Reifying Control of Multi-Owned Network Resources

Nadeem Jamali[1] and Chen Liu[2]

[1]University of Saskatchewan
Dept. of Computer Science
176 Thorv. Bldg., Saskatoon, SK, Canada
n.jamali@agents.usask.ca

[2]University of Alberta
Dept. of Computing Science
2-21 Athab. Hall, Edmonton, AB, Canada
cliu@cs.ualberta.ca

Abstract

Communication delay is a key source of uncertainty in distributed systems. Existing approaches to reduce this uncertainty focus on maintaining sufficient surplus bandwidth; applications, on their part, are designed in ways to tolerate certain degree of uncertainty in communication delays. This leads to contention between the goals of optimal utilization and acceptable delays.

We argue that the multi-owned nature of today's networks offers opportunities to reason about and scalably control networks at a fine grain. An explicit treatment of network resource ownership and trade allows reasoning about acceptable delays. This can lead to scalable mechanisms for fine-grained accounting and reification of control, which make it possible to quantify and control network utilization.

We bring together ownership, fine-grained accounting, and reification of control in a model for resource acquisition and control called CyberOrgs. Cyberorgs encapsulate distributed computations with resources required for their execution. A cyberorg acquires resources required by its computations by buying them from other cyberorgs using eCash.

We present a novel approach for implementing fine-grained network resource control based on the CyberOrgs model. A prototype implementation is described with experimental results illustrating the effectiveness of control.

.

1 Introduction

Computations sharing an execution space inevitably compete for the resources in that space. Even sub-computations working on parts of the same problem can compete for resources, hampering progress toward the shared goal. When a computation's choice of the next action depends on actions taken by other computations, *coordination* is required to achieve optimal results [5]. Not only is coordination recognized as a key concern in distributed computing [2], it has also been argued that computation and coordination are separate and orthogonal dimensions of all useful computing [6], necessitating coordination to be addressed explicitly.

Communication delay is a key source of uncertainty in distributed systems. Existing approaches to reduce this uncertainty focus on maintaining sufficient surplus bandwidth; applications, on their part, are designed in ways to tolerate certain degree of uncertainty in communication delays, and distributed implementations of applications which do not tolerate uncertainty well are limited. Because there are few ways of fine-grained measurement and control of network usage, this approach leads to contention between the goals of optimal utilization and acceptable delays. Because there is no way of addressing communication load in a targeted manner, it is not possible to resolve a localized bottleneck locally; attempts are made to limit communication over a wider network than necessary. This leads to over-reservation and under-utilization of large parts of the network, resulting in waste, and adverse consequences for other applications.

We argue that uncertainty in networks results in part from not considering a key notion: ownership. In the absence of explicit consideration of ownership, it is difficult to quantify acceptable delay. Similarly, in the absence of fine-grained accounting, it is not possible to quantify optimality of utilization. Quantifying network utilization at a finer grain can lead to measurement of surplus resource, and an overall better snapshot of remaining opportunities.

Our approach is to bring together ownership and fine-grained accounting in a model for resource acquisition and control. Cyberorgs are resource encapsulations which host

This research was supported by NSERC and CFI

distributed computations and trade with each other in resources. A cyberorg acquires resources required by its computations by negotiating contracts with owners of needed resources. Ownership is defined in time and space; so, for the span of a contract, the buyer would be the owner of the resource. Cyberorgs trade in resources using eCash.

We present a prototype implementation of this model for network resource control, with results illustrating the effectiveness of control.

2 Related Work

There is a significant body of work in network resource management. Related approaches fall into two categories [13]: congestion control and QoS. Congestion control adjusts the behavior of senders according to the current state of congestion, which attempts to maximize overall network utilization rather than address Individual flow requests. Network QoS is closer to our approach in that it provides service assurances to satisfy requirements of applications. Research in this area can be placed in three classes: Integrated Service, Differentiated Service and Overlay QoS. Integrated Service architecture is intended for supporting real-time applications, which require a bound (either statistical or absolute) on the delivery delay of each packet in an Integrated Services Packet Network (ISPN) [3]. Differentiated Service aims to guarantee QoS requirements for applications. However, rather than treating individual flows differently, it guarantees quality of service to a traffic aggregate, which is a bundle of flows. OverQoS [14], is an example of Overlay-based QoS, which uses an abstraction CLVL (controlled loss virtual link) to put a bound on the loss rate of a traffic aggregate and provide services of smoothing packet losses, prioritizing packets within an aggregate and statistical loss and bandwidth guarantees. Q-RAM-based QoS model [7] supports network resource allocation. The network resource allocation approach in Q-RAM-based QoS model is close to our approach in that it treats bandwidth separately from delay and packet loss and is implemented by optimized route discovery and bandwidth reservation. In active networks, network switches perform customized computations on the messages flowing through them [15]. This allows users to specify their application-oriented control requirements and build up user-aware networks.

Generalized Processor Sharing (GPS) [12] allows flows to have different service shares in accordance with their desired quality of service, and the service one flow is not influenced by other flows. Weighted Fair Queueing (WFQ), an approximation of GPS, aims to allow different flows to share the same link and have different guaranteed bandwidth allocated to them. This mechanism ensures that the bandwidth guarantee for each flow is independent and is not influenced by other flows. Besides, this scheme also allows configurable number of flows and guarantees delay and throughput as well.

3 CyberOrgs

CyberOrgs [8] is a model for resource sharing in a network of self-interested peers, where application agents may migrate in order to avail themselves of remotely located peer-owned resources. CyberOrgs organize computational and communication resources as a market, and their control as a hierarchy. Specifically, each cyberorg encapsulates one or more distributed computations (to be referred to as computations contained in the cyberorg), and an amount of *eCash* in a shared currency. Cyberorgs act as principals in a market of distributed resources, where they may use *eCash* to buy or sell resources among themselves. A cyberorg may use the resources so acquired for carrying out its computations, or it may sell them to other cyberorgs.

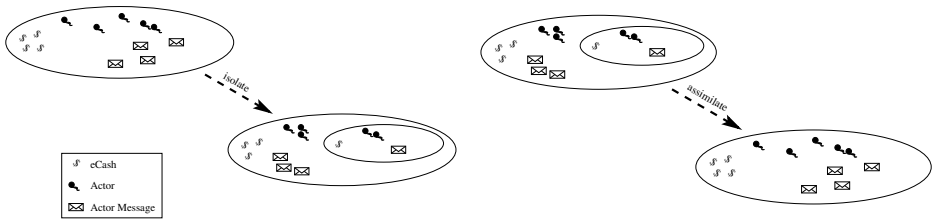
CyberOrgs treat computational and communication resources as being defined in time and space. In other words, a resource is not available for use before or after the instant of time at which it exists. Sale of a resource is represented by a *contract* stipulating availability of resources to the buyer for a cost. Delivery of resources to cyberorgs is determined by a hierarchy of control decisions. In other words, cyberorg a makes control decisions required for delivery of resources purchased from it by cyberorg b. Cyberorgs may pre-pay to buy resources which will exist in the future. In other words, after signing a contract, a cyberorg must migrate to the prospective host cyberorg in order to avail itself of newly acquired resources.

The CyberOrgs model separates concerns of computations from those of the resources required to complete them. We assume that computations are carried out by actors [1], and we represent the resource requirements of each computation by the sequence of resources required to complete it. *Ticks* serve as the unit of a consumable resource such as processor time. Every computation requires a certain number of ticks to complete.

Progress is represented by transitions occurring with introduction of ticks into the system. When a tick is inserted into a cyberorg, it may pass the tick on to a client cyberorg, use it for progressing on its system tasks or on its actors. Whether a tick is passed on to a client or used locally depends on the contracts that the cyberorg has with its clients.

As illustrated in Figure 1, a new cyberorg is created by using the `isolate` primitive, which collects a set of actors, messages, and electronic cash, and creates a new cyberorg hosted locally. A cyberorg disappears by assimilating into its host cyberorg using the `asmlt` primitive, relinquishing control of its contents to its host.

A cyberorg may realize that its resource requirements have exceeded what is available by its contract with the host



a) CyberOrg Isolation

b) CyberOrg Assimilation

Figure 1. Creation and Absorption

cyberorg, triggering an attempt to migrate. The tasks required for migration are: search (for potential hosts), negotiate (potential contracts), and migrate (to a selected host).

A more formal treatment of the operational semantics may be found in [10].

4 Design and Implementation

An execution environment of our system on a computer node is called a *CyberOrg platform*. A CyberOrg platform is composed of three layers: CyberOrgs interface layer, CyberOrgs management layer, and Resource scheduling layer.

4.1 CyberOrgs Interface Layer

CyberOrgs interface layer is composed of two parts: a class library and an application programming interface (API) for CyberOrgs. Using these facilities, programmers are able to specify high-level resource requests for their applications, and define their resource control policies to manage resources.

4.1.1 CyberOrgs Class Library

The CyberOrgs library contains a set of classes corresponding to major components in CyberOrgs. There are six main classes: CyberOrg, SysComCyberOrg, Actor, Facilitator, vLink and Contract.

The CyberOrg class defines a cyberorg, which is the basic unit of resource control in the CyberOrgs model. Each cyberorg contains a *facilitator* actor which helps it control its resource requirements.

The Actor class defines an actor, which carries out computations or communicates with other actors while consumes resources. The Actor class contains a thread and a message queue which buffers received messages.

A *facilitator* is a special actor, which facilitates the resource control function of a cyberorg. Every request for resource or primitive operations is eventually arrived at a corresponding facilitator of a cyberorg as a message.

The vLlink class defines abstract network resources called *virtual links*.

The Contract class defines contracts, in which the supplier and consumer fields match the cyberorgs who grant and receive resources respectively. The quantity of purchased resources and corresponding price are also specified.

4.1.2 CyberOrgs APIs

We implement CyberOrgs by extending Actor Architecture [11], a Java library for implementing Actor systems.

Actor APIs Most of the Actor API in our CyberOrgs implementation is borrowed from Actor Architecture, except for the interface for actor creation. Specifically, two new methods are added.

Actor Creation

- ActorName createActor (ActorName p_anCreator, String p_strActorClass, Object[] p_objaArgs)

Creates an application actor. Parameters are: p_anCreator, the creating the actor; p_strActorClass, class of new actor; p_objaArgs, creation arguments.

- ActorName createActor (CyberOrg p_coMyCyberOrg, String p_strFacilitatorClass, Object[] p_objaArgs)

Creates a facilitator actor. Parameters are: actor creation parameters, plus p_coCyberOrg, the cyberorg represented by the facilitator.

CyberOrg APIs APIs for Cyberorg provide interfaces for creation, absorption, and migration of cyberorgs, plus negotiation of contracts between cyberorgs.

Cyberorg Creation

- Cyberorg createCyberOrg (String p_strCyberOrgClass, long p_lECash, Contract p_cContract, String p_strFacilitatorClass, Object[] p_objaArgs)

Creates a cyberorg, which can be the system cyberorg or a newly isolated cyberorg. In the latter case, the method is called by the Isolate() method described below. Parameters are: p_strCyberOrgClass, the new cyberorg's class; p_lECash, eCash provided to the new cyberorg; p_strFacilitatorClass, facilitator class; p_cContract, the contract imposed on the new cyberorg,[1] which specifies the network resource available to the new cyberorg, and its cost; p_objaArguments, facilitator creation arguments.

- CyberOrg isolateSysComCyberOrg (long p_lECash, String p_strFacilitatorClass, Contract p_cSysComContract, String p_strFacilitatorClass, Object[] p_objaArgs)

Creates system communication cyberorg. Parameters are: createCyberOrg parameters plus p_cSysComContract, contract between the system cyberorg and the system communication cyberorg, which specifies the network resource reserved for system communications, and its cost.

- Cyberorg Isolate (long p_lECash, ActorName[] p_anaActors, Contract p_cNewContract)

Creates new child cyberorg. Parameters are: p_lECash, eCash given to new cyberorg; p_anaActors, an array of existing actors to be isolated into new cyberorg; p_cNewContract, contract between new cyberorg and creating cyberorg, specifying the resources available to the new cyberorg, and their cost.

Cyberorg Absorption

- void Assimilate()

Assimilates cyberorg into its host cyberorg. After the invocation of this method, all resources, actors and eCash held by the assimilating cyberorg are released to the host cyberorg.

Cyberorg Negotiation and Migration

- Contract Negotiate (ActorName p_anSupplierFacActor, Contract p_cProposedContract)

Initiates negotiation with another cyberorg to purchase required resources. Parameters are: p_anSupplierFacActor, name of facilitator of potential host cyberorg; p_cProposedContract, proposed contract which specifies resource request and the proposed price. If proposal is acceptable to the potential host, a the proposed contract becomes effective.

- void Migrate(ActorName p_anDestinationActor, Contract p_cNewContract)

[1] If the cyberorg being created is a system cyberorg, the contract is between this cyberorg and the system. Otherwise, the contract is between this cyberorg and its creating cyberorg.

Service	Bandwidth Request	Transfer Limit
ChatActor	0.1MBps	50MB
ConferenceMemberActor	1MBps	100MB
FileShareActor	0.05MBps	10MB
VideoShareActor	10MBps	500MB

Figure 2. Sample Resource Specification File

Migrates cyberorg to destination cyberorg. Parameters are: p_anfacActorOfdesCyberorg, name of facilitator of destination cyberorg; p_cNewContract, pre-negotiated contract between the migrating cyberorg and the destination cyberorg.

4.1.3 Example: Chat Room Service

Consider a chat room service, in which the service provider offers multiple types of services, such as live video conference, text chatting, file transferring, audio and video exchange, and so on. Here, we illustrate how such a system may be constructed using CyberOrgs. We specifically look at: (1) how to set up an application in the system; (2) how to develop a resource allocation policy and (3) how to develop application-specific resource control policies. In this example, it is assumed that a chat application and relevant utilities already exist in the system.

In order to isolate a new cyberorg for chat room service, the name of the master facilitator of the system cyberorg has to be known in advance. This is done by retrieving the actor name of the master facilitator of the system cyberorg from the local CyberOrg Manager. The CyberOrg Manager, which manages local cyberorg hierarchy, has a public name (uan://host_address:2). Then, the resource allocation specification file (described in the next section) is loaded to set default allocation policy held by the new cyberorg. After a contract is generated for this isolation, a synchronous message is sent to the master facilitator of the system cyberorg to isolate a cyberorg for this chat room service.

Resource Allocation Specification

In order to offer different qualities of service to users, the service provider has to specify an allocation policy. There are different possible ways to achieve this purpose (e.g. using a resource specification language such as RSL provided by Globus [4])). Here a simple example is given of a specification file.

It is assumed that the service provider allocates vLinks with different bandwidth and data transfer limit to different services. Therefore users of the same service are allocated with equalized vLinks. Figure 2 shows an allocation specification file in which resource allocation policy is in

terms of the service type, such as chat service (represented by the corresponding actor class), the amount of bandwidth required, and the data transfer limit. Alternatively, bandwidth requirement can be specified qualitatively.

Each cyberorg has such a specification file, and allocates resources to actors according to the specified policy.

Resource Control Policy

Although a resource allocation specification file allows service providers to set default allocation policy for their resources, this is only one side of the story. Service providers may also want to change the default allocation policy to meet some special requests. For example, users may wants to have better bandwidth for file transfer service. With these concerns, application programmers should define policies about how to control resource allocation.

4.2 CyberOrgs Management Layer

CyberOrgs management layer enforces resource management mechanisms in CyberOrgs. This layer consists of three components: CyberOrgs Manager, Task Distributor and Contract Manager. CyberOrg Manager manages the hierarchy of CyberOrgs and cooperates with the Task Distributor to carry out resource allocation. The Contract Manager is in charge of guaranteeing the execution of contracts made between cyberorgs.

Resource allocation is the core mechanism in the CyberOrgs system. Resource allocation has different meanings at CyberOrgs and lower levels. At the CyberOrgs level, resource allocation create vLinks upon requests and assigns these vLinks to the requester, either a cyberorg or a communication. In contrast, at the lower level, resource allocation searches a path to meet the source, destination and bandwidth requirements specified in a resource request, establishes a connection along the path for communication, and disestablishes this connection at the end of the corresponding communication. The resource allocation enforcement is achieved by resource scheduling layer (see next section). We refer to the procedure of vLink creation and path searching as *resource discovery*, and vLink assignment and connection establishment as *resource allocation task distribution*. The procedure of vLink consumption and connection disestablishment is called *post-allocation maintenance*.

The procedure of resource discovery plays two important roles. First of all, this procedure controls the admission of resource requests. If a resource request cannot be satisfied, it is refused. Further, this procedure maps an application-level vLink to a low-level network connection.

Upon receiving a vLink request, a concrete physical path which meets the source, destination and bandwidth requirements specified in the request has to be found before the vLink creation. This is in fact a routing problem. Different from routing on the Internet, which aims to find the shortest path between a source and destination, this routing also concerns bandwidth requirement. It is necessary to ensure that the searched path has bandwidth greater than or at least equal to the requested value. We customize Dijkstra's algorithm to solve this problem.

Once a path has been found, a corresponding vLink is created to represent the resource at the CyberOrg level. At the lower level, a connection has to be established. Each node along the path has to be informed about their scheduling tasks corresponding to the resource request. The Task Distributor is responsible for distributing a resource allocation task to related nodes.

The CyberOrg Manager passes a resource allocation task to the Task Distributor. A resource allocation task stipulates which flow a vLink is created for, the underlying path mapping to this vLink as well as the bandwidth of the vLink. Each node along the path has to schedule packets belonging to the flow according to the allocated bandwidth. There are two ways for a Task Distributor to inform related nodes about the task. First, the Task Distributor generates a single control message which contains the resource allocation task, and relays this message along the assigned path. Each node along the path receives the message and sets a state in itself. It is the last node to receive the control message that sends back a response message to the Task Distributor. Then, the Task Distributor informs the Message Manager to start the corresponding communication which requires this resource allocation.

4.3 Resource Scheduling Layer

Resource scheduling layer is implemented by modifying an existing Actor system, Actor Architecture [11] (AA). AA has a well-designed layer architecture, which is easy to extend. We keep most original components in AA, and customize some of them to support our scheduling schemes.

In resource scheduling layer, we schedule messages, packets and threads. Message scheduling supports link sharing between different flows. Packet scheduling realizes rate-based flow control, and thread scheduling ensures any thread at a node has a chance to progress. Aided by these three scheduling schemes, bandwidth allocation can be achieved. For efficiency, we employ the approach of flattening a hierarchical schedule developed in [9].

Message Scheduling We decompose an actor message into a number of fixed length packets, and control the sending rate and order of these packets to enforce bandwidth allocation. Although packets are scheduled at a rate corresponding to allocated bandwidth, we still need to schedule messages for the purpose of multiplexing the link utilization.

For each communication, there is a corresponding message queue to buffer messages for this communication, and a packet queue which buffers decomposed packets accordingly. Message decomposing and dequeuing happen in a round robin fashion. For example, at each message scheduling cycle, the first message in each message queue is allowed to decompose 20 packets and these packets are dequeued from the message queue and then put into the corresponding packet queue.

Algorithm 1 Packet Scheduling

```
1:  while true do
2:      start timing /*start time is Tstart*/
3:      calculate the end time /* Tend = Tstart + Tgranularity*/
4:      for each packet queue i do
5:          if packet queue i is not empty then
6:              retrieve allocated bandwidth B_i
7:              dequeue N_i packets /* N_i = B_i * Tgranularity*/
8:              send these packets to the next hop
9:          end if
10:     end for
11:     message scheduling
12:     if currenttime < endtime then
13:         wait until the end time
14:     end if
15: end while
```

Packet Scheduling Packet level scheduling is based on the weighted round robin algorithm, and supports adjustable time granularity. By granularity we mean the length of each time cycle in which each flow has received its due share. The weight of each flow is determined by its bandwidth request. When the granularity of a time cycle is set, the number of packets to be scheduled for a flow is the product of its weight and time granularity. Algorithm 1 shows our packet scheduling scheme.

Thread Scheduling Threads are explicitly scheduled by the system to manage CPU sharing between the network control system and other activities of a node.

4.4 Overhead

CyberOrgs primitives incur both computational and communication overheads. Because cyberorgs in the prototype system are internally distributed – i.e., one cyberorg may have actors located on different nodes – primitive operations are implemented in a distributed manner as well. Correspondingly, the invocation of a primitive operation causes system message communication and may also result in resource allocation computation. Table 1 summarizes communication and computation overhead for each primitive operation.

Table 1. System Overhead (n is the number of nodes covered by a cyberorg; r is the number of vLink requests)

Operation	Comm Overhead	Comm Overhead	Comput Overhead
	Loc Msgs	Rmt Msgs	
Isolate	$O(n)$	$O(n)$	$O(rn^2)$
Assimilate	$O(n)$	$O(n)$	
Migrate	$O(n)$	$O(n)$	$O(rn^2)$
Allocate	0	$O(r)$	$O(n^2)$

For communication overhead, the number of triggered messages is counted to demonstrate the amount of system communication caused by a primitive operation. The amount of system communication for a primitive operation is linear with the number of nodes covered by a cyberorg. In addition to the amount of system data transferring, system resource consumed in transferring data is also considered. Here, local messages are differentiated from remote messages, because local messages may sometimes be optimized as function calls. Since remote messages may have different source and destination requests, vLinks consumed by transferring these messages are different. However, as system resources are reserved in the system communication cyberorg in advance, these messages do not compete with application messages for resources.

Primitive operations require recalculation of resource allocation. Both the isolate and migrate operations have to calculate the resource allocation in order to create the new cyberorg. Suppose there are r resource requests, then the overhead for calculating resource allocation for these requests is $O(rn^2)$.

5 Experimental Results

We have carried out a preliminary evaluation of our packet scheduling scheme for fine-grained network control by conducting experiments on a prototype implementation. Specifically, we tested for effectiveness of control for a flow given different numbers of flows in the system.

Three sets of two-node results were obtained. Figure 3 shows the bandwidth maintained for requests of different bandwidths during 2 minutes with granularity of 200 milliseconds in the presence of 9 additional flows existing in the system (which are not shown). Inaccuracies observed

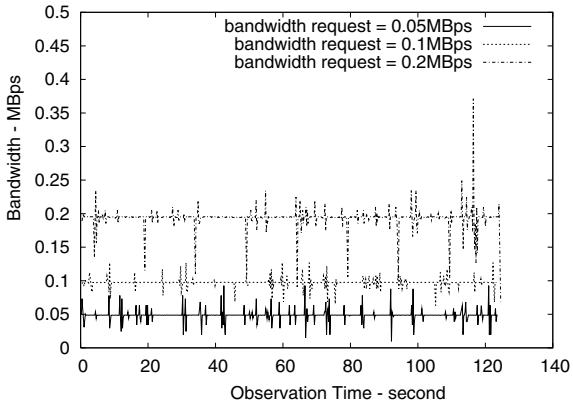


Figure 3. Performance of Bandwidth Allocation (bandwidth request: 0.05MBps-0.2MBps; time granularity: 200 ms; observ time: 2 min; env: 2-node; flows: 10)

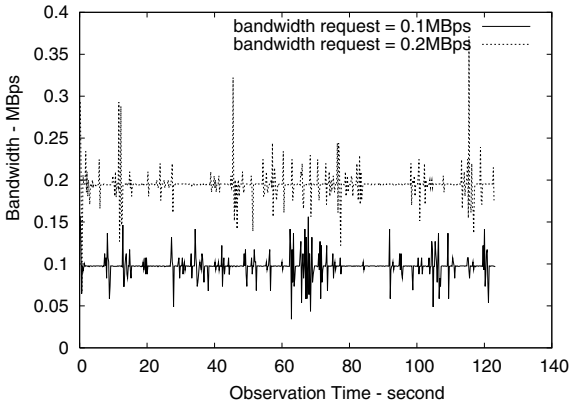


Figure 4. Performance of Bandwidth Allocation (bandwidth request: 0.05MBps-0.2MBps; time granularity: 200 ms; observ time: 2 min; env: 3-node; flows: 10)

Table 2. Inaccuracy Comparison of Scheduling With Different Number of Flows

Request/Flows	1 flow	5 flows	10 flows
0.05MBps	5.894 %	5.08 %	6.695 %
0.1MBps	6.187 %	4.236 %	2.314 %
0.2MBps	4.869 %	4.512 %	4.312 %

Table 3. Inaccuracy Comparison of Scheduling with Different Number of Flows in Three-Node Environment

Request/Flows	1 flow	5 flows	10 flows
0.1MBps	3.739 %	4.011 %	5.364 %
0.2MBps	3.51 %	4.231 %	3.294 %

are summarized in Table 2. Although some of this fluctuation is because of measurement errors (which is confirmed by down-spikes followed by up-spikes), others are caused by packets dropped because of packet buffer overflows, because we did not employ flow control.[2]

Figure 4 shows the bandwidth maintained when there is an intermediary node between the source and the destination nodes. Figure 4 shows flow maintained for a single flow in the presence of 9 other flows (which are not shown).

Finally, experiments were carried out on multi-node paths with the packets being rescheduled at each intermediary node. Three physical machines were used to create the path of length 8, with packets traveling between source and intermediary node for a number of times before arriving at the destination. Figure 5 shows the bandwidth maintained for one flow in the presence of 9 additional flows (which are not shown). Figure 6 shows the difference in performance between the two cases, representing a greater need for flow control in multi-hop paths.

[2]Because of possibility of buffer overflows in the absence of flow control, percentage of dropped packets increased dramatically when bandwidths larger than 2MBps were requested. For this reason, our experiments avoided cumulative bandwidth requests of above 2MBps.

6 Conclusion

We have introduced an approach to fine-grained network control based on the CyberOrgs model. Design and implementation of a prototype system have been presented with experimental results.

We are encouraged by the preliminary results which show effectiveness of our approach in maintaining desired bandwidth for communication flow. Additional experimental work is ongoing on a more efficient implementation of the packet scheduler with flow control as part of a CyberOrgs-based prototype for controlling different types of computational and communication resources. We are particularly interested in studying network and processor overheads involved in using CyberOrgs for supporting resource requirements of actual applications, especially as the number of nodes and communication flows scale up.

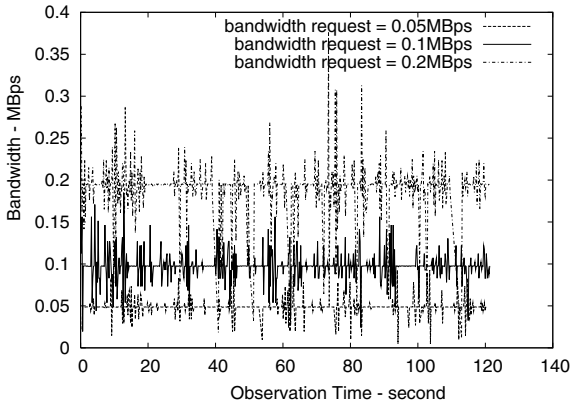


Figure 5. Performance of Bandwidth Allocation (bandwidth request: 0.05MBps-0.2MBps; time granularity: 200 ms; observ time: 2 min; env: n-node; flows: 10)

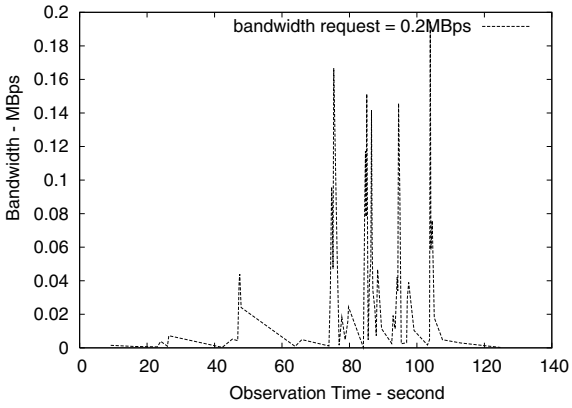


Figure 6. Degradation in Accuracy of Bandwidth Allocation from 1 to 10 Flows in System (bandwidth request: 0.05MBps-0.2MBps; time granularity: 200 ms; observ time: 2 min; env: n-node)

References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.

[2] A. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufman Publishers, San Mateo, California, 1988.

[3] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633, June 1994.

[4] I. Foster and C. Kesselman. The Globus project: a status report. *Future Generation Computer Systems*, 15(6):607–621, 1999.

[5] L. Gasser. DAI approaches to coordination. In N. M. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 31–51. Kluwer Academic Publishers, 1992.

[6] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[7] S. Ghosh. *Scalable QoS-based Resource Allocation*. PhD thesis, Carnegie Mellon University, 2004.

[8] N. Jamali. *CyberOrgs: A Model for Resource Bounded Complex Agents*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[9] N. Jamali and X. Zhao. A scalable approach to multi-agent resource acquisition and control. In *Proceedings of the 2005 international joint conference on Autonomous agents and multiagent systems*, pages 868–875, July 2005.

[10] N. Jamali and X. Zhao. Hierarchical resource usage coordination for large-scale multi-agent systems. In T. Ishida, L. Gasser, and H. Nakashima, editors, *LNAI: Massively Multi-agent Systems I*, volume 3446, pages 40–54. Springer Verlag, 2005.

[11] Open System Laboratory. The Actor Architecture. Technical report, University of Illinos at Urbana-Champaign, 2004.

[12] A. K. Parekh. *A generalized processor sharing approach to flow control in integrated services networks*. PhD thesis, Massachusetts Institute of Technology, February 1992.

[13] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan-Kaufmann, 2000.

[14] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An Overlay based Architecture for Enhancing Internet QoS, March 2004.

[15] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D.J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.