Load balancing algorithms for TINA networks

Maria Kihl, Niklas Widell and Christian Nyberg

Department of Communication Systems, Lund Institute of Technology,
BOX 118, 221 00 Lund, Sweden

TINA is an open, object oriented, distributed telecom architecture, with many concepts taken directly from the latest computer research. In TINA, instances of the same object type can be placed on different physical nodes. Therefore, the network performance can be improved by introducing load balancing algorithms. These algorithms should distribute the traffic between the object instances in such way that the overall throughput and setup time are improved. We discuss and examine a number of simple distributed load balancing algorithms, that do not require any extra load information exchange between the nodes. The results show that it is difficult to find an algorithm that behave well for all traffic situations. The main problem is that the algorithms have not enough information about the load situation on the different nodes, since no load information is exchanged between the nodes. This problem can be solved by adding the feasibility of load status information to the TINA protocols.

1. Introduction

The trend in modern telecommunication systems is to provide a whole array of services, such as Freephone and Video conferences. To support these services, computers are integrated in the networks, adding for example central databases and allowing more complex service logic to be implemented. The Intelligent Network (IN) is an attempt to create a set of reusable components that can be connected to perform a new service. IN uses special nodes, called Service Control Points (SCPs), that are capable of executing complex services. If a service cannot be performed locally in the switch, it is passed on to the SCP. However, INs are not flawless. For instance, SCPs are potential bottlenecks and management of the network is difficult. Further, there is little vendor independence, which is an important factor in today’s deregulated telecom market.

To solve these problems, the Telecommunication Information Networking Architecture (TINA) is being developed. TINA is an open, object oriented, distributed telecom architecture, with many concepts taken directly from the latest computer research. TINA is intended to be a good platform for service development in a rapidly expanding telecom environment. TINA is developed by a consortium which includes network operators and telecommunication and computer equipment suppliers. It is based on the Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG), that is a standard for object distribution and communication in computer networks. Services that may be provided by a TINA system include voice-based services, interactive multi-media services, information services and management services.

In TINA, instances of the same object type can be placed on several nodes. Since the traffic thereby must be distributed among the object instances, good load balancing algorithms could increase the network performance. The main objectives of a load balancing algorithm is to improve the throughput and lower the setup times by distributing more traffic to lightly loaded nodes than to nodes with a high load.

Several papers have discussed load balancing and load sharing in computer networks. However, none of these papers examine load balancing in TINA networks. Kremien and Kramer [6] discussed load sharing algorithms for distributed systems. Kunz [7] examined how the network nodes should exchange load status information to be used in load balancing schemes. Jordan and Varaiya [8] investigated call admission control policies for communication networks that support several services. Here, it is assumed that the control scheme knows the status of all network resources. Ramakrishnan *et. al.* [9] defined a optimization problem for how to assign a number of tasks to a number of processors in order to minimize, for example, the maximum completion time.

This paper extends the work by Kihl *et. al.* [14][15] and it is partly based on the Master Thesis by Widell [18]. We discuss and examine feasible load balancing algorithms for TINA. The algorithms do not require any load status communication between the nodes, since they only consider the number of rejected signals. The investigations show that the algorithms behave differently depending on the traffic situation. Therefore, we also have a discussion about alternative algorithms that require some load status communication between nodes. Such an algorithm should have a good behaviour during all traffic situations.

2. TINA

The TINA architecture provides a set of concepts and principles to be applied in the development of software for telecommunication systems (see Chapman and Montessi [1]). TINA is very comprehensive. In order to be a fully distributed architecture, TINA contains a number of domains. The objects in two different domains are logically separated from each other. One domain is the *retailer domain*, which is intended to contain components necessary to create a “market place” for services provided in the *service provider domain.* The *user domain* contains the users of services and items closely related to the user.

Further, there are in TINA three types of sessions, that perform a set of activities during a specific period in time. The *Access session* deals with authentication and authorization of users, support of users’ preferences, support for mobility and control of service interactions.The *Service session* provides users with an environment to support the execution of a service. Finally, the *Communication session* supports the activities needed to establish the communication between users.

The *service component* (SC) is an abstraction that encapsulates data and processing. It consists of a number of computational objects and provides a set of capabilities that can be used by other objects through two types of interfaces; the *stream interface* and the *operational interface*. The stream interface connects communication endpoints producing or consuming information flows, for example video or voice bitstreams. The operational interface deals with operations and requests from one object to another.

The TINA-consortium has defined a set of generic service components. Here follows a list of the more important ones:

- *Communication Session Manager (CSM):* The computational counterpart of the Communication session.
- *Service Session Manager (SSM):* Supports the capabilities that are shared among the users in a Service session.
- *User Session Manager (USM):* The software representation of a user in the retailer domain.
- *Initial Agent (IA) and User Agent (UA):* Act on behalf of a user within the network.
- *Provider Agent (PA):* Is the user’s point of contact with the provider.
- *User Application (UAP):* Represents in the user domain the application that the end-user needs to use a service.
- *Terminal CSM (TCSM):* Supports the Communication Session in the user domain.
- *Service Factory (SF):* Creates instances of SSMs and USMs.
- *Generic Session End Point (GSEP):* Connects the UAP and the USM.

A TINA application consists of computational objects that interact with each other. One feature in TINA is that an object does not have to know on which node another object is implemented. Instead, logical addresses are used in the communication. To accomplish this, TINA has a *Distributed Processing Environment (DPE)* that is used for the communication between objects (see Graubmann *et al.* [2]). The DPE knows on what nodes the objects are placed on. This means that from the computational objects' point of view, there is no difference between local and remote communication.

The DPE supports a number of services that are available for all applications. Here follows some examples of these services:

- *Trading service:* Helps computational objects to find appropriate objects to interact with.
- *Notification service:* Sends broadcasted messages, that may be picked up by any listening computational object.
- *Performance monitoring service:* Allows operators to monitor network performance.

3. A TINA service

Minetti and Utsunomiya [3] describe a simple TINA service in which two end-users exchange data via stream interfaces on the users' application objects (for example, an ordinary telephone call). In this paper the same service is used in the investigations. This section contains a description of how the service is modelled in the investigations. In the model only the setup of the service is considered.

The service uses ten different service components belonging to the user domain and the provider domain. The user domain components are Provider Agent (PA), User Application (UAP), Generic Session End Point (GSEP) and Terminal Communication Session Manager (TCSM). The components in the provider domain are User Agent (UA), Initial Agent (IA), User Session Manager (USM), Service Session Manager (SSM), Communication Session Manager (CSM) and Service Factory (SF).

In the investigations a simplified SC model is used. First, the SCs in the user domain are modelled as one SC called *USER*. The reason for this is that the user domain components will probably be placed on the same physical node close to the users. Second, the IA and PA are modelled as one SC called *AGENT*. The other SCs are called the same as before, that is *USM*, *SSM*, *CSM* and *SF*.

The setup of a call (that is, the setup of the stream interface) consists of a number of signals being exchanged between the objects. Sometimes, the objects have to use a so called DPE service, for example the trader service. Table 1 contains the number of signals and DPE service requests that is needed to setup a call.

4. Network model

The network consists of K nodes that communicate via a signalling network. We assume that the network is very fast which means that the switching times are negligible. The network supports the service described in Section 3. Each node handles a number of SCs. The communication between the SCs is performed via a DPE that is placed on each node. A particular SC type can be placed on several nodes. In TINA, the computational objects in a service component can be placed on different nodes. However, we assume that all objects in an SC are placed on the same node. Further, SCs do not migrate in the network, since we assume that a steady state concerning the placement of SCs has been reached. Figure 1 shows a network with three nodes and three types of SCs.

New service requests arrive at the network in a Poisson stream. The requests are evenly distributed among the USER objects. When a USER object receives a service request, the setup sequence starts. The setup sequence consists of a number of signals that are sent between the SCs (see Section 3). Each signal generates a job that must be processed on the node on which the receiving SC is placed. After the processing, the next signal in the setup sequence is transmitted.

Table 1
Signalling model

	USER	AGENT	USM	SSM	CSM	SF
Number of normal signals	10	9	11	7	2	5
Number of DPE service requests	0	4	0	1	0	0

The nodes are modelled as single server systems with infinite FIFO job queues. We assume that they have unlimited memory capacity. The jobs belonging to the SCs on a node are executed concurrently with a simple time-sharing algorithm. The execution time for a job depends on what kind of SC it belongs to. If the SC uses a DPE service, the execution time is multiplied with a certain factor.

The transmission times are modelled as an added execution time in the sending and receiving nodes. These times represent the packing and unpacking of protocols. We assume that the DPE is intelligent enough to be able to separate between local and remote communication. Therefore, the transmission time is zero if a signal is sent between two SCs that are placed on the same node. Note that the SCs themselves have no knowledge about the physical addresses. Only the DPE knows on which network node a particular SC is placed.

Each node has an internal overload control scheme for protecting themselves. The objective of the internal overload control scheme is to keep the load on the node at a target load, ρ_t . The scheme measures the average load on the node during intervals of length τ seconds. If the measured load is above the target load, only a certain fraction of the incoming signals is accepted. The acceptance probability during interval *k*, *p(k)*, is calculated as follows. Let the measured load during interval *k-1* be $\rho(k-1)$. If $\rho(k-1) > \rho_t$ then *p(k)=p(k-1)-step*, else *p(k)=p(k-1)+step*. *p(k)* can never be less than zero or greater than one.

One problem with an internal control is that it is not very efficient (see Kihl [15] or Houck et. al. [11]). In order to achieve a high network throughput, calls should be rejected as early as possible. If a call is rejected late, much processing capacity is wasted. Therefore, in order to improve the internal control scheme a call can be rejected only the first time it uses a service component. If a call has had one accepted signal to a particular SC, all other signals to this SC are accepted. In this way, the overall network throughput is improved. However, there will still be much rejection in the network, since a call consists of signals to several different SCs. To achieve a maximum throughput, a global control scheme which tries to optimise the overall network performance should be implemented. We have decided not to implement such a control scheme, since this scheme would interfere with the load balancing algorithms that are the main concern of this article.

5. Load balancing algorithms

In TINA networks, multiple instances of a service component can be placed on different nodes. One result of this is the feasibility of load balancing. If one node suffers from heavy traffic, the other nodes can help this node by sending the signals elsewhere. The main objectives of a load balancing algorithm are to improve the *throughput* and the *setup time* in the network.

Since load balancing has not yet been considered during the TINA development, it is not certain how much a load balancing algorithm might know about the network. Therefore, we have investigated a number of simple load balancing algorithms. We have decided to place the load balancing mechanism

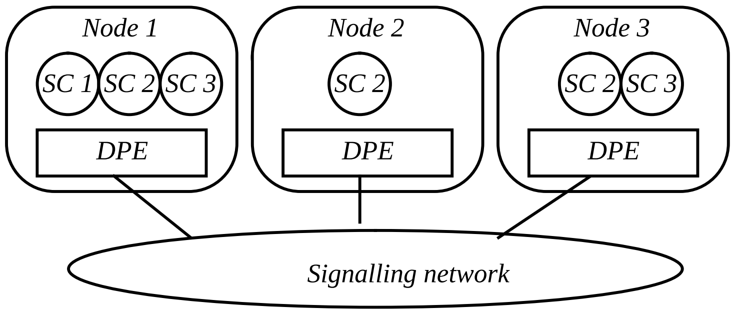

Figure 1. Network with three nodes and three SC types

in the DPE on each node, since it is only the DPE that has knowledge about the other nodes. The DPE uses the algorithm to find an appropriate SC instance. The algorithm is used only the first time a signal is sent to a particular SC type during the setup sequence. The next time a signal in the same setup sequence is sent to this SC, the same SC instance is used.

The algorithms use measurements to decide the current load status of the nodes. These measurements are performed during time intervals, called control intervals, of length τ seconds. At the end of each control interval, the load status is updated. The algorithms are described below.

5.1 Random

In this algorithm, an SC is chosen randomly, with the same probability for each node that contains instances of the particular SC type. This algorithm is of course very simple to implement in the network, and can be considered as a “default” algorithm.

5.2 Shortest queue

In this algorithm, the DPE selects the SC instance that is placed on the node with the shortest job queue. The shortest queue algorithm can be considered as an optimal algorithm concerning the setup time. However, it would be very difficult to implement this algorithm in a real network, since the DPE will not have this knowledge about other nodes. Therefore, this algorithm is only used for comparison with the other three strategies.

5.3 Acceptance probabilities

In this algorithm, each node uses two metrics: $N_{tot}(i)$ and $N_{rej}(i)$, where $N_{tot}(i)$ stands for the number of signals sent to node *i* and $N_{rej}(i)$ stands for the number of signals sent to node *i* that has been rejected. At the end of the control intervals, each node estimates the acceptance probabilities for the other nodes. Let $A(i)$ denote the estimated acceptance probability for node *i*. $A(i)$ is estimated as

$$A(i) = \frac{N_{tot}(i) - N_{rej}(i)}{N_{tot}(i)}$$

The node chooses an SC instance on node *i* with the probability $P(i)$. $P(i)$ is given by

$$P(i) = \frac{A(i)}{\sum_{i \in V} A(i)}$$

where *V* is the set of nodes that contain the particular SC type.

In order to delete the effects due to statistical fluctuations, $P(i)$ is low pass filtered. If $P_k(i)$ is the estimated probability in the *k*th interval, the low pass filtered probability, *P*(i),* is calculated as

$$P^*(i) = \alpha \cdot P_k(i) + (1 - \alpha) \cdot P_{k-1}^*(i)$$

where $P_{k-1}^*(i)$ is the low pass filtered probability from the previous interval.

5.4 Load status values

In this algorithm, each node uses a metric *L(i),* which denotes the load status of node *i*. The load status is updated at the end of the control intervals. The update is performed as follows. If there have been any rejected messages from node *i*, *L(i)* is decreased with one. Otherwise, *L(i)* is increased with one. L(i) can never be less than L_{min} and more than L_{max}.

The node chooses an SC instance on node *i* with the probability *P(i)*. *P(i)* is given by

$$P(i) = \frac{L(i)}{\sum_{i \in V} L(i)}$$

where *V* is the set of nodes that contain the particular SC type.

6. Investigations

The load balancing algorithms were examined in a number of simulation cases. The main objective of the investigations was to find better load balancing algorithms than the Random algorithm. Therefore, all results should be compared with the results for the Random algorithm. The parameters used in the simulations are shown in Table 2.

6.1 Execution times

It is of course very hard to estimate appropriate execution times for the SCs, since there are no TINA systems in operation today. The execution times depend on the amount of work related to each signal. If the signal requires a DPE service, even more execution time is needed. We have decided to use the following execution times. Signals belonging to USER, AGENT, USM and SSM objects have an execution time of 1 ms. Signals belonging to CSM objects require 4 ms, since we assume that the CSM perform more complex tasks than the previous objects. Further, signals belonging to SF objects require an execution time of 2 ms. If the SC uses a DPE service, the execution time of this signal is multiplied with five. Further, the transmission times are modelled as extra execution times in the sending and receiving nodes. This extra execution time is 0.25 ms both for the sending and receiving node.

Table 2
Parameter settings

	Value
Number of nodes, *K*	10
Control interval length, τ	1 second
Parameter in control scheme, *step*	0.05
Parameter in low pass filter, α	0.2
Minimum load status value, L_{min}	1
Maximum load status value, L_{max}	30

Table 3
Service component distributions

	USER	AGENT	USM	SSM	CSM	SF
Balanced distribution	1-5	6-8	6-8	9 and 10	9 and 10	9 and 10
Focused distribution	1-5	6-8	6-8	6 and 9	6 and 10	6 and 10

Table 4
Arrival rates (calls/second)

	Balanced distribution		Focused distribution	
Traffic profile	Low	High	Low	High
Arrival rate	29	67	25	60

6.2 Service component distributions

Since there can be several instances of each SC in a TINA network, the instances must be distributed among the nodes in some way. We used two SC distributions, one *balanced* distribution in which the load was evenly shared among the nodes and one *focused* distribution in which node 6 received more load than the other nodes.

If nodes containing the USER component become overloaded, calls will be rejected before they enter the network. This means that they are rejected before the load balancing can have any effect. Therefore, nodes 1-5 only contain the USER component. Table 3 shows the SC types on each node.

6.3 Traffic profiles

Two basic traffic profiles were used in the simulations. The first profile is *Low* traffic, in which the arrival rate is so low that no nodes are overloaded in the network.This means that all calls that arrive at the network are accepted. The second profile is *High* traffic, in which the arrival rate is high enough to cause overload in one or several nodes. This means that some calls will be rejected.

In the balanced distribution, the load is evenly shared among the nodes. This means that a network using this distribution can have a higher arrival rate before it becomes overloaded than a network using the focused distribution. Therefore, the actual arrival rates for Low and High traffic depend on the SC distribution. The arrival rates are shown in Table 4.

6.4 Simulation cases

The load balancing algorithms described in Section 5 were examined for all combinations of algorithms and traffic profiles. Each case was simulated with both the balanced and focused SC distributions.

7. Results and discussion

This section contains the results from the simulations. Since the confidence intervals are small for all results, we have decided not to shown these. The resulting throughputs and setup times must be considered together, since a low throughput usually results in a low setup time for the calls that are finished without being rejected. Therefore, an algorithm with a low setup time is not always the best algorithm.

Table 5
Results for the balanced SC distribution

	Low Traffic		High traffic	
	Throughput	Setup time (sec)	Throughput	Setup time (sec)
Random	100%	0.13	75%	0.31
Shortest queue	100%	0.11	70%	0.20
Acc. prob.	100%	0.13	75%	0.31
Load status	100%	0.12	70%	0.48

7.1 Results with the balanced SC distribution

The results for the balanced SC distribution are shown in Table 5. During Low traffic, all cases have 100% throughput. However, the Shortest queue algorithm has the lowest setup times. During High traffic, the algorithm with Acceptance probabilities and the Random algorithm behave similarly. The Shortest queue algorithm has low setup times, though the throughput is lower as well.

During High traffic, the algorithm with Load status values behaves strangely. The throughput is rather low, which usually result in shorter setup times as well. However, the setup times are much higher than for the other algorithms. The reason for this is that, compared to the other algorithms, the calls are rejected later in the setup phase. This results in more wasted capacity which means lower throughput and longer setup times for the calls that are finished. This means that the algorithm with load status values cannot be considered good when the SC distribution is balanced, since it behave worse than the Random algorithm.

7.2 Results with the focused SC distribution

The results for the focused SC distribution are shown in Table 6. Here, the difference between the algorithm with Acceptance probabilities and the algorithm with Load status values is seen more clearly. In the case with High traffic, the node with a high load will get a small load status value in the other nodes. The nodes with a low load will get high load status values, since they never reject any messages. Thereby, these nodes will receive relatively more traffic and this helps the node with high load (node 6). In the algorithm with Acceptance probabilities, the acceptance probabilities are compared for each node. Since, node 6 has about 90% acceptance probability, and the nodes have about 100% acceptance probability, there is not much difference, which means that the traffic will be distributed rather evenly among the nodes. Therefore, the algorithm with Acceptance probabilities and the Random algorithm have a similar behaviour during high traffic. During low traffic, all three algorithm behave similarly. This because the nodes reject very few messages, which means that the algorithms will distribute the traffic evenly among the nodes.

Further, the Shortest queue algorithm minimises the setup time, however it cannot maintain a high throughput in the case with high traffic. This is probably due to the fact that it is not certain that a node with a short queue has a low load. During low traffic the Shortest queue algorithm is definitely the best algorithm since it has both the highest throughput and the lowest setup time.

Table 6
Results for the focused SC distribution

	Low Traffic		High traffic	
	Throughput	Setup time (sec)	Throughput	Setup time (sec)
Random	97%	0.14	45%	0.26
Shortest queue	100%	0.11	67%	0.18
Acc. prob.	98%	0.14	49%	0.25
Load status	98%	0.14	76%	0.30

8. Alternative load balancing algorithms

As can be seen in the results it is feasible to improve the network performance in TINA by introducing simple load balancing algorithm. The algorithms we have suggested need no extra communication between the nodes, since they only consider the rejected messages from other nodes. However, one problem with these types of algorithms is that the nodes have no complete knowledge about the traffic situation (see, for example Kihl and Nyberg [13]). If for example, a node has not sent any messages to a specific node for a long time, it has no knowledge about the load situation on that specific node. Therefore, it is very difficult to develop a load balancing algorithm that behave well for all load situations if we assume that the nodes cannot exchange any load status information.

However, the main problem with these simple algorithms is that they only work when there is overload in the network. If there is a normal load situation in the network, that is no overload, there are no rejected messages to use in the load balancing algorithms. This means that the traffic is distributed evenly among the nodes. However, also during normal load situations it would be feasible to improve the setup times by sending relatively more traffic to lightly loaded nodes. This requires that the nodes exchange load information, which means that it is necessary to add the feasibility of load status communication to the protocols used in TINA. One simple way to implement this is to use a load status field in all messages. This is already implemented in the Automatic Congestion Control (ACC) used in telephone networks (see ITU-T [16]). Here, the node adds its load status to all messages it sends to the other nodes. In ACC this load status is defined to be zero, one or two, where zero means that there is no overload and two means that it is severe overload. Of course, it is better to use more load status values, for example 0-10 (Northcote and Rumsewicz [12]). The other nodes could use this load status value to update the parameters in a load balancing algorithm.

9. Conclusions

In TINA, instances of the same service component can be placed on different physical nodes. Therefore, the network performance can be improved by introducing load balancing algorithms. These algorithms should distribute the traffic among the SC instances in such way that the overall throughput and setup time are improved.

We have examined four simple load balancing algorithms. The first algorithm randomly distributes the traffic, with the same probability for all instances. The second one sends a call to the node with the shortest job buffer. The third and fourth algorithms use the number of rejected messages to decide an appropriate traffic distribution. All algorithms except the second one can easily be implemented in a TINA network, since they do not require any extra communication between the nodes.

The results show that it is difficult to find a simple algorithm that behave well for all traffic situations. The main problem is that the algorithms have not enough information about the load situation on the different nodes, since no load information is exchanged between the nodes. This problem can be solved by adding the feasibility of load status information to the TINA protocols.

References

1. M. Chapman and S. Montesi, "Overall Concepts and Principles of TINA", TINA Consortium, 1995.
2. P. Graubmann, W. Hwang, M. Kudela, K. MacKinnon, N. Mercouroff and N. Watanabe, "Engineering Modelling Concepts (DPE Architecture)", TINA Consortium, December 1994
3. R. Minetti and E. Utsunomiya, "The TINA Service Architecture", Proceedings of the TINA Workshop at TINA'96 Conference, Heidelberg, Germany, 1996
4. R. Minerva, "TINA Service Architecture: some Issues in Service Control", Proceedings of the TINA'95 Conference, Melbourne, Australia, 1995.
5. L. Kristiansen, "TINA Service Architecture", version 5.0, TINA Consortium, 1997.
6. O. Kremien and J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, No. 6, Nov. 1992.
7. T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", *IEEE Trans. on Software Engineering*, Vol.17, No.7, July 1991.
8. S. Jordan and P. Varaiya, "Control of Multiple Service, Multiple Resource Communication Networks", Proceedings of Infocom'91, Bal Harbour, Florida, USA, 1991.
9. S. Ramakrishnan, I. Cho and L.A. Dunning, "A Close Look at Task Assignment in Distributed Systems", Proceedings of Infocom'91, Bal Harbour, Florida, 1991.
10. A. Parhar and M.P Rumsewicz, "A Preliminary Investigation of Performance Issues Associated with Freephone Service in a TINA consistent network", Proceedings of the TINA'95 Conference, Melbourne, Australia, 1995.
11. D.J. Houck, K.S Meier-Hellstern, F. Saheban and R.A. Skoog, "Failure and Congestion Propagation Through Signalling Controls", Proceedings of the 14th International Teletraffic Congress, Juan-les-Pines, France, 1994.
12. B. Northcote and M. Rumsewicz, "An Investigation of Tandem Overload Control Issues", Proceedings of the International Conference on Communications, Seattle, USA, 1995.
13. M. Kihl and C. Nyberg, "Investigation of overload control algorithms for SCPs in the Intelligent Network", *IEE Proceedings*, Vol. 144, No. 6, Dec 1997, pages 419-424.
14. M. Kihl, C. Nyberg, H. Warne and P. Wollinger, "Performance Simulation of a TINA Network", Proceedings of Globecom'97, Phoenix, Arizona, USA, 1997.
15. M. Kihl, "On Overload Control in TINA Networks", Proceedings of the 6th IEE Conference on Telecommunications, Edinburgh, United Kingdom, 1998.
16. ITU-T, Recommendation Q.723 "Specifications of Signalling System Number 7".
17. ITU-T, Recommendation X.900, "Basic Reference Model for Open Distributed Processing (RM-ODP)", 1993.
18. N. Widell, "STINA - Performance Simulation of TINA Networks", Master Thesis, Dep. of Communication Systems, Lund Institute of Technology, Sweden, 1998.