Multi-resolution Network Simulations using Dynamic Component Substitution*

Dhananjai M. Rao and Philip A. Wilsey
Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221–0030.
dmadhava@ececs.uc.edu, philip.wilsey@uc.edu

Abstract

Modeling and simulation of large, high resolution network models is a time consuming task even when parallel simulation techniques are employed. Processing voluminous, detailed simulation data further increases the complexity of analysis. Consequently, the models (or parts of the models) are abstracted to improve performance of the simulations by trading-off model details and fidelity. However, abstraction defeats the purpose of studying high resolution network models and magnifies the problems of validation! An alternative approach is to dynamically *(i.e., during the course of simulation) change the resolution of the model (or parts of the model). In our component based Network Modeling and Simulation Framework (NMSF), we have enabled dynamic changes to the resolution of a model using a novel methodology called* Dynamic Component Substitution *(DCS). Using DCS, a set of components can be substituted by a functionally equivalent component (or vice versa) to change the resolution (or the level of abstraction) of a network model. DCS improves the overall efficiency of simulations through dynamic tradeoffs between resolution of a model, simulation performance, and analysis overheads. This paper presents an overview of DCS and the issues involved in enabling DCS in NMSF, an optimistically synchronized parallel simulation framework. The experiments conducted to evaluate the effectiveness of DCS are also illustrated. Our studies indicate that DCS provides an effective technique to considerably improve the overall efficiency of network simulations.*

1 Introduction

Computer and communication networks have steadily grown in size and complexity to meet the growing needs and demands of modern computing [7]. Today's networks involve complex interactions between a few thousand to several million networking components. Study and analysis of modern networks is usually performed using computer based simulations. Parallel simulation techniques are often employed to enable simulation of large network models in acceptable time frames [3]. In simulation studies, the validity of the models plays a crucial role [9]. The models should reflect the size and complexity of the system in order to ensure that crucial scalability issues do not dominate during validation of simulation results [7, 10]. High resolution models are important for studying events that are rare — events that do not even occur in small, low fidelity models may be common in the actual network [7]. Detailed simulation of the complete network is necessary to study large scale characteristics, long term phenomena, and to analyze the network as a whole [9].

However, simulation of high fidelity, high resolution models of large networks is a time consuming task even when parallel simulation techniques are exploited [9]. Furthermore, study of large networks is typically conducted in phases [3], wherein each phase focuses on a particular aspect or portion of the network. In such studies, detailed simulation data from other parts of the network model is inconsequential (depending on the analysis requirements) [3, 10]. Processing voluminous, inconsequential simulation data further aggravates the problems associated with modeling, simulation and analysis. Consequently, parts of the network model that are not critical for a given study are suitably abstracted to minimize inconsequential simulation results and to improve simulation time [3]. Abstraction of selected parts of a model is an effective technique to optimize the overheads associated with modeling, simulation, and analysis by trading-off resolution or details. However, abstraction results in low resolution and possibly low fidelity network models [3, 10] which not only defeats the goal of studying detailed models but also brings us back to the problems of validity of the simulation!

An alternative approach to improve the overall efficiency of large, high resolution network simulations is to *dynamically* (*i.e.,* during the course of simulation) change the resolution (or "level of abstraction") of selected parts of a network model. In this methodology, the resolution of the model is dynamically altered to suit the needs of the

*Support for this work was provided in part by the Ohio Board of Regents.

0-7695-1315-8/01 $10.00 © 2001 IEEE

simulation study – scenarios of interest (or parts of the model) can be simulated in high resolution while the remainder of the simulation (or the model) can be simulated in low resolution. In our component based network modeling and parallel simulation framework, we have enabled dynamic changes to the resolution of a model through a novel methodology called *Dynamic Component Substitution* (DCS). Using DCS, a *set* of components (constituting a network model) can be substituted with *a* (functionally) equivalent component (or vice versa) during simulation, to achieve dynamic changes to the resolution of a model. DCS improves the overall efficiency of a simulation by enabling dynamic tradeoffs between several parameters such as: cost of modeling, resolution (or details) of simulation data, and simulation overheads.

This paper presents an overview of DCS along with the issues involved in enabling DCS in an optimistically synchronized (based on Time Warp) parallel network simulation framework. Section 2 presents an overview of some of the closely related research activities. An overview of DCS along with a brief description of the component based modeling methodology used in this study are presented in Section 3. To ease study and effective use of DCS, an existing Network Modeling and Simulation Framework (NMSF) [9] has been extended to provide support for DCS. An overview of the NMSF is presented in Section 4. The issues involved in extending NMSF to support DCS are discussed in Section 5. The results obtained from some of the experiments conducted to evaluate the overall effectiveness of DCS are presented in Section 6. Section 7 concludes the paper and presents some pointers to future work.

2 Related Research

A number of studies have been reported on selectively abstracting parts of a model to enable efficient tradeoffs between several model and simulation related parameters, such as: model resolution, fidelity, and simulation performance. In this section we present a brief overview of some of the closely related research activities. One of the recent studies on selective abstraction was presented by Huang *et al* [3]. In their work, they present two abstraction techniques for abstracting network and transport layer protocols. They apply the abstraction techniques to the simulation study of reliable multicast protocols. Their studies indicate that, although the abstract simulations are not identical to more detailed simulations, the abstract models provide good approximations and considerably improve simulation performance [3]. Ahn *et al* [1] demonstrate that abstraction can be employed to adjust the simulation granularity of packet network models in order to efficiently study flow and congestion control algorithms. Hybrid simulation models, wherein the network model is a combination of discrete-event and analytical components, have been used to yield efficient, yet accurate simulations [11]. Selective abstraction of models have also been applied to other domains of simulations, particularly for digital logic simulations. Levelized code compilation techniques, that selectively replace parts of combinatorial logic circuits with equivalent behavioral descriptions, are widely used to improve performance of circuit simulations [12]. McBrayer *et al* have described techniques for combining processes descriptions specified in VHDL to yield more abstract models, in order to improve performance of parallel logic simulations [5]. These techniques are all *static*, *i.e.,* selective abstraction of the model is done prior to commencement of the simulation; while, in this study we propose to dynamically change the resolution of selected parts of the model. The use of mixed resolution models to enable effective simulation and analysis of large systems have also been studied [2, 6]. Natrajan *et al* [6] present the issues involved in the use of Multiple Resolution Entities (MRE) in parallel simulations. MREs are entities that are capable of maintaining internal consistency across multiple, concurrent levels of resolution. In other words, a MRE is a model of a sub-system, that is capable of modeling the behavior of the sub-system at different levels of resolution. On the other hand, we propose to dynamically substitute a given set of components with an equivalent component (or vice versa) to change the resolution (and consequently the level of abstraction) of a model.

3 Dynamic Component Substitution

Component based modeling techniques are widely used because they offer a number of advantages [9]. In a component based model, a system is represented as a set of interconnected *components*. A component is a well defined entity which is viewed as a "black box", *i.e.,* only its interface is of interest and not its implementation. A component could in turn be specified using a set of sub-components. During simulation, each *atomic* component is associated with a specific, well defined software module called a *simulation object* or a *logical process* (LP) that implements its behavior and functionality. In such models, a set of components can be substituted by a functionally equivalent component (or vise versa), without altering the basic characteristics of the model. For example, consider the network model shown in Figure 1(a), consisting of 9 nodes interconnected by 3 switches (labeled $1 \cdots 12$); logically organized as 3 sub-networks. Figure 1(b) illustrates a typical component based simulation layout for the network model shown in Figure 1(a). The interconnection between the networking entities (shown as black lines in Figure 1(a)) are represented using logical links (shown as gray lines in Figure 1(b) and Figure 1(c)); *i.e.,* the logical links indicate the pattern in which the components communicate (*i.e.,* exchange events)

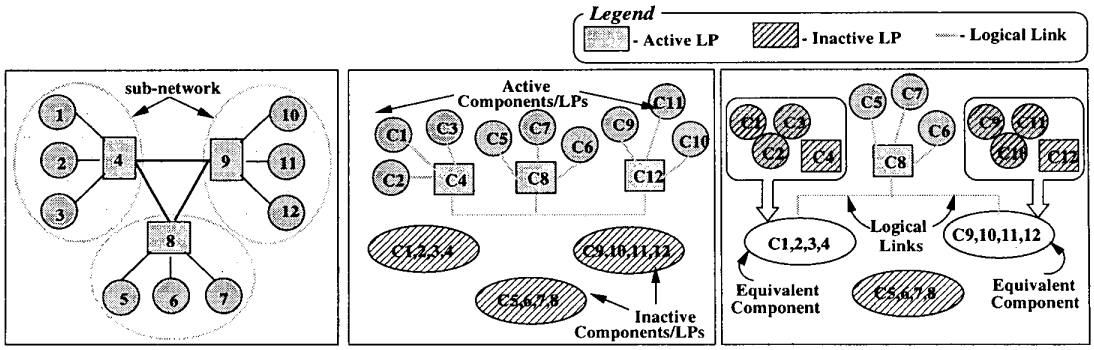


Figure 1. A sample network, typical component based model, and changes induced by DCS

during simulation.

Figure 1(b) also illustrates equivalent components (labeled $C_{1,2,3,4}$, $C_{5,6,7,8}$, and $C_{9,10,11,12}$) for the three sub-networks constituting the model. An equivalent component is capable of closely imitating (within acceptable error margins) the behavior and functionality of the set of components it represents. For example, as shown in Figure 1(c), the component $C_{1,2,3,4}$ is equivalent in behavior and functionality to the set of components $\{C_1, C_2, C_3, C_4\}$. Figure 1(c) illustrates a scenario that could arise when a given set of components are substituted by their equivalent component. The set of components $\{C_1, C_2, C_3, C_4\}$ and $\{C_9, C_{10}, C_{11}, C_{12}\}$ have been substituted by the corresponding equivalent components. As illustrated by Figure 1(b) and Figure 1(c), substitution of components involves updating the logical links between the components and possibly updating the states of the newly activated (or created) components to reflect the current state of the simulation. The simulation configuration in Figure 1(b) is a higher resolution equivalent of the simulation configuration shown in Figure 1(c). It must be noted that the comparison of resolution (or level of abstraction) is only relative and not absolute.

Substitution of components may be done *statically* (*i.e.*, prior to simulation) or *dynamically i.e.*, during the course of simulation [9]. Static component substitution is widely used in different flavors, to address capacity and performance issues of large scale simulations [3, 5, 12]. They are also prevalently used for "What-if" type of simulation analysis. The primary drawback of static component substitution is that functionality, observability, and model details cannot be altered during simulation. However, resolution and fidelity are crucial for effectively studying large scale networks. On the other hand, substituting components during simulation provides a dynamic tradeoff between model details and performance of the simulation. Dynamic Component Substitution (DCS) not only encompasses the utility of its static counterpart but also provides a number of other useful features [9]. It can be used to dynamically change the resolution of selected parts of a model; thereby optimizing simulation overheads and volume of inconsequential simulation data. DCS provides an effective solution for improving the overall efficiency of large scale network simulation and analysis. The improvement in efficiency is achieved by striking a tradeoff between several modeling and simulation related parameters [9]. However, enabling support for DCS in a parallel simulation environment involves additional overheads during kernel and model development. The issues involved in the design and development of a parallel simulation framework capable of supporting DCS is presented in the following sections.

4 Network Modeling and Simulation Framework (NMSF)

A simulation framework capable of supporting DCS has been developed by suitably extending an existing Network Modeling and Simulation Framework (NMSF) [9]. Although, a detailed description of the framework is available in the literature [9], a brief description of NMSF is presented in this section for completeness of this paper and for the discussion (in Section 5) on extending NMSF to support DCS. Figure 2 presents an over view of NMSF. As shown in Figure 2, the primary input to the framework is the model (or topology) of the network to be simulated. The topology of the model is described using the Topology Specification Language (TSL) [9]. TSL is a component based, modular network topology modeling language wherein a network is specified as a set of interconnected networking components (such as traffic generators, nodes, and routers). The components are developed using the Application Program Interface (API) provided by NMSF (as shown in Fig-

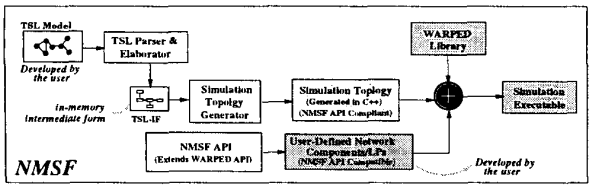


Figure 2. Overview of NMSF

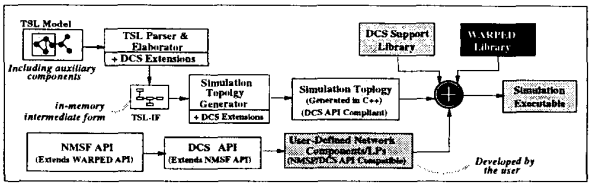


Figure 3. NMSF with DCS Extensions

ure 2). As shown in Figure 2, the input TSL description is parsed into an object-oriented in-memory intermediate form called TSL-IF. TSL-IF is used by a backend code-generator to generate a simulatable network topology. The generated code contains the necessary calls to instantiate the various components (developed using the NMSF API), pass parameters specified in the TSL description to them, configure, and establish the simulation. The generated code, in conjunction with all the other components of NMSF, is in C++. The generated code is compiled along with the WARPED [8] (a parallel simulation kernel) library and the user-defined modules to obtain the final executable; which when run performs the actual simulation. A more detailed description of the different components relevant to this study are presented in the following subsections, while detailed descriptions of the other components constituting NMSF are available in the literature [9].

4.1 Topology Specification Language (TSL)

TSL provides a hierarchical, component based modeling techniques for specifying the topology of a network for simulation. A TSL specification consists of a set of interconnected sub-topology specifications. Each sub-topology specification consists of three main sections, namely the *object definition section*, the *object instantiation section*, and the *netlist section*. The object definition section contains the details of the components (such as the name of the C++ class used to model the component along with necessary parameters) used in the sub-topology. The object instantiation section specifies the set of components constituting the sub-topology. Each object instantiation must be associated with an object definition. The object definitions and instantiations are synonymous to defining a new type and variables of a given type in a programming language. The netlist section defines the interconnectivity (or communication pattern) between the component instantiations. An optional *label* may be associated with each sub-topology. The labels may be used as an object definition in subsequent sub-topology specifications to nest one sub-topology within another. In other words, a sub-topology encapsulates a set of interconnected components and provides a predefined interface to them. When sub-topologies are nested within one another, they get interconnected through the predefined interface to form larger networks. Hierarchical topologies are elaborated or "flattened" prior to simulation [9]. An example TSL source code is presented in Section 5 while further details on TSL is available in the literature [9].

4.2 WARPED

The parallel simulation capabilities of NMSF have been enabled using WARPED [8]. WARPED is an API for a general purpose discrete event simulation kernel with different implementations [8]. NMSF utilizes the Time Warp synchronized simulation kernel of WARPED for parallel simulation. A Time Warp synchronized simulation is organized as a set of communicating asynchronous logical processes (LPs). The LPs operate as asynchronous discrete event simulators and communicate between each other by exchanging *virtual time*-stamped event messages [4]. Virtual Time is used to model the passage of time and defines order on the events in the system. Accordingly, the WARPED kernel [8] provides an API to build different LPs with unique definitions of state. It provides the basic functionality for sending and receiving events between LPs. Control is exchanged between the application and the kernel through cooperative use of function calls. In WARPED, LPs are placed into groups called "clusters" to optimize communication overheads. Although, LPs are grouped together into clusters they are not coerced into synchronizing with each other. Causal violations are detected by a LP when it receives an event with time-stamp lower than its local time (LVT). Such events are called *straggler* events. On receiving a straggler event, a *rollback* mechanism [4] is invoked to recover from the causality error. The WARPED kernel insulates the application from the intricacies of rollbacks, optimistic synchronization, and other simulation overheads [8].

5 Implementing Support for DCS in NMSF

A simulation environment capable of supporting DCS was developed by suitably extending the NMSF. Care has been taken to ensure that the extensions do not invalidate any existing models for the NMSF. Figure 3 presents an overview of NMSF along with the modifications for supporting DCS. As shown in Figure 3, the first phase of im-

```
subNet : abstSubNet "nodes=3 trGens=3" {
    node : simpleNode "CBR size=50";
    router : Router;
    trGen : Generator "const delay=1ms pkts=50"; }
{ n1, n2, n3 : Node;
    tr1, tr2, tr3 : trGen;
    r1 : router; }
{ t1 : n1; t2 : n2; t3 : n3;
    n1 : r1; n2 : r2; n3 : r3;
    r1 : n1 n2 n3 subNet; }
```

```
mainNet {
    controller : DCSController;
    router : Router; }
{ ctrl1 : controller "10ns trigger";
    ctrl2 : controller "20ns trigger";
    subNet1, subNet2,
        subNet3 : subNet;
    r2 : router; }
{ ctrl1 : $subNet1;
    ctrl2 : $subNet2;
    r2 : subNet1 subNet2 subNet3;
}
```

Figure 4. Sample TSL source

plementing support for DCS in NMSF involved extensions to the modeling infrastructure. Changes to the modeling infrastructure involved extending TSL and correspondingly modifying the parser, elaborator, and the backend code-generator. TSL was extended to include additional constructs for associating a set of components with an equivalent component. This was achieved by permitting an *auxiliary* object definition to be associated with each sub-topology in a TSL description. On encountering an auxiliary object definition an auxiliary object instantiation is implicitly created (in TSL-IF). The auxiliary component represents the higher level abstraction (or lower resolution model) of the set of components contained in the sub-topology. In other words, when DCS is triggered for a sub-topology, the set of components encapsulated by that sub-topology are substituted using by the auxiliary component or vice versa, as the case maybe.

The TSL source code for the network model shown in Figure 1(a) is shown in Figure 4. The source code also includes auxiliary component specifications and references (shown in bold, in Figure 4) for the sub-topologies constituting the network model. As shown in Figure 4, each `subNet` is associated with an auxiliary component – an abstract sub network (`abstSubNet`). The main network (`mainNet`) consists of three instances of `subNet`. It also includes specifications for controller components that are used to trigger DCS during simulation. It must be noted that, the controllers are not special components but just regular components (developed using NMSF API and an user can modify them to suit the needs) that are geared to perform specific tasks. Each controller is interconnected to the corresponding auxiliary components using suitable netlist entities (such as "`ctrl1: $subNet1;`").

TSL-IF was also extended to correspondingly reflect the changes to the grammar. As shown in Figure 3, the elaborator was also modified to account for the auxiliary components. The elaborator also creates unique instances of the auxiliary components for each unique occurrence (or usage) of a sub-topology. The auxiliary components are an integral part of the elaborated TSL-IF and are identified using special flags in the various data structures. The elaborator was also extended to identify components that form the interface for a sub-topology. For example, the component `r1` shown in Figure 4 is the primary interface for the sub-topology `subNet`. In other words, any interaction with a sub-topology occurs through the primary interface components. It must be noted that several components could constitute the primary interface. This information is utilized during simulation to optimally update netlist entries (or communication links) during DCS (as explained in detail further below). The elaborator also collates information on the set of components contained by each sub-topology. The data collated by the elaborator is embedded into the generated code. The data is utilized during simulation (by the DCS support library modules shown in Figure 3) to achieve DCS.

The second phase of implementing support for DCS involved extending the simulation infrastructure of NMSF. As illustrated in Figure 3, the API supported by NMSF was extended to include necessary support structures for DCS. The DCS API extends the NMSF API such that the modifications do not invalidate existing models and eases use of DCS by insulating the models from the intricacies of enabling DCS. The API classes also provide necessary interfaces for the DCS support library modules that perform the actual task of achieving DCS. As shown in Figure 3, a support library that provides the necessary simulation-time support for enabling DCS has also been developed. These modules not only implement several interface methods of the DCS API but also perform the actual tasks of achieving DCS. In NMSF, an event driven approach has been adopted for sequencing the different stages involved in achieving DCS. The event driven mechanism was utilized because it offers several advantages. The primary advantage is that, it exploits the inherent simulation capabilities of the underlying kernel; thereby abstracting away the intricacies of parallel simulation. On the other hand, the drawbacks of this design are: (i) it introduces additional events (to achieve DCS) during simulation; and (ii) it adds to the state saving overheads in a Time Warp simulation. However, several Time Warp related optimizations can be exploited to minimize these overheads.

A typical sequence of operations performed to achieve DCS are shown in Figure 5. The figure also illustrates the corresponding sequence of transformations that occur to the model during the different phases. As shown in Figure 5, the initial phase involves triggering DCS in the simulation by scheduling an `Activate` or a `DeActivate` event, as the case may be, to the corresponding auxiliary component(s). On receiving an `Activate` or a `DeActivate` event, the auxiliary component schedules corresponding events to the set of LPs (or components) that it is going to substitute. The information on the set of components to be replaced is collated during elaboration and is passed onto the DCS modules (as explained earlier). The necessary state data is also passed on by the deactivating

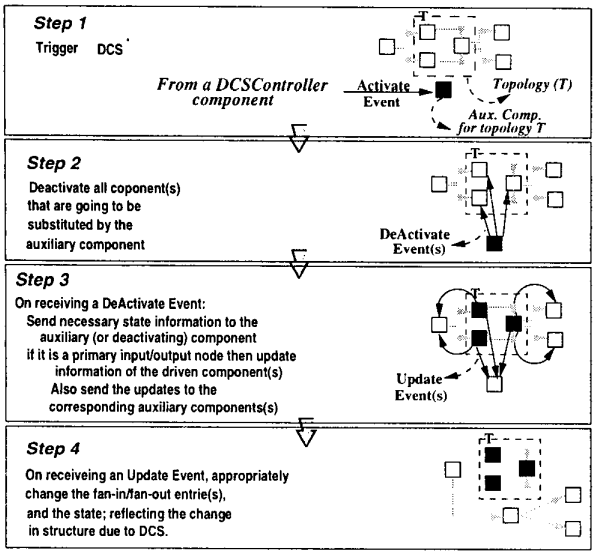


Figure 5. Sequence of operations during DCS

component(s) to the activating component(s). A simple API has been developed for enabling the transfer of states from one component to another. It is the modeler's responsibility to suitably map the states of the various components. In the next phase, necessary `Update` events are scheduled . During the last phase, the `Update` events are processed wherein the LPs update their netlists and states, reflecting the change in structure. As shown in Figure 5, during subsequent simulation cycles, the events generated would be passed on to the new components while the old components get deactivated. It must be noted that the transient events that were already scheduled for the old set of LPs do not get reassigned to the new set of components. They continue to get processed by the substituted (or deactivated) set of components.

6 Experiments

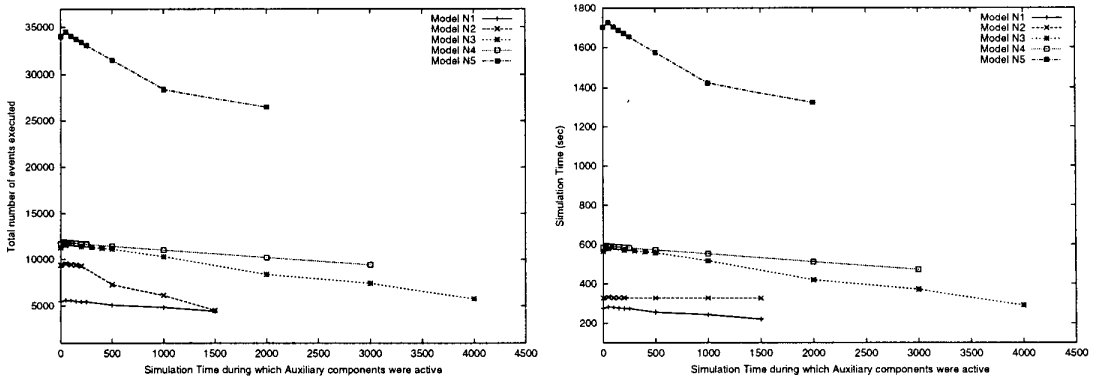
The experiments conducted to evaluate the support for DCS were performed using a set of network models. The network models were built using the hierarchical modeling constructs provided by TSL. The models consisted of a set of interconnected sub-networks. Each sub-network contained a set of nodes connected to a router. The nodes were driven by a set of traffic generators. In the experiments the nodes and the traffic generators were configured to yield a Constant Bit Rate (CBR) type of network traffic with a packet size of 500 bytes. The router component used in the experiments behaves as a typical router that stores and forwards the packets generated by the nodes to the corresponding destination nodes. Interconnectivity between routers is specified using suitable netlist entries in the TSL description of the network model. The routers build necessary routing tables when the simulation is initialized (*i.e.,* just before simulation commences) by exchanging information regarding the set of nodes connected to them. A more detailed description of these components is available in the literature [9].

Model Name	Number of Sub Nets.	Number of Components per subNet		
		Nodes	Traffic Gens.	Routers
N1	3	3	3	1
N2	10	5	5	1
N3	5	3	3	1
N4	50	5	5	1
N5	100	6	6	2

Table 1. Models used in experiments

An abstract sub-topology component that is capable of reflecting the behavior of a sub-network built using the router, node, and traffic generator components was also developed. This component is essentially a router that is also capable of generating network traffic similar to the traffic generated by the nodes in the sub-topology. For example, if a sub-network contained 3 nodes, each node generating traffic at a CBR of 500 bytes/ms, the abstract sub-topology component would *internally* generate 3 packet streams at a CBR of 500 bytes/ms. The necessary information is provided when DCS is triggered (as illustrated in Section 5). The set of events generated by the abstract (or lower resolution) component and the detailed sub-network are equivalent. In other words, the abstract sub-topology component is an equivalent lower resolution model of a sub-topology. An example of a TSL source utilizing these components is shown in Figure 4.

The above described networking components were used to develop different network models by specifying different topologies using TSL. The salient characteristics of the models using in the experiments is shown in Table 1. The TSL descriptions also included equivalent abstract sub-network component (or auxiliary component) specifications for the different sub-topologies. The number of components (*i.e.,* nodes, traffic generators, and routers) substituted by each auxiliary components is shown in Table 1 (column `Number of Components per subNet`). All the simulation experiments were conducted using a network of shared memory multi-processor (SMP) workstations running Linux. Each workstation consisted of two 300MHz Pentium II processors with 128MB of main memory. The workstations were inter-connected using fast Ethernet. Validity of the models and the simulations were verified by comparing the event traces obtained from the top most hierarchical level of each network model. Since, the top level did not involve any DCS, the event traces at that level remain the same, immaterial of the resolution of the sub-



(a) Number of events versus duration of DCS (b) Simulation Time versus duration of DCS

Figure 6. Effect of duration of DCS on number of events and simulation time

networks.

The graph in Figure 6(a) presents the change in the total number of events processed with respect to the duration of simulation time in which the auxiliary components (or lower resolution models) were active. These statistics were collated from the experiments conducted using a single processor where in no rollbacks occur. The data points shown with zero durations did not involve the use of lower resolution modules and represent the basic number of events executed by each model. As illustrated by the graphs shown in Figure 6(a), for short durations during which the auxiliary components are active, the total number of events processed is higher. The increase in number of events is due to the additional events used during DCS. However, as the duration during which the lower level abstractions are active increases, the number of events processed decreases (as shown in Figure 6(a)). The number of events decreases because a set of components are replaced by a single component which results in the elimination of a number intermediate events. Figure 6(b) presents the corresponding simulation times for the different models. As shown by the graphs in Figure 6, the simulation time is proportional to the number of events. However, the change in simulation time for smaller models is not very pronounced because of the low event granularities. As shown in Figure 6(b), the gains in simulation time accrued by utilizing the lower resolution component is significant for the larger models. As illustrated by the graphs in Figure 6, the duration of simulation time for which DCS reduces the number of events varies with respect to the model characteristics. If DCS is triggered at a rate faster than this value the overall simulation time would increase and vice versa. Therefore, this threshold value plays a crucial role in the overall effectiveness of DCS to improve performance of the simulations.

Figure 7 presents the time for simulating model N5 in parallel using a varying number of processors. The LPs were randomly partitioned across the different processors used for simulation. The timing information shown in the graph is the average of 10 simulation runs. As illustrated by the graph, the performance of the simulations increases as the duration during which the auxiliary components are active increase. The improvement in performance is due to the decrease in the total number of events that need to be processed (as shown in Figure 6(a)). As illustrated by Figure 7, the parallel simulations performed using 3 processors performs better than those performed using a single processor. The performance improves because the simulation overheads get distributed across the three processors. In the two processor case the computational overheads dominate the gains accrued by parallel simulation. On the other hand, in the 4 processors case, communication overheads dominate the performance gains. Hence, in these cases the overheads outweigh the gains accrued by employing parallel simulation and the performance of the simulations do not improve. As illustrated by Figure 6 and Figure 7 the overall efficiency of network simulations can be effectively improved using dynamic component substitution.

7 Conclusions

Dynamic change to the level of abstraction of a model (*i.e.,* during simulation) enables more optimal tradeoffs between the resolution (or observability) of a model, model details, and simulation performance. Dynamic Component Substitution is a novel methodology for achieving abstraction of selective parts of a component based model during simulation. In this study, DCS has been applied to improve the overall efficiency of network simulations. The paper presented the issues involved in the design and implementation of the support for DCS in an existing network mod-

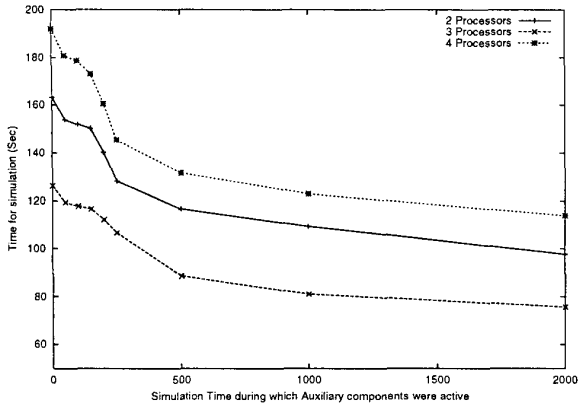


Figure 7. Time for Parallel Simulation

eling and simulation framework. A similar approach can be adopted for implementing support for DCS in other network simulators. The experiments in which DCS was used to dynamically change the level of abstraction of the model were described. As illustrated by the results obtained from the experiments, DCS may not always improve simulation performance (which may not be the goal of DCS). For instance, if DCS is triggered very frequently, the overall performance of the simulations may deteriorate. Moreover, reduction in the level of abstraction may not necessarily improve performance of the simulations, parallel simulations in particular. Furthermore, changes in the levels of resolution may introduce errors into the simulation data. Hence, care must be taken while applying DCS. The use of DCS also involves the development of valid components at different levels of abstraction. Albeit some of the modeling overheads, our studies coupled with the experiments presented in this paper highlight that considerable improvements in the overall efficiency of networks simulations can be accrued by employing DCS.

References

[1] AHN, J., AND DANZIG, P. B. Speedup vs. simulation granularity. *IEEE/ACM Transactions on Networking 4*, 5 (Oct. 1996), 743–757.

[2] DAVIS, P. K., AND HILLESTAD, R. J. Families of models that cross levels of resolution: Issues for design , calibration and management. In *Proceedings of the 1993 Winter Simulation Conference* (1993), ACM.

[3] HUANG, P., ESTRIN, D., AND HEIDEMANN, J. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *In Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Networks* (Oct. 1998).

[4] JEFFERSON, D. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3 (July 1985), 405–425.

[5] MCBRAYER, T., AND WILSEY, P. A. Process combination to increase event granularity in parallel logic simulation. In *9th International Parallel Processing Symposium* (Apr. 1995), pp. 572–578.

[6] NATRAJAN, A., REYNOLDS, P. F., AND SRINIVASAN, S. MRE: A flexible approach to multi-resolution modeling. In *Proceedings of the 11th Workshop on Parallel and Distributed Simu lation (PADS'97)* (June 1997), pp. 156–163.

[7] PAXSON, V., AND FLOYD, S. Why we don't know how to simulate the internet. In *In Proceedings of the 1997 Winter Simulation Conference (WSC'97)* (Dec. 1997), pp. 44–50.

[8] RADHAKRISHNAN, R., MARTIN, D. E., CHETLUR, M., RAO, D. M., AND WILSEY, P. A. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, D. Caromel, R. R. Oldehoeft, and M. Tholburn, Eds., vol. LNCS 1505. Springer-Verlag, Dec. 1998, pp. 13–23.

[9] RAO, D. M. A network simulation tool kit. Master's thesis, University of Cincinnati, Aug. 2000.

[10] ROBINSON, S. Simulation model verficiation and validation: Increasing the users' confidence. In *Proceedings of the 1997 Winter Simulation Conference* (Dec. 1997).

[11] SCHWETMAN, H. D. Hybrid simulation models of computer systems. *Communications of the ACM 21*, 9 (1979), 718–723.

[12] WANG, Z., AND MAURER, P. M. Lecsim: A levelized event driven compiled logic simulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC'90)* (June 1990), pp. 491–496.