Modeling and Analysis of a Cooperative Control Protocol

G. Fortino, C. Mastroianni, W. Russo

RT-ICAR-CS-03-18 **Dicembre 2003**

Modeling and Analysis of a Cooperative Control Protocol

G. Fortino[2], C. Mastroianni[1], W. Russo[2]

Rapporto Tecnico N.:
RT-ICAR-CS-03-18

Data:
Dicembre 2003

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS)
[2] Università degli Studi della Calabria, DEIS, Via P. Bucci 41C, Rende (CS)

Modeling and Analysis of a Cooperative Control Protocol

Abstract

This paper proposes COCOP – COoperative COntrol Protocol, which enables a synchronous cooperative group to interactively control an on-demand server which multicasts time-dependent data streams. Multicast-based streaming on-demand systems such as video on-demand systems, web casters, and networks of real/virtual sensors can beneficially exploit COCOP to provide cooperative control sessions as a mainstream service. In order to improve efficiency and scalability, the protocol relies on a reliable multicast transport layer which can be based either on IP-multicast or on Application Layer Multicast. Performance evaluation of COCOP is carried out on multicast delivery trees by using a discrete-event simulation framework.

1. Introduction

Due to advances in communications and computing technology, Internet is now a world-wide platform which can be ubiquitously exploited for delivering time dependant-services such as media streaming, web casting, measurements, etc [CHW99]. One key enabling technology is multicast [M99] which efficiently concurs to save network resources and easily feature multi-party applications. Multicast is exploitable over Internet at network level through IP-multicast or at application level through an application level multicast infrastructure [BB03]. Both real-time data services and synchronous group coordination services have been demonstrated to greatly benefit from multicast [DG99a]. In particular, synchronous group services entail synchronization of content and activities in remote workspaces with regard to time and space and notably mutual exclusion in resource access.

The integration of on-demand multicast data delivery with group-based remote control can enable the construction of synchronous cooperative environments in which a group of clients shares the data service control.

Such an integration can be incorporated in peer-to-peer based or grid-based Content Distribution Networks (CDNs) [CGK01] as a basic low-level service. Thus, a variety of high-level group services such as cooperative playbacks [FGM02], cooperative browsing and collaborative visualization of remote measurements [DG99a] can be provided.

A challenging issue is the development of a real-time distributed control logic of the server entity which multicasts data streams. To this end several floor control protocols and architectures have been proposed in the last years which facilitate users, organized in even large groups, to coordinate access to shared resources through coordination primitives called floors [DG99a]. Since it is needed to request and grant floors in a session-wide contention scheme, floor management can very likely introduce overhead diminishing interactivity and increasing response time of the server control in highly responsive environments.

This paper presents a high level cooperative control protocol (COCOP) suitable for timely control of a shared server by a cooperative group of clients. It relies on a distributed and implicit coordination policy centred on cooperation among clients and final contention resolution at server side. COCOP can be supported by a lightweight reliable multicast protocol purposely enhanced to detect events which can affect the logical thread of a cooperative control session. Notably, the protocol also embeds a mechanism coping with the problem of unfairness among clients. COCOP is validated by means of an event-driven simulation framework ad-hoc customized to analyse (i) the high-level protocol performances and (ii) the dynamics of a cooperative control session over both basic topologies, namely star and linear chain, and tree topologies.

The rest of the report is structured as follows. §2 details the proposed high-level cooperative control protocol along with a reference application context. §3 describes the simulation scenario for the protocol validation and then comments and analyses the simulation results. Finally, conclusions are drawn and directions of future work delineated.

2. The COCOP Protocol

The aim of the COoperative Control Protocol (COCOP) is to enable a synchronous group of clients to cooperatively share the control of a server, which typically provides time-dependant services. COCOP complies with a multicast client/server model [FGM02], which is an extension of the unicast client/server one. It allows a group member to send, on the behalf of the group, a request to the server that, after processing the request, replies to all group members. According to the peculiarity of the service offered, group/server interactions can be stateful or stateless. We focus on stateful interactions which originate cooperative control sessions. The server holds the session state and changes it each time a client request is accepted. Every client request is provisional which means that it is the server reply that modifies the group state. In order to increase the flexibility and the resilience of the session state management, we adopt a soft-state like paradigm [M99].

COCOP relies on the idea of cooperativity-featured competitive access to a shared resource, the server, combined with mechanisms aimed at increasing the protocol efficiency and effectiveness.

The cooperative competitive access is regulated by the following mechanisms:

1. During session contention periods the server resolves conflicts by accepting the first incoming request and discarding the others. Such mechanism drives the session state by dynamically establishing which client request is accepted and processed by the server.
2. After forwarding a request to the server, a client inhibits itself by blocking every successive user request until a server reply is received. Such a mechanism limits the network load and avoids meaningless request sequences coming from the same client.
3. After a server reply, both the server and the clients block themselves for a given amount of time, in order to make users aware of the session state change.
4. When a client senses another client request, it auto-inhibits in order to fairly give priority to the remote client so avoiding to forward requests, that would be probably discarded and would only increase the network load.

COCOP has characteristics similar to the activity sensing floor control protocol (RAS) belonging to the Random-access Floor Control (RFC) class [DG99b]. The main differences are that, in COCOP, the floor is the server itself and, in RAS, the auto-inhibition is realized by using back-off, i.e., by delaying a request of a random amount of time. Moreover, higher level policies such as token-based [DG99b] or voting-based [FN00] can be adapted atop of the basic protocol to fulfill specific synchronization requirements.

In order to enable cooperativity in COCOP, a client request is multicast to the group so that it can be received by each client and by the server. To this end, it is assumed the existence of a virtual logical channel connecting all the parties (clients and server) which, from a semantic point of view, complies with the idea of multicast group [CHW99]. It provides a simple way to send one-to-many messages and allows an efficient realization of the logical channel by either IP-multicast [M99] or an Application Level Multicast (ALM) protocol [BB03].

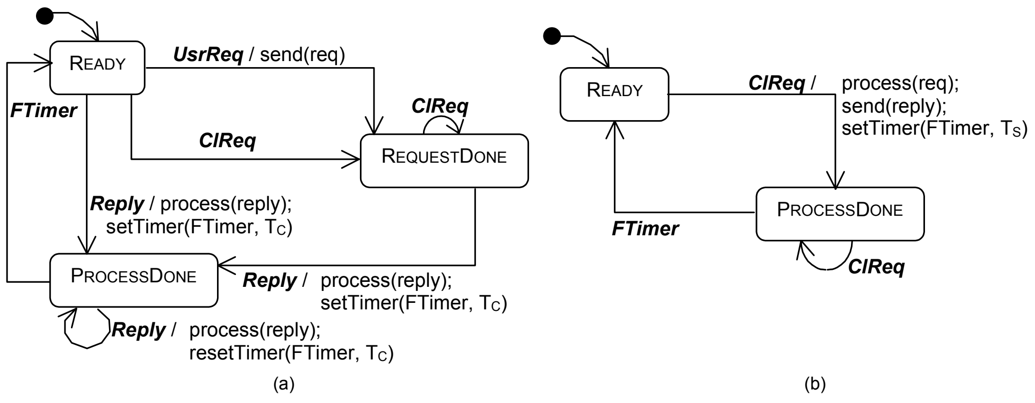
COCOP is not intended for the transport of application data but it should be coupled with a data delivery protocol tied to a specific service (e.g., file bulk transfer, media streaming). It is worth noting that the proposed protocol focuses on the server control and implicit low-level group coordination; challenging issues such as group formation, security, and fault-tolerance as highlighted in [FN00, SRC98] are out of the scope of this paper.

2.1. Client and server high-level behaviors

The behaviors of client and server processes are visually defined by the finite state machines (FSMs) depicted in Figure 2.1.

Client Automaton. In the `Ready` state, the client process (or client) can (i) accept a request (`UsrReq`) from the local user in order to forward the corresponding client request to the multicast group, (ii) sense a client request (`ClReq`) sent from a remote client so performing the cooperative mechanism, (iii) process a server reply (`Reply`). If the guard f is true, a client request forwarding is delayed of the time T_w, dynamically calculated by each client. Such a mechanism allows smoothing the unfairness problem as analyzed in §3.2.2. In the `RequestDone` state, the client ignores other `ClReq` and, once it receives a server `Reply`, it passes into the `ProcessDone` state and sets the timer T_C, disabling the user to perform new requests. The client gets `Ready` again after receiving the `FTimer` (timer expiration event).

Server Automaton. In the `Ready` state, the server process (or server) is available to receive and process a client request (`ClReq`). After processing the request and sending the reply, the server rests inactive for an amount of time T_S that depends on the specific application. T_S is introduced both to make the clients aware of the session state change, and to regulate the session interactivity. In fact, the group has to wait at least T_S to be able to get another request accepted. In the `ProcessDone` state, the server refuses all incoming `ClReqs`. As soon as the timer expires, the server gets ready again. Usually, T_S is set at session set-up whereas T_C is dimensioned upon T_S.



READY
UsrReq / send(req)
FTimer
ClReq
ClReq
REQUESTDONE
Reply / process(reply); setTimer(FTimer, Tc)
PROCESSDONE
Reply / process(reply); setTimer(FTimer, Tc)
Reply / process(reply); resetTimer(FTimer, Tc)
(a)
READY
ClReq / process(req); send(reply); setTimer(FTimer, Ts)
PROCESSDONE
FTimer
ClReq
(b)

Figure 2.1 (a) the client process automaton; (b) the server process automaton.

2.2. Reliability requirements

Since a control message (request or reply) require both reliability and fast delivery, COCOP relies on a lightweight reliable multicast protocol [FJM97] which only guarantees reliability. It is worth noting that different multicast applications require many different levels of reliability and ordering guarantees in the face of transient network failures such as dropped packets. In [WMK94] and [CKV01] the main QoS levels provided by multicast transport protocols are discussed: unreliable, unordered, source ordered, causal ordered, totally ordered, K resilient, majority resilient, totally resilient.

Although the exploitation of more constraining protocols such as source-ordered, causal-ordered or total-ordered protocols [WMK94] would avoid exceptional situations by providing a more logical message delivery within a control session, such an exploitation would result in an unavoidable performance degradation in terms of message delivery rate and hence session interactivity. As an example, consider the sending of a pause command which temporarily halts a streaming transmission. The command should reach the server "as fast as possible", in order to provide the issuing client with a reasonable degree of interactivity.

However, the reliable multicast protocol should be further enhanced at a higher-level in order to detect the loss of source-ordering on a per-process (client or server) basis.

To this end, COCOP integrates a high-level mechanism which filters messages violating some source-ordering and causal-ordering requirements in the hypothesis of no clock synchronization among the parties. Particularly, we isolated the following set of requirements whose not fulfilling would cause the violation of the logical thread of control activities:

1) *Source-ordering of Requests*. The requests issued by each client should arrive at the server according to their transmission order.
2) *Source-ordering of Replies*. The replies sent by the server should orderly arrive at each client, so respecting the logical order of the global state change.
3) *Causal-ordering of Request/Reply pairs*. The arrivals of a request/reply pair should never be inverted.

In order to fulfill such requirements, we have appositely defined the control structure of messages by inserting some information which is exploited for building dynamic session state tables maintained by each process (see Appendix A). Messages are discarded on the basis of these tables if they violate the above requirements, otherwise they are admitted to drive the automata of server and client processes (Fig. 2.1).

2.3. An application context: cooperative playback sessions

Cooperative playback sessions are multicast sessions where an explicit and synchronous group of clients cooperatively controls a media server (MS) which is multicastly streaming an archived media file. Basically two interaction levels exist within a cooperative playback session: media and control. Media is unidirectionally streamed by the MS to all the group members over a multicast channel. Control is applied under the form of transmissions of VCR-like commands (e.g., PLAY, PAUSE, SEEK, etc) sent from the clients to the MS.

Cooperative playback sessions are usually provided by a cooperative playback system (CPS) [FN00, SRC98]. A fundamental building block of a CPS is the archive control protocol which is the protocol used to contact the media server in order to organize the cooperative group, request and control a selected playback. Specifically, the archive control protocol should allow a client to issue control commands by resolving possible conflicts with all the other clients of the group.

In the MASH Rover [SRC98], the SSAC (Soft State Archive Control) protocol is based on a soft state approach and on an announce/listen (A/L) underlying transport mechanism [M99]. A/L protocols are robust and can be implemented atop of an unreliable transport protocol such as unicast or multicast UDP. No specific control policies are introduced, the implicit policy exploited is of type laissez-faire cooperative (e.g., with user-level feedback). Moreover, control commands can be lost so that the requesting client loses "a priori" the contention against other clients.

Conversely, in our CPS [FN00], the control protocol, called MACπ (Multicast Archive Control Protocol), is realized as a variant of RTSP (Real-Time Streaming Protocol) [CHW99] adapted atop of the lightweight reliable multicast protocol (LRMP). Two basic policies regulating conflicts are introduced: (i) a pure FC policy for the PAUSE command and (ii) a voting-based policy for the PLAY, SEEK and STOP commands. The former is embedded in the MACπ whereas the latter is enforced by using an higher level protocol, called COπ (COllaborative protocol) which also enables the multicast exchange of questions and annotations. The usability analysis of the system [FN00] highlights that as soon as the request rate increases the group members are overwhelmed by too many voting procedures and are affected by consistent refusals of the PAUSE command at the server side. So it would be more convenient if the protocol, for some control messages (e.g. PAUSE), relied on an implicit coordination mechanism like COCOP.

The above-mentioned CPSs are strongly IP-multicast enabled so limiting their widespread diffusion to only IP-multicast enabled network such as MBone and private LANs. Efficient application level multicast protocols within peer-to-peer networks and grid-based infrastructures coupled with lightweight cooperative control protocols can boost the exploitation of cooperative playbacks as a mainstream service.

3. Protocol analysis through simulation

In order to analyse the dynamics of a cooperative control session, COCOP is implemented in an object-oriented simulation framework.

The aim of the simulations is to analyse and compare the performances of:

- the basic COCOP protocol (see §3.1), referred as `COOP`;
- a protocol obtained from `COOP` by deleting the transitions labelled by `ClReq` in the client automaton (Fig. 2.1(a)). With this modification, a client does not sense requests from remote clients. This protocol, named `NoCOOP`, is analysed to evaluate the effect of client cooperation introduced by COCOP. The coordination mechanism of `NoCOOP` is the same one which is provided by the protocols SSAC and MACπ (see § 2.2);
- the version of COCOP denoted as `FAIRNESS`, which was enhanced with a fairness mechanism. The fairness mechanism (described in §4.2.1) allows for augmenting the fairness degree of the coordination mechanism in cooperative control sessions where client nodes experience different delays from the server node.

The main purpose of the analysis is to investigate the capability of a generic client to obtain the control of the server as well as the server load and the network system load. To this end, the analysis parameters in Table 3.1 were defined. Note that the first six parameters (in bold style) are independent variables that must be set before simulation runs, while the other parameters are calculated at the end of the simulation to analyse protocol performances.

Analysis was carried out first in basic topologies such as star and chain [FMR03], and then in more complex and realistic multicast topologies, such as binary and N-ary trees, which are fully representative of multicast delivery trees [NB97].

$\mathbf{T_{session}}$	**Duration of the session**	
$\mathbf{N}$	**Number of clients**	
$\mathbf{\Lambda}$	**User request rate**	
$\boldsymbol{\delta}$	**Link delay between two adjacent nodes**	
$\mathbf{T_s}$	**Server timer**	
$\mathbf{T_C}$	**Client timer**	
N_{UsrReq}	Total number of user requests	
N_{ClReq}	Total number of client requests	
$N_{UsrReqBlk}$	Number of user requests blocked by respective clients	
$N_{ClReqDis}$	Number of client requests discarded by the server	
$N_{ClReqAcpt}$	Number of client requests accepted by the server	
$P_{blocking}$	$\frac{N_{UsrReqBlk}}{N_{UsrReq}}$	*Blocking probability*: the probability that a user request is not forwarded by the client
P_{denial}	$\frac{N_{ClReqDis}}{N_{ClReq}}$	*Denial probability:* the probability that a client request is discarded by the server
P_{DS}	*Server denial probability*: the number of the requests discarded by the server out of the total number of received requests	
L_{Ser}	$\frac{N_{ClReqAcpt}}{T_{session}}$	*Server load:* the number of requests per second that the server accepts
L_{Net}	$\lambda N \sum_{i=1}^{N} (1 - P_{BLKi})(2 - P_{DS})$	*Network load*: the number of messages per second that circulate in the network

Table 3.1. Analysis parameters.

3.1. The Discrete-Event Simulator

The discrete-event simulator [FMR03], written in C++, is fully object-oriented and designed upon the following kinds of objects:

1. `Server Node`, which models the server process, according to the automaton shown in Figure 2.1(a).
2. `Client Node`, which models the client process, according to the automaton shown in Figure 2.1(b).
3. `Passive Node`, which represents either a router node in a multicast tree (i.e. a node that forwards messages according to the multicast logic but does not have a client process attached on it), or a client process that does not originate messages because the attached user is a "passive" user, i.e. non interactive.
4. `User Agent`, which generates new requests on behalf of the user. The request generation process follows a chosen statistical distribution, as discussed below.
5. `Event`, which embodies a message exchanged among `Client` and `Server` objects. Upon event reception, a `Node` responds according to its automaton.
6. `Event Scheduler/Dispatcher`, which manages events, stores them in a queue ordered by message delivery times, and dispatches them to destination nodes.

A simple script language is exploited to define the network configuration.

In order to feed the simulation of a cooperative control session with users' requests, the behaviour of group members should be carefully modelled. In [PK99], an empirical characterization of user behaviour, observed in the context of a Web-based courseware system, is reported. The authors found out that lognormal and gamma distributions are a good approximation of the empirical distribution of user requests, though the exponential distribution is often used for analytical and simulation studies. In the simulation runs, the request rate of each user in the group is therefore modelled as a gamma distribution with a given shape parameter.

3.2. Simulation parameters

Prior to simulation runs, the first six parameters listed in Table 3.1 were accurately tuned and varied in order to achieve a deep comprehension of the COCOP protocol.

In particular, the *duration of the session* ($T_{Session}$) was set, for each simulation run, to an amount of time that allowed for deriving performance values with a pre-determined statistical relevance (i.e. with at least a 0.95 probability that the statistical error was below 5%). Simulations are carried out on tightly-coupled sessions which correspond to static logical topologies.

Cooperative control sessions are mainly intended for small/medium sized groups of users. In particular, a cooperative playback session [FN03] is averagely composed of 2 to 10 members. Accordingly, the *number of clients* N was varied from 2 to 15 in the performed simulations, even though large groups ($N>50$) were also investigated in the case of basic topologies.

User activity is performed according to a simple statistical model based on the *Gamma* probability distribution function with shape parameter equal to 2. The parameter characterizing the user activity is the *Mean Request Interarrival Time* ($1/\lambda$, MRIT), which is the average interarrival time between two successive requests issued by the same user. User activity can be further classified in very low (MRIT>=15m), low (10m<=MRIT<15m), medium (5m<=MRIT<10m), high (120s<=MRIT<5m) and very high (MRIT<120s). In the simulations, the value of MRIT was varied within the range {10s, 180s}, to account for analyses of COCOP in sessions with very high/high user activity.

The *link delay* between two adjacent nodes (δ) was set according to the following model:

$$\delta_i = K_f\delta_m + N(K_v\delta_m, \sqrt{K_v\delta_m}) \qquad (1)$$

$$K_f + K_v = 1 \quad K_f, K_v \geq 0 \qquad (2)$$

where δ_m is the mean delay and δ_i is the instantaneous delay for a given message. δ_i is the sum of a fixed part and a variable part. Equation (2) guarantees that the mean of δ_i is equal to δ_m. The variable part of δ_i is generated by a normal random variable whose mean and variance are set to $K_v\delta_m$. The distribution of the normal variable is truncated to $-K_f\delta_m$ in order to assure that δ_i cannot assume negative values. The choice of the normal distribution was made according to the considerations presented in [GL96]. The EED model parameters were set as follows: $\delta_m=0.1s$, which accommodates small regional areas, and $K_f=0.7$, to limit the EED variability.

Moreover, the average time (ΔT_{Elab}) spent by the server to process a client request was set to 0.2s, a typical value in a single-session VoD system, with a uniform variability of ±0.1s.

The *server timer* T_S, used to control the server reactivity and the overall system interactivity degree, was set to 3.0s; T_C, the *client timer*, was set to the same value, thus avoiding the occurrence of deadlock situations, as shown in [FGM02]. The timer values were chosen on the basis of the performance analysis of the ViCRO^C system [FN03].

4. Simulation Results

In this Section we first report the most relevant results concerning the performances of COCOP with star and chain topologies (see also [FMR03]), since these basic topologies can be considered as the building blocks of generic multicast trees.
Then we analyze the behaviour of the COCOP with more complex topologies: binary and n-ary multicast trees, with both active and passive nodes.

4.1 Basic Topologies: Star and Chain

Star.

The star topology (Figure 4.1) can be considered as a model of a particular multicast topology having the peculiarity that all clients experiment similar average delays to reach the server. A passive node (R) routes client requests to the multicast group and to the server. Since all clients are in the same conditions, the performances of a generic client are reported.

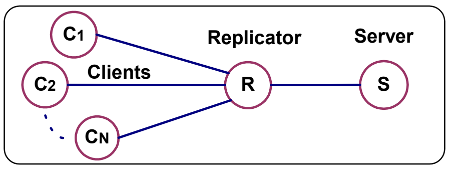


Figure 4.1. A star topology.

In Figure 4.2, the blocking and denial probabilities of `COOP` and `NoCOOP` versus the mean request interarrival time for different values of `N` are shown. This comparison shows that the use of cooperation leads to a light increase of the blocking probability, but to a considerable decrease of the denial probability. This is surely a beneficial effect, and a confirmation of the validity of the cooperation-based protocol. In fact, it is worth noting that a relatively high value of the blocking probability can be accepted by the user, who in a cooperative control session is aware that she/he cannot always be able to issue a request due to the need of limiting the system dynamics and to the existence of contention periods. The blocking probability can also be interpreted as the fraction of time in which the user is unable to submit a request. On the contrary, the denial probability should be as low as possible since the server reject of a client request is to be considered a very unpleasant event.

Looking at Figure 4.2, interesting considerations about the scalability of COCOP can be done. Suppose that denial probability is considered acceptable when it is lower than `0,1`. It can be noted that acceptable denial probabilities are achieved, having a number of clients equal to `2` or `5`, for every value of `MRIT` used in the simulation. However, if the number of clients is `10`, denial probabilities are not acceptable if the `MRIT` is lower than `20s`. Simulations carried out with higher numbers of clients (not reported in Figure 4.2 for sake of readability) confirmed this behaviour. Thus COCOP is exploitable by a high number of clients only if the user activity is moderate or low (`MRIT>5min`). However, it is argued that this is not a strong limitation for two reasons: (i) applications for which COCOP was thought are tailored to small/medium groups of clients (`<15`); (ii) protocol simulations were carried out using very high/high request rates: in real applications the request rate is often far lower w.r.t. the values used in the performed experiments.

In Figures 4.3 the server load and the network load are portrayed for `COOP` and `NoCOOP`. `COOP` exhibits a considerable decrease of the server load, because client cooperation limits the number of client requests, especially of requests that would be likely discarded. For what concerns the network load, there are no remarkable differences between the two protocols.

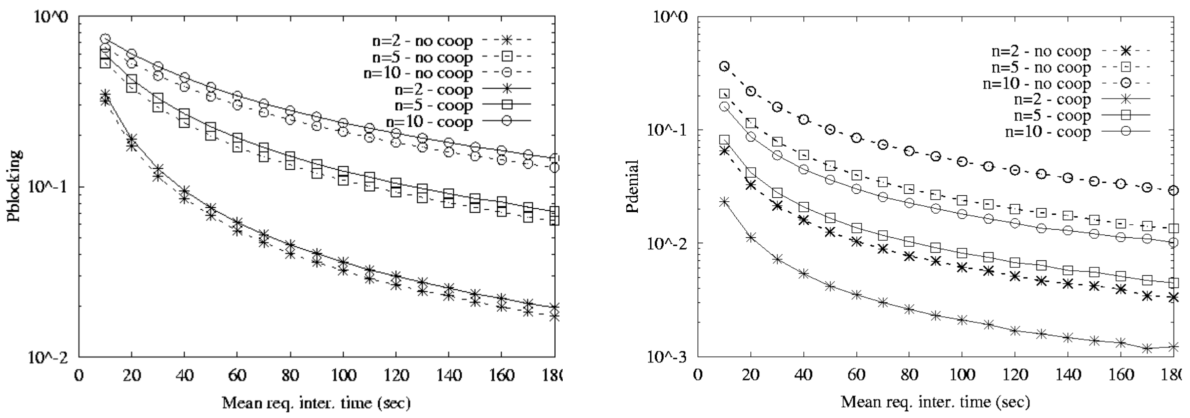


Figure 4.2. Star topology analysis: blocking probability and denial probability of Coop and NoCoop.

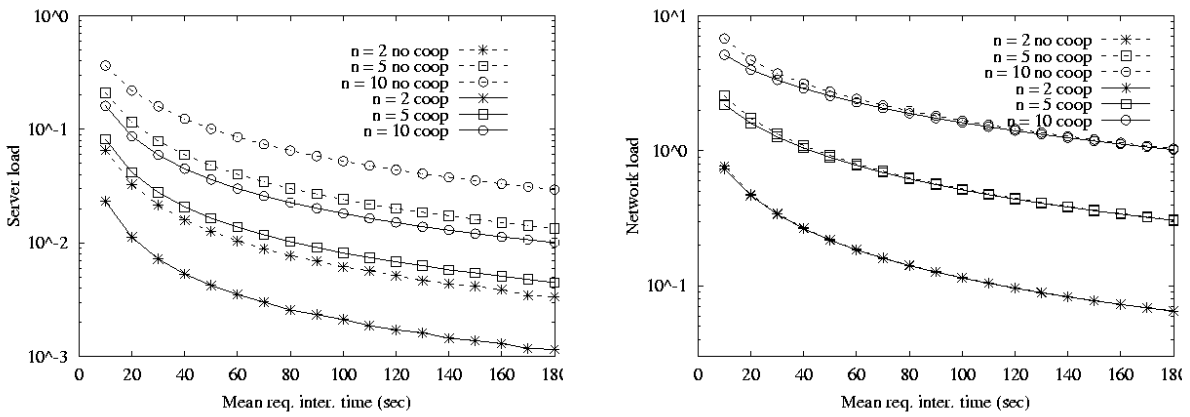


Figure 4.3. Star topology analysis: server load and network load of Coop and NoCoop.

***Chain*.**

The chain topology (Figure 4.4) allows for exploring the case in which clients experiment different delays from the server, and therefore have different chances to control the server process. A linear chain can be formed either as a consequence of a tree degeneration or through a peer-to-peer approach based on an invitation protocol [CHW99] by which the client C_1, already connected to the server, invites C_2, which in turn invites C_3 and so forth.

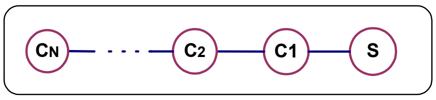


Figure 4.4. A chain topology.

Figure 4.5 shows blocking and denial probabilities of `COOP` for chains with 2, 3, 5, 7 and 10 clients. Since clients are located at different distances from the server, results are reported w.r.t. the client ID, for a given value of the mean request interarrival time (90 s).

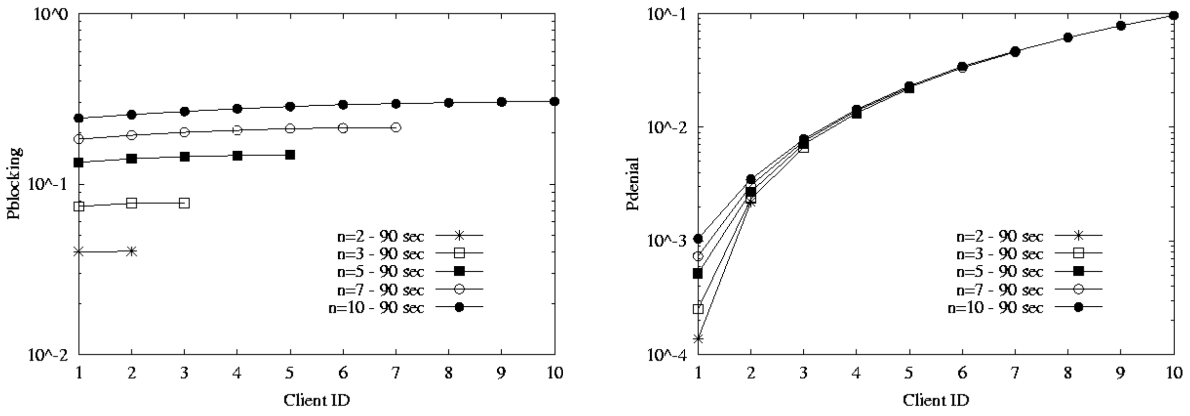


Figure 4.5 Chain topology analysis: blocking and denial probability of Coop.

Figure 4.6 shows the blocking and denial probabilities of `COOP and NoCOOP` for a chain of 5 clients, with given values of the Mean Request Interarrival Time (`20s` and `120s`). It emerges that the major benefit of `COOP`, i.e., the denial probability decrease is even more consistent in a chain configuration than in a star configuration.

However, Figure 4.6 also shows that fairness among clients is not guaranteed with `COOP`. In fact, clients located in the vicinity of the server (or `near` clients) have more chances to control the server than `far` clients, located at the chain tail. In long chains, `near` client requests can preclude `far` clients to control the server. This behaviour is general, even if it is shown for chains with 5 clients. Thus, the `FAIRNESS` version of COCOP was defined by introducing the following fairness mechanism:

- a client, even if being in the `READY` state, has to wait an amount of time T_w to forward a user request to the multicast group. The value of T_w is different from client to client, and is computed depending on the location of the client with respect to the server and the chain tail. In particular, for the client C_i, T_w should be approximately equal to the time delay that a message issued by client C_N (the farthest client) would take to reach client C_i. In order to exploit this approach, each client should know the delay from itself to the farthest client. This information can be easily computed at group formation time. Each time a client joins the group, it measures the delay to the node which it is connecting to, and propagates this information to the other clients, that can recursively calculate their delays to the farthest client. A similar approach is used to periodically update the values of T_w.

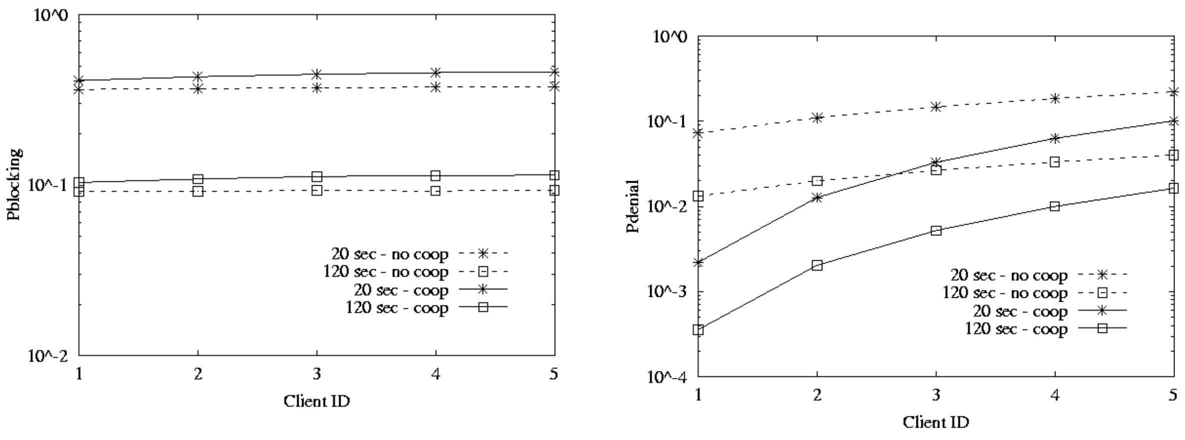


Figure 4.6. Chain topology analysis: denial probability of COOP and NOCOOP for a chain with 5 clients.

Figure 4.7 shows that FAIRNESS is able to assure a much higher degree of fairness with respect to COOP; results are shown for chains with different numbers of clients (2, 3, 5, 7, 10) and a Mean Request Interarrival Time equal to 90s. For short chains, near clients are still slightly favoured because they "get ready" (i.e. their automata get into the Ready state) earlier than far clients after a session state change. This effect is smoothed for long chains, in which "middle" clients are favoured because they are statistically able to sense other client requests earlier than clients located at the head or at the tail of the chain.

For what concerns the server and the network load (see Figure 4.8), simulations show that, as in the star configuration, COOP, if compared with NOCOOP, lowers the server load, whereas little effect can be appreciated on the network load. Furthermore, FAIRNESS causes a slight increment of the server load w.r.t. COOP (but values are still lower if compared with NOCOOP), since far client requests are favoured.

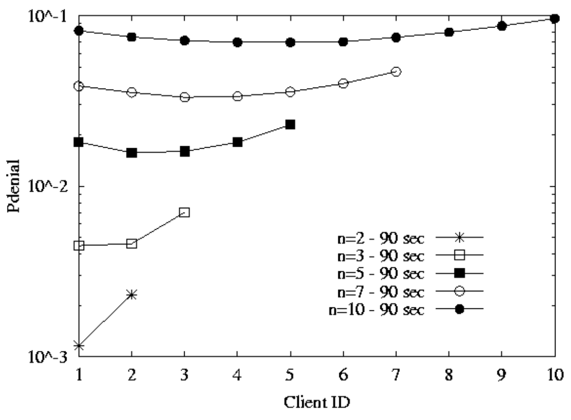


Figure 4.7. Chain topology analysis: denial probability with the FAIRNESS protocol.

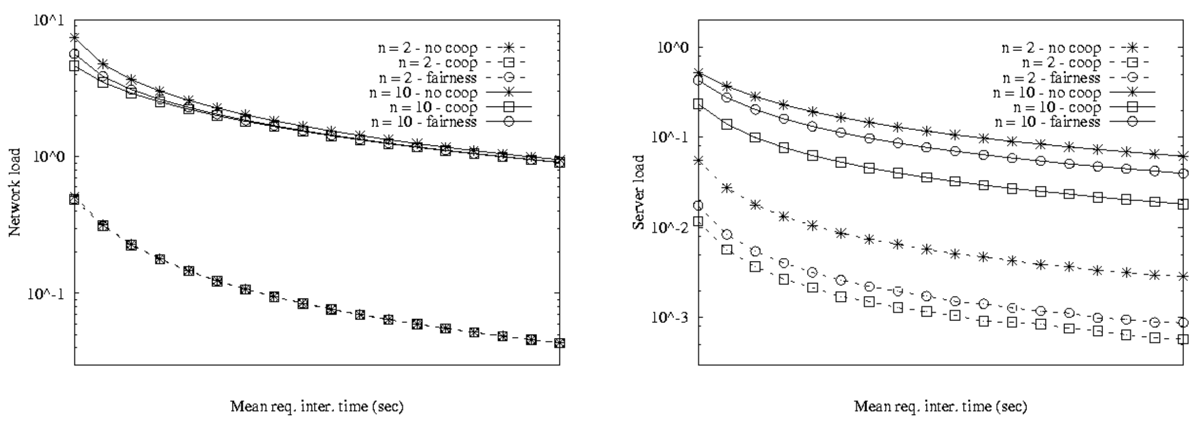


Figure 4.8. Chain topology analysis: network load and server load.

4.2 Tree Topologies: Binary Trees and N-ary Trees

Binary Trees.

The binary tree topology allows the analysis of COCOP in the context of peer-to-peer networks whose nodes coincide with client and server processes. Such processes can be organized according to an application-level multicast (ALM) protocol based on delivery trees (such as mesh-first and tree-first ALM approaches [BB03]). In particular, binary trees are fully representative of multicast data delivery paths, as stated in [NB97].

Our analysis has taken into account all balanced and unbalanced binary tree topologies having a tree depth <= 3 and up to 14 client nodes. We found that several results obtained with balanced trees are similar to those obtained with star and chain topologies:

- the cooperation mechanism leads to a considerable decrease of denial probability
- client behaviours are similar for clients belonging to the same level, while clients closer to the server perform slightly better than farther clients
- the fairness approach is effective in limiting unfairness

We will focus our attention to the FAIRNESS protocol variant, possibly in comparison to the other two variants.

When analyzing performances of unbalanced binary trees, we found that the denial probabilities of a client depend not only on the client's distance from the server, but also on the number of clients that belong to the same branch of the tree. Observing clients that have similar distances from the server, we saw that clients belonging to "heavier" branches (i.e. branches with a higher number of nodes) are favored with respect to clients located in lighter "branches", as an effect of cooperation among clients. This can be seen in Figure 4.9, where an unbalanced tree with 5 nodes is considered, and the three protocol variants (COOP, NoCOOP and FAIRNESS) are compared. In particular, if we consider nodes belonging to the same level, we observe that nodes populating the heavier branch of the tree perform better (e.g., see node 1 vs. node 2, nodes 3-4 vs. node 5). The reason is that the cooperation among nodes belonging to heavier branches (e.g., 1, 3, 4) is more effective than the cooperation between nodes in lighter branches (2, 5). In other words, the competition appears to be more between the opposite branches of a tree than between the nodes belonging to the same branch. Figure 4.9 also shows that FAIRNESS smoothes the performance differences among nodes.

An unbalancing of trees is also experimented when, in a topologically balanced structure, one or more nodes are passive. For example, starting form a balanced tree with 6 clients, if one of the nodes is switched from the active to the passive modality, the tree becomes unbalanced. Figure 4.10 shows the effect of this variation: a comparison is made between denial probabilities obtained when node 1 (grey in Figure 4.10) is active and when it is passive. Since, in the unbalanced tree, the number of active nodes is reduced, all nodes experiment lower denial probabilities with respect to the balanced tree. However, this effect is negligible for nodes belonging to the same branch as the passive node (nodes 3 and 4), while it is more relevant on the nodes of the opposite branch. Therefore, if one node becomes passive (either because the corresponding client process disconnects or the user does not originate any request), performances of neighbour nodes are slightly affected, while far nodes are remarkably favoured. This can be considered as a further proof of benefits deriving from the collaboration among active nodes belonging to the same tree branch.

The performance comparison shown in Figure 4.10 is remarkable also because it represents a first step into the analysis of the dynamic behaviour of the system: in fact client 1 may switch form the active to the passive modality during a single multicast session, either because it disconnects from the multicast application (but continues to forward messages according to the multicast logic), or because the attached user, during a significant amount of time, does not generate requests.

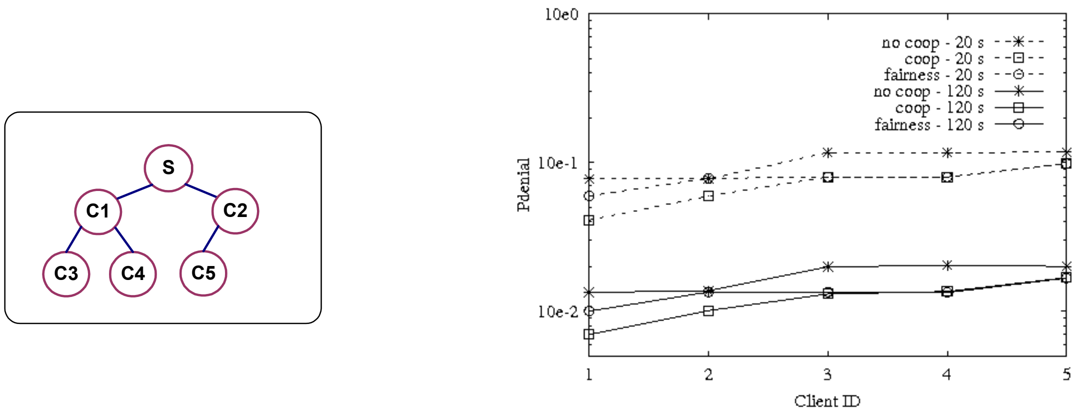


Figure 4.9. Binary tree topology analysis: Pdenial for a unbalanced binary tree with `N=5`**.**

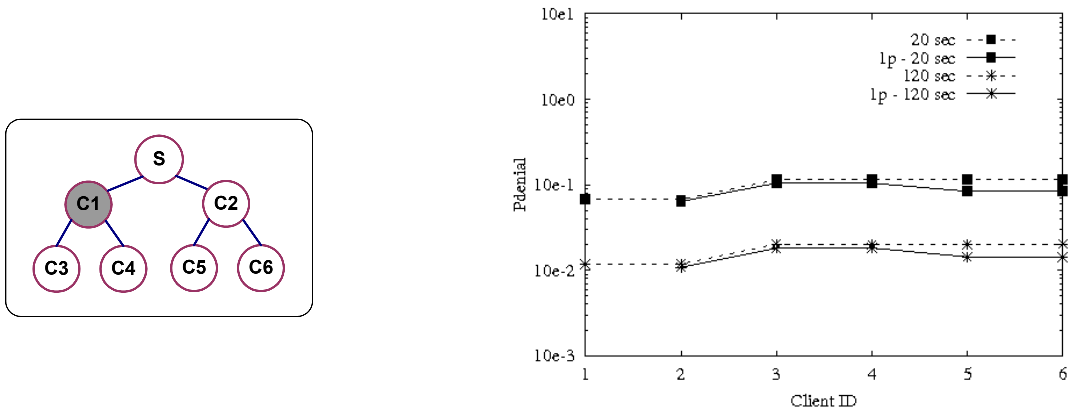
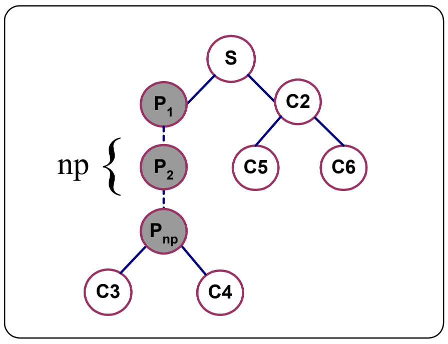


Figure 4.10. Binary tree topology analysis, Pdenial: comparison between a balanced binary tree with `N=6` **and a similar tree where one node is passive.**

Several simulations have been performed to investigate the effect of the presence of passive nodes between the server and the clients. In Figure 4.11, a tree with 5 active nodes is filled with a number of `np` passive nodes interposed between the server and nodes 3 and 4: if `np` is equal to 0, nodes 3 and 4 are directly connected to the server. Note that increasing the parameter `np` is equivalent to increasing the link delay between the server and nodes 3 and 4. From the performance graph comes out that a higher value of `np` causes higher denial probabilities for nodes 3 and 4, while a little positive effect is noted on nodes 2, 5 and 6, belonging to the opposite branch. Therefore, we can conclude that the right branch nodes are favored if the number of nodes in the left branch is reduced, but they are only slightly affected if that number is kept invariant and the left branch nodes are pushed away from the server.



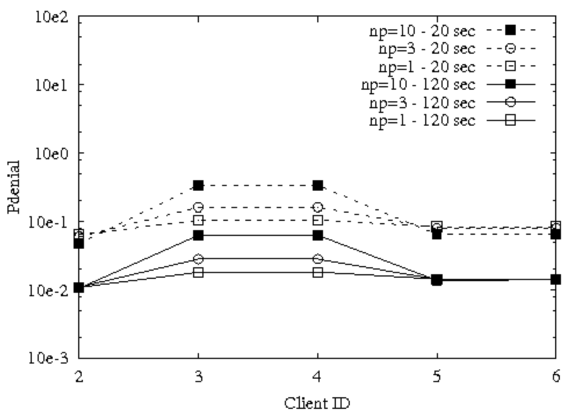
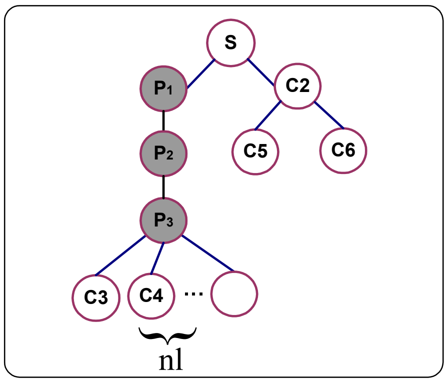


Figure 4.11. A binary tree with a sub-chain composed of `np` **passive nodes. Denial probabilities versus** `np`**.**

N-ary Trees.

After examining performance of binary trees, in this section we conclude with the analysis of general n-ary trees, with both active and passive nodes. In particular, the impact of passive nodes on client performances, along with the effect of increasing the number of active nodes, was further investigated by attaching a variable number of active nodes to a fixed chain of passive nodes. As shown in Figure 4.12, we connected `nl` siblings nodes to a chain of three passive nodes. Performances obtained with `nl` = 2, 4, 6, and 8 are compared. In Figure 4.12, we only show the performances of two sibling nodes (i.e. nodes 3 and 4), since other sibling nodes are topologically identical and consequently their performances are similar. Again, increasing of the number of active nodes on one branch causes significant performance variations (i.e. increasing of denial probabilities) only on the opposite branch.



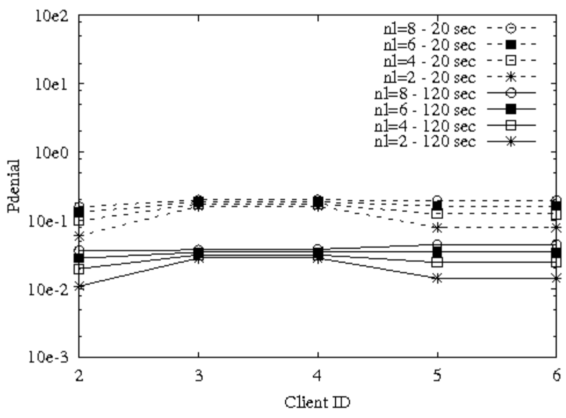


Figure 4.12. A binary tree with a sub-chain composed of `np` **passive nodes. Denial probabilities versus** `np`**.**

Finally, in Figure 4.13 a more complex n-ary tree topology, featuring 12 active nodes distributed over 5 depth levels, is shown; performances of the three protocol variants are compared. Once again, the benefits resulting from the cooperation mechanism are evident since, with `NoCOOP`, denial probabilities are always the highest. Also interesting is the comparison between `COOP` and `FAIRNESS`. With `FAIRNESS`, performance values of different nodes are all included in a small range, opposite to what happens with `COOP`: with `COOP`, for example, node 2 experiments denial probabilities that are significantly lower with respect to other nodes. Furthermore, Figure 4.13 shows that the use of `FAIRNESS` can sometimes invert performance differences among nodes. As an example, consider node 6 and 7: with `COOP`, node 6 experiments lower denial probabilities since it belongs to a more "populated" branch; however, with `FAIRNESS`, this advantage is inverted because node 6 must wait a considerable time (equals to the delay from node 6 to node 10, 11 or 12) before sending its requests, while node 7 has no nodes below itself, so it can send its requests immediately. Moreover, the blocking probability of the client nodes in the case of COOP, NoCOOP, and FAIRNESS, is stable and no remarkable differences can be appreciated.

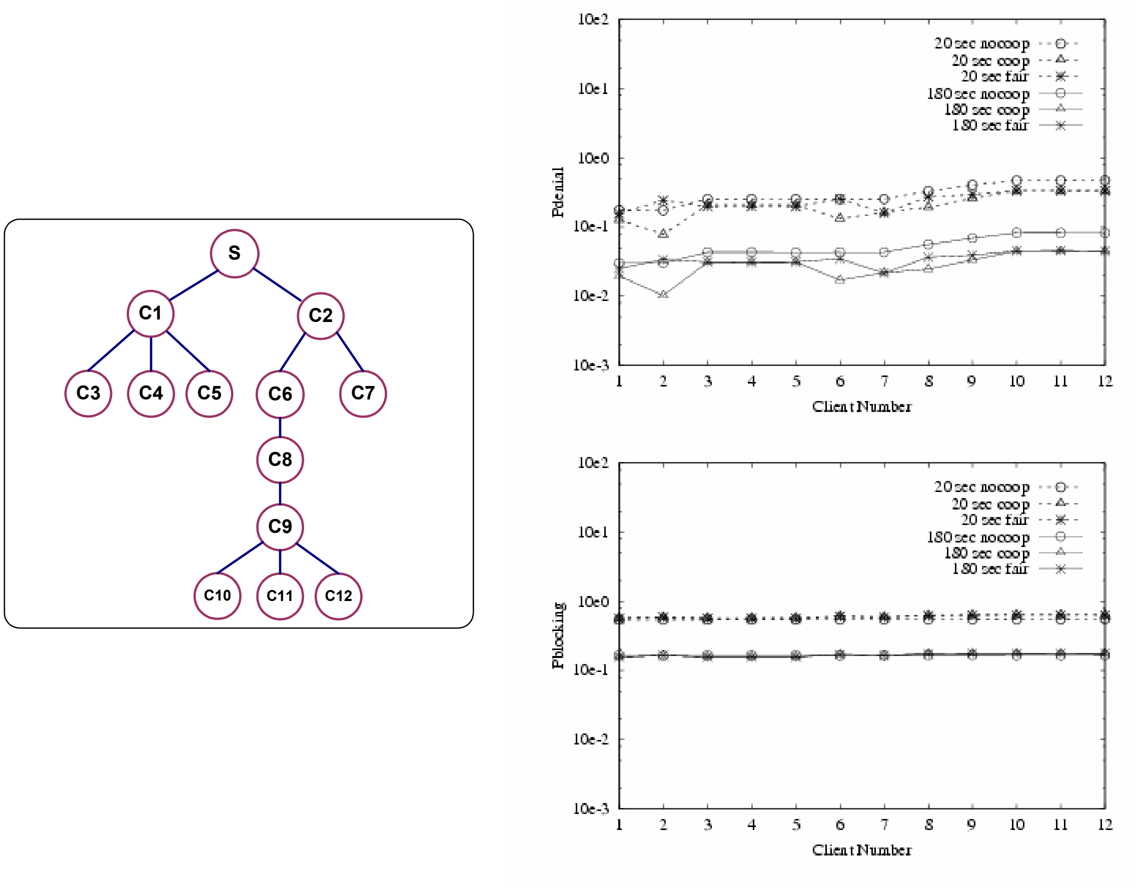


Figure 4.13. Comparison of COOP, NoCOOP and FAIRNESS for an n-ary tree with 12 nodes and 5 levels.

5. Conclusions

Integrating data multicasting on-demand with efficient group-based control protocols can feature new kinds of media services which future Content Distribution Networks as well as spontaneous media peer-to-peer networks can incorporate and provide. This paper has presented the modelling and the analysis of an application-level control protocol, COCOP, which is suited for the cooperative control of a server providing real-time, multicast services. Novelty rests in the integration of a competitive access mechanism with a cooperation-based policy, which basically avoids the adoption of higher-level policies (e.g. token-based) so increasing client/server interactivity. Protocol validation is performed on several basic and general topologies (star, chain, binary tree, n-ary tree) by using an event-driven simulation framework. Simulations show that the cooperative approach improves performances by guaranteeing that a client has a higher probability to control the server. Moreover, unfairness among clients is mitigated by enabling a fairness mechanism.

References

[BB03] S. Banerjee and B. Bhattacharjee, “A Comparative Study of Application Layer Multicast Protocol,” http://www.cs.umd.edu/users/suman/pubs/compare.ps.gz.

[CGK01] C.D. Cranor, M. Green, C. Kalmanek, D. Shur, S. Sibal, C.J. Sreenan, and J.E. Van der Merwe, “Enhanced Streaming Services in a Content Distribution Network”, *IEEE Internet Computing,* **5**(4), pp 66-75, 2001.

[CHW99] J. Crowcroft, M. Handley, and I. Wakeman, “*Internetworking Multimedia*,” Morgan Kaufmann Pub., San Francisco (USA), 1999.

[CKV01] G.V. Chockler, I. Keidar, and R. Vitenberg, “Group communication specifications: a comprehensive study”, *ACM Computing Surveys,* **33**(4), pp. 1-43, 2001.

[DG99a] H.P. Dommel and J.J. Garcia-Luna-Aceves, “Group Coordination Support for synchronous Internet Collaboration,” *IEEE Internet Computing*, pp. 74-80, Mar/Apr. 1999.

[DG99b] H.P. Dommel and J.J. Garcia-Luna-Aceves, "Efficacy of Floor Control Protocols in Distributed Multimedia Collaboration," *Cluster Computing*, **2**(4), 1999.
[FGM02] G. Fortino, J.C. Guerri, C. Mastroianni, C. Palau, and W. Russo, "Development and validation of a multicast client/server model for cooperative control sessions," *Proc. of IASTED Communications and Computer Networks (CCN'02)*, Cambridge (MA), USA, pp. 514-519, Nov. 2002.
[FJM97] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing", *IEEE/ACM Transactions on Networking,* **5**(6), pp. 784-803, 1997.
[FMR03] G. Fortino, C. Mastroianni, and W. Russo, "Performance Analysis of an Application-level Cooperative Control Protocol," *Proc. of IEEE Symposium, on Network Computing and Applications (NCA'03)*, Cambridge (MA), USA, Apr. 2003.
[FN03] G. Fortino and L. Nigro, "Collaborative Learning on-Demand on the Internet MBone", in *Usability Evaluation of Online Learning Programs*, C. Ghaoui Ed., Idea Publishing Group, USA, 2003.
[GL96] J.F. Gibbon and T.D.C. Little, "Use of Network Delay Estimation for Multimedia Data Retrieval", *IEEE Journal on Selected Areas in Communications* **14**(7), pp. 1376-1387, 1996.
[M99] S. McCanne, "Scalable Multimedia Communication using IP multicast and Lightweight Sessions," IEEE Internet Computing, pp. 33-45, Mar/Apr. 1999.
[NB97] J. Nonnenmacher and E.W. Biersack, "Performance Modelling of Reliable Multicast Transmission," *Proc. of IEEE INFOCOMM'97*, Kobe, Japan, Apr. 1997.
[PK99] J. Padhye and J. Kurose, "Continuous Media Courseware Server: a Study of Client Interactions," *IEEE Internet Computing,* **3**(2), pp. 65-72, 1999.
[SRC98] A. Schuett, S. Raman, Y. Chawathe, S. McCanne, and R. Katz, "A Soft State Protocol for Accessing Multimedia Archives," *Proc. of NOSSDAV*, Cambridge, UK, July 1998.
[WMK94] B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," *Theory and Practice in Distributed Systems*, LCNS 938, Springer Verlag, 1994.

Appendix A

In the following we give a high-level definition of the request and reply messages and their exploitation for formalizing the set of rules which meets the requirements 1-3 described in §2.2.

- $\mathtt{Rq}_i^{C_K} = \left[\mathtt{T}_i;\mathtt{REQ}\right]^{C_K}$: the i-th request sent by the k-th client. It contains a timestamp $\mathtt{T_i}$ set to the request transmission time and a request content `REQ` including the request payload and related data. The requests are partially ordered whereas the total ordering is on a per-client basis:
  $\forall \mathtt{Rq}_i^{C_K}, \mathtt{Rq}_j^{C_K} \quad \mathtt{Rq}_i^{C_K} \succ \mathtt{Rq}_j^{C_K} \Leftrightarrow \mathtt{T}_i < \mathtt{T}_j$, where "$\succ$" means "temporally precedes".
- $\mathtt{Rp}_i = \left\lfloor \mathtt{T}_i;\mathtt{C}_K;\mathtt{T}_{C_k};\mathtt{REP}\right\rfloor$: the i-th reply sent by the server. It contains a timestamp $\mathtt{T_i}$ set to the reply transmission time, the identifier $\mathtt{C_K}$ of the client issued the related request, the timestamp $\mathtt{T_{Ck}}$ of this request, and the reply content `REP` possibly containing reply data. The session time is kept by the server so that the reply timestamp embodies the instant in which a session state change occurred. Server replies are totally ordered: $\forall \mathtt{Rp}_i, \mathtt{Rp}_j \quad \mathtt{Rp}_i \succ \mathtt{Rp}_j \Leftrightarrow \mathtt{T}_i < \mathtt{T}_j$
- $\mathtt{SST(IDC,\ T_{REQ},\ N_{REQ})}$: the session state table managed by the server. For each session client it contains the identifier of the client IDC, the timestamp $\mathtt{T_{REQ}}$ of the last request accepted by the server, and the number of requests $\mathtt{N_{REQ}}$ accepted.
- `CST(ID, T)`: a session state table managed by the client. It contains for each client and for the server the identifier `ID` and the timestamp T (also called $\mathtt{T_{LRR}}$) of the last reply received if `ID` identifies the server, or the timestamp T (also called $\mathtt{T_{ACP}}$) of the last request issued by the client `ID` and accepted by the server, if `ID` identifies a client.
- *Requirements 1* (*Source-ordering of Requests)*: given an incoming request msg $\mathtt{Rq}_i^{C_K}$, if $\mathtt{T_i > T_{REQ}}$ it is passed to the server state machine (ssm) for processing and the timestamp $\mathtt{T_{REQ}}$ related to the client $\mathtt{C_K}$ in the table `SST` is updated to $\mathtt{T_i}$; otherwise it is discarded.
- *Requirement 2* (*Source-ordering of Replies*): given an incoming reply msg $\mathtt{Rp}_j$ if $\mathtt{T_i > T_{LRR}}$ the $\mathtt{T_{LRR}}$ is updated to $\mathtt{T_i}$, the timestamp $\mathtt{T_{ACP}}$ of the client $\mathtt{C_k}$ in the table `CST` is update to $\mathtt{T_{Ck}}$, and finally the message is passed to the client state machine (csm) for processing; otherwise it is discarded.
- *Requirement 3* (*Causal ordering of Request/Reply pairs*): given an incoming remote request msg $\mathtt{Rq}_i^{C_K}$ if $\mathtt{T_i > T_{ACP}}$ the request is passed to csm; otherwise it is discarded.

The filters associated to the server and the clients defined by means of this set of rules are presented using a script language in Figures A.1-2. In particular the rule applied by the server filter for each incoming request (msg

$\mathrm{Rq}_i^{C_K}$) meets the requirement 1, whereas the rules applied by the client for each incoming remote request (msg $\mathrm{Rq}_i^{C_K}$) and reply (msg Rp_j) meet the requirements 2-3:

```
void serverFilter(Message msg){
 if (timestamp(msg) > SST[identifier(msg)].getTimestamp()){
  SST[identifier(msg)].setTimestamp(timestamp(msg));
  ssm.handler(msg);
 }
 else discard(msg);
}
```

Figure A.1. Script of the Server Filter.

```
void clientFilter(Message msg){
 if (identifier(msg)==ID_SERVER) {
  if (timestamp(msg) > CST[ID_SERVER].getTimestamp()){
   SST[ID_SERVER]=timestamp(msg);
   SST[clientIdentifier(msg)].setTimestamp(clientTimestamp(msg));
   csm.handler(msg);
  }else discard(msg);
 }else {
   if (timestamp(msg) > CST[identifier(msg)].getTimestamp())
    csm.handler(msg);
   else discard(msg);}
}
```

Figure A.2. Script of the Client Filter.