procedure also prevents misordering of packets. As a consequence, at most one packet payload per session is buffered in an active node. We assume that all payloads have the same size.

After that, the capsule can take a decision to either proceed, buffer its payload (for mixing), or discard itself. This decision depends on the budget it carries, and on the prices of the memory and link resources it needs. A number of alternative decision strategies can be envisaged:

Null strategy: Corresponds to the trivial case when no congestion control is performed. In this case the capsule always goes intact to the outgoing interface.

First strategy: If there are other audio payloads from the same group waiting in the memory buffer, it adds its own payload to the buffered one and adds its list of sources to the list of sources already buffered (union operation). Then it terminates execution. If no other audio payloads from the same group are buffered yet, it decides for the cheapest resource: if the price of memory (to buffer the payload for future mixing) is currently lower than the transmission price for the capsule, then it decides to buffer itself; otherwise it decides to move on to the next hop.

Second strategy: The arriving capsule mixes its own data with the buffered one (if existent), then it chooses the cheapest resource, either memory or bandwidth.

The difference between the first strategy and the second strategy is that the first strategy always decides for storage when another payload from the same session is already buffered, while the second strategy always chooses the cheapest resource, independent on the fact that another payload is already buffered or not.

Although the first and second strategies are very naive, they already give quite reasonable results as we will see below. However, their behaviour is suboptimal and they do not take into account the different preferences for resources.

Third strategy: It first mixes its own data with the buffered one (if existent), as in the second strategy. It obtains a new payload that combines samples from n sources (n is known from the list of source addresses). It also knows N, the maximum number of sources that can be mixed together without saturation. Then it calculates the amounts of link and memory resources according to equations 4 (for link) and 5 (for memory), where the a parameter is also carried in the capsule, and expresses its preferences for link resources with respect to memory. It then tries to keep the resources in the proportions obtained, as follows:

If $\frac{N-n}{n} > \frac{x}{y}$ then store, else move on.

Where x is the demand for link resources according to equation 4, and y is the demand for memory resources according to equation 5.

Since $\frac{x}{y} = \frac{a \cdot p_y}{(1-a)p_x}$, the decision is independent on the capsule's budget I. The a parameter will play a role in the proportion of link resources used with respect to memory resources. The higher a, the higher the amount of link resources used,

and therefore less capsules will be mixed together, for given memory and link prices.

Note that in all cases, capsules that run out of budget are automatically discarded by the resource managers, thus there is no need to explicitely indicate this operation.

5 Simulations

The audio mixing AA has been simulated with the help of an AN module that we developed for the NS simulator [27]. This module implements a simplified AN architecture consisting of a NodeOS, an EE, and some resource managers. The simulated EE executes capsules written in TCL language.

The topology for the simulations is shown in figure 3. It consists of n sessions of m sources and one receiver each. The sessions traverse a bottleneck (link L), so that the capsules in active node N must decide to mix or to proceed intact to the receiver node, according to the prices of link or memory resources available from the resource managers.

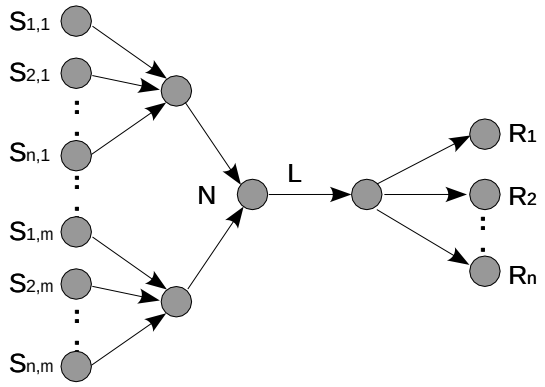

Fig. 3. Topology used in the simulations.

The price function used is based on the one in [22]:

$$price = 1000 \cdot \sqrt{\frac{1.01}{(1.01 - load)}}) - 1000 \tag{9}$$

This function is a practical implementation of a convex increasing function that would go to infinite as the load approaches 100%. It forces the price to rise sharply as we approach high loads, which discourages applications from using a

resource when its load is too high. This gives the applications a clear indication of the "dangerous zone" to avoid, while at the same time encouraging a relatively high utilisation.

In the case of the memory manager, the load is given by the ratio between the average number of memory units occupied, and the total number of memory units available to user capsules (which is of course assumed to be much smaller than the actual amount of memory available). The average is calculated using an exponential weighted moving average (EWMA [8]).

For the link manager, the load is given by the average queue occupancy ratio at the outgoing link interface. This average ratio is obtaining by calculating the average queue length as an EWMA, and then dividing by the maximum queue size. The resulting congestion indication is a bit similar to RED, except that here the binary feedback is replaced by an explicit price indication to the arriving capsules. Note that the actual usage of bandwidth is not taken into account in the price function, if it does not cause queues to build up. This is therefore a very simplified version of a link manager, but it already serves the purpose of controlling congestion.

We first run an example where 2 sessions are active. There are 5 sources per session, each sending an audio stream of 100kbps to a single receiver, resulting in a total of 1Mbps of traffic arriving at N. The capacity of link L is set to 500kbps.

Figure 4 (top) shows what happens in the trivial case when no congestion control is used. In this case, link overflow occurs at L, and half of the packets are dropped. The remaining packets receive an unequal share of the bottleneck link as we can see in the figure.

Figure 4 (middle) shows what happens when the first strategy is used. First we can notice that the two sessions (left and right) get approximately the same share of the bottleneck. Additionally, no packet losses were observed during the simulation. However, the link is underutilised. This can be explained by the fact that this strategy always favours memory when there is already an item in memory.

The second strategy is a bit more clever (Figure 4, bottom). It always chooses the cheapest resource, either memory or bandwidth. Therefore it is able to grab any bandwidth when it becomes available. Here again, no packet losses occur. However, with this strategy it is not possible to specify different weights for each resource.

Now let us look at the third strategy, which allows us to specify utility weights through the a parameter. Figure 5 (top) shows its behaviour when the weights of the two sessions are the same. We see that this strategy is able to share the bandwidth efficiently. It also leads to more stable rates when compared to the previous strategies. The same happens when the weights are different (bottom side of Figure 5), but in this case each session receives link resources in proportion to its respective weight, expressed by the a parameter value.

Until this point only the rates have been shown, since our main goal is to achieve congestion control. Table 1 shows average memory and link parameters taken over the complete duration of the simulations shown in figures 4 and 5. The

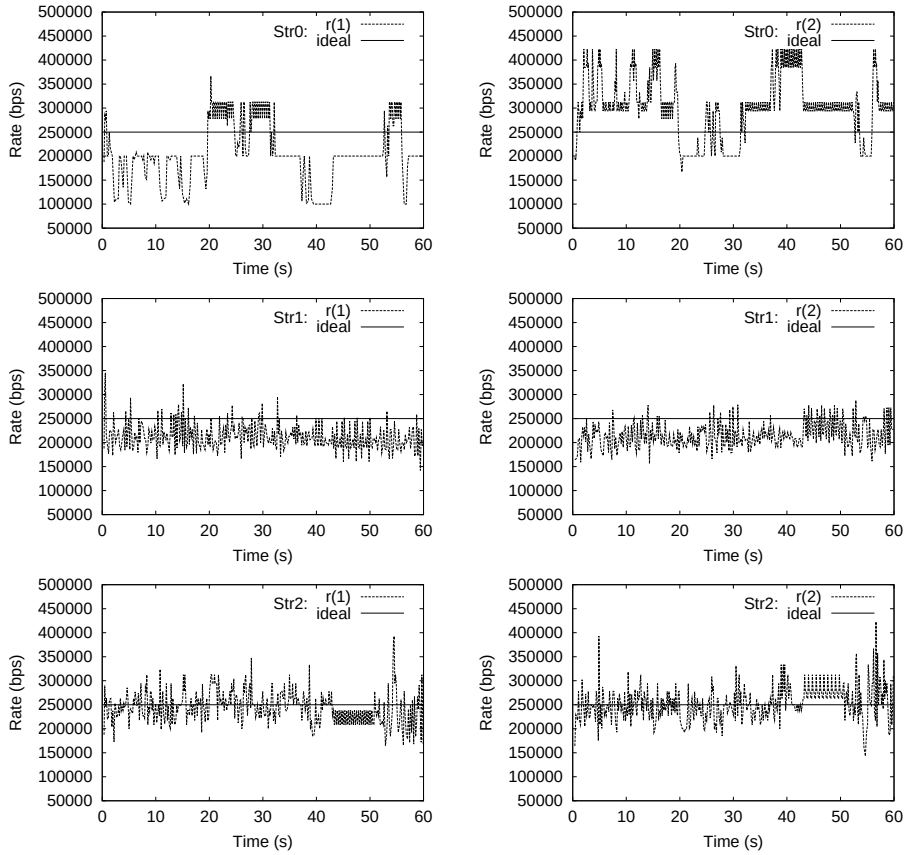

Fig. 4. Evolution of rates in time for 2 sessions, as perceived by their respective receivers. Left: first session (receiver r_1). Right: second session (receiver r_2). Top: no congestion control. Middle: first strategy. Bottom: second strategy

null strategy (Str.0, no congestion control) occupies most of the link resources and causes the link price to rise, since the link queue is most of the time full. Strategy 1 reduces link utilisation by using memory for mixing, however it does that in an inefficient way when compared to strategy 2, which is able to use more bandwidth while decreasing both link buffer occupancy and memory utilisation. The result is a decrease in link and memory prices.

Strategy 3, when $a_1 = a_2$ (Str.3a in Table 1), improves further by keeping the average link queue occupancy at a very low level, in spite of a high bandwidth utilisation. This can be explained by the fact that this strategy tries to find an optimum balance between the usage of memory (which delays packets) and the usage of link resources. It therefore waits for the good moment to send a packet over the link, the moment when link prices are favourable (which in our case corresponds to low queue occupancy).

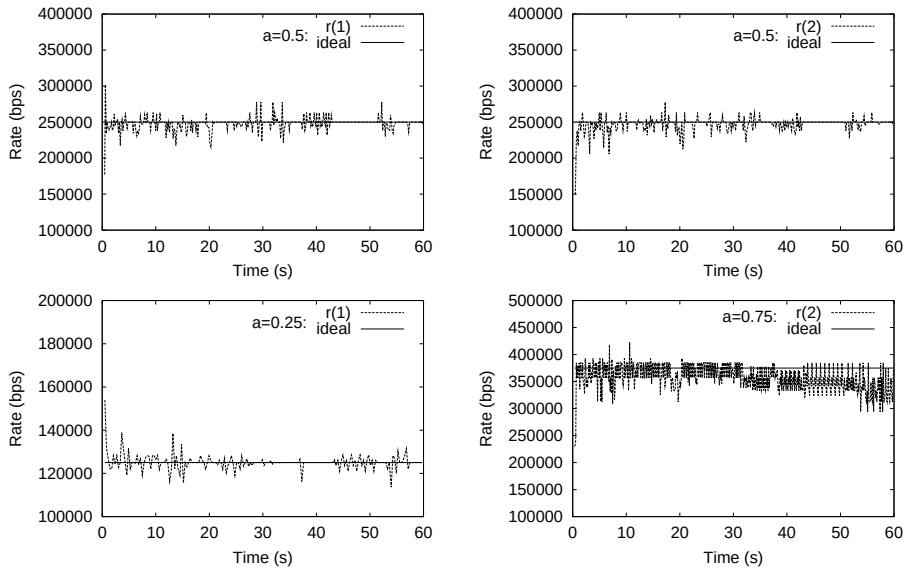

Fig. 5. Evolution of rates in time for 2 sessions using the third strategy. Top: $a_1 = a_2 = 0.5$ for link resources. Bottom: $a_1 = 0.25$, $a_2 = 0.75$

Table 1. Memory and link usage parameters for different strategies

Parameter (average)	Str.0	Str.1	Str.2	Str.3a	Str.3b
memory utilisation (%)	0	7.18	5.14	4.98	5.33
bandwidth utilisation (%)	99.91	84.54	98.18	98.93	96.37
link queue occupancy (%)	93.58	5.83	4.97	1.61	2.98
memory price ($)	0	37.71	26.57	25.62	27.37
link price ($)	2914.60	32.35	27.71	10.17	17.22

Column Str.3b in Table 1 shows the average resource parameters when $a_1 \neq a_2$, corresponding to the situation illustrated at the bottom side of Figure 5. There are no significant changes in the parameters at node N, with respect to Str.3a, but the prices are higher, which is an unexpected result, as we would expect the two sessions to concentrate each one on its respective preferred resource, making the load equally distributed. We are still investigating the reason for this discrepancy.

In order to have a better insight on the mixing procedure, we now look at the number of audio sources carried by each capsule that arrives at its destination. Table 2 shows the percentage of packets that arrived at both receivers, over the total number of packets sent, which is the mix of a given number of sources. The first row ("0 (lost)") represents lost packets. The second row (one source) represents the percentage of packets that arrive intact from the source. The third row (two sources) represents packets that mix samples from two different sources,

Table 2. Average percentage of packets carrying the sum of samples from a given number of sources

# sources	Str.0	Str.1	Str.2	Str.3a	Str.3b(r1)	Str.3b(r2)
0 (lost)	49.61	0	0	0	0	0
1	50.39	66.22	39.87	1.54	0.05	60.86
2	0	0.02	33.53	96.16	0.48	39.14
3	0	0	15.80	2.30	1.93	0
4	0	0.43	6.26	0	97.54	0
5	0	33.34	4.53	0	0	0

and so on for the rest of the rows. The columns represent the strategies used. For the null strategy, roughly half of the packets are lost, and the remaining packets arrive intact (no mixing). Strategy 1 is an all-or-nothing strategy: two thirds of the arriving packets contain data from only one source, while one third contains data from all the sources for a given session. The second strategy distributes the mixing effort more evenly.

As for the third strategy, column Str.3a of Table 2 shows the results when $a_1 = a_2$, corresponding to the simulation result shown at the top side of Figure 5. Columns Str.3b(r1) and Str.3b(r2) show the results for receivers r_1 and r_2 respectively, when $a_1 \neq a_2$ (Figure 5, bottom). We can see that when $a_1 = a_2$, most of the packets arrive at the receivers containing samples from two sources mixed together, while for $a_1 = 0.25$ (r_1), most of the packets contain samples from four sources, and for $a_2 = 0.75$ (r_2), more than one third of the arriving packets contain a mix of only two sources, and the rest only one source (no mixing). This shows that strategy 3 tries to stabilise at a target mixing level, which is characterised by the a parameter. The lowest the a parameter value is for a given session, the highest the mixing level which is achieved.

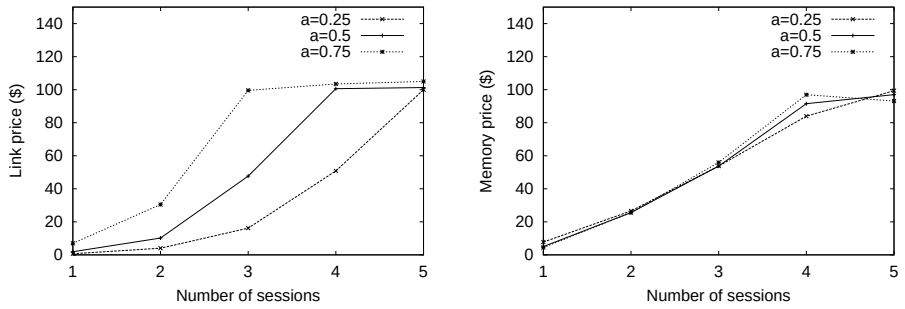

Fig. 6. Average link and memory prices when varying the number of sessions in parallel.

Finally, we vary the number of sessions in parallel using the third strategy, in three separate runs: during the first run all sessions have $a = 0.25$, during the

second, $a = 0.5$, and the third, $a = 0.75$. The total average prices for memory and link buffer occupancy are depicted in Figure 6. We can see that the a parameter has a clear influence on the link prices, that increase with a for a given number of sessions, as expected. However it has little influence on the memory prices. This can probably be explained by the fact that, although the application avoids using the memory during a too long period due to delay constraints, at any time each session has at most one packet stored in memory. In the simulations shown, memory does not become a bottleneck, therefore the small impact on prices.

6 Conclusions and Future Work

We have described a model for trading resources inside an active network node, which draws many elements from agent technology. We apply it to a concast audio mixing application which trades off link resources against memory in the presence of bottleneck links. The concast application is able to take congestion control decisions locally at each active node, such that no closed loop feedback between source and destination is needed. Using simulations, we have studied three different strategies to make a decision on the amount of resources to use: two naive strategies based on the cheapest price, and a strategy that makes use of utility function weights. The results indicate that first two strategies are already able to make improvements over the case when no congestion control is used, but they use resources inefficiently. The third strategy gives better results, achieving a stable and efficient sharing of resources.

We have several research directions to pursue: the most immediate one is to perform more complex simulations involving multiple node and link types, resource manager types, active and non-active nodes, different user strategies, etc. An implementation over a real active networking platform is also envisaged for the near future. We also plan to investigate the issues of dynamic resource manager upgrade with the help of mobile agents. The precise communication abstractions among the various kinds of agents need further attention too.

References

1. S. Athuraliya, S. Low, "Optimization Flow Control with Newton-Like Algorithm", Journal of Telecommunication Systems, to appear, 2000.
2. J. Bredin et al., "A Game-Theoretic Formulation of Multi-Agent Resource Allocation", Proceedings of the 2000 International Conference on Autonomous Agents, Barcelona, Spain, June 2000.
3. K.L. Calvert (ed) et al., "Architectural Framework for Active Networks", (DARPA) AN Working Group, draft version 1.0, July 1999, work in progress.
4. K. Calvert, "Toward an Active Internet", Active and Programmable Networks Miniconference, Networking 2000, Paris, France, May 2000.
5. A.T. Campbell, et al., "A Survey of Programmable Networks", ACM SIGCOMM Computer Communication Review, April 1999, p.7-23.
6. T.M. Chen, A.W. Jackson, "Active and Programmable Networks", Guest Editorial, IEEE Network, May/June 1998, p.10-11.

7. D. F. Ferguson, C. Nikolaou, J. Sairamesh, Y. Yemini, "Economic Models for Allocating Resources in Computer Systems", Market based Control of Distributed Systems, Ed. Scott Clearwater, World Scientific Press, 1996.
8. S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.
9. M.A. Gibney, N.J. Vriend, J.M. Griffths, "Market-Based Call Routing in Telecommunication Networks Using Adaptive Pricing and Real Bidding", LNAI n.1699, Proceedings of the IATA'99 Workshop, Stockholm, Sweden, August 1999.
10. M. Hicks et al., "PLANet: An Active Internetwork", Proceedings of IEEE INFOCOM'99, New York, 1999.
11. G. Hjlmtýsson, "The Pronto Platform: A Flexible Toolkit for Programming Networks using a Commodity Operating System", Proceedings of IEEE OPENARCH'2000, Tel-Aviv, Israel, March 2000, p. 98-107.
12. K. Jun, L. Blni, D. Yau, D.C. Marinescu, "Intelligent QoS Support for an Adaptive Video Service", To appear in the Proceedings of IRMA 2000.
13. D. Lapsley, S. Low, "Random Early Marking for Internet Congestion Control", Proceedings of Globecom'99, pp. 1747-1752, Rio de Janeiro, Brazil, December, 1999.
14. S. Low, D.E. Lapsley, "Optimization Flow Control, I: Basic Algorithm and Convergence", IEEE/ACM Transactions on Networking, 1999.
15. N. Miller, P. Steenkiste, "Collecting Network Status Information for Network-Aware Applications", Proceedings of IEEE INFOCOM'2000, Tel-Aviv, Israel, March 2000.
16. H. Mizuta, K. Steiglitz, E. Lirov, "Effects of Price Signal Choices on Market Stability", 4th Workshop on Economics with Heterogenous Interacting Agents, Genoa, June 4-5, 1999.
17. K. Najafi, A. Leon-Garcia, "A Novel Cost Model for Active Networks", Proc. of Int. Conf. on Communication Technologies, World Computer Congress 2000.
18. L. Peterson (ed) et al., "NodeOS Interface Specification", (DARPA) AN NodeOS Working Group, draft, January 2000, work in progress.
19. R.S. Pindyck, D.L. Rubinfeld, "Microeconomics", Forth Edition, Prentice Hall International Inc., 1998.
20. R. Sivakumar, S. Han, V. Bharghavan "A Scalable Architecture for Active Networks", Proceedings of IEEE OPENARCH'2000, Tel-Aviv, Israel, March 2000.
21. D. L. Tennenhouse et al., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.
22. C. Tschudin, "Open Resource Allocation for Mobile Code", Proceedings of the Mobile Agent'97 Workshop, Berlin, Germany, April 1997.
23. D. J. Wetherall, J. V. Guttag, D. L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", Proceedings of IEEE OPENARCH'98, San Francisco, USA, April 1998.
24. H. Yamaki, M.P. Wellman, T. Ishida, "A market-based approach to allocating QoS for multimedia applications", Proceedings of ICMAS'96, Kyoto, Japan, December 1996.
25. L. Yamamoto, G. Leduc, "Adaptive Applications over Active Networks: Case Study on Layered Multicast", Proceedings of ECUMN'2000, Colmar, France, October 2000.
26. L. Yamamoto, G. Leduc, "An Active Layered Multicast Adaptation Protocol", Proceedings of IWAN'2000, Tokyo, Japan, October 2000.
27. *UCB/LBNL/VINT Network Simulator - ns (version 2), http://www-mash.cs.berkeley.edu/ns/.*